# Administering SQL Server

# Administering SQL Server Overview

Microsoft® SQL Server™ 2000 administration applications, and the accompanying services, are designed to assist the system administrator with all administrative tasks related to maintaining and monitoring server performance and activities.

| Topic | Description |
|---|---|
| Starting, Pausing, and Stopping SQL Server | Explains how to start an instance of SQL Server, and what you need to do before, during, and after you log in. |
| Failover Clustering | Describes how to set up and use a failover cluster. |
| Importing and Exporting Data | Describes how to retrieve data from external sources and feed data to other applications. |
| Backing Up and Restoring Databases | Describes how to protect and restore data over a wide range of potential system problems. |
| Using the Copy Database Wizard | Describes how to copy or move databases between servers and upgrade databases from SQL Server version 7.0 to SQL Server 2000. |
| Managing Servers | Describes how to register and configure remote and linked servers, add or remove servers, and modify server settings. |
| Managing Clients | Describes how to configure client connections with server components and change the default network protocol to meet the needs of your site. |
| Automating Administrative Tasks | Describes how to establish which administrative responsibilities will occur regularly, define jobs and alerts, and run SQL Server Agent. |

| | |
|---|---|
| Managing Security | Describes how to protect and safeguard database access by restricting permissions to include only authorized users. |
| Monitoring Server Performance and Activity | Describes how to develop a strategy for ensuring that server and activity performance are at acceptable levels. |
| Using the Web Assistant Wizard | Explains how to use the wizard to create Web pages. |

# Administering SQL Server

# Starting, Pausing, and Stopping SQL Server

Before you log in to an instance of Microsoft® SQL Server™, you need to know how to start, pause, and stop an instance of SQL Server. After you are logged in, you can perform tasks such as administering the server or querying a database.

## Using the SQL Server Service

When you start an instance of SQL Server, you are starting the SQL Server service. After you start the SQL Server service, users can establish new connections to the server. The SQL Server service can be started and stopped as a Microsoft Windows NT® 4.0 or Windows® 2000 service, either locally or remotely. The SQL Server service is referred to as MSSQLServer if it is the default instance, or MSSQL$*instancename* if it is a named instance.

## Using SQL Server Service Manager

If you are running Microsoft Windows 98, SQL Server Service Manager can be used start, pause, stop and check the state of local services, though it cannot remotely administer services.

If you have to restart your computer, SQL Server Service Manager appears automatically and the default service is displayed. It is possible to change the default service on the local computer through the SQL Server Service Manager. When you restart the computer, the default service will now be displayed in SQL Server Service Manager. For example, if you change the default service to SQL Server Agent service, and then shut down the computer, the next time you start it, SQL Server Agent service will be displayed in SQL Server Service Manager.

SQL Server Service Manager can also be used to start, pause, or stop an instance of SQL Server 2000 Analysis Services.

### To change the default service

# Administering SQL Server

# Starting SQL Server

You can start an instance of Microsoft® SQL Server™ automatically, manually, or from the command prompt. Both the automatic and manual methods start an instance of SQL Server as a Microsoft Windows NT® 4.0 or Windows® 2000 service. If you run **sqlservr** from a command prompt, you cannot pause, stop, or resume an instance of SQL Server as a Windows NT 4.0 or Windows 2000 service using any **net** commands.

## See Also

SQL Server Service Manager

Using Startup Options

# Administering SQL Server

## Starting SQL Server Automatically

During installation, you can configure Microsoft® SQL Server™ to start automatically in the following ways:

- You can configure an instance of SQL Server to start automatically each time you start the Microsoft Windows NT® 4.0 or Windows® 2000 operating system.

- You can configure a server running Microsoft Windows 98 to start automatically. Select the **Auto-start service when OS starts** check box in SQL Server Service Manager. Windows 98 does not have a component that corresponds to Window NT 4.0 and Windows 2000 services. The SQL Server database engine and SQL Server Agent run as executable programs on Windows 98. These SQL Server components cannot be started as services automatically.

  **Note** The SQL-DMO **AutoStartServer** property does not work with Windows 98.

- You can also use the Services application in Control Panel.

After SQL Server is installed, you can enable or disable the server configuration using SQL Server Enterprise Manager. For more information, see the Windows NT 4.0 and Windows 2000 documentation.

### To start an instance of SQL Server automatically

# Administering SQL Server

## Starting SQL Server Manually

You can start an instance of Microsoft® SQL Server™ manually using the following methods.

| Method | Description |
|---|---|
| SQL Server Enterprise Manager | Start, pause, continue, and stop an instance of a local or remote SQL Server or the SQL Server Agent service in the same window in which you administer other servers and databases. |
| SQL Server Service Manager | Start, pause, continue, and stop an instance of a local or remote SQL Server or the SQL Server Agent service. |
| Services application in Control Panel | Start, pause, continue, and stop an instance of SQL Server or the SQL Server Agent service on the local server. |
| Command prompt | Start an instance of SQL Server or the SQL Server Agent service from a command prompt by typing: **net start mssqlserver** or **sqlservr**, or **net start SQLServerAgent** or by running **SQLSERVR.EXE**. If you are referring to a named instance of SQL Server, you must specify **mssql$*instancename*** or **SQLAgent$*instancename***. |

Before you choose a startup method, consider the following:

- If you start an instance of SQL Server using **sqlservr** from a command prompt (independent of the Service Control Manager):
    - All system messages appear in the window used to start an instance of SQL Server.

    - You cannot pause, stop, or resume an instance of SQL Server as a Windows NT 4.0 or Windows 2000 service using SQL Server Enterprise Manager, SQL Server Service Manager, the Services application in Control Panel, or any **net** commands (for example, **net start**, **net pause**, **net stop**, and **net continue**).

    - You must shut down an instance of SQL Server before logging off Windows NT 4.0 or Windows 2000.
- If you start an instance of SQL Server from a command prompt:
    - Any command prompt options that you type take precedence over the default command prompt options written to the Windows 2000 registry by SQL Server Setup.

    - SQL Server Service Manager and SQL Server Enterprise Manager show the service as stopped.
- You can log off the Windows NT 4.0 or Windows 2000 network without shutting down an instance of SQL Server.

### To start the default instance of SQL Server

# Administering SQL Server

## Starting SQL Server in Single-User Mode

Under certain circumstances, you may need to start an instance of Microsoft® SQL Server™ in single-user mode using the startup option **-m**. For example, you may want to change server configuration options or recover a damaged **master** database or other system database. Both actions require starting an instance of SQL Server in single-user mode.

When you start an instance of SQL Server in single-user mode:

- Only one user can connect to the server.
- The CHECKPOINT process is not executed. By default, it is executed automatically at startup.
- The **sp_configure** system stored procedure **allow updates** option is enabled. By default, the **allow updates** option is disabled.

### To start SQL Server in single-user mode

# Administering SQL Server

## Starting SQL Server with Minimal Configuration

If you have configuration problems that prevent the server from starting, you can start an instance of Microsoft® SQL Server™ using the minimal configuration startup option. This is the startup option **-f**. Starting an instance of SQL Server with minimal configuration places the server in single-user mode automatically.

When you start an instance of SQL Server in minimal configuration mode:

- Only a single user can connect, and the CHECKPOINT process is not executed.
- Remote access and read-ahead are disabled.
- Startup stored procedures do not run.
- The **sp_configure** stored procedure **allow updates** option is enabled. By default, the **allow updates** option is disabled.

After the server has been started with minimal configuration, you should change the appropriate server option value or values, stop, and then restart the server.

**IMPORTANT** Stop the SQL Server Agent service before connecting to an instance of SQL Server in minimal configuration mode. Otherwise, the SQL Server Agent service uses the connection, thereby blocking it.

### To start SQL Server with minimal configuration

## Administering SQL Server

## Using Startup Options

When you install Microsoft® SQL Server™, SQL Server Setup writes a set of default startup options in the Microsoft Windows® 2000 registry. You can use these startup options to specify an alternate **master** database file, **master** database log file, or error log file.

| Default startup options | Description |
|---|---|
| **-d***master_file_ path* | The fully qualified path for the **master** database file (typically, C:\Program Files\Microsoft SQL Server\MSSQL\Data\Master.mdf). If you do not provide this option, the existing registry parameters are used. |
| **-e***error_log_ path* | The fully qualified path for the error log file (typically, C:\Program Files\Microsoft SQL Server\MSSQL\Log\Errorlog). If you do not provide this option, the existing registry parameters are used. |
| **-l***master_log_path* | The fully qualified path for the **master** database log file (typically C:\Program Files\Microsoft SQL Server\MSSQL\Data\Mastlog.ldf). |

You can override the default startup options temporarily and start an instance of SQL Server by using the following additional startup options.

| Other startup options | Description |
|---|---|
| **-c** | Shortens startup time by starting an instance of SQL Server independently of the Service Control Manager, so that SQL Server does not run as a Microsoft Windows NT® 4.0 or Windows 2000 service. |
| **-f** | Starts an instance of SQL Server with minimal configuration. Useful if the setting of a configuration value (for example, over-committing memory) has prevented the server from starting. |

| | |
|---|---|
| | Enables the **sp_configure allow updates** option. By default, **allow updates** is disabled. |
| **-g** | Specifies the amount of virtual address space (in megabytes) SQL Server will leave available for memory allocations within the SQL Server process, but outside the SQL Server memory pool. This is the area used by SQL Server for loading items such as extended procedure .dll files, the OLE DB providers referenced by distributed queries, and automation objects referenced in Transact-SQL statements. The default is 128 megabytes (MB).<br><br>Use of this option may help tune memory allocation, but only when physical memory exceeds 2 gigabytes (GB) for the SQL Server 2000 Personal Edition or SQL Server 2000 Standard Edition, or 3 GB for SQL Server 2000 Enterprise Edition. Configurations with less physical memory will not benefit from using this option. Use of this option may be appropriate in large memory configurations in which the memory usage requirements of SQL Server are atypical and the virtual address space of the SQL Server process is totally in use. Incorrect use of this option can lead to conditions under which an instance of SQL Server may not start or may encounter run-time errors.<br><br>Use the default for the **-g** parameter unless you see the following warning in the SQL Server error log:<br><br>WARNING: Clearing procedure cache to free contiguous memory<br><br>This message may indicate that SQL Server is trying to free parts of the SQL Server memory pool in order to find space for items such as extended stored procedure .dll files or automation objects. In |

| | |
|---|---|
| | this case, consider increasing the amount of memory reserved by the **-g** switch. Using a value lower than the default will increase the amount of memory available to the buffer pool and thread stacks; this may, in turn, provide some performance benefit to memory-intensive workloads in systems that do not use many extended stored procedures, distributed queries, or automation objects. |
| **-m** | Starts an instance of SQL Server in single-user mode. When you start an instance of SQL Server in single-user mode, only a single user can connect, and the CHECKPOINT process is not started. CHECKPOINT guarantees that completed transactions are regularly written from the disk cache to the database device. (Typically, this option is used if you experience problems with system databases that should be repaired.) Enables the **sp_configure allow updates** option. By default, **allow updates** is disabled. |
| **-n** | Does not use the Windows application log to record SQL Server events. If you start an instance of SQL Server with **-n**, it is recommended that you use the **-e** startup option too; otherwise, SQL Server events are not logged. |
| **-s** | Allows you to start a named instance of SQL Server 2000. Without the **-s** parameter set, the default instance will attempt to start. You must switch to the appropriate BINN directory for the instance at a command prompt before starting sqlservr.exe. For example, if Instance1 were to use \mssql$Instance1 for its binaries, the user must be in the \mssql$Instance1\binn directory to start sqlservr.exe-sinstance1. |
| **/T***trace#* | Indicates that an instance of SQL Server should be started with a specified trace flag (*trace#*) in effect. |

| | |
|---|---|
| | Trace flags are used to start the server with nonstandard behavior. |
| **-x** | Disables the keeping of CPU time and cache-hit ratio statistics. Allows maximum performance. |

**IMPORTANT** When specifying a trace flag with the **/T** option, use an uppercase "T" to pass the trace flag number. A lowercase "t" is accepted by SQL Server, but this sets other internal trace flags that are required only by SQL Server support engineers. (Parameters specified in the Control Panel startup window are not read.)

## See Also

CHECKPOINT

# Administering SQL Server

# Logging In to SQL Server

You can log in to an instance of Microsoft® SQL Server™ by using any of the graphical administration tools or from a command prompt.

When you log in to an instance of SQL Server using a graphical administration tool such as SQL Server Enterprise Manager or SQL Query Analyzer, you are prompted to supply the server name, a login ID, and a password, if necessary. How you log in to an instance of SQL Server depends on whether SQL Server is using Windows Authentication or mixed mode (SQL Server Authentication and Windows Authentication). If SQL Server is using Windows Authentication, you do not have to provide a login ID each time you access a registered SQL Server. Instead, SQL Server logs you in automatically using your Microsoft Windows NT® 4.0 or Windows® 2000 account.

**Note** If you selected a case-sensitive sort order when you installed SQL Server, your login ID is also case-sensitive.

### To log in to SQL Server

# Administering SQL Server

# Running SQL Server

Microsoft® SQL Server™ can run over the network or without a network.

## Running SQL Server on a Network

For SQL Server to communicate over the network, the SQL Server service must be running. By default, Microsoft Windows NT® 4.0 and Windows® 2000 automatically start the built-in SQL Server service. To find out whether the SQL Server service has been started, at the command prompt, type:

```
net start
```

If the SQL Server service has been started, the following appears in the **net start** output:

```
C:\> net start
These Windows NT services are started:

   ClipBook Server
   Computer Browser
   EventLog
   Messenger
   Network DDE
   Network DDE DSDM
   Server
   Workstation

The command completed successfully.
```

If the SQL Server service has not been started, at the command prompt, type:

```
net start server
```

The following message indicates that the service has been started:

```
The Server service was started successfully.
```

You can also use the Services application in Control Panel to check service status and to start and stop services. For more information, see the Windows NT 4.0 and Windows 2000 documentation.

## Running SQL Server Without a Network

When running an instance of SQL Server without a network, you do not need to start the built-in SQL Server service. Because SQL Server Enterprise Manager, SQL Server Service Manager, and the **net start** and **net stop** commands are functional even without a network, the procedures for starting and stopping an instance of SQL Server are identical for a network or stand-alone operation.

When connecting to an instance of a stand-alone SQL Server from a local client such as **osql**, you bypass the network and connect directly to the instance of SQL Server by using a local pipe. The difference between a local pipe and a network pipe is whether you are using a network. Both local and network pipes establish a connection with an instance of SQL Server by using the standard pipe (\pipe\sql\query), unless otherwise directed.

When you connect to an instance of a local SQL Server without specifying a server name, you are using a local pipe. When you connect to an instance of a local SQL Server and specify a server name explicitly, you are using either a network pipe or another network interprocess communication (IPC) mechanism, such as Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX) (assuming you have configured SQL Server to use multiple networks). Because a stand-alone SQL Server does not support network pipes, you must omit the unnecessary **/S***server_name* argument when connecting to the instance of SQL Server from a client. For example, to connect to a stand-alone instance of SQL Server from **osql**, type:

```
osql /Usa /P
```

### See Also

Named Pipes Clients

# Administering SQL Server

## Pausing and Resuming SQL Server

When you pause an instance of Microsoft® SQL Server™, users who are connected to the server can finish tasks, but new connections are not allowed. For example, you can pause an instance of SQL Server for a few minutes and send a shutdown message to connected users before shutting it down. You can also resume a SQL Server service.

You can pause an instance of SQL Server before stopping the server. Pausing an instance of SQL Server prevents new users from logging in and gives you time to send a message to current users asking them to log out before you stop the server.

**Note** You cannot pause an instance of SQL Server if it was started by running **sqlservr**. Only SQL Server services started as a Microsoft Windows NT® 4.0 or Windows® 2000 service can be paused.

For more information about pausing and resuming an instance of SQL Server from the Services application in Control Panel, see the Windows NT 4.0 or Windows 2000 documentation.

**To pause and resume SQL Server**

# Administering SQL Server

## Stopping SQL Server

You can stop an instance of Microsoft® SQL Server™ locally from the server or remotely from a client or another server. If you stop an instance of SQL Server without pausing it, all server processes are terminated immediately. Stopping an instance of SQL Server prevents new connections and disconnects current users.

The following table describes the available methods for stopping an instance of SQL Server.

| Method | Description |
|---|---|
| SQL Server Enterprise Manager | Stops a local or remote instance of SQL Server or a SQL Server Agent service. |
| SQL Server Service Manager | Stops a local or remote instance of SQL Server or a SQL Server Agent service from a single window or from the Windows® taskbar. |
| SHUTDOWN statement | Stops an instance of SQL Server when executed within **osql** or another query tool. Using the WITH NOWAIT option stops an instance of SQL Server immediately. |
| **net stop mssqlserver** | Stops an instance of SQL Server either remotely or locally if you are running the Microsoft Windows NT® 4.0 or Windows® 2000 operating systems. To stop a named instance of SQL Server 2000, you must enter **net stop mssql$instancename** from the command prompt. |
| Control Panel | Stops an instance of SQL Server using the Services application in Control Panel. |
| CTRL+C | Stops an instance of SQL Server if it was started as a program from the command prompt. |

When you stop an instance of SQL Server, the server performs these services before it shuts down:

- Disables logins (except for system administrators).

- Performs a CHECKPOINT in every database. However, if you stop an instance of SQL Server using CTRL+C at the command prompt, it does not perform a CHECKPOINT in every database. Therefore, the next time the server is started, recovery time takes longer.

- Waits for all Transact-SQL statements or stored procedures currently executing to finish.

**Note** To bring the system to an immediate halt, you can issue the SHUTDOWN WITH NOWAIT statement from the **osql** utility.

### To stop SQL Server

# Administering SQL Server

# Broadcasting a Shutdown Message

Before you stop an instance of Microsoft® SQL Server™, you can broadcast a message to warn users of an impending shutdown. In the message, you can include the time the instance of SQL Server will be stopped so users can finish their tasks.

## To broadcast a shutdown message

# Administering SQL Server

# Failover Clustering

In Microsoft® SQL Server™ 2000 Enterprise Edition, SQL Server 2000 failover clustering provides high availability support. For example, during an operating system failure or a planned upgrade, you can configure one failover cluster to fail over to any other node in the failover cluster configuration. In this way, you minimize system downtime, thus providing high server availability.

To install, configure, and maintain a failover cluster, use SQL Server Setup. For information about upgrading to a SQL Server 2000 failover cluster, see Upgrading to a SQL Server 2000 Failover Cluster.

Use failover clustering to:

- Install SQL Server on multiple nodes in a failover cluster. You are limited only by the number of nodes supported by the operating system.

  Before installing failover clustering, you must install Microsoft Windows NT® 4.0, Enterprise Edition, Microsoft Windows® 2000 Advanced Server or Windows 2000 Datacenter Server, and the Microsoft Cluster Service (MSCS).

  There are specific installation steps that must be followed to use failover clustering. For more information, see Installing Failover Clustering and Handling a Failover Cluster Installation.

- Specify multiple IP addresses for each virtual server.

  SQL Server 2000 allows you to use all available network IP subnets, thereby providing alternate ways to connect if one subnet fails and increasing network scalability. For example, with a single network adaptor, a network failure can disrupt communications. However, with multiple network cards in the server, each network can be on a different IP subnet. If one subnet fails, at least one connection can continue to function. If a router fails, MSCS continues to function, and all IP addresses still work. However, if the network card on the local computer fails, communication still may be disrupted. For more information, see Creating a Failover Cluster.

- Administer a failover cluster from any node in the clustered SQL Server configuration. To perform setup tasks, you must be working from the node in control of the cluster disk resource. For more information, see Creating a Failover Cluster.

- Allow one virtual server to fail over to any other node on the failover cluster configuration. For more information, see Creating a Failover Cluster.

- Add or remove nodes from the failover cluster configuration using the Setup program. For more information, see Maintaining a Failover Cluster.

- Reinstall or rebuild a virtual server on any node in the failover cluster without affecting the other nodes. For more information, see Maintaining a Failover Cluster.

- Perform full-text queries by using Microsoft Search service with failover clustering. For more information, see Using SQL Server Tools with Failover Clustering.

## Multiple Instance Support

Failover clustering also supports multiple instances. Multiple instance support makes it easier to build, install, and configure virtual servers in a failover cluster. Applications can connect to each instance on a single computer in much the same way as they connect to instances of SQL Server running on multiple computers. For more information about virtual servers, see Creating a Failover Cluster.

With multiple instance support, you can isolate work environments (for example, testing from production) or volatile application environments and provide different system administrators for each instance of SQL Server on the same computer. For more information, see Multiple Instances of SQL Server.

## See Also

Failover Clustering Architecture

# Administering SQL Server

## Failover Clustering Support

In Microsoft® SQL Server™ 2000 Enterprise Edition, the number of nodes supported in SQL Server 2000 failover clustering depends on the operating system you are running:

- Microsoft Windows NT® 4.0, Enterprise Edition, Microsoft Windows® 2000 Advanced Server, and Microsoft Windows 2000 Datacenter Server support two-node failover clustering.

- Windows 2000 Datacenter Server supports up to four-node failover clustering, including an active/active/active/active failover clustering configuration.

The following tools, features and components are supported with failover clustering:

- Microsoft Search service. For more information, see Using SQL Server Tools with Failover Clustering.

- Multiple instances. For more information, see Failover Clustering.

- SQL Server Enterprise Manager. For more information, see Using SQL Server Tools with Failover Clustering.

- Service Control Manager. For more information, see Using SQL Server Tools with Failover Clustering.

- Replication. For more information, see Creating a Failover Cluster.

- SQL Profiler. For more information, see Using SQL Server Tools with Failover Clustering.

- SQL Query Analyzer. For more information, see Using SQL Server Tools with Failover Clustering.

- SQL Mail. For more information, see Using SQL Server Tools with Failover Clustering.

The following component is not supported for failover clustering:

- SQL Server 2000 Analysis Services

**Note** Microsoft Data Access Components (MDAC) 2.6 is not supported for SQL Server version 6.5 or SQL Server 7.0, when either version is in a failover cluster configuration.

Before using failover clustering, consider the following:

- Failover clustering resources, including the IP addresses and network name, must be used only when you are running an instance of SQL Server 2000. They should not be used for other purposes, such as file sharing.

- In a failover cluster configuration, SQL Server 2000 supports Windows NT 4.0, Enterprise Edition but requires that the service accounts for SQL Server services (SQL Server and SQL Server Agent) be local administrators of all nodes in the cluster.

**IMPORTANT** SQL Server 2000 supports both Named Pipes and TCP/IP Sockets over TCP/IP within a failover cluster. However, it is strongly recommended that you use TCP/IP Sockets in a clustered configuration.

# Administering SQL Server

# Creating a Failover Cluster

To create a Microsoft® SQL Server™ 2000 failover cluster, you must create and configure the virtual servers on which the failover cluster runs. You create virtual servers during SQL Server Setup. Virtual servers are not provided by Microsoft Windows NT® 4.0 or Microsoft Windows® 2000.

To create a failover cluster, you must be a local administrator with rights to log on as a service and to act as part of the operating system on all computers in the failover cluster.

## Elements of a Virtual Server

A virtual server contains:

- A combination of one or more disks in a Microsoft Cluster Service (MSCS) cluster group.

  Each MSCS cluster group can contain at most one virtual SQL Server.

- A network name for each virtual server. This network name is the virtual server name.

- One or more IP addresses that are used to connect to each virtual server.

- One instance of SQL Server 2000, including a SQL Server resource, a SQL Server Agent resource, and a full-text resource.

  If an administrator uninstalls the instance of SQL Server 2000 within a virtual server, the virtual server, including all IP addresses and the network name, is also removed from the MSCS cluster group.

A failover cluster can run across one or more actual Windows 2000 Advanced Server or Windows 2000 Datacenter Server servers or Windows NT 4.0, Enterprise Edition servers that are participating nodes of the cluster. However, a SQL Server virtual server always appears on the network as a single Windows 2000 Advanced Server, Windows 2000 Datacenter Server, or Microsoft

Windows NT 4.0, Enterprise Edition server.

## Naming a Virtual Server

SQL Server 2000 depends on distinct registry keys and service names within the failover cluster so that operations will continue correctly after a failover. Therefore, the name you provide for the instance of SQL Server 2000, including the default instance, must be unique across all nodes in the failover cluster, as well as across all virtual servers within the failover cluster. For example, if all instances failed over to a single server, their service names and registry keys would conflict. If INST1 is a named instance on virtual server VIRTSRV1, there cannot be a named instance INST1 on any node in the failover cluster, either as part of a failover cluster configuration or as a stand-alone installation.

Additionally, you must use the VIRTUAL_SERVER\Instance-name string to connect to a clustered instance of SQL Server 2000 running on a virtual server. You cannot access the instance of SQL Server 2000 by using the computer name that the clustered instance happens to reside on at any given time. SQL Server 2000 does not listen on the IP address of the local servers. It listens only on the clustered IP addresses created during the setup of a virtual server for SQL Server 2000.

## Usage Considerations

Before you create a failover cluster, consider the following:

- If you are using the Windows 2000 Address Windowing Extensions (AWE) API to take advantage of memory greater than 3 gigabytes (GB), make certain that the maximum available memory you configure on one instance of SQL Server will still be available after you fail over to another node. If the failover node has less physical memory than the original node, instances of SQL Server may fail to start or may start with less memory than they had on the original node. You must:
    - Give each server in the cluster the same amount of physical RAM.

    - Ensure that the summed value of the **max server memory** settings for all instances is less than the lowest amount of physical RAM available on any of the virtual servers in the failover cluster.

For more information about AWE, see Using AWE Memory on Windows 2000.

- If you need high-availability servers in replication, it is recommended that you use an MSCS cluster file share as your snapshot folder when configuring a Distributor on a failover cluster. In the case of server failure, the distribution database will be available and replication will continue to be configured at the Distributor.

  Also, when creating publications, specify the MSCS cluster file share for the additional storage of snapshot files or as the location from which Subscribers apply the snapshot. This way, the snapshot files are available to all nodes of the cluster and to all Subscribers that must access it. For more information, see Publishers, Distributors, and Subscribers and Alternate Snapshot Locations.

- If you want to use encryption with a failover cluster, you must install the server certificate with the fully qualified DNS name of the virtual server on all nodes in the failover cluster. For example, if you have a two-node cluster, with nodes named test1.redmond.corp.microsoft.com and test2.redmond.corp.microsoft.com and a virtual SQL Server "Virtsql", you need to get a certificate for "virtsql.redmond.corp.microsoft.com" and install the certificate on both nodes. You can then check the **Force protocol encryption** check box on the Server Network Utility to configure your failover cluster for encryption.

- You should not remove the BUILTIN/Administrators account from SQL Server. The IsAlive thread runs under the context of the cluster service account, and not the SQL Server service account. The cluster service must be part of the administrator group on each node of the cluster. If you remove the BUILTIN/Administrators account, the IsAlive thread will no longer be able to create a trusted connection, and you will lose access to the virtual server.

## Creating a Failover Cluster

Here are the basic steps for creating a failover cluster using the Setup program:

1. Identify the information you need to create your virtual server (for example, cluster disk resource, IP addresses, and network name) and the nodes available for failover.

   The cluster disks to use for failover clustering should all be in a single cluster group and owned by the node from which the Setup program is run. This configuration must take place before you run the Setup program. You configure this through Cluster Administrator in Windows NT 4.0 or Windows 2000. You need one MSCS group for each virtual server you want to set up.

2. Start the Setup program to begin your installation. After all necessary information has been entered, the Setup program installs a new instance of SQL Server binaries on the local disk of each computer in the cluster and installs the system databases on the specified cluster disk. The binaries are installed in exactly the same path on each cluster node, so you must ensure that each node has a local drive letter in common with all the other nodes in the cluster.

   In SQL Server 2000, during a failover only the databases fail over. In SQL Server version 6.5 and SQL Server version 7.0, both the SQL Server databases and binaries fail over during a failover.

   If any resource (including SQL Server) fails for any reason, the services (SQL Server, SQL Server Agent, Full-Text Search, and all services in the failover cluster group) fail over to any available nodes defined in the virtual server.

3. You install one instance of SQL Server 2000, creating a new virtual server and all resources.

### How to create a new failover cluster

# Administering SQL Server

## Failover Clustering Example

The following example illustrates how you configure Microsoft® SQL Server™ 2000 failover clustering.

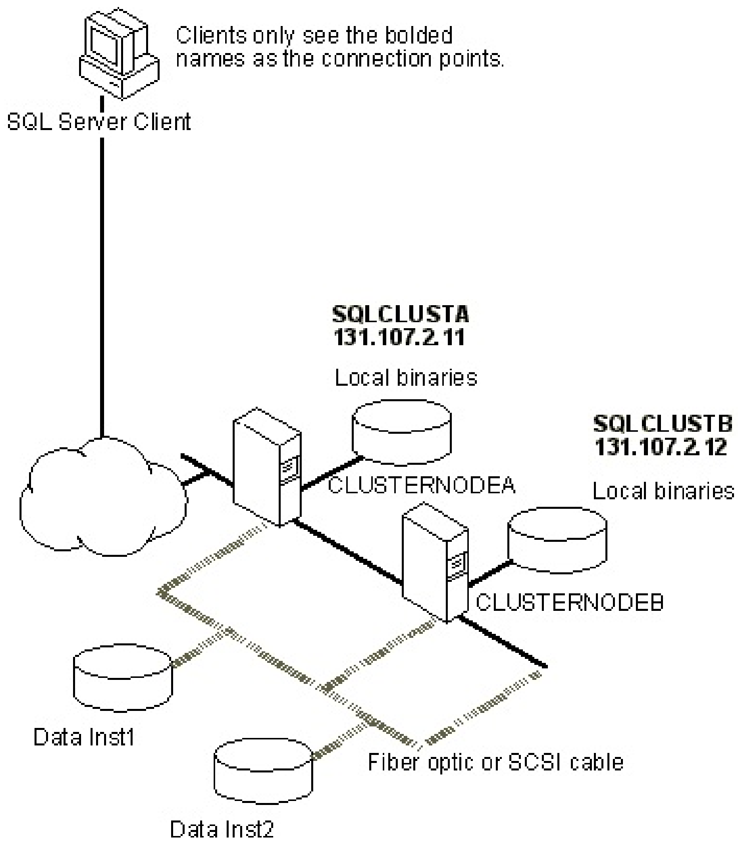
CLUSTERNODEA and CLUSTERNODEB are two computers in a failover cluster. Run SQL Server Setup on CLUSTERNODEA and create a virtual server named "SQLCLUSTA." Then install a default instance of SQL Server 2000, which can run on both CLUSTERNODEA and CLUSTERNODEB. From this point forward, connect to the server by specifying "SQLCLUSTA" as the server name in the connection string.

Run the Setup program again on CLUSTERNODEB. Create a new virtual server named "SQLCLUSTB" (in a different Microsoft Cluster Service (MSCS) cluster group) and install an instance named "Inst1" that can run on both CLUSTERNODEA and CLUSTERNODEB. From this point forward, connect to the server by specifying "SQLCLUSTB\Inst1" as the connection string.

The two virtual servers are running in the MSCS cluster consisting of CLUSTERNODEA and CLUSTERNODEB. Other than that, they are completely separate from each other. Each virtual server resides in a different MSCS cluster group, and each has a different set of IP addresses, a distinct network name, and data files that reside on a separate set of shared cluster disks.

When a failover occurs for any resource in an MSCS cluster group, all resources that are members of that group also fail over. For SQLCLUSTA, any failure (from the disk resources, IP address, the network name, or the installations of SQL Server 2000 within the virtual server) causes all members of the cluster group to fail over when the failover threshold is reached.

The following illustration is a two-node cluster with binaries and data. Each virtual server in this illustration must have exclusive ownership of the disk on which the data and log files are located.



## See Also

Failover Clustering Architecture

# Administering SQL Server

## Upgrading to a SQL Server 2000 Failover Cluster

When you are upgrading to a Microsoft® SQL Server™ 2000 failover cluster, only one default instance is allowed. Use the Cluster Wizard in SQL Server version 6.5 or SQL Server 7.0 to uncluster any existing SQL Server 6.5 or SQL Server 7.0 clustered instances before upgrading to SQL Server 2000. Then run SQL Server Setup on SQL Server 2000.

SQL Server 6.5 or SQL Server 7.0 failover clusters cannot exist on the same computer as a SQL Server 2000 failover cluster. In SQL Server 6.5 or SQL Server 7.0, in an active/active configuration or in an active/passive configuration where one server contains an unclustered SQL Server, there is a name conflict. Both servers are default instances.

**IMPORTANT** You cannot run the Cluster Wizard in SQL Server 6.5 or SQL Server 7.0 after SQL Server 2000 has been installed.

For SQL Server 2000, you must use a domain account for the services (SQL Server, SQL Server Agent, and all services in the clustered group). That account must be an administrator on all computers in the cluster, if those computers are running on Microsoft Windows NT® Server 4.0, Enterprise Edition.

**Note** If you are using replication on a SQL Server 6.5 or 7.0 failover cluster and upgrading to a SQL Server 2000 failover cluster, you must uncluster the previous installation. Delete all publications, remove replication, and then reconfigure replication after upgrading. This will not be a requirement when upgrading from SQL Server 2000 in future releases.

### To upgrade from a SQL Server 6.5 active/passive failover cluster

# Administering SQL Server

## Handling a Failover Cluster Installation

When you install a Microsoft® SQL Server™ 2000 failover cluster, you must:

- Ensure that the operating system is installed properly and designed to support failover clustering. For more information about what to do before installing a failover cluster, see Before Installing Failover Clustering. For more information about the order of installation, see Installing Failover Clustering.

- Consider whether the SQL Server tools, features, and components you want to use are supported with failover clustering. For more information, see Failover Clustering Support.

- Consider whether failover clustering is dependent on the products you want to use. For more information, see Failover Clustering Dependencies.

- Consider how to create a new failover cluster. For more information about creating a new failover cluster configuration, see Creating a Failover Cluster.

- Review the instructions for upgrading from a SQL Server version 6.5 or SQL Server version 7.0 cluster to a SQL Server 2000 failover cluster. For more information, see Upgrading to a SQL Server 2000 Failover Cluster.

## Administering SQL Server

# Before Installing Failover Clustering

Before you install a Microsoft® SQL Server™ 2000 failover cluster, you must select the operating system on which your computer will run. You can use Microsoft Windows NT® 4.0, Enterprise Edition, Microsoft Windows® 2000 Advanced Server, or Microsoft Windows 2000 Datacenter Server. You also must install Microsoft Cluster Service (MSCS).

## Preinstallation Checklist

Before you begin the installation process, verify that:

- There is no IRQ sharing between network interface cards (NICs) and drive/array (SCSI) controllers. Although some hardware may support this sharing, it is not recommended.

- Your hardware is listed on the Windows NT Hardware Compatibility List.

  For a complete list of supported hardware, see the Hardware Compatibility List at the Microsoft Web site.

  The hardware system must appear under the category of cluster. Individual cluster components added together do not constitute an approved system. Only systems purchased as a cluster solution and listed in the cluster group are approved. When checking the list, specify cluster as the category. All other categories are for OEM use.

- MSCS has been installed completely on at least one node before you run Windows NT 4.0, Enterprise Edition or Windows 2000 Advanced Server or Windows 2000 Datacenter Server simultaneously on all nodes.

  When using MSCS, you must make certain that one node is in control of the shared SCSI bus prior to the other node(s) coming online. Failure to do this can cause application failover to go into an online pending state. As a result, the cluster either fails on the other node or fails totally. However, if your hardware manufacturer has a proprietary installation process, follow the hardware manufacturer instructions.

- WINS is installed according to the following article in the Product Support Services Microsoft Web site:

  Q258750 Recommended Private "Heartbeat" Configuration on Cluster Server

- The disk drive letters for the cluster-capable disks are the same on both servers.

- You have disabled NetBIOS for all private network cards before beginning SQL Server Setup.

- You have cleared the system logs in all nodes and viewed the system logs again. Ensure that the logs are free of any error messages before continuing.

# Administering SQL Server

## Installing Failover Clustering

If you are installing Microsoft® SQL Server™ 2000 failover clustering on Microsoft Windows NT® 4.0, Enterprise Edition, you need to install programs in the order specified below. However, this is not necessary if you are installing failover clustering on Microsoft Windows® 2000 Advanced Server or Windows 2000 Datacenter Server.

**CAUTION** If you do not install the programs in the following order, the software products can fail on installation and require that you completely reinitialize the disk and restart installation.

Before installing SQL Server 2000 in a failover cluster configuration, you must upgrade any pre-release versions of SQL Server 2000.

### To install failover clustering on Windows NT 4.0

1. Install Windows NT 4.0, Enterprise Edition.

   Windows NT 4.0, Enterprise Edition includes Windows NT 4.0 Service Pack 3. Service Pack 3 is required to install Microsoft Cluster Service (MSCS).

   - Do not go directly to Service Pack 4 or later if you intend to install the Windows NT Option Pack.

   - Do not install Microsoft Internet Information Server (IIS).

   **IMPORTANT** IIS is installed by default. It is recommended that you clear this option during the Windows NT 4.0 installation.

2. Install MSCS.

3. Install Microsoft Internet Explorer version 5.0 or later.

4. Manually create a Microsoft Distributed Transaction Coordinator (MS DTC) compatible resource group where MS DTC setup can create its

resources. This should contain an IP address, network name, and cluster disk resource. Any group with these three things is compatible with MS DTC.

SQL Server Setup will install MS DTC in a later step. Install Windows NT 4.0 Option Pack only if you require components of the Windows NT 4.0 Option pack besides MS DTC.

5. Install the latest Windows NT 4.0 Service Pack, Service Pack 5 at the latest. Click **Create an uninstall directory**, click **Year 2000 Setup**, and then select the **Service Pack install for Intel based systems** check box.

   Do not select Microsoft Message Queue Server (MSMQ 1.0) or IIS. MSMQ 1.0 is not supported on SQL Server 2000. It is recommended that IIS functionality be used with Windows NT Load Balancing Service (WLBS). For more information about WLBS, search on "WLBS Features Overview" on the NT Server Microsoft Web site.

   Prior to Step 5, it is recommended that you rename the hidden directory $NTServicePackUninstall$ to $NTServicePackUninstall$.*service packnumber*. After installing the service pack, add a new directory. This way you have uninstall directories available, which prevents the directories from being accidentally overwritten.

6. Install SQL Server 2000.

**Note** Install any additional server products before installing any other applications.

### To install failover clustering on Windows 2000

1. Install Windows 2000 and accept the default application choices.

2. After installing Windows 2000 on the first node and prior to installing MSCS, click **Start\Programs\Administrative Tools\Configure Your Server**.

3. Click **Advanced\Cluster Service**, and then in the right pane, click **Learn More**.

4. From **Help**, review Item 2 under Windows Clustering.

   Windows Clustering is used during the installation of Windows 2000 and with SQL Server 2000 failover clustering. Follow these instructions to install MSCS.

   **IMPORTANT** It is necessary to read the section on Planning for Windows Clustering\Requirements for server clusters and to follow the Checklist for server clusters called Checklist: Creating a server cluster. This is found under the Server Clusters section\Checklist for server clusters.

5. After you have successfully installed MSCS, you need to configure MS DTC to run on a cluster.

   For more information about MS DTC, see Failover Clustering Dependencies.

6. On the **Start** menu, point to **Programs\Administrative Tools\Cluster Administrator**, and click **View Groups\Cluster Group**. If the group contains an MS DTC resource, proceed to Step 9. If not, complete the following two steps.

7. On the **Start** menu, point to **Command Prompt**. Enter comclust.exe from the command prompt.

8. Repeat Step 7 on the remaining nodes of the cluster, one node at a time.

9. Install SQL Server 2000.

**Note** Install any additional server products before installing any user applications.

# Administering SQL Server

# Failover Clustering Dependencies

There are several products that interact with Microsoft® SQL Server™ 2000 failover clustering. To ensure that your failover cluster functions properly, you need to understand the underlying dependencies that failover clustering has on other products.

## Microsoft Distributed Transaction Coordinator (MS DTC)

SQL Server 2000 requires Microsoft Distributed Transaction Coordinator (MS DTC) in the cluster for distributed queries and two-phase commit transactions, as well as for some replication functionality. After you install Microsoft Windows® 2000 and configure your cluster, you must run the Cluster Wizard (the comclust.exe program) on all nodes to configure MS DTC to run in clustered mode.

The Cluster Wizard makes the following changes to the MS DTC configuration:

- It creates an MS DTC resource in a resource group containing a shared cluster disk resource and a network name resource.

- It creates an MS DTC log file on the shared cluster disk contained in the MS DTC resource group. Placing the MS DTC log file on the shared cluster disk makes it possible for the MS DTC transaction manager to access the MS DTC log file from any system in the cluster.

- It copies critical MS DTC registry entries to the shared cluster registry.

## Running MS DTC in Clustered Mode

When MS DTC is running in clustered mode, only one node in the cluster runs the MS DTC transaction manager at a time.

Any process running on any node in the cluster can use MS DTC. These processes simply call the MS DTC Proxy and the MS DTC Proxy automatically forwards MS DTC calls to the MS DTC transaction manager that is controlling the entire cluster.

If the node running the MS DTC transaction manager fails, the MS DTC transaction manager is automatically restarted on another node in the cluster. The newly restarted MS DTC transaction manager reads the MS DTC log file on the shared cluster disk to determine the outcome of pending and recently completed transactions. Resource managers reconnect to the MS DTC transaction manager and perform recovery to determine the outcome of in-doubt transactions. Applications reconnect to MS DTC so they can initiate new transactions.

For example, suppose the MS DTC transaction manager is active on system B. The application program and resource manager on system A call the MS DTC proxy. The MS DTC proxy on system A forwards all MS DTC calls to the MS DTC transaction manager on system B.

If system B fails, the MS DTC transaction manager on system A will take over. It will read the entire MS DTC log file on the shared cluster disk, perform recovery, and then serve as the transaction manager for the entire cluster.

**Note** The MS DTC transaction manager, MS DTC Proxy, and Component Services administrative tools are installed on each node of a Windows 2000 cluster using MSCS as part of Windows 2000 Setup.

### To manually install MS DTC on a Windows 2000 system running MSCS

1. Install Windows 2000 on each node in the cluster.

2. Use the Windows 2000 Configure Your Server facility to configure your cluster.

3. From a command prompt, run comclust.exe on each node in the cluster. Comclust.exe can be found in the system32 directory.

### To automatically install MS DTC on a Windows 2000 cluster system

1. Install Windows 2000 on each node in the cluster and configure your cluster using automatic installation scripts.

2. From a command prompt, run comclust.exe on each node in the

cluster. Comclust.exe can be found in the system32 directory.

### To upgrade a non-clustered Windows NT 4.0 SP4 system to a Windows 2000 cluster

1. Upgrade each system that will be part of the cluster to Windows 2000.

2. Use the Windows 2000 Configure Your Server facility to configure your server.

3. From a command prompt, run comclust.exe on each node in the cluster. Comclust.exe can be found in the system32 directory.

### To upgrade a clustered Windows NT 4.0 SP4 system to a Windows 2000 cluster

1. Install Windows 2000 on each node in the cluster.

   MS DTC requires that all nodes in the cluster be upgraded to Windows 2000 at the same time.

2. From a command prompt, run comclust.exe on each node in the cluster. Comclust.exe can be found in the system32 directory.

**IMPORTANT** Microsoft System Management Server 1.2 is not supported with SQL Server or Microsoft Cluster Service (MSCS).

### To recover from a cluster failure and rebuild MS DTC on a Windows 2000 cluster

1. When a node is lost, MS DTC will continue to work on the remaining nodes in the cluster. It does not matter whether the node that is lost is the primary or secondary node.

2. When you are ready to restore the lost node, join the lost node back to the cluster. After the node has joined the cluster, run Comclust.exe, which can be found in the system32 directory. This will reconfigure MS DTC on the node.

# Administering SQL Server

# Maintaining a Failover Cluster

After you have installed a Microsoft® SQL Server™ 2000 failover cluster, you can change or repair your existing setup. For example, you can add additional nodes to a virtual server in a failover cluster, run a clustered instance as a stand-alone instance, remove a node from a clustered instance, or recover from failover cluster failure.

## Adding a Node to an Existing Virtual Server

During SQL Server Setup, you are given the option of maintaining an existing virtual server. If you choose this option, you can add other nodes to your failover cluster configuration at a later time. You can add up to three additional nodes to an existing virtual server configured to run on one node.

### To add a node to an existing virtual server

# Administering SQL Server

# Using SQL Server Tools with Failover Clustering

You can use Microsoft® SQL Server™ 2000 failover clustering with a variety of SQL Server tools and features. However, review the following usage considerations.

## Full-Text Queries

To use the Microsoft Search service to perform full-text queries with failover clustering, consider the following:

- An instance of SQL Server 2000 must run on the same system account on all failover cluster nodes in order for full-text queries to work on failover clusters.

- You must change the start-up account for SQL Server 2000 in the failover cluster using SQL Server Enterprise Manager. If you use Control Panel or the Services Application in Microsoft Windows® 2000, you will break the full-text configuration for SQL Server.

## SQL Server Enterprise Manager

To use SQL Server Enterprise Manager with failover clustering, consider the following:

- You must change the start-up account for SQL Server 2000 in the failover cluster by using SQL Server Enterprise Manager. If you use Control Panel or the Services Application in Microsoft Windows 2000, you could break your server configuration.

- When creating or altering databases, you will only be able to view the cluster disks for the local virtual server.

- If you are browsing a table through SQL Server Enterprise Manager and lose the connection to SQL Server during a failover, you will see the error message, "Communication Link Failure". You must press ESC and undo the changes to exit out of the SQL Server Enterprise Manager window. You cannot click **Run Query**, save any changes, or edit the grid.

- If you use Enterprise Manager to reset the properties of the SQL Server service account, you will be prompted to restart SQL Server. When SQL Server is running in a failover cluster configuration, this will bring the full text and SQL Agent resources offline, as well as SQL Server. However, when SQL Server is restarted, it will not bring the full text or SQL Agent resources back online. You must start those resources manually using the Windows Cluster Administrator utility.

## Service Control Manager

Use the Service Control Manager to start or stop a clustered instance of SQL Server. You cannot pause a clustered instance of SQL Server.

### To start a clustered instance of SQL Server using Service Control Manager

## Administering SQL Server

# Failover Cluster Troubleshooting

This topic provides information about:

- Resolving the most common Microsoft® SQL Server™ 2000 failover clustering usage issues.
- Optimizing failover cluster performance.
- Using failover clustering with extended stored procedures that use COM objects.

## Resolving Common Usage Issues

The following list describes common usage issues and explains how to resolve them:

- SQL Server 2000 cannot log on to the network after it migrates to another node.

  SQL Server service account passwords must be identical on all nodes or else the node cannot restart a SQL Server service that has migrated from a failed node.

  If you change the SQL Server service account passwords on one node, you must change the passwords on all other nodes. However, if you change the account using SQL Server Enterprise Manager, this task will be done automatically.

- SQL Server cannot access the cluster disks.

  A node cannot recover cluster disks that have migrated from a failed node if the shared cluster disks use a different letter drive. The disk drive letters for the cluster disks must be the same on both servers. If they are not, review your original installation of the operating system and Microsoft Cluster Service (MSCS). For more information, see the Microsoft Windows NT® 4.0, Enterprise Edition, Windows® 2000

Advanced Server, or Windows 2000 Datacenter Server documentation.

- You do not want a failure of a service, such as full-text search or SQL Server Agent, to cause a failover.

  To prevent the failure of specific services from causing the SQL Server group to fail over, configure those services using Cluster Administrator in Windows NT 4.0 or Windows 2000. For example, to prevent the failure of the Full-Text Search service from causing a failover of SQL Server, clear the **Affect the Group** check box on the **Advanced** tab of the **Full Text Properties** dialog box. However, if SQL Server causes a failover, the full-text search service will restart.

- SQL Server will not start automatically.

  You cannot start a failover cluster automatically using SQL Server. You must use Cluster Administrator in MSCS to automatically start a failover cluster.

- The error message "No compatible resource groups found" is displayed during SQL Server Setup.

  This error is caused by the Microsoft Distributed Transaction Coordinator (MS DTC) setup on Windows NT 4.0, Enterprise Edition. MS DTC requires a group containing a network name, IP address, and shared cluster disk to be owned by the local node when the Setup program is run. If this error is displayed, open Cluster Administrator and make certain there is a group that meets these requirements owned by the local node. The easiest way to do this is to move a disk into the cluster group that already contains a network name and IP address. After you have this group on the local node, click **Retry**.

- The error message "All cluster disks available to this virtual server are owned by other node(s)" is displayed during Setup.

  This message is displayed when you select the drive and path for installing data files, and the drive you selected is not owned by the local node. Move the disk to the local node using Cluster Administrator.

- The error message "Unable to delete SQL Server resources. They must be manually removed. Uninstallation will continue." is displayed during

SQL Server Setup.

This message is displayed if SQL Server Setup cannot delete all of the SQL Server resources. You must go into Control Panel and uninstall the instance you were trying to remove on every node.

- You cannot enable the clustering operating system error log.

  The operating system cluster error log is used by MSCS to record information about the cluster. Use this error log to debug cluster configuration issues. To enable the cluster error log, set the system environment variable CLUSTERLOG=<path to file> (for example, CLUSTERLOG=c:\winnt\cluster\cluster.log). This error log is on by default in Windows 2000.

- If the Network Name is offline and you cannot connect using TCP/IP, you must use Named Pipes.

  To connect using Named Pipes, create an alias using the Client Network Utility to connect to the appropriate computer. For example, if you have a cluster with two nodes (Node A and Node B), and a virtual server (Virtsql) with a default instance, you can connect to the server that has the Network Name resource offline by doing the following:

  1. Determine on which node the group containing the instance of SQL Server is running by using the Cluster Administrator. For this example, it will be Node A.

  2. Start the SQL Server service on that computer using **net start**. For more information about using **net start**, see Starting SQL Server Manually.

  3. Start the SQL Server Network Utility on Node A. View the pipe name on which the server is listening. It should be similar to \\.\$$\VIRTSQL\pipe\sql\query.

  4. On the client computer, start the Client Network Utility.

5. Create an alias SQLTEST1 to connect via Named Pipes to this pipe name. To do this, put Node A as the server name and edit the pipe to be \\.\pipe\$$\VIRTSQL\sql\query. Connect to this instance using the alias SQLTEST1 as the server name.

For more information, see Client Net-Libraries and Network Protocols.

## Optimizing Failover Clustering Performance

To optimize performance when using failover clustering, consider the following:

- If your disk controller is not external to your clustered computer, you must turn off write-caching within the controller to prevent data loss during a failover.

- Write-back caching cannot be used on host controllers in a cluster without hindering performance. However, if you use external controllers, you continue to provide performance benefits. External disk arrays are not affected by failover clustering and can sync the cache correctly, even across a SCSI bus.

- It is recommended that you do not use the cluster drive for file shares. Using these drives impacts recovery times and can cause a failover of the cluster group due to resource failures.

## Using Extended Stored Procedures and COM Objects

When you use extended stored procedures with a failover clustering configuration, all extended stored procedures need to be installed on the shared cluster disk. This is to ensure that when a node fails over, the extended stored procedures can still be used.

If the extended stored procedures use COM components, the administrator needs to register the COM components on each node of the cluster. The information for loading and executing COM components must be in the registry of the active node in order for the components to be created. Otherwise, the information will remain in the registry of the computer on which the COM components were first registered. For more information, see Extended Stored Procedure Architecture.

# Administering SQL Server

## Importing and Exporting Data

Importing data is the process of retrieving data from sources external to Microsoft® SQL Server™ (for example, an ASCII text file) and inserting it into SQL Server tables. Exporting data is the process of extracting data from an instance of SQL Server into some user-specified format (for example, copying the contents of a SQL Server table to a Microsoft Access database).

Importing data from an external data source into an instance of SQL Server is likely to be the first step you perform after setting up your database. After data has been imported into your SQL Server database, you can start to work with the database.

Importing data into an instance of SQL Server can be a one-time occurrence (for example, migrating data from another database system to an instance of SQL Server). After the initial migration is complete, the SQL Server database is used directly for all data-related tasks, rather than the original system. No further data imports are required.

Importing data can also be an ongoing task. For example, a new SQL Server database is created for executive reporting purposes, but the data resides in legacy systems updated from a large number of business applications. In this case, you can copy new or updated data from the legacy system to an instance of SQL Server on a daily or weekly basis.

Usually, exporting data is a less frequent occurrence. SQL Server provides tools and features that allow applications, such as Access or Microsoft Excel, to connect and manipulate data directly, rather than having to copy all the data from an instance of SQL Server to the tool before manipulating it. However, data may need to be exported from an instance of SQL Server regularly. In this case, the data can be exported to a text file and then read by the application. Alternatively, you can copy data on an ad hoc basis. For example, you can extract data from an instance of SQL Server into an Excel spreadsheet running on a portable computer and take the computer on a business trip.

SQL Server provides tools for importing and exporting data to and from data sources, including text files, ODBC data sources (such as Oracle databases), OLE DB data sources (such as other instances of SQL Server), ASCII text files, and Excel spreadsheets.

Additionally, SQL Server replication allows data to be distributed across an enterprise, copying data between locations and synchronizing changes automatically between different copies of data.

## Administering SQL Server

## Choosing a Tool to Import or Export Data

Data can be imported to and exported from instances of Microsoft® SQL Server™ using several SQL Server tools and Transact-SQL statements. You can also write your own programs to import and export data using the programming models and application programming interfaces (APIs) available with SQL Server.

You can copy data to and from instances of SQL Server by:

- Using the Data Transformation Services (DTS) Import/Export Wizard or DTS Designer to create a DTS package that can be used to import, export and transform data.

  For more information, see DTS Tools.

- Using SQL Server replication to distribute data across an enterprise.

  The replication technology in SQL Server allows you to make duplicate copies of your data, move those copies to different locations, and synchronize the data automatically so that all copies have the same data values. Replication can be implemented between databases on the same server or different servers connected by LANs, WANs, or the Internet.

  For more information, see Replication Overview.

- Using the **bcp** command prompt utility to import and export data between an instance of SQL Server and a data file.

- Selecting data from an OLE DB provider and copying it from external data sources into an instance of SQL Server.

- Using a distributed query to select data from another data source and specify the data to be inserted.

  For more information, see Distributed Queries.

- Using the INSERT statement to add data to an existing table.

For more information, see INSERT.

- Using the BULK INSERT statement to import data from a data file to an instance of SQL Server.

  For more information, see BULK INSERT.

- Using the SELECT INTO statement to create a new table based on an existing table.

  For more information, see SELECT.

The method chosen to import or export data depends on user requirements, for example:

- The format of the source and destination data.
- The location of the source and destination data.
- Whether the import or export is a one-time occurrence or an ongoing task.
- Whether a command prompt utility, Transact-SQL statement, or graphical interface is preferred.
- The performance of the import or export operation.

This table describes the capabilities of various import and export options in SQL Server.

| **Required functionality** | **DTS wizards** | **Replication** | **bcp** | **BULK INSERT** | **SELECT INTO/ INSERT** |
|---|---|---|---|---|---|
| Import text data | YES | | YES | YES | YES 1 |
| Export text data | YES | | YES | | |
| Import from ODBC data | YES | YES | | | |

| | | | | | |
|---|---|---|---|---|---|
| sources | | | | | |
| Export to ODBC data sources | YES | YES | | | |
| Import from OLE DB data sources | YES | YES | | | YES (1) |
| Export to OLE DB data sources | YES | YES | YES | | |
| Graphical user interface (GUI) | YES | YES | | | |
| Command prompt/batch scripts | YES | YES | YES | | |
| Transact-SQL scripts | | YES | | YES | YES |
| Automatic scheduling | YES | YES | YES 2 | YES 2 | |
| Ad hoc import/export | YES | | YES | YES | YES |
| Recurring import/export | YES | YES | YES | | |
| Maximum performance | | | YES | YES | |
| Data transformation | YES | | | | |
| Programmatic interface | YES | YES | YES | | |

1 Using a distributed query that retrieves data from an external source by using an OLE DB provider.
2 By explicitly creating a job scheduled using SQL Server Agent.

## See Also

bcp Utility

# Administering SQL Server

## Preparing Data for Importing and Exporting

In order for the **bcp** and BULK INSERT utilities to insert data, the data file must be in row and column format. Microsoft® SQL Server™ can accept data in any ASCII or binary format as long as the terminators (characters used to separate columns and rows) can be described. The structure of the data file does not need to be identical to the structure of the SQL Server table because **bcp** and BULK INSERT allow columns to be skipped or reordered during the bulk copy process.

Data that is bulk copied into an instance of SQL Server is appended to any existing contents in a table. Data that is bulk copied from an instance of SQL Server to a data file overwrites the previous contents of the data file.

To bulk copy data:

- If importing data, the destination table must already exist. If exporting to a file, **bcp** will create the file.

  The number of fields in the data file does not have to match the number of columns in the table or be in the same order.

- The data in the data file must be character format or a format generated previously by the **bcp** utility, such as native format.

  Each column in the table must be compatible with the field in the data file being copied. For example, it is not possible to copy an **int** field to a **datetime** column using native format **bcp**.

- Relevant permissions to bulk copy data are required on source and destination files and tables.

  To bulk copy data from a data file into a table, you must have INSERT and SELECT permissions on the table. To bulk copy a table or view to a data file, you must have SELECT permission on the table or view being bulkcopied.

Before using bulk copy operations, consider the following:

- It is possible to specify the number of rows to load from the data file rather than loading the entire file. For example, to load only the first

150 rows from a 10,000 row data file, specify the **-L** *last_row* switch when loading the data. This can be useful for testing a batch load process.

- When using the **-F** *first_row* switch to specify the first row in the table or view to bulk copy, all rows in the table or view are first returned to the client, and then the **bcp** utility determines which rows to skip and write to the data file. Therefore, specifying **-F** *first_row* does not limit the amount of data returned to the client and does not necessarily cause the bulk copy operation to execute any faster.

- Because SQL Server can use parallel scans to retrieve data, the data bulk copied from an instance of SQL Server is not guaranteed to be in any specific order unless you bulk copy from a query and specify an ORDER BY clause.

- To copy data from earlier versions of SQL Server using native format data files, use the same version of **bcp** for importing, exporting, and formatting files.

## Importing and Exporting Data Example

To bulk copy data from the **publishers** table in the **pubs** database to the Publishers.txt data file in ASCII text format, from the command prompt, execute:

```
bcp pubs..publishers out publishers.txt -c -Sservername -Usa -Ppasswc
```

The contents of the Publishers.txt file:

| | | | | |
|---|---|---|---|---|
| 0736 | New Moon Books | Boston | MA | USA |
| 0877 | Binnet & Hardley | Washington | DC | USA |
| 1389 | Algodata Infosystems | Berkeley | CA | USA |
| 1622 | Five Lakes Publishing | Chicago | IL | USA |
| 1756 | Ramona Publishers | Dallas | TX | USA |
| | | | | |

|  |  |  |  |  |
|---|---|---|---|---|
| 9901 | GGG&G | München | -- | Germany |
| 9952 | Scootney Books | New York | NY | USA |
| 9999 | Lucerne Publishing | Paris | -- | France |

Conversely, to bulk copy data from the Publishers.txt file into the **publishers2** table in the **pubs** database, from the command prompt, execute:

```
bcp pubs..publishers2 in publishers.txt -c -Sservername -Usa -Ppasswc
```

Alternatively, you can use the BULK INSERT statement from a query tool, such as SQL Query Analyzer, to bulk copy data:

```
BULK INSERT pubs..publishers2 FROM 'c:\publishers.txt'
WITH (DATAFILETYPE = 'char')
```

**Note** The **publishers2** table must be created first.

## See Also

bcp Utility

BULK INSERT.

Character Format

ExportData Method

ImportData Method

Managing Security Accounts

Native Format

Administering SQL Server

## Using bcp and BULK INSERT

The **bcp** command prompt utility copies Microsoft® SQL Server™ data to or from a data file. It is used most frequently to transfer large volumes of data into a SQL Server table from another program, usually another database management system (DBMS). The data is first exported from the source program to a data file, and then imported from the data file into a SQL Server table using **bcp**. Alternatively, **bcp** can be used to transfer data from a SQL Server table to a data file for use in other programs. For example, the data can be copied from an instance of SQL Server into a data file. From there, another program can import the data.

**Note** The **bcp** utility is written using the ODBC bulk copy application programming interface (API). Earlier versions of the **bcp** utility were written using the DB-Library bulk copy API.

Data can also be transferred into a SQL Server table from a data file using the BULK INSERT statement. However, the BULK INSERT statement cannot bulk copy data from an instance of SQL Server to a data file. The BULK INSERT statement allows you to bulk copy data to an instance of SQL Server using the functionality of the **bcp** utility with a Transact-SQL statement, rather than from the command prompt.

It is also possible to write programs to bulk copy SQL Server data to or from a data file using the bulk copy API. The bulk copy API can be used in ODBC, OLE DB, SQL-DMO, and DB-Library-based applications.

### Trigger Execution

All bulk copy operations (the BULK INSERT statement, **bcp** utility, and the bulk copy API) support a bulk copy hint, FIRE_TRIGGERS. If FIRE_TRIGGERS is specified on a bulk copy operation that is copying rows into a table, INSERT and INSTEAD OF triggers defined on the destination table are executed for all rows inserted by the bulk copy operation. By default, bulk copy operations do not execute triggers.

These considerations apply to bulk copy operations that specify FIRE_TRIGGERS:

- Bulk copy operations that would usually be minimally logged are fully logged.

- Triggers are fired once for each batch in the bulk copy operation. The **inserted** table passed to the trigger contains all of the rows inserted by the batch. Specify FIRE_TRIGGERS only when bulk copying into a table with INSERT and INSTEAD OF triggers that support multiple row inserts.

- No result sets generated by the insert triggers are returned to the client performing the bulk copy operation.

## See Also

SQL Server Backward Compatibility Details

Bulk-Copy Functions

Bulk-Copy Rowsets

BulkCopy Object

BULK INSERT.

Performing Bulk Copy Operations

# Administering SQL Server

# Using Native, Character, and Unicode Formats

The **bcp** utility can create or read data files in the following default data formats by specifying a switch at the command prompt.

| Data format | bcp utility switch | BULK INSERT clause |
|---|---|---|
| Native | **-n** | DATAFILETYPE = **'native'** |
| Character | **-c** | DATAFILETYPE = **'char'** |
| Unicode character | **-w** | DATAFILETYPE = **'widechar'** |
| Unicode native | **-N** | DATAFILETYPE = **'widenative'** |

By default, the **bcp** utility operates in interactive mode and queries Microsoft® SQL Server™ and the user for information required to specify the data format. However, when using the **-n**, **-c**, **-w**, or **-N** switches, **bcp** does not query for information about the SQL Server table on a column-by-column basis. It reads or writes the data using the default format specified.

By default, the BULK INSERT statement operates in character mode (**char**). Interactive mode does not apply.

Additionally, the **–V** switch causes the **bcp** utility to modify native (**-n**) or character (**-c**) data to a format compatible with earlier versions of SQL Server clients. For more information, see Copying Native and Character Format Data from Earlier Versions of SQL Server.

Native mode bulk copies are best for **sql_variant** columns. Unlike character or Unicode bulk copies, native mode bulk copies preserve the meta data for each **sql_variant** value.

The recommended default data format depends on the type of bulk copy operation.

| Bulk copy operation | Native | Character | Unicode character | Unicode native |
|---|---|---|---|---|
| Bulk copying data between | | -- | -- | -- |

| | | | | |
|---|---|---|---|---|
| multiple instances of SQL Server using a data file (no extended/double-byte character set (DBCS) characters involved). | YES [1] | | | |
| Bulk copying data between multiple instances of SQL Server using a data file (extended/DBCS characters involved). | -- | -- | -- | YES |
| Exporting data to a text file to be used in another program. | -- | YES | -- | -- |
| Importing data from a text file generated by another program. | -- | YES | -- | -- |
| Bulk copying data between multiple instances of SQL Server using a data file (Unicode data/no extended/DBCS characters). | -- | -- | YES | -- |

1 Fastest method for bulk copying data from SQL Server using bcp.

## See Also

Specifying Data Formats

## Native Format

The **-n** switch (or **native** value for the DATAFILETYPE clause of the BULK INSERT statement) uses native (database) data types. Storing information in native format is useful when information must be copied from one instance of Microsoft® SQL Server™ to another. Using native format saves time and space, preventing unnecessary conversion of data types to and from character format. However, a data file in native format cannot be read by any program other than **bcp**.

For example, the command to bulk copy the **publishers** table in the **pubs** database to the Publ.txt data file using native data format is:

bcp pubs..publishers out publ.txt -n -S*servername* -Usa -P*password*

**sql_variant** data stored as a SQLVARIANT in a native mode data file maintains all of its characteristics. The meta data recording the data type of each data value is stored along with the data value and is used to re-create the data value with the same data type in a destination **sql_variant** column. If the data type of the destination column is not **sql_variant**, each data value is converted to the data type of the destination column, following the normal rules of implicit data conversion. If a data conversion error occurs, the current batch is rolled back. **char** and **varchar** values transferred between **sql_variant** columns may have code page conversion issues. For more information, see Copying Data Between Different Collations.

The **bcp** utility adds an ASCII character to the beginning of each **char** or **varchar** field equivalent to the length of the data in those fields. Noncharacter data in the table is written to the data file in the SQL Server internal binary data format.

**Important** Using native mode, **bcp**, by default, always converts characters from the data file to ANSI characters before bulk copying them into SQL Server and converts characters from SQL Server to OEM characters before copying them to the data file. Extended character data can be lost during the OEM to ANSI or ANSI to OEM conversions. To prevent loss of extended characters, use Unicode native format, or specify a code page for the bulk copy operation using **-C** (or the CODEPAGE clause for the BULK INSERT statement).

## Nonidentical and Improperly Defined Tables

Using native format to bulk copy data into an improperly defined table can cause the table to be loaded incorrectly. The incorrect loading may appear as an unusual formatting of data in the target table. This also applies to client tools that use the **bcp** API in native mode.

Native format is intended for high-speed data transfer between identically defined SQL Server tables. To achieve the optimum transfer rate, few checks are performed regarding data formatting. If the table is not defined correctly, use character format.

Correct table definition includes the correct number of columns, data type, length, and NULL status.

## Loading ASCII Files

Often, users attempt to load an ASCII file in the SQL Server native format. This leads to misinterpretation of the hexadecimal values in the ASCII file and sometimes the "unexpected end of file" error message. The correct method of loading the ASCII file is to represent each field in the data file as a character string (character format **bcp**) and let SQL Server do the data conversion to internal data types (for example, **int**, **float**, or **datetime**) as rows are inserted into the table.

## See Also

BULK INSERT.

ServerBCPDataFileType Property

Unicode Native Format

## Character Format

The **-c** switch (or **char** value for the DATAFILETYPE clause of the BULK INSERT statement) uses the character (**char**) data format for all columns, providing tabs between fields and a newline character at the end of each row as default terminators. Storing information in character format is useful when the data is used with another program, such as a spreadsheet, or when the data needs to be copied into an instance of Microsoft® SQL Server™ from another database. Character format tends to be used when copying data from other programs that have the functionality to export and import data in plain text format.

For example, the command to bulk copy the **publishers** table in the **pubs** database to the Publ.txt data file using character format is:

bcp pubs..publishers out publ.txt -c -S*servername* -Usa -P*password*

The following table shows the contents of the Publ.txt file.

| | | | | |
|---|---|---|---|---|
| 0736 | New Moon Books | Boston | MA | USA |
| 0877 | Binnet & Hardley | Washington | DC | USA |
| 1389 | Algodata Infosystems | Berkeley | CA | USA |
| 1622 | Five Lakes Publishing | Chicago | IL | USA |
| 1756 | Ramona Publishers | Dallas | TX | USA |
| 9901 | GGG&G | München | | Germany |
| 9952 | Scootney Books | New York | NY | USA |
| 9999 | Lucerne Publishing | Paris | | France |

To use field and row terminators other than the default provided with character format, specify the following.

| Terminator | bcp utility switch | BULK INSERT clause |
|---|---|---|
| Field | **-t** | FIELDTERMINATOR |
| Row | **-r** | ROWTERMINATOR |

For example, the command to bulk copy the **publishers** table in the **pubs** database to the Publ.txt data file using character format, with a comma as a field terminator and the newline character (\n) as the row terminator, is:

```
bcp pubs..publishers out publ.txt -c -t , -r \n -Sservername -Usa -Ppass
```

Here are the contents of the Publ.txt file:

```
0736,New Moon Books,Boston,MA,USA
0877,Binnet & Hardley,Washington,DC,USA
1389,Algodata Infosystems,Berkeley,CA,USA
1622,Five Lakes Publishing,Chicago,IL,USA
1756,Ramona Publishers,Dallas,TX,USA
9901,GGG&G,München,Germany
9952,Scootney Books,New York,NY,USA
9999,Lucerne Publishing,Paris,France
```

**IMPORTANT** Using character mode, **bcp**, by default, always converts characters from the data file to ANSI characters before bulk copying them into an instance of SQL Server, and converts characters from SQL Server to OEM characters before copying them to the data file. Extended character data can be lost during the OEM to ANSI or ANSI to OEM conversions. To prevent loss of extended characters, use Unicode character format, or specify a code page for the bulk copy operation using **-C** (or the CODEPAGE clause for the BULK INSERT statement).

**sql_variant** data stored in a character mode file is stored without any meta data. Each data value is converted to **char** following the rules of implicit data conversion. When it is bulk copied into a **sql_variant** destination column, the data is imported as **char**. When it is bulk copied into a destination column with a data type other than **sql_variant**, the values are converted from **char** following the rules of implicit conversion.

**Note** The **bcp** utility exports **money** values in character format data files without digit grouping symbols such as comma separators, but with four digits after the decimal point. For example, a **money** column containing the value

1,234,567.123456 is bulk copied to a data file as the character string 1234567.1235.

## See Also

Copying Data Between Different Collations

ServerBCPDataFileType Property

## Copying Native and Character Format Data from Earlier Versions of SQL Server

To copy native and character format data from Microsoft® SQL Server™ 7.0 or earlier, use the **–V** switch. When this switch is specified, SQL Server 2000 uses data types from earlier versions of SQL Server. Use the **–V** switch to specify whether the **bcp** data file is at the level of SQL Server version 6.0 (**-V 60**), SQL Server version 6.5 (**-V 65**), or SQL Server version 7.0 (**-V 70**).

The **–V** switch extends the functionality of the **–6** switch used in SQL Server 7.0. Using **–6** is the same as using **–V 60** or **–V 65**. Although SQL Server 2000 still supports the **–6** switch, the use of **–V** is recommended.

**Note** The **-V** switch does not apply to the BULK INSERT statement.

If you bulk copy data from SQL Server 7.0 or earlier into a data file, consider the following:

- **bcp** does not generate SQL Server 6.0 or SQL Server 6.5 date formats for any **datetime** or **smalldatetime** data. Dates are always written in ODBC format.

- Null values in **bit** columns are written as the value 0 because SQL Server 6.5 and earlier versions do not support nullable **bit** data.

- In SQL Server 6.5 or earlier, **bcp** represented null values as a length value of 0, whereas null is now stored as the length value -1. In SQL Server 7.0 and SQL Server 2000, the value 0 represents a zero-length column.

- **bigint** data copied to a SQL Server 7.0, SQL Server 6.5, or SQL Server 6.0 native mode or Unicode native mode data file is stored as **decimal(19,0)**. **bigint** data in a character mode or Unicode character mode data file is stored as a character or Unicode string of *[-]digits,* (for example, –25688904432).

- In a table with **char** or **varchar** fields, the **bcp** utility adds an ASCII character to the beginning of each data file field equivalent to the length of the data. In a table with **numeric** data, the information is written to the data file in the SQL Server internal binary data format.

## Copying Date Values

In SQL Server 7.0 and SQL Server 2000, **bcp** uses the ODBC bulk copy API. Therefore, **bcp** uses the ODBC date format (*yyyy-mm-dd hh:mm:ss*[*.f...*]) to import date values. However, in SQL Server 6.5 or earlier, **bcp** uses the DB-Library bulk copy API and the DB-Library date format. Use the **–V 65** switch to copy date formats from SQL Server 6.5 or earlier to SQL Server 7.0 and SQL Server 2000. If you specify **–V 65**, the **bcp** utility first attempts to convert the date value in the data file using ODBC date format. If the conversion fails, **bcp** attempts to convert the date value using DB-Library formats.

Even if **–V 65** is specified, however, the **bcp** utility always exports character format data files using the ODBC default format for **datetime** and **smalldatetime** values. For example, a **datetime** column containing the date 12 Aug 1998 is bulk copied to a data file as the character string 1998-08-12 00:00:00.000.

**IMPORTANT** When importing data into a **smalldatetime** field using **bcp**, be sure the value for seconds is 00.000; otherwise the **bcp** operation will fail. The **smalldatetime** data type only holds values to the nearest minute. BULK INSERT will not fail in this instance but will truncate the seconds value.

Using the **–V 65** switch can affect performance because of the overhead required to support multiple date conversions.

## See Also

Using bcp and BULK INSERT

SQL Server Backward Compatibility Details

CAST and CONVERT

Use6xCompatible Property

## Unicode Character Format

The **-w** switch (or **widechar** value for the DATAFILETYPE clause of the BULK INSERT statement) uses the Unicode character data format for all columns, providing, as default terminators, tabs between fields and a newline character at the end of each row. This allows data to be copied both from a server using a code page different from the code page used by the client running **bcp**, and to another server with the same (or a different) code page as the original server:

- Without loss of any character data, if the source and destination are Unicode data types.

- With minimal loss of extended characters in the source data that cannot be represented at the destination if the source and destination are not Unicode data types.

For example, the command to bulk copy the **publishers** table in the **pubs** database to the Publ.txt file using Unicode character format is:

bcp pubs..publishers out publ.txt -w -S*servername* -Usa -P*password*

Unicode character format data files follow the conventions for Unicode files: the first two bytes of the file are either of the hexadecimal numbers 0xFEFF or 0xFFFE. These bytes serve as byte-order marks, specifying whether the high-order byte is stored first or last in the file.

To use field and row terminators other than the default provided with Unicode character format, specify the following.

| Terminator | bcp utility switch | BULK INSERT clause |
|---|---|---|
| Field | **-t** | FIELDTERMINATOR |
| Row | **-r** | ROWTERMINATOR |

For example, the command to bulk copy the **publishers** table to the Publ.txt data file using Unicode character format, with a comma as a field terminator and the newline character (\n) as the row terminator, is:

```
bcp pubs..publishers out publ.txt -w -t , -r \n -Sservername -Usa -Ppas
```

Two character positions are used for each character in the Publ.txt data file, with each field separated by a comma, and each row separated by a newline character.

**sql_variant** data stored in a Unicode character mode data file operates the same way it does in a character mode data file, except that the data is stored as **nchar** instead of **char** data.

## See Also

ServerBCPDataFileType Property

## Unicode Native Format

The **-N** switch (or **widenative** value for the DATAFILETYPE clause of the BULK INSERT statement) uses native (database) data types for all noncharacter data, and Unicode character data format for all character (**char**, **nchar**, **varchar**, **nvarchar**, **text**, and **ntext**) data.

Storing information in Unicode native format is useful when information must be copied from one Microsoft® SQL Server™ installation to another. Using native format for noncharacter data saves time, preventing unnecessary conversion of data types to and from character format. Using Unicode character format for all character data prevents loss of any extended characters when bulk loading data between servers using different code pages. However, a data file in Unicode native format can be read only by the **bcp** utility and the BULK INSERT statement.

For example, the command to bulk copy the **sales** table in the **pubs** database to the Sales.dat data file using Unicode native data format is:

```
bcp pubs..sales out Sales.dat -N -Sservername -Usa -Ppassword
```

**sql_variant** data stored as a SQLVARIANT in a Unicode native mode data file operates the same as it does in a native mode data file, except that **char** and **varchar** values are converted to **nchar** and **nvarchar**. The original meta data is preserved, and the values are converted back to their original **char** and **varchar** data type when bulk copied into the destination column.

### See Also

ServerBCPDataFileType Property

# Administering SQL Server

## Specifying Data Formats

If data is being copied between an instance of Microsoft® SQL Server™ and other programs, such as another database program, the default data type formats (native, character, or Unicode) may not be compatible with the data structures expected by the other programs. Therefore, the **bcp** utility allows more detailed information regarding the structure of the data file to be specified.

If the **-n**, **-c**, **-w**, or **-N** switches are not specified, the **bcp** utility prompts for further information interactively on each column of data being copied:

- File storage type
- Prefix length
- Field length
- Field terminator

  **Note** Interactive mode is not available when using the BULK INSERT statement.

The **bcp** utility provides default values at each of these prompts based on the SQL Server data type of the source or destination column. Accepting the default values supplied by **bcp** at these prompts produces the same result as native format (**-n**), and provides a way to bulk copy data out of other programs for later reloading into SQL Server.

A format file can be created to store the responses of the prompts for each field in the data file, allowing the same responses to be reused without having to enter them again. The format file can be used to provide all the format information required to bulk copy data to and from an instance of SQL Server. A format file provides a flexible system for writing data files that requires little or no editing to conform to other data formats, or for reading data files from other software.

For example, the command to bulk copy the **publishers** table interactively to the

Publ.txt file is:

```
bcp pubs..publishers out publ.txt -Sservername -Usa -Ppassword
```

A series of prompts appears for each column of the **publishers** table, with the **bcp**-supplied default displayed in brackets. This example is for the **pub_id** column in the **publishers** table only.

```
Enter the file storage type of field pub_id [char]:
Enter prefix length of field pub_id [0]:
Enter length of field pub_id [4]:
Enter field terminator [none]:
```

Pressing ENTER accepts the supplied default. To specify a value other than the default, enter the new value at the command prompt.

## See Also

Using Format Files

## File Storage Type

The file storage type describes how data is stored in the data file. Data can be copied to a data file as its database table type (native format), as a character string in ASCII format (character format), or as any data type where implicit conversion is supported (for example, copying a **smallint** as an **int**). User-defined data types are copied as their base types.

To bulk copy data from an instance of Microsoft® SQL Server™ to a data file in the most compact storage possible (native data format), accept the default file storage types provided by **bcp**.

To bulk copy data from an instance of SQL Server to a data file as ASCII text, specify **char** as the file storage type for all columns in the table.

To bulk copy data to an instance of SQL Server from a data file, specify the file storage type as **char** for ASCII-only files, and the following appropriate file storage type for data stored in native data type format.

| File storage type | Enter at command prompt |
|---|---|
| **char** | **c**[**har**] |
| **varchar** | **c**[**har**] |
| **nchar** | **w** |
| **nvarchar** | **w** |
| **text** | **T**[**ext**] |
| **ntext** | **W** |
| **binary** | **x** |
| **varbinary** | **x** |
| **image** | **I**[**mage**] |
| **datetime** | **d**[**ate**] |
| **smalldatetime** | **D** |
| **decimal** | **n** |
| **numeric** | **n** |
| **float** | **f**[**loat**] |
| | |

| | |
|---|---|
| **real** | **r** |
| **Int** | **i**[**nt**] |
| **bigint** | **B[igint]** |
| **smallint** | **s**[**mallint**] |
| **tinyint** | **t**[**inyint**] |
| **money** | **m**[**oney**] |
| **smallmoney** | **M** |
| **Bit** | **b**[**it**] |
| **uniqueidentifier** | **u** |
| **sql_variant** | **V[ariant]** |
| **timestamp** | **x** |

Entering a file storage type that represents an invalid implicit conversion causes **bcp** to fail. For example, specifying **smallint** for **int** data causes overflow errors, but specifying **int** for **smallint** data is valid. Specifying **char** as the file storage type when bulk copying any data type from an instance of SQL Server to a data file is always valid.

When noncharacter data types (for example, **float**, **money**, **datetime**, or **int**) are stored as their database types, the data is written to the data file in the SQL Server internal binary data format.

A format file can also be generated to save the responses of the file storage type for each field. This format file can be used to provide the default information used to bulk copy the data in the data file back into an instance of SQL Server, or to bulk copy data out from the table another time, without needing to respecify the format.

Each native file storage type is recorded in the format file as a corresponding host file data type.

| **File storage type** | **Host file data type** |
|---|---|
| **char** | SQLCHAR |
| **varchar** | SQLCHAR |
| **nchar** | SQLNCHAR |
| **nvarchar** | SQLNCHAR |
| **text** | SQLCHAR |
| | |

| | |
|---|---|
| **ntext** | SQLNCHAR |
| **binary** | SQLBINARY |
| **varbinary** | SQLBINARY |
| **image** | SQLBINARY |
| **datetime** | SQLDATETIME |
| **smalldatetime** | SQLDATETIM4 |
| **decimal** | SQLDECIMAL |
| **numeric** | SQLNUMERIC |
| **float** | SQLFLT8 |
| **real** | SQLFLT4 |
| **int** | SQLINT |
| **bigint** | SQLBIGINT |
| **smallint** | SQLSMALLINT |
| **tinyint** | SQLTINYINT |
| **money** | SQLMONEY |
| **smallmoney** | SQLMONEY4 |
| **bit** | SQLBIT |
| **uniqueidentifier** | SQLUNIQUEID |
| **sql_variant** | SQLVARIANT |
| **timestamp** | SQLBINARY |

Because data files stored as ASCII text use **char** as the file storage type, only SQLCHAR appears in the format file in those instances.

## See Also

Using Format Files

## Prefix Length

To provide the most compact file storage when bulk copying data in native format to a data file, **bcp** precedes each field with one or more characters that indicates the length of the field. These characters are called length prefix characters. The number of length prefix characters required is called the prefix length.

The number of length prefix characters required to store the length of the data field depends on the file storage type, the nullability of a column, and whether the data is being stored in the data file in its native (database) data type or as ASCII characters (character format). A **text** or **image** data type requires four prefix characters to store the field length, whereas a **varchar** data type requires two characters.

**Note** These length prefix characters are stored in the data file in Microsoft® SQL Server™ internal binary data format.

Null values are represented as an empty field when copied from an instance of SQL Server to a data file. To indicate that the field is empty (represents NULL), the field prefix contains the value -1. Any SQL Server column that allows null values requires a prefix length of 1 or greater, depending on the file storage type.

Use these prefix lengths when bulk copying data from an instance of SQL Server to a data file, storing the data using either native data types or as ASCII characters (text file).

| **SQL Server** | **Native format** | | **Character format** | |
|---|---|---|---|---|
| **data type** | **NOT NULL** | **NULL** | **NOT NULL** | **NULL** |
| **char** | 2 | 2 | 2 | 2 |
| **varchar** | 2 | 2 | 2 | 2 |
| **nchar** | 2 | 2 | 2 | 2 |
| **nvarchar** | 2 | 2 | 2 | 2 |
| **text** | 4 | 4 | 4 | 4 |
| **ntext** | 4 | 4 | 1 | 1 |
| **binary** | 1 | 1 | 2 | 2 |

| | | | | |
|---|---|---|---|---|
| **varbinary** | 1 | 1 | 2 | 2 |
| **image** | 4 | 4 | 4 | 4 |
| **datetime** | 0 | 1 | 1 | 1 |
| **smalldatetime** | 0 | 1 | 1 | 1 |
| **decimal** | 1 | 1 | 1 | 1 |
| **numeric** | 1 | 1 | 1 | 1 |
| **float** | 0 | 1 | 1 | 1 |
| **real** | 0 | 1 | 1 | 1 |
| **int** | 0 | 1 | 1 | 1 |
| **bigint** | 0 | 1 | 1 | 1 |
| **smallint** | 0 | 1 | 1 | 1 |
| **tinyint** | 0 | 1 | 1 | 1 |
| **money** | 0 | 1 | 1 | 1 |
| **smallmoney** | 0 | 1 | 1 | 1 |
| **bit** | 0 | 1 | 0 | 1 |
| **uniqueidentifier** | 1 | 1 | 1 | 1 |
| **timestamp** | 1 | 1 | 2 | 2 |

When storing data as **nchar** rather than **char**, the prefix length for all data types is the same as the native data type value, except **char**, **varchar**, **text**, **ntext**, and **image**, which all have a prefix length of 1.

When bulk copying data to an instance of SQL Server, the prefix length is the value specified when the data file was created originally. If the data file was not created with **bcp**, it is unlikely that length prefix characters exist. In this instance, specify 0 for the prefix length.

**Note**  The default values provided at the prompts indicate the most efficient prefix lengths.

## Field Length

When bulk copying **char**, **nchar**, or **binary** data with a prefix length of 0 from Microsoft® SQL Server™, **bcp** also prompts for a field length. The field length indicates the maximum number of characters needed to represent data in character format. A column of type **tinyint** can have values from 0 through 255; the maximum number of characters needed to represent any number in that range is three (representing values 100 through 255). When **bcp** converts noncharacter data to character, it suggests a default field length large enough to store the data.

If the file storage type is noncharacter, data is stored in the SQL Server native data representation (native format) and the **bcp** utility does not prompt for a field length.

These are the default field lengths for data to be stored as **char** file storage type (nullable data is the same length as nonnull data).

| **Data type** | **Default length (characters)** |
|---|---|
| **Char** | Length defined for the column |
| **Varchar** | Length defined for the column |
| **Nchar** | Twice the length defined for the column |
| **Nvarchar** | Twice the length defined for the column |
| **Text** | 0 |
| **Ntext** | 0 |
| **Bit** | 1 |
| **Binary** | Twice the length defined for the column + 1 |
| **Varbinary** | Twice the length defined for the column + 1 |
| **Image** | 0 |
| **Datetime** | 24 |
| **Smalldatetime** | 24 |
| **Float** | 30 |
| **Real** | 30 |

| | |
|---|---|
| **Int** | 12 |
| **Bigint** | 19 |
| **Smallint** | 7 |
| **Tinyint** | 5 |
| **Money** | 30 |
| **Smallmoney** | 30 |
| **Decimal** | 41* |
| **Numeric** | 41* |
| **Uniqueidentifier** | 37 |
| **Timestamp** | 17 |

*For more information about the **decimal** and **numeric** data types, see decimal and numeric.

These are the default field lengths for data to be stored as native file storage type (nullable data is the same length as nonnull data, and character data is always stored in character format).

| **Data type** | **Default length (characters)** |
|---|---|
| **bit** | 1 |
| **binary** | Length defined for the column |
| **varbinary** | Length defined for the column |
| **image** | 0 |
| **datetime** | 8 |
| **smalldatetime** | 4 |
| **float** | 8 |
| **real** | 4 |
| **int** | 4 |
| **bigint** | 8 |
| **smallint** | 2 |
| **tinyint** | 1 |
| **money** | 8 |
| **smallmoney** | 4 |
| **decimal** | * |
| **numeric** | * |
| **uniqueidentifier** | 16 |
| **timestamp** | 8 |

*For more information about the **decimal** and **numeric** data types, see decimal and numeric.

Accepting the **bcp** default values for the field length is recommended.

**Note** Using default data type sizes (field length) can lead to an "unexpected end of file" error message. This generally occurs with the **money** and **datetime** data types when only part of the field occurs in the data file (for example, a **datetime** value of *mm/dd/yy* with no time component) rather than an entire string, as expected by SQL Server. When using the default size option, SQL Server expects to read 24 characters (the length of the **datetime** data type when stored in **char** format). To avoid this problem, bulk copy data using field terminators, or fixed-length data fields.

Specifying a field length too short for numeric data when bulk copying data causes **bcp** to print an overflow message and not copy the data. When **datetime** data is copied to a data file as a character string of less than 26 bytes, the data is truncated without an error message. When creating an ASCII data file, use the default field length to ensure that data is not truncated and that numeric overflow errors causing **bcp** to fail do not occur. To change the default field length, supply another value.

**Note** To create a data file for later reloading into SQL Server and keep the storage space to a minimum, use a length prefix character with the default file storage type and the default field length.

The amount of storage space allocated in the data file for noncharacter data stored as **char** file storage type also depends on whether a prefix length or terminators are specified:

- If specifying a prefix length of 1, 2, or 4, the field length is not used. The data file storage space used is the length of the data, the length of the prefix, plus any terminators.

- If specifying a prefix length of 0 and no terminator, **bcp** allocates the maximum amount of space shown in the field length prompt because this is the maximum space that may be needed for the data type in question. The field is treated as if it were of fixed length so that it is possible to determine where one field ends and the next begins.

- If specifying a prefix length of 0 and a terminator, the field length specified is ignored. The data file storage space used is the length of the data, plus any terminators.

SQL Server **char** data is always stored in the data file as the full length of the defined column. For example, a column defined as **char**(10) always occupies 10 characters in the data file regardless of the length of the data stored in the column; spaces are appended to the data as padding. For more information, see SET ANSI_PADDING.

The interaction of prefix lengths (P), terminators (T), and field length on data determines the storage space used in the data file. In this example, the field length is 8 for each column, and the 6-character value "string" is stored each time. Dashes (-) indicate appended spaces and ellipses (...) indicate that the pattern repeats for each field.

This is the pattern for SQL Server **char** data.

| | **Prefix length = 0** | **Prefix length = 1, 2, or 4** |
|---|---|---|
| **No terminator** | *string*--*string*--... | P*string*--P*string*--... |
| **Terminator** | *string*--T*string*--T... | P*string*--TP*string*--T... |

This is the pattern for other data types converted to **char** storage.

| | **Prefix length = 0** | **Prefix length = 1, 2, or 4** |
|---|---|---|
| **No terminator** | *string*--*string*--... | P*string*P*string*... |
| **Terminator** | *string*T*string*T... | P*string*TP*string*T... |

## Field Terminator

It is possible to use optional terminating characters to mark the end of a field or row, separating one field or row in the data file from the next. Terminating characters indicate to a program reading the data file where one field or row ends and another begins. The default provided by the **bcp** utility is to use no terminating characters between fields and rows in the data file.

Field terminators are needed when the data file does not contain:

- Length prefixes to indicate the length of each field (perhaps because the program reading the data file does not understand length prefixes).

- Fixed-length data fields (perhaps because storage space needs to be minimized).

The **bcp** utility allows many characters to be used as field or row terminators.

| Terminator | Indicated by |
|---|---|
| Tab | \t |
| Newline character | \n |
| Carriage return | \r |
| Backslash | \\ |
| Null terminator (no visible terminator) | \0 |
| Any printable character (control characters are not printable, except null, tab, newline, and carriage return) | (*, A, t, l, and so on) |
| String of up to 10 printable characters, including some or all of the terminators listed earlier | (**\t**, end, !!!!!!!!!!, \t--\n, and so on) |

**Note** Only the t, n, r, \, and 0 characters work with the backslash escape character to produce a control character.

It is possible to change the default field and row terminators using the **-t** and **-r** switches of **bcp**. When using these switches, the bracketed default listed in the interactive **bcp** prompt changes for all fields and rows to the value specified at the command prompt. Use **-t** to change the default field terminator and **-r** to change the default row terminator.

The command to change the default field terminator to a comma (,) and the default row terminator to the newline character (\n):

```
bcp pubs..publishers out publ.txt -t , -r \n -Sservername -Usa -Ppasswo
```

**IMPORTANT** Terminators must be chosen to ensure that their pattern does not appear in any of the data. For example, when using tab terminators with a field that contains tabs as part of the data, **bcp** does not know which tab represents the end of the field. The **bcp** utility always looks for the first possible character(s) that matches the terminator it expects. Using a character sequence with characters that do not occur in the data avoids this conflict.

Native format data can also conflict with terminators because this file is in the SQL Server internal binary data format. When using native format, use length prefixes rather than field terminators.

Any data column that contains null values is considered variable length for bulk copy purposes. Therefore, a length prefix or field terminator needs to be used to specify the length of each field.

**Note** The no terminator value is different from the null terminator (\0) value. The no terminator value places no row terminator character(s). The null terminator value puts a null character after the column. A null character is invisible but real.

Because **bcp** does not prompt for a row terminator, the field terminator for the last column in a row serves that purpose. Given a row with 10 columns, the field terminator for the tenth column is also the row terminator. Therefore, the terminator for the last field can be (but is not required to be) different from the field terminator used for other fields in the same row. For tabular output, terminate the last field with the newline character (\n) and all other fields with the tab character (\t).

A common row terminator used when exporting SQL Server data to ASCII data files is \r\n (carriage return, newline). Using both characters as the row terminator ensures that each row of data appears on its own line in the data file. However, it is only necessary to enter the characters \r\n as the terminator when manually editing the terminator column of a **bcp** format file. When you use **bcp** interactively and specify \n (newline) as the row terminator, **bcp** prefixes the \r (carriage return) character automatically.

## See Also

ColumnDelimiter Property

RowDelimiter Property

# Administering SQL Server

## Using Format Files

When bulk copying data using interactive mode, the **bcp** utility prompts you to store information regarding the storage type, prefix length, field length, and field and row terminators. The file used to store the format information for each field in the data file is called the format file:

```
Do you want to save this format information in a file? [Y/n] y
Host filename: [bcp.fmt]
```

Although the default name for the format file is Bcp.fmt, a different file name can be specified.

This format file provides the default information used either to bulk copy the data in the data file back into an instance of Microsoft® SQL Server™ or to bulk copy data out from the table another time, without needing to respecify the format. When bulk copying data into or out of an instance of SQL Server with an existing format file, **bcp** does not prompt for the file storage type, prefix length, field length, or field terminator because it uses the values already recorded.

To use a previously created format file when importing data into an instance of SQL Server, use the **-f** switch with the **bcp** utility or the FORMATFILE clause with the BULK INSERT statement. For example, the command to bulk copy the contents of New_auth.dat data file into the **authors2** table in the **pubs** database using the previously created format file (Authors.fmt) is:

```
bcp pubs..authors2 in c:\new_auth.dat -fc:\authors.fmt -Sservername -
```

The BULK INSERT statement can use format files saved by the **bcp** utility. For example:

```
BULK INSERT pubs..authors2 FROM 'c:\new_auth.dat'
WITH (FORMATFILE = 'c:\authors.fmt')
```

The format file is a tab-delimited text file with a specific structure.

The following table describes the file format structures.

| Field | Description |
|---|---|
| Version | Version number of **bcp**. |
| Number of fields | Number of fields in the data file. This must be the same for all rows. |
| Host file field order | Position of each field within the data file. The first field in the row is 1, and so on. |
| Host file data type | Data type stored in the particular field of the data file. With ASCII data files, use SQLCHAR; for native format data files, use default data types. For more information, see File Storage Type. |
| Prefix length | Number of length prefix characters for the field. Legal prefix lengths are 0, 1, 2, and 4. To avoid specifying the length prefix, set this to 0. A length prefix must be specified if the field contains null data values. For more information, see Prefix Length. |
| Host file data length | Maximum length, in bytes, of the data type stored in the particular field of the data file. For more information, see Field Length. |
| Terminator | Delimiter to separate the fields in a data file. Common terminators are comma (,), tab (\t), and end of line (\r\n). For more information, see Field Terminator. |
| Server column order | Order that columns appear in the SQL Server table. For example, if the fourth field in the data file maps to the sixth column in a SQL Server table, then for the fourth field the server column order is 6.<br>To omit a column in the table from receiving any data in the data file, set the server column order value to 0. |
| Server column name | Name of the column taken from the SQL Server table. It is not necessary to use the actual name of the field. The only condition is that the field in the format file not be blank. |
| Collation | The collation used to store character and Unicode data in the bulk copy data file. |

**Note** It is possible to skip importing a table column if the field does not exist in the data file by specifying 0 prefix length, 0 length, 0 server column order, and no terminator. This effectively states that the data field does not exist in the data file, and that the server column should not have data loaded into it.

## Selectively Copying Data

A format file provides a way to bulk copy data selectively from a data file to an instance of SQL Server. This allows the transfer of data to a table when there is a mismatch between fields in the data file and columns in the table. This approach can be used when the fields in the data file are:

- Fewer than the columns in the table.
- More than the columns in the table.
- In a different order from the columns in the table.

By using a format file, it is possible to bulk copy data into an instance of SQL Server without having to add or delete unnecessary data, or reorder existing data, in the data file.

The following three topics contain examples of selectively copying data. For the following examples, first make a copy of the **authors** table, named **authors2**, in the **pubs** database. To create a copy of the **authors** table, execute:

```
USE pubs
GO
SELECT * INTO authors2 FROM authors
GO
```

### See Also

FormatFilePath Property

## Using a Data File with Fewer Fields

In some cases, a data file may have fewer fields than there are columns in the table. For example, the New_auth.dat data file (ASCII, or character format) does not contain matching fields for the **address** and **zip** columns in the **authors2** table.

The New_auth.dat file:

```
777-77-7777,Smith,Chris,303 555-1213,Denver,CO,1
888-88-8888,Doe,John,206 555-1214,Seattle,WA,0
999-99-9999,Door,Jane,406 555-1234,Bozeman,MT,1
```

To bulk copy data selectively to the correct columns in **authors2**, create a default format file (Authors.fmt) with the following command:

```
bcp pubs..authors2 out c:\authors.txt -Sservername -Usa -Ppassword
```

The **bcp** utility prompts for the file storage type, prefix length, field length, and field terminator of each column of **authors2**. The field terminator for every column should be a comma (,), except for the **contract** column, which should use the row terminator \n (newline) because it is the last column in the row. Also, the **contract** column has a file storage type of **char**, because the data file is an ASCII file. The **address** and **zip** columns should not have field terminators and should have their field length set to 0. When prompted for the format file name, specify Authors.fmt.

The Authors.fmt file:

```
8.0
9
1  SQLCHAR  0  11 ","     1  au_id      SQL_Latin1_General_Cp437_
2  SQLCHAR  0  40 ","     2  au_lname    SQL_Latin1_General_Cp43
3  SQLCHAR  0  20 ","     3  au_fname    SQL_Latin1_General_Cp43
4  SQLCHAR  0  12 ","     4  phone      SQL_Latin1_General_Cp437_
5  SQLCHAR  0  0  ""      5  address    SQL_Latin1_General_Cp437_
```

```
6  SQLCHAR  0  20 ","      6  city        SQL_Latin1_General_Cp437_E
7  SQLCHAR  0  2  ","     7  state       SQL_Latin1_General_Cp437_B
8  SQLCHAR  0  0  ""      8  zip         SQL_Latin1_General_Cp437_B
9  SQLCHAR  0  1  "\r\n"   9  contract    SQL_Latin1_General_Cp437_
```

The format file contains all the information necessary to bulk copy data from the data file to the Microsoft® SQL Server™ table. A prefix length of 0, field length of 0, and no field terminator for **address** and **zip** means that these columns do not exist in the data file. However, the format file must be modified further with a text editor to ensure that no data will be loaded into **address** and **zip**. The server column numbers (sixth field in the format file) for these columns should be 0:

```
8.0
9
1  SQLCHAR  0  11 ","      1  au_id       SQL_Latin1_General_Cp437_
2  SQLCHAR  0  40 ","      2  au_lname    SQL_Latin1_General_Cp43
3  SQLCHAR  0  20 ","      3  au_fname    SQL_Latin1_General_Cp43
4  SQLCHAR  0  12 ","      4  phone       SQL_Latin1_General_Cp437_
5  SQLCHAR  0  0  ""      0  address     SQL_Latin1_General_Cp437_
6  SQLCHAR  0  20 ","      6  city        SQL_Latin1_General_Cp437_E
7  SQLCHAR  0  2  ","     7  state       SQL_Latin1_General_Cp437_B
8  SQLCHAR  0  0  ""      0  zip         SQL_Latin1_General_Cp437_B
9  SQLCHAR  0  1  "\r\n"   9  contract    SQL_Latin1_General_Cp437_
```

The data in the data file can now be bulk copied into **authors2** using the command:

```
bcp pubs..authors2 in c:\new_auth.dat -fc:\authors.fmt -Sservername -U
```

Alternatively, you can use the BULK INSERT statement from a query tool, such as SQL Query Analyzer, to bulk copy data:

```
BULK INSERT pubs..authors2 FROM 'c:\new_auth.dat'
WITH (FORMATFILE = 'c:\authors.fmt')
```

**Note** Because **address** and **zip** are not present in the data file, those columns will contain NULL in the SQL Server table if no DEFAULT values have been defined. Therefore, **authors2** must allow null values in those columns.

## Using a Data File with More Fields

In some cases, a data file may have more fields than there are columns in the table. For example, the New_auth.dat data file (ASCII, or character format) contains two fields (**age** and **salutation**) not contained on **authors2**. These fields will be omitted, or skipped, during the bulk copy procedure.

The New_auth.dat file:

```
777-77-7777,Smith,Chris,303 555-1213,27 College Ave,Denver,CO,8(
888-88-8888,Doe,John,206 555-1214,123 Maple Street,Seattle,WA,95(
999-99-9999,Door,Jane,406 555-1234,45 East Main,Bozeman,MT,597
```

To bulk copy data selectively to the correct columns in **authors2** only, create a default format file (Authors.fmt) with the command:

```
bcp pubs..authors2 out c:\authors.txt -Sservername -Usa -Ppassword
```

The **bcp** utility prompts for the file storage type, prefix length, field length, and field terminator of each column of **authors2**. The field terminator for every column should be a comma (,). Also, the **contract** column has a file storage type of **char** because the data file is an ASCII file. When prompted for the format file name, specify Authors.fmt.

The Authors.fmt file:

```
8.0
9
1  SQLCHAR  0  11 ","     1  au_id       SQL_Latin1_General_Cp437_
2  SQLCHAR  0  40 ","     2  au_lname    SQL_Latin1_General_Cp43
3  SQLCHAR  0  20 ","     3  au_fname    SQL_Latin1_General_Cp43
4  SQLCHAR  0  12 ","     4  phone       SQL_Latin1_General_Cp437_
5  SQLCHAR  0  40 ","     5  address     SQL_Latin1_General_Cp437_
6  SQLCHAR  0  20 ","     6  city        SQL_Latin1_General_Cp437_E
7  SQLCHAR  0  2  ","     7  state       SQL_Latin1_General_Cp437_B
```

```
8  SQLCHAR  0  5  ","      8  zip        SQL_Latin1_General_Cp437_B
9  SQLCHAR  0  1  "\r\n"   9  contract   SQL_Latin1_General_Cp437_
```

The format file contains all the information necessary to bulk copy data from the data file to the Microsoft® SQL Server™ table. However, the format file needs to be modified further with a text editor to reflect the addition of two new columns: **age** and **salutation**. The second line of the format file specifies the number of columns and should now be changed to 11 because there are 11 fields in the data file. Two new rows need to be added to the end of the format file to provide format information for the additional fields. The row terminator needs to be moved from the **contract** column to the **salutation** column and the server column numbers (sixth field in the format file) for the **age** and **salutation** columns should be 0:

```
8.0
11
1  SQLCHAR  0  11 ","      1  au_id      SQL_Latin1_General_Cp437_
2  SQLCHAR  0  40 ","      2  au_lname   SQL_Latin1_General_Cp43
3  SQLCHAR  0  20 ","      3  au_fname   SQL_Latin1_General_Cp43
4  SQLCHAR  0  12 ","      4  phone      SQL_Latin1_General_Cp437_
5  SQLCHAR  0  40 ","      5  address    SQL_Latin1_General_Cp437_
6  SQLCHAR  0  20 ","      6  city       SQL_Latin1_General_Cp437_E
7  SQLCHAR  0  2  ","      7  state      SQL_Latin1_General_Cp437_B
8  SQLCHAR  0  5  ","      8  zip        SQL_Latin1_General_Cp437_B
9  SQLCHAR  0  1  ","      9  contract   SQL_Latin1_General_Cp437_
10 SQLCHAR  0  0  ","      0  age        SQL_Latin1_General_Cp437_I
11 SQLCHAR  0  0  "\r\n"   0  salutation  SQL_Latin1_General_Cp43
```

The data in the data file can now be bulk copied into **authors2** using the command:

```
bcp pubs..authors2 in c:\new_auth.dat -fc:\authors.fmt -Sservername -
```

Alternatively, you can use the BULK INSERT statement from a query tool, such as SQL Query Analyzer, to bulk copy data:

```
BULK INSERT pubs..authors2 FROM 'c:\new_auth.dat'
WITH (FORMATFILE = 'c:\authors.fmt')
```

## Using a Data File with Fields in a Different Order

In some cases, a data file may have fields in an order different from the corresponding columns in the table. For example, the New_auth.dat data file (ASCII, or character format) contains the same number of fields as the **authors2** table, but the **au_lname** and **au_fname** fields are reversed. These fields will be reordered during the bulk copy procedure.

The New_auth.dat file:

```
777-77-7777,Chris,Smith,303 555-1213,27 College Ave,Denver,CO,8(
888-88-8888,John,Doe,206 555-1214,123 Maple Street,Seattle,WA,95
999-99-9999,Jane,Door,406 555-1234,45 East Main,Bozeman,MT,597
```

To bulk copy data selectively to the correct columns in **authors2**, create a default format file (Authors.fmt) with the command:

```
bcp pubs..authors2 out c:\authors.txt -Sservername -Usa -Ppassword
```

The **bcp** utility prompts for the file storage type, prefix length, field length, and field terminator of each column of **authors2**. The field terminator for every column should be a comma (,), except for the **contract** column, which should use the row terminator \n (newline) because it is the last column in the row. Also, the **contract** column has a file storage type of **char** because the data file is an ASCII file. When prompted for the format file name, specify Authors.fmt.

The Authors.fmt file:

```
8.0
9
1  SQLCHAR  0  11 ","     1  au_id       SQL_Latin1_General_Cp437_
2  SQLCHAR  0  40 ","     2  au_lname    SQL_Latin1_General_Cp43
3  SQLCHAR  0  20 ","     3  au_fname    SQL_Latin1_General_Cp43
4  SQLCHAR  0  12 ","     4  phone       SQL_Latin1_General_Cp437_
5  SQLCHAR  0  40 ","     5  address     SQL_Latin1_General_Cp437_
6  SQLCHAR  0  20 ","     6  city        SQL_Latin1_General_Cp437_E
```

```
7  SQLCHAR  0  2  ","     7  state       SQL_Latin1_General_Cp437_B
8  SQLCHAR  0  5  ","     8  zip         SQL_Latin1_General_Cp437_B
9  SQLCHAR  0  1  "\r\n"   9  contract    SQL_Latin1_General_Cp437
```

The format file contains all the information necessary to bulk copy data from the data file to the Microsoft® SQL Server™ table. However, the format file needs to be further modified with a text editor to change the server column order (sixth field in the format file) of the **au_lname** and **au_fname** fields.

```
8.0
9
1  SQLCHAR  0  11 ","     1  au_id       SQL_Latin1_General_Cp437_
2  SQLCHAR  0  40 ","     3  au_lname    SQL_Latin1_General_Cp43
3  SQLCHAR  0  20 ","     2  au_fname    SQL_Latin1_General_Cp43
4  SQLCHAR  0  12 ","     4  phone       SQL_Latin1_General_Cp437_
5  SQLCHAR  0  40 ","     5  address     SQL_Latin1_General_Cp437_
6  SQLCHAR  0  20 ","     6  city        SQL_Latin1_General_Cp437_E
7  SQLCHAR  0  2  ","     7  state       SQL_Latin1_General_Cp437_B
8  SQLCHAR  0  5  ","     8  zip         SQL_Latin1_General_Cp437_B
9  SQLCHAR  0  1  "\r\n"   9  contract    SQL_Latin1_General_Cp437
```

The data in the data file can now be bulk copied into **authors2** using the command:

```
bcp pubs..authors2 in c:\new_auth.dat -fc:\authors.fmt -Sservername -
```

Alternatively, you can use the BULK INSERT statement from a query tool such as SQL Query Analyzer to bulk copy data:

```
BULK INSERT pubs..authors2 FROM 'c:\new_auth.dat'
WITH (FORMATFILE = 'c:\authors.fmt')
```

# Administering SQL Server

# Copying Data

There are six options for copying data using bcp or BULK INSERT.

| Topic | Description |
|---|---|
| Copying Data Between Servers | Describes which data format to use when copying data between instances of Microsoft® SQL Server™. |
| Copying Data from a Data File to SQL Server | Describes how to copy data from a data file to an instance of SQL Server, including how to handle identity values and image data. |
| Copying Data From a Query to a Data File | Describes how to copy the result set from a Transact SQL statement to a data file. |
| Copying Data To or From a Temporary Table | Describes how to copy data to or from a temporary table. |
| Copying Data To or From a View | Describes how to copy data to or from a view. |
| Copying Data Between Different Collations | Describes how to copy data between different collations, including the use of column-level collations. |

## Copying Data Between Servers

To bulk copy data from one Microsoft® SQL Server™ database to another, data from the source database must first be bulk copied into a file. The file is then bulk copied into the destination database.

After bulk copying data into a table, if the recovery model is simple, then a full or differential backup is recommended. If the recovery model is bulk-logged or full, a log backup is sufficient.

**Note** Native, character, and Unicode format **bcp** can be used to bulk copy data between different instances of SQL Server on different processor architectures. However, the same format must be used when importing as exporting.

Storing information in Unicode native format is useful when information must be copied from one instance of SQL Server to another. Using native format for noncharacter data saves time, preventing unnecessary conversion of data types to and from character format. Using Unicode character format for all character data prevents loss of any extended characters when bulk loading data between servers using different code pages (character loss is possible if extended characters are copied into non-Unicode columns and the extended character cannot be represented). However, a data file in Unicode native format cannot be read by any program other than **bcp** or the BULK INSERT statement.

It is also possible to copy data from one SQL Server database to another using:

- The DTS Import/Export Wizard.
- The Transact-SQL statements BACKUP and RESTORE (to copy entire databases).
- Distributed queries as part of an INSERT statement.
- The SELECT INTO statement.

## See Also

BACKUP

Distributed Queries

DTS Import/Export Wizard

INSERT

Optimizing Bulk Copy Performance

RESTORE

SELECT

Unicode Character Format

## Copying Data From a Data File to SQL Server

To bulk copy a data file to an instance of Microsoft® SQL Server™, follow these guidelines:

- When bulk copying data to a table with no indexes, set the recovery model to bulk-logged if you usually use full recovery.

  This is recommended to help prevent the transaction log from running out of space because row inserts are not logged. The system administrator or database owner can set this option. For more information, see Logged and Minimally Logged Bulk Copy Operations.

- If you are loading a large amount of data relative to the amount of data already in the table, it can be quicker to drop the indexes on the table before performing the bulk copy operation.

  Conversely, if you are loading a small amount of data relative to the amount of data already in the table, dropping the indexes may not be necessary because the time taken to rebuild the indexes can be longer than performing the bulk copy operation. For more information, see Optimizing Bulk Copy Performance.

- Be sure that the user account used to log in to SQL Server using **bcp** (or the query tool when using the BULK INSERT statement) has SELECT and INSERT permissions on the table (assigned by the table owner).

  **Note**  Only members of the **sysadmin** fixed server role can execute the BULK INSERT statement.

- If the recovery model is simple, then a full or differential backup is recommended; for bulk-logged recovery and full recovery, a log backup is sufficient. For more information, see Backup and Restore Operations.

- To bulk copy data successfully into a table from a data file with the **bcp** utility or BULK INSERT statement, the terminators in the data file must be known and specified.

**Note** A hidden character in an ASCII data file can cause problems when trying to bulk copy data into an instance of SQL Server, resulting in an "unexpected null found" error message. Many utilities and text editors display hidden characters which can usually be found at the bottom of the data file. Finding and removing these characters should resolve the problem.

The Newpubs.dat file:

```
1111,Stone Age Books,Boston,MA,USA
2222   ,Harley & Davidson,Washington,DC,USA
3333   ,Infodata Algosystems,Berkeley,CA,USA
```

Because the data file is all character data, the following options and switches need to be specified.

| Bulk copy option | bcp utility switch | BULK INSERT clause |
|---|---|---|
| Character mode format | **-c** | DATAFILETYPE = **'char'** |
| Field terminator | **-t** | FIELDTERMINATOR |
| Row terminator | **-r** | ROWTERMINATOR |

In the Newpubs.dat file, each field in a row ends with a comma (,); each row ends with a newline character (\n).

The **publishers2** table in the following example can be created by executing:

```
USE pubs
GO
SELECT * INTO publishers2 FROM publishers
GO
```

The command to bulk copy data from Newpubs.dat into **publishers2** is:

```
bcp pubs..publishers2 in newpubs.dat -c -t , -r \n -Sservername -Usa -P
```

Alternatively, you can use the BULK INSERT statement from a query tool, such as SQL Query Analyzer, to bulk copy data:

```
BULK INSERT pubs..publishers2 FROM 'c:\newpubs.dat'
```

```
WITH (
   DATAFILETYPE = 'char',
   FIELDTERMINATOR = ',',
   ROWTERMINATOR = '\n'
)
```

Data from the Newpubs.dat file has been now appended to **publishers2**:

| Pub_id | pub_name | city | state | Country |
|---|---|---|---|---|
| ------ | ---------------- | ---------- | ----- | ----- |
| 0736 | New Moon Books | Boston | MA | USA |
| 0877 | Binnet & Hardley | Washington | DC | USA |
| 1111 | Stone Age Books | Boston | MA | USA |
| 1389 | Algodata Infosystems | Berkeley | CA | USA |
| 1622 | Five Lakes Publishing | Chicago | IL | USA |
| 1756 | Ramona Publishers | Dallas | TX | USA |
| 2222 | Harley & Davidson | Washington | DC | USA |
| 3333 | Infodata Algosystems | Berkeley | CA | USA |
| 9901 | GGG&G | München |  | Germany |
| 9952 | Scootney Books | New York | NY | USA |
| 9999 | Lucerne Publishing | Paris |  | France |

## Copying Data Containing Identity Values

The **bcp** utility and BULK INSERT statement allow data files containing identity values to be bulk copied into an instance of SQL Server. To prevent SQL Server from supplying identity values, the **bcp** utility accepts the **-E** switch, and the BULK INSERT statement accepts the KEEPIDENTITY clause. While the rows in the data file are bulk copied into the table, SQL Server does not assign unique identity values automatically; the identity values are taken from the data file.

If these options are not supplied, the values for the identifier column in the data file being imported are ignored and SQL Server assigns unique values automatically based on the seed and increment values specified during table creation. If the data file does not contain values for the identifier column in the table, use a format file to specify that the identifier column in the table should be skipped when importing data. SQL Server assigns unique values automatically for the column.

## Importing Image Data

It is possible to bulk copy a data file as **image** data into an instance of SQL Server. The command to load the data file Test.doc into the **bitmap** table in the **pubs** database using the **bcp** utility is:

```
bcp pubs..bitmap in test.doc -Usa -Ppassword -Sservername
```

**bcp** prompts:

```
Enter the file storage type of field c1 [image]:
Enter the prefix length of field c1 [4]: 0
Enter length of field c1 [4096]: 5578
Enter the field terminator [none]:
```

In this example, the data file will be loaded into column **c1**, and **5578** is the length of the data file.

Using the BULK INSERT statement, a format file needs to be created first and then used to provide the format information. To create the format file, use the **bcp** utility:

```
bcp pubs..bitmap out c:\bitmap.txt -Sservername -Usa -Ppassword
```

The **bcp** utility prompts for the file storage type, prefix length, field length, and field terminator of each column of **bitmap**. The values for the **c1** column are listed in this table.

| Prompt | Value |
|---|---|
| File storage type | **Image** |
| Prefix length | 0 |
| Field length | 5578 |
| | |

|  |  |
|---|---|
| Field terminator | None |

The Bcp.fmt file:

```
8.0
1
1   SQLIMAGE   0   5578      ""      1   c1
```

Using the BULK INSERT statement to bulk copy the Test.doc data file into the **bitmap** table in the **pubs** database, execute from a query tool, such as SQL Query Analyzer:

```
BULK INSERT pubs..bitmap FROM 'c:\test.doc'
WITH (
   FORMATFILE = 'c:\Bcp.fmt'
)
```

**Note**  You cannot bulk copy data into **text**, **ntext**, and **image** columns that have DEFAULT values.

## See Also

bcp Utility

BULK INSERT.

ImportData Method

IncludeIdentityValues Property

SuspendIndexing Property

UseBulkCopyOption Property

Using a Data File with Fewer Fields

## Copying Data From a Query to a Data File

The **bcp** utility allows you to copy the result set from a Transact-SQL statement to a data file. The Transact-SQL statement can be any valid statement that returns a results set, such as a distributed query or a SELECT statement joining several tables. For example, to copy the names of all the authors, ordered by surname, from the **authors** table in the **pubs** database to the Authors.txt data file, execute at the command prompt:

```
bcp "SELECT au_fname, au_lname FROM pubs..authors ORDER BY
```

Bulk copying data from a query is useful if you want to ensure that the order of the data is preserved in the data file; bulk copying data from a table or view does not guarantee the order of the data written to the data file. Preserving the order of the data in the data file allows you to make use of the **ORDER** hint when bulk copying data from the data file back into a table. Using the **ORDER** hint can significantly improve bulk copy performance. For more information, see Optimizing Bulk Copy Performance.

If the Transact-SQL statement returns multiple result sets, such as a SELECT statement that specifies the COMPUTE clause, or the execution of a stored procedure that contains multiple SELECT statements, only the first result set is copied; subsequent result sets are ignored.

### See Also

bcp Utility

Ordered Data Files

## Copying Data To or From a Temporary Table

When using **bcp** or BULK INSERT to bulk copy data using a global temporary table, the table name must be specified at the command prompt, including initial number signs (##). For example, to bulk copy data from the global temporary table **##temp_authors** to the Temp_authors.txt data file, execute at the command prompt:

```
bcp ##temp_authors out temp_authors.txt -c -Sservername -Usa -Ppas
```

However, do not specify the database name when using global temporary tables because temporary tables exist only in **tempdb**. It is possible to use a local temporary table (for example, **#temp_authors**) only when bulk copying data using the BULK INSERT statement.

## Copying Data To or From a View

Data can be bulk copied to or from a view. This includes copying data from multiple joined tables, adding a WHERE clause, or performing special formatting, such as changing data formats using the CONVERT function. For example, to bulk copy data from the view **titleview** in the **pubs** database to the Titleview.txt data file, execute at the command prompt:

```
bcp pubs..titleview out titleview.txt -c -Sservername -Usa -Ppassword
```

To bulk copy data into a view using **bcp** or the BULK INSERT statement, the rules for inserting data into a view apply.

**Note** When data is bulk copied into a view, NULL values will be inserted even if a default value is defined for the field.

### See Also

Modifying Data Through a View

## Copying Data Between Different Collations

When bulk copying data using native or character format, **bcp**, by default, converts character data to:

- OEM code page characters when exporting data from an instance of Microsoft® SQL Server™.

- ANSI/Microsoft Windows® code page characters when importing data into an instance of SQL Server.

This can cause the loss of extended or DBCS characters during the conversion between OEM and ANSI code pages. To prevent the loss of extended or DBCS characters, **bcp** can create data files using:

- Unicode native data format (**-N**).

- Unicode character data format (**-w**).

- A specific code page (**-C**).

Unicode native format and Unicode character format convert character data to Unicode during the bulk copy, resulting in no loss of extended characters.

Using the **-C** (code page) switch, the **bcp** utility can create or read data files using the code page specified by the user. For example, to bulk copy the **authors2** table in the **pubs** database to the Authors.txt data file using code page 850, execute from the command prompt:

```
bcp pubs..authors2 out authors.txt -c -C850 -Sservername -Usa -Ppass
```

Alternatively, using the CODEPAGE clause, the BULK INSERT statement can read data files using the code page specified by the user. For example, to bulk copy the Authors.txt data file into the **authors2** table in the **pubs** database using code page 850, execute from a query tool such as SQL Query Analyzer:

```
BULK INSERT pubs..authors2 FROM 'c:\authors.txt'
WITH (
   CODEPAGE = 850
)
```

The following are valid values for the code page.

| Code page value | Description |
|---|---|
| ACP | Columns of **char**, **varchar**, or **text** data type are converted from the ANSI/Windows code page (ISO 1252) to the SQL Server code page when importing data to an instance of SQL Server, and vice versa when exporting data from an instance of SQL Server. |
| OEM (default) | Columns of **char**, **varchar**, or **text** data type are converted from the system OEM code page to the SQL Server code page when importing data to an instance of SQL Server, and vice versa when exporting data from an instance of SQL Server. |
| RAW | This is the fastest option because no conversion from one code page to another occurs. |
| <value> | Specific code page number (for example, 850). |

## Column-level Collations

In SQL Server 2000, you can specify column-level collations for bulk copy operations. These collations define how character and Unicode data is stored in the specified columns of the data file.

Users and applications specify only the collation in which the data is stored in the data file. The bulk copy components perform internally any required translations between the data file collation and the collations of the source or destination columns in the database.

On a bulk copy out operation, the column and default collation specifications define the code pages used to build all SQLCHAR data in the resulting bulk copy data file. On a bulk copy in operation, the column and default collation specifications define the code pages used to read SQLCHAR data from the source data file.

If the SORTED hint is specified on a bulk copy in operation, the collations defined for any character and Unicode columns referenced in the SORTED hint define the expected sequence of the data.

On a bulk copy in operation, you must ensure that the collation specifications you make match the collations present in the bulk copy data file.

Format files in SQL Server 2000 support an eighth column in which you can provide a collation specification that defines how the data for that column is stored in the data file:

- "RAW" specifies the data is stored in the collation specified in the **–C** switch, BCPFILECP hint, or CODEPAGE option. If none of these is specified, the collation of the data file is that of the OEM code page of the bulk copy client computer.

- "*name*" specifies the name of the collation used to store the data in the data file.

- "" has the same meaning as RAW.

This is an example of a format file with column collations specified:

```
8.0
5
1  SQLCHAR  0   4  "/t" pub_id   1   "SQL_LATIN1_General_Cp1_CI
2  SQLCHAR  0  40  "/t" pub_name 2   "SQL_LATIN1_General_Cp85
3  SQLCHAR  0  20  "/t" city     3   "RAW"
4  SQLCHAR  0   2  "/t" state    4   "RAW"
5  SQLCHAR  0  30  "/t" country  5   ""
```

Column collation specifications are ignored for columns that do not have SQLCHAR or SQLNCHAR specified as their host data type. Collations for SQLNCHAR columns are ignored on bulk copy out operations; they apply only to bulk copy in operations where the SQLNCHAR column is referenced in a

SORTED hint. Collations apply to SQLCHAR columns on both in and out operations.

On a bulk copy out operation, the collation specification controls only the code page used to store character data in the bulk copy data file. It applies to:

- All columns in a character mode data file.
- Any column in a native mode file where SQLCHAR is specified as the host file data type.
- SQLCHAR characters whose values are greater than 127 or less than 32. Collations are applied to characters whose values are between 32 and 127, but all code pages map the same characters to the values from 32 to 127, so applying different collations has no noticeable effect.

The rules for determining which collation is used on a bulk copy out are:

- If a column collation is specified in either a format file or by using **bcp_setcolfmt**, the character data is stored using the ANSI code page associated with the collation. This overrides all other methods of specifying a collation.
- If a column collation was not specified, but either the **bcp –C** switch or the **bcp_control** BCPFILECP hint was specified, all SQLCHAR data from columns having no column collation specification is stored using the code page specified in BCPFILECP or **–C**. Column collations are not specified for any columns when producing a character mode data file with no format file. This rules also applies when "" or "RAW" is specified for a column collation.
- If no collations are specified at all (no column collation specifications, no **–C** switch, and no BCPFILECP hint), SQLCHAR data is stored using the OEM code page of the bulk copy client computer.

On a bulk copy in operation, the collation specification controls:

- How bulk copy attempts to interpret the code page of SQLCHAR columns in the data file.

- How bulk copy applies the ORDER hint.

For a bulk copy in operation, code page interpretation applies only to columns stored as SQLCHAR in a data file. All columns in a character mode data file are stored as SQLCHAR in a data file. It also applies to any column for which SQLCHAR is specified in a format file or using **bcp_setcolfmt**:

- If a column collation is specified in a format file or using **bcp_setcolfmt**, the SQLCHAR data in a data file is interpreted using the ANSI code page associated with the specified column collation.

- If a column collation is not specified, but a default code page is specified using the BULK INSERT CODEPAGE option, the **bcp –C** switch, or the **bcp_control** BCPFILECP hint, the SQLCHAR data is interpreted using the code page specified in either CODEPAGE, **–C**, or BCPFILECP.

- If the user did not specify any collations (no column collation, no BULK INSERT CODEPAGE option, no **bcp –C** switch, no BCPFILECP hint), then data in SQLCHAR columns is interpreted using the OEM code page of the client computer.

A bulk copy in operation also uses collations to properly interpret the ORDER bulk copy hint. This applies to both SQLCHAR and SQLNCHAR columns. The data in the columns referenced by a SORTED hint must be in the sequence defined by the collation mapped to those columns.

## See Also

bcp Utility

BULK INSERT.

SetCodePage Method

Unicode Character Format

Unicode Native Format

# Administering SQL Server

## Bulk Copy Performance Considerations

To bulk copy data as fast as possible, it is important to understand how data is copied, and what options are available to specify how data should be copied.

| Topic | Description |
|---|---|
| The Query Processor | Describes how **bcp** and BULK INSERT work in conjunction with the query processor. |
| Logged and Nonlogged Bulk Copies | Describes when bulk copy operations are logged and how to perform nonlogged bulk copy operations. |
| Parallel Data Loads | Describes bulk copying data in parallel from multiple clients to a single table. |
| Batch Switches | Describes the switches used to control the size of batches used in bulk copy operations. |
| Constraint Checking | Describes how to specify if constraints are checked during bulk copy operations. |
| Ordered Data Files | Describes how to specify the ordering of data in a data file. |
| Bypassing DEFAULT Definitions | Describes how to bypass default values specified in the destination table. |
| Controlling the Locking Behavior | Describes how to specify the locking behavior used during bulk copy operations. |

### See Also

Optimizing Bulk Copy Performance

## The Query Processor

The **bcp** utility works in conjunction with the query processor to insert data into an instance of Microsoft® SQL Server™. The **bcp** utility generates client OLE DB rowsets that are sent to SQL Server and are inserted into the table by the query processor. This has the advantage of allowing the query processor to plan and optimize queries that import and export data from an instance of SQL Server. It also allows optimized index maintenance, constraint checking, and parallel data load operations. The BULK INSERT statement works in conjunction with the query processor to bulk copy data into an instance of SQL Server.

Any program written using the bulk copy API takes advantage of using client OLE DB rowsets and the SQL Server query processor to insert data.

### See Also

Bulk-Copy Rowsets

Constraint Checking

Parallel Data Loads

Query Processor Architecture

## Logged and Minimally Logged Bulk Copy Operations

When using the full recovery model, all row-insert operations performed by **bcp** are logged in the transaction log. For large data loads, this can cause the transaction log to fill rapidly. To help prevent the transaction log from running out of space, a minimally logged bulk copy can be performed if all of these conditions are met:

- The recovery model is simple or bulk-logged.
- The target table is not being replicated.
- The target table does not have any triggers.
- The target table has either 0 rows or no indexes.
- The **TABLOCK** hint is specified. For more information, see Controlling the Locking Behavior.

Any bulk copy into an instance of Microsoft® SQL Server™ that does not meet these conditions is logged.

Before doing bulk copy operations, it is recommended that you set the recovery model to bulk-logged if you usually use full recovery. This will prevent the bulk copy operations from using excessive log space and possibly filling the log. However, even with bulk-logged recovery, some transaction log space will be used. You may want to create transaction log backups during the bulk copy operation to free up transaction log space.

When bulk copying a large number of rows into a table with indexes, it can be faster to drop all the indexes, perform the bulk copy, and re-create the indexes. For more information, see Optimizing Bulk Copy Performance.

**Note** Although data insertions are not logged in the transaction log when a minimally logged bulk copy is performed, SQL Server still logs extent allocations each time a new extent is allocated to the table.

## See Also

BACKUP

sp_dboption

SuspendIndexing Property

UseBulkCopyOption Property

## Parallel Data Loads

Microsoft® SQL Server™ allows data to be bulk copied into a single table from multiple clients in parallel using the **bcp** utility or BULK INSERT statement. This can improve the performance of data load operations. To bulk copy data into an instance of SQL Server in parallel:

- Set the database to Bulk-Logged Recovery if you usually use the Full Recovery model.
- Specify the **TABLOCK** hint. For more information, see Controlling the Locking Behavior.
- Ensure the table does not have any indexes.

**Note** Any application based on the DB-Library client library supplied with SQL Server version 6.5 or earlier, including the **bcp** utility, is not able to participate in parallel data loads into an instance of SQL Server. Only applications using the ODBC or SQL OLE DB-based APIs can perform parallel data loads into a single table.

After data has been bulk copied into a single table from multiple clients, any nonclustered indexes that need to be created can also be created in parallel by simply creating each nonclustered index from a different client concurrently.

**Note** Any clustered index on the table should be created first from a single client before creating the nonclustered indexes.

### See Also

bcp Utility

Logged and Nonlogged Bulk Copy Operations

Optimizing Bulk Copy Performance

## Batch Switches

The **bcp** utility and BULK INSERT statement accept two switches that allow the user to specify the number of rows per batch sent to Microsoft® SQL Server™ for the bulk copy operation.

| Bcp utility switch | BULK INSERT clause |
|---|---|
| **-b** batch_size | BATCHSIZE = *batch_size* |
| **-h "ROWS_PER_BATCH =** ***bb*****"** | ROWS_PER_BATCH = *rows_per_batch* |

The use of these switches has a large effect on how data insertions are logged.

### Using the -b Switch or BATCHSIZE Clause

Each batch of rows is inserted as a separate transaction. If, for any reason, the bulk copy operation terminates before completion, only the current transaction is rolled back. For example, if a data file has 1,000 rows, and a batch size of 100 is used, SQL Server logs the operation as 10 separate transactions; each transaction inserts 100 rows into the destination table. If the bulk copy operation terminates while copying row 750, only the previous 49 rows are removed as SQL Server rolls back the current transaction. The destination table still contains the first 700 rows.

### Using ROWS_PER_BATCH

If the **-b** switch or BATCHSIZE clause is not used, the entire file is sent to SQL Server and the bulk copy operation is treated as a single transaction. In this case, the **ROWS_PER_BATCH** hint or ROWS_PER_BATCH clause can be used to give an estimate of the number of rows. SQL Server optimizes the load automatically, according to the batch size value, which may result in better performance.

**Note** Generally, the larger the batch size is, the better the performance of the bulk copy operation will be. Make the batch size as large as is practical, although accuracy in the hint is not critical.

If, for any reason, the operation terminates before completion, the entire transaction is rolled back, and no new rows are added to the destination table.

Although all rows from the data file are copied into an instance of SQL Server in one batch, **bcp** displays the message "1000 rows sent to SQL Server" after every 1000 rows. This message is for information only and occurs regardless of the batch size.

**Note** Supplying both switches with different batch sizes will generate an error message.

When bulk copying large data files into an instance of SQL Server, it is possible for the transaction log to fill before the bulk copy is complete, even if the row inserts are not logged, from the extent allocation logging. In this situation, enlarge the transaction log, allow it to grow automatically or perform the bulk copy using the **-b** or BATCHSIZE switch, and set the recovery model to simple. Because only committed transactions can be truncated, this option does not free up space during the bulk copy operation if the **-b** switch is not used; the entire operation is logged as a single transaction.

The **bcp** utility and BULK INSERT statement also accept the **KILOBYTES_PER_BATCH** hint or KILOBYTES_PER_BATCH clause, respectively, which can be used to specify the approximate amount of data (in kilobytes) contained in a batch. SQL Server optimizes the bulk load according to the value set.

Batch sizes are not applicable when bulk copying data from an instance of SQL Server to a data file.

## See Also

BACKUP

bcp Utility

BULK INSERT.

ImportRowsPerBatch Property

Optimizing Bulk Copy Performance

sp_dboption

## Constraint Checking

The **bcp** utility and BULK INSERT statement accept the **CHECK_CONSTRAINTS** hint and CHECK_CONSTRAINT clause, respectively, which allows the user to specify whether constraints are checked during a bulk load.

By default, constraints are ignored during the bulk load. This improves the performance of the bulk load but allows the possibility of data being inserted into the table that violates existing constraints. **CHECK_CONSTRAINTS** specifies that constraints are enforced during the bulk load. This reduces the performance of the bulk load but ensures that all data inserted does not violate any existing constraints. For example, to bulk copy data from the Authors.txt data file to the **authors2** table in the **pubs** database, specifying that any constraints should be enforced, execute from the command prompt:

```
bcp pubs..authors2 in authors.txt -c -t, -Sservername -Usa -Ppassword
```

Alternatively, you can use the BULK INSERT statement from a query tool, such as SQL Query Analyzer, to bulk copy data:

```
BULK INSERT pubs..authors2 FROM 'c:\authors.txt'
WITH (
  DATAFILETYPE = 'char',
  FIELDTERMINATOR = ',',
  CHECK_CONSTRAINTS
)
```

When data is copied into a table, any triggers defined for the table are ignored.

To find any rows that violate constraints or triggers, you must check the copied data manually using queries. Bulk copy data into the table and run queries or stored procedures that test the constraint or trigger conditions, such as:

```
UPDATE pubs..authors2 SET au_fname = au_fname
```

Although this query does not change data to a different value, it causes Microsoft® SQL Server™ to update each value in the **au_fname** column to itself. This causes any constraints or triggers to be tested.

**Note** Although, by default, constraints on the table are not checked for the bulk copy operation unless **CHECK_CONSTRAINTS** is specified, constraints act as expected for other concurrent operations, such as INSERT, UPDATE, or DELETE.

## See Also

bcp Utility

BULK INSERT.

DBCC CHECKCONSTRAINTS

## Ordered Data Files

The **bcp** utility and BULK INSERT statement accept the **ORDER** hint and ORDER clause, respectively, which allows the user to specify how data in the data file is sorted. Although it is not necessary for data in the data file to be sorted in the same order as the table, the same ordering can improve performance of the bulk copy operation.

The order of data in the table is determined by the clustered index. The order and columns listed in the **ORDER** hint or ORDER clause must match the columns and be in the same order in the clustered index to improve the performance of the bulk copy operation.

For example, to bulk copy data from the Authors.txt data file to the **authors2** table in the **pubs** database, specifying that the data file is in ascending order on the **au_id** column, execute from the command prompt:

```
bcp pubs..authors2 in authors.txt -c -t, -Sservername -Usa -Ppassword
```

Alternatively, you can use the BULK INSERT statement from a query tool, such as SQL Query Analyzer, to bulk copy data:

```
BULK INSERT pubs..authors2 FROM 'c:\authors.txt'
WITH (
   DATAFILETYPE = 'char',
   FIELDTERMINATOR = ',',
   ORDER (au_id ASC)
)
```

By default, the bulk copy operation assumes that the data file is unordered.

### See Also

bcp Utility

BULK INSERT

Optimizing Bulk Copy Performance

## Bypassing DEFAULT Definitions

The **bcp** utility and the BULK INSERT statement accept the **-k** switch and the KEEPNULLS clause, respectively, which can be used to specify that empty columns should retain a null value during the bulk copy operation, rather than have any default values for the columns inserted.

**Note** If default values are not inserted, the column must be defined to allow null values.

By default, when data is copied into a table using the **bcp** utility or BULK INSERT statement, any defaults defined for the columns in the table are observed. For example, if there is a null field in a data file, the default value for the column is loaded instead.

For example, the data file Publishers.txt has two rows:

```
0111,New Moon Books,Boston,MA,
0222,Binnet & Hardley,Washington,DC,USA
```

Commas separate the fields; a newline character separates the rows. There is no country for the first row. If the **country** column of the **publishers** table had a default of "USA", the rows bulk loaded into the table by **bcp** or the BULK INSERT statement when the **-k** switch or KEEPNULLS clause is not specified are:

```
0111   New Moon Books          Boston          MA     USA
0222   Binnet & Hardley        Washington      DC     USA
```

Alternatively, to bulk copy data from the Publishers.txt data file into the **publishers** table in the **pubs** database and insert the value null into the **country** column, rather than the default value of "USA", execute from the command prompt:

```
bcp pubs..publishers in publishers.txt -c -t, -Sservername -Usa -Ppassw
```

Alternatively, you can use the BULK INSERT statement from a query tool, such as SQL Query Analyzer, to bulk copy data:

```
BULK INSERT pubs..publishers FROM 'c:\publishers.txt'
WITH (
   DATAFILETYPE = 'char',
   FIELDTERMINATOR = ',',
   KEEPNULLS
)
```

**Note** Although DEFAULT definitions on the table are not checked for the bulk copy operation if **-k** or KEEPNULLS is specified, DEFAULT definitions are expected for other concurrent INSERT statements.

## See Also

BACKUP

bcp Utility

Creating and Modifying DEFAULT Definitions

ServerBCPKeepNulls Property

## Controlling the Locking Behavior

The **bcp** utility and BULK INSERT statement accept the **TABLOCK** hint, which allows the user to specify the locking behavior used. **TABLOCK** specifies that a bulk update table-level lock is taken for the duration of the bulk copy. Using **TABLOCK** can improve performance of the bulk copy operation due to reduced lock contention on the table. For example, to bulk copy data from the Authors.txt data file to the **authors2** table in the **pubs** database, specifying a table-level lock, execute from the command prompt:

```
bcp pubs..authors2 in authors.txt -c -t, -Sservername -Usa -Ppassword
```

Alternatively, you can use the BULK INSERT statement from a query tool, such as SQL Query Analyzer, to bulk copy data:

```
BULK INSERT pubs..authors2 FROM 'c:\authors.txt'
WITH (
   DATAFILETYPE = 'char',
   FIELDTERMINATOR = ',',
   TABLOCK
)
```

If **TABLOCK** is not specified, the default uses row-level locks, unless the **table lock on bulk load** option is set to **on**. Setting the **table lock on bulk load** option using **sp_tableoption** sets the locking behavior for a table during a bulk load.

| Table lock on bulk load | Table locking behavior |
|---|---|
| **Off** | Row-level locks used |
| **On** | Table-level lock used |

If the **TABLOCK** hint is specified, the default setting for the table set with **sp_tableoption** is overridden for the duration of the bulk load.

**Note** It is not necessary to use the **TABLOCK** hint to bulk load data into a table from multiple clients in parallel, but doing so can improve performance.

## See Also

bcp Utility

BULK INSERT.

sp_tableoption

Understanding Locking in SQL Server

# Administering SQL Server

# Backing Up and Restoring Databases

The backup and restore component of Microsoft® SQL Server™ 2000 provides an important safeguard for protecting critical data stored in SQL Server databases.

With proper planning, you can recover from many failures, including:

- Media failure.
- User errors.
- Permanent loss of a server.

Additionally, backing up and restoring databases is useful for other purposes, such as copying a database from one server to another. By backing up a database from one computer and restoring the database to another, a copy of a database can be made quickly and easily.

This section provides the information necessary to implement a complete backup and recovery plan.

| Topic | Description |
|---|---|
| Designing a Backup and Restore Strategy | Helps you analyze and refine your data availability requirements and choose a recovery model for each database. |
| Using Recovery Models | Describes each recovery model in detail, as well as appropriate backup and restore strategies. This topic also describes how to switch between recovery models. |
| Backup and Restore Operations | Describes the various types of backups available and how they are used. This topic also describes point-in-time recovery, restarting a failed backup or restore, recovering to a particular |

| | |
|---|---|
| | transaction, and recovering part of a database. |
| Managing Backups | Describes backup devices, the backup format, and removable media terminology. This section also describes password security and media management including formatting, appending, overwriting, listing, and verifying media contents. |
| Backing Up and Restoring the System Databases | Describes the procedures necessary to protect and recover the system databases. |
| Handling Large Mission-Critical Environments | Describes features and techniques appropriate for highly available or very large production databases. These include using multiple backup devices, file and filegroup backups, file differential backups, and snapshot backups. |
| Copying Databases to Other Servers | Describes the use of backup and restore to quickly transport a database to another server. |

## See Also

Backup/Restore Architecture

Copying Databases to Other Servers

Databases

Transactions

# Administering SQL Server

## Designing a Backup and Restore Strategy

You must identify the requirements for the availability of your data in order to choose the appropriate backup and restore strategy. Your overall backup strategy defines the type and frequency of backups and the nature and speed of the hardware required for them.

It is strongly recommended that you test your backup and recovery procedures thoroughly. Testing helps to ensure that you have the required backups to recover from various failures, and that your procedures can be executed smoothly and quickly when a real failure occurs.

This section includes the following topics.

| Topic | Description |
|---|---|
| Analyzing Availability and Recovery Requirements | Explains the basic requirements for developing a backup and restore plan. |
| Planning for Disaster Recovery | Explains how to plan for a disaster (for example, the complete loss of a server). |
| Selecting a Recovery Model | Introduces Microsoft® SQL Server™ 2000 recovery models, which you implement after analyzing your availability requirements. |

# Administering SQL Server

## Analyzing Availability and Recovery Requirements

In order to develop a successful backup and restore plan, you must understand when your data needs to be accessible and the potential impact of data loss on your business. Answering the following questions can help you determine your availability requirements and sensitivity to data loss. Then you can choose the correct Microsoft® SQL Server™ 2000 recovery models for your databases and make the necessary technical and financial tradeoffs.

Here are some basic questions to help you analyze your availability and recovery requirements:

- What are your availability requirements? What portion of each day must the database be online?
- What is the financial cost of downtime to your business?
- If you experience media failure, such as a failing disk drive, what is the acceptable downtime?
- In case of a disaster, such as the loss of a server in a fire, what is the acceptable downtime?
- How important is it to never lose a change?
- How easy would it be to re-create lost data?
- Does your organization employ system or database administrators?
- Who will be responsible for performing backup and recovery operations, and how will they be trained?

Here are some questions to help you choose the tools, techniques, and hardware appropriate for your site:

- How large is each database?
- How often does the data in each database change?
- Are some tables modified more often than others?
- What are your critical database production periods?
- When does the database experience heavy use, resulting in frequent inserts and updates?
- Is transaction log space consumption likely to be a problem due to heavy update activity?
- Is your database subject to periodic bulk data loading?
- Is your database subject to risky updates or application errors that may not be detected immediately?
- Is your database server part of a SQL Server 2000 failover cluster for high availability?
- Is your database in a multi-server environment with centralized administration?

## Managing Media

When you back up and restore a database, you need to back up the data onto media (for example, tapes and disks). It is recommended that your backup plan include provisions for managing media, such as:

- A tracking and management plan for storing and recycling backup sets.
- A schedule for overwriting backup media.
- In a multi-server environment, a decision to use either centralized or distributed backups.
- A means of tracking the useful life of media.
- A procedure to minimize the effects of the loss of a backup set or backup media (for example, a tape).
- A decision to store backup sets on or offsite, and an analysis of how this will affect recovery time.

## Administering SQL Server

## Planning for Disaster Recovery

You need to create a disaster recovery plan in order to ensure that all your systems and data can be quickly restored to normal operation in the event of a natural disaster (for example, a fire) or a technical disaster (for example, a two-disk failure in a RAID-5 array). When you create a disaster recovery plan, you prepare all the actions that must occur in response to a catastrophic event. It is recommended that you verify your disaster recovery plan through the simulation of a catastrophic event.

Consider disaster recovery planning in light of your own environment and business needs. For example, suppose a fire occurs and wipes out your 24-hour data center. Are you certain you can recover? How long will it take you to recover and have your system available? How much data loss can your users tolerate?

Ideally, your disaster recovery plan states how long recovery will take and the final database state the users can expect. For example, you might determine that after the acquisition of specified hardware, recovery will be completed in 48 hours, and data will be guaranteed only up to the end of the previous week.

A disaster recovery plan can be structured in many different ways and can contain many types of information, including:

- A plan to acquire hardware.

- A communication plan.

- A list of people to be contacted in the event of a disaster.

- Instructions for contacting the people involved in the response to the disaster.

- Information on who owns the administration of the plan.

## Running a Base Functionality Script

Usually, you include a base functionality script as part of your disaster recovery plan in order to confirm that everything is working as intended. The base functionality script provides a dependable tool for the system administrator or database administrator to be able to see that the database is back in a viable state, without depending on end users for verification. Most commonly, this is an .sql file with batched SQL statements run into the server from **osql**. For other applications, a .bat file is more appropriate because it can contain **bcp** and **osql** commands. This base functionality script is very application specific, and it can take many different forms. For example, on a decision support/reporting system, the script may merely be a copy of several of your key reporting queries. For an online transaction processing (OLTP) application, the script may execute a batch of stored procedures that execute INSERT, UPDATE, and DELETE statements.

## Preparing for a Disaster

To prepare for disaster, it is recommended that you periodically perform the following steps:

- Perform regular database and transaction log backups to minimize the amount of lost data. It is recommended that both system and user databases be backed up.

- Maintain system logs in a secure fashion. Keep records of all service packs installed on Microsoft® Windows NT® 4.0 or Windows® 2000 and Microsoft SQL Server™. Keep records of network libraries used, the security mode, and the **sa** password.

- Maintain a base functionality script for quickly assessing minimal capability.

- Assess the steps you need to take to recover from a disaster ahead of time on another server, and amend the steps as necessary to suit your environment.

## Recovering from a Disaster

To recover from a disaster, perform the following steps after acquiring suitable replacement hardware:

1. Install Windows NT 4.0 or Windows 2000, and apply the appropriate service pack. Verify that appropriate domain functionality exists.

2. Install SQL Server, and apply the appropriate service pack. Restore the **master** and **msdb** database backups. Restart the server after restoring the **master** database.

3. Reconfigure the server for the appropriate network libraries and security mode.

4. Confirm that SQL Server is running properly by checking SQL Server Service Manager and the Windows application log. If the Windows NT 4.0 or Windows 2000 name was changed, use **sp_dropserver** and **sp_addserver** to match it with the SQL Server computer name.

5. Restore and recover each database according to its recovery plan.

6. Verify the availability of the system. Run a base functionality script to ensure correct operation.

7. Allow users to resume normal usage.

## See Also

Managing Permissions

sqlservr Application

# Administering SQL Server

## Selecting a Recovery Model

Microsoft® SQL Server™ provides three recovery models to:

- Simplify recovery planning.
- Simplify backup and recovery procedures.
- Clarify tradeoffs between system operational requirements.

These models each address different needs for performance, disk and tape space, and protection against data loss. For example, when you choose a recovery model, you must consider the tradeoffs between the following business requirements:

- Performance of large-scale operation (for example, index creation or bulk loads).
- Data loss exposure (for example, the loss of committed transactions).
- Transaction log space consumption.
- Simplicity of backup and recovery procedures.

Depending on what operations you are performing, more than one model may be appropriate. After you have chosen a recovery model or models, plan the required backup and recovery procedures.

This table provides an overview of the benefits and implications of the three recovery models.

| **Recovery model** | **Benefits** | **Work loss exposure** | **Recover to point in time?** |
|---|---|---|---|
| **Simple** | Permits high- | Changes since the | Can recover to the |

| | | | |
|---|---|---|---|
| | performance bulk copy operations.<br><br>Reclaims log space to keep space requirements small. | most recent database or differential backup must be redone. | end of any backup. Then changes must be redone. |
| **Full** | No work is lost due to a lost or damaged data file.<br><br>Can recover to an arbitrary point in time (for example, prior to application or user error). | Normally none.<br><br>If the log is damaged, changes since the most recent log backup must be redone. | Can recover to any point in time. |
| **Bulk-Logged** | Permits high-performance bulk copy operations.<br><br>Minimal log space is used by bulk operations. | If the log is damaged, or bulk operations occurred since the most recent log backup, changes since that last backup must be redone.<br><br>Otherwise, no work is lost. | Can recover to the end of any backup. Then changes must be redone. |

When a database is created, it has the same recovery model as the **model** database. To alter the default recovery model, use ALTER DATABASE to change the recovery model of the model database. You set the recovery model with the RECOVERY clause of the ALTER DATABASE statement. For more information, see ALTER DATABASE.

## Simple Recovery

Simple Recovery requires the least administration. In the Simple Recovery model, data is recoverable only to the most recent full database or differential backup. Transaction log backups are not used, and minimal transaction log space is used. After the log space is no longer needed for recovery from server failure, it is reused.

The Simple Recovery model is easier to manage than the Full or Bulk-Logged models, but at the expense of higher data loss exposure if a data file is damaged.

**Important** Simple Recovery is not an appropriate choice for production systems where loss of recent changes is unacceptable.

When using Simple Recovery, the backup interval should be long enough to keep the backup overhead from affecting production work, yet short enough to prevent the loss of significant amounts of data.

For more information, see Simple Recovery.

## Full and Bulk-Logged Recovery

Full Recovery and Bulk-Logged Recovery models provide the greatest protection for data. These models rely on the transaction log to provide full recoverability and to prevent work loss in the broadest range of failure scenarios.

The Full Recovery model provides the most flexibility for recovering databases to an earlier point in time. For more information, see Full Recovery.

The Bulk-Logged model provides higher performance and lower log space consumption for certain large-scale operations (for example, create index or bulk copy). It does this at the expense of some flexibility of point-in-time recovery. For more information, see Bulk-Logged Recovery.

Because many databases undergo periods of bulk loading or index creation, you may want to switch between Bulk-Logged and Full Recovery models. For more information, see Switching Recovery Models.

## See Also

ALTER DATABASE

# Administering SQL Server

## Using Recovery Models

You can select one of three recovery models for each database in Microsoft® SQL Server™ 2000 to determine how your data is backed up and what your exposure to data loss is. The following recovery models are available:

- Simple Recovery

  Simple Recovery allows the database to be recovered to the most recent backup.

- Full Recovery

  Full Recovery allows the database to be recovered to the point of failure.

- Bulk-Logged Recovery

  Bulk-Logged Recovery allows bulk-logged operations.

The recovery model of a new database is inherited from the **model** database when the new database is created.

**Note** The recovery model for a new database in SQL Server 2000 Personal Edition and SQL Server 2000 Desktop Engine defaults to Simple Recovery.

# Administering SQL Server

## Simple Recovery

With the Simple Recovery model, the database can be recovered to the point of the last backup. However, you cannot restore the database to the point of failure or to a specific point in time. To do that, choose either the Full Recovery or Bulk-Logged Recovery model.

The backup strategy for simple recovery consists of:

- Database backups.
- Differential backups (optional).

**Note** This model is similar to setting the **trunc. log on chkpt**. database option in Microsoft® SQL Server™ version 7.0 or earlier.

### To recover in the event of media failure

1. Restore the most recent full database backup.
2. If differential backups exist, restore the most recent one.

Changes since the last database or differential backup are lost.

### To create a database backup

## Administering SQL Server

## Full Recovery

The Full Recovery model uses database backups and transaction log backups to provide complete protection against media failure. If one or more data files is damaged, media recovery can restore all committed transactions. In-process transactions are rolled back.

Full Recovery provides the ability to recover the database to the point of failure or to a specific point in time. To guarantee this degree of recoverability, all operations, including bulk operations such as SELECT INTO, CREATE INDEX, and bulk loading data, are fully logged.

The backup strategy for full recovery consists of:

- Database backups.
- Differential backups (optional).
- Transaction log backups.
- Full and bulk-logged recovery are similar and many users of the Full Recovery model will use the Bulk-Logged model on occasion. For more information, see Bulk-Logged Recovery.

### Recovering in the Event of Media Failure

You can restore a database to the state it was in at the point of failure if the current transaction log file for the database is available and undamaged. To restore the database to the point of failure:

1. Back up the currently active transaction log. For more information, see Transaction Log Backups.
2. Restore the most recent database backup without recovering the database.

3. If differential backups exist, restore the most recent one.

4. Restore each transaction log backup created since the database or differential backup in the same sequence in which they were created without recovering the database.

5. Apply the most recent log backup (created in Step 1), and recover the database.

   **Important** To protect against loss of transactions under the Full Recovery model, the transaction log must be protected against damage. It is strongly recommended that fault-tolerant disk storage be used for the transaction log.

## To create a database backup

# Administering SQL Server

## Bulk-Logged Recovery

The Bulk-Logged Recovery model provides protection against media failure combined with the best performance and minimal log space usage for certain large-scale or bulk copy operations. These operations are minimally logged:

- SELECT INTO.
- Bulk load operations (**bcp** and BULK INSERT).
- CREATE INDEX (including indexed views).
- **text** and **image** operations (WRITETEXT and UPDATETEXT).

In a Bulk-Logged Recovery model, the data loss exposure for these bulk copy operations is greater than in the Full Recovery model. While the bulk copy operations are fully logged under the Full Recovery model, they are minimally logged and cannot be controlled on an operation-by-operation basis under the Bulk-Logged Recovery model. Under the Bulk-Logged Recovery model, a damaged data file can result in having to redo work manually.

In addition, the Bulk-Logged Recovery model only allows the database to be recovered to the end of a transaction log backup when the log backup contains bulk changes. Point-in-time recovery is not supported.

In Microsoft® SQL Server™ 2000, you can switch between full and bulk-logged recovery models easily. It is not necessary to perform a full database backup after bulk copy operations complete under the Bulk-Logged Recovery model. Transaction log backups under this model capture both the log and the results of any bulk operations performed since the last backup.

The backup strategy for bulk-logged recovery consists of:

- Database backups.
- Differential backups (optional).

- Log backups.

  Backing up a log that contains bulk-logged operations requires access to all data files in the database. If the data files are not accessible, the final transaction log cannot be backed up and all committed operations in that log will be lost.

### To recover in the event of media failure

1. Back up the currently active transaction log. For more information, see Transaction Log Backups.

2. Restore the most recent full database backup.

3. If differential backups exist, restore the most recent one.

4. Apply in sequence all transaction log backups created since the most recent differential or full database backup.

5. Manually redo all changes since the most recent log backup.

**IMPORTANT** If the active transaction log is lost (for example, due to hardware failure on the disk containing the transaction log files), all transactions in that log are lost. To prevent loss of the active transaction log, place the transaction log files on mirrored disks.

### To create a database backup

Administering SQL Server

## Switching Recovery Models

You can switch a database from one recovery model to another in order to meet changing business needs. For example, a mission-critical online transaction processing (OLTP) system requires full recoverability but periodically undergoes bulk load and indexing operations. The recovery model for the database can be changed to Bulk-Logged for the duration of the load and indexing operations and then returned to Full Recovery. This increases performance and reduces the required log space while maintaining server protection.

**Note** Switching recovery models during a bulk load operation is permitted. The logging of the bulk operation changes appropriately.

The following table indicates what action to take when switching from one recovery model to another.

| From | To | Action | Description |
|---|---|---|---|
| Full Recovery | Bulk-Logged Recovery | No action | Requires no change in backup strategy. Continue to perform periodic database, log, and (optionally) differential backups. |
| Full Recovery | Simple Recovery | Optionally back up the transaction log prior to the change | Executing a log backup immediately before the change permits recovery to that point. After switching to the simple model, stop executing log backups. |
| Bulk-Logged Recovery | Full Recovery | No action | Requires no change in backup strategy. Recovery to any point in time is enabled after the next log backup. If point-in-time recovery is important, execute a log backup immediately after switching. |
| | | | |

| | | | |
|---|---|---|---|
| Bulk-Logged Recovery | Simple Recovery | Optionally back up the transaction log prior to the change | Executing a log backup immediately before the change permits recovery to that point. After switching to the simple model, stop executing log backups. |
| Simple Recovery | Full Recovery | Back up the database after the change | Execute a database or differential backup after switching to the Full Recovery model. Begin executing periodic database, log, and (optionally) differential backups. |
| Simple Recovery | Bulk-Logged Recovery | Back up the database after the change | Execute a database or differential backup after switching to the bulk-logged model. Begin executing periodic database, log, and (optionally) differential backups. |

## Administering SQL Server

# Backup and Restore Operations

Microsoft® SQL Server™ supports various types of backups to be used separately or in combination. The recovery model you choose will determine your overall backup strategy, including the types of backups available to you. For more information, see Designing a Backup and Restore Strategy and Using Recovery Models.

The following table illustrates the types of backups that are available for each recovery model.

| **Model** | **Backup Type** | | | |
|---|---|---|---|---|
| | **Database** | **Database differential** | **Transaction log** | **File or file differential** |
| **Simple** | Required | Optional | Not allowed | Not allowed |
| **Full** | Required (or file backups) | Optional | Required | Optional |
| **Bulk-Logged** | Required (or file backups) | Optional | Required | Optional |

Backups are created on backup devices, such as disk or tape media. With SQL Server, you can decide how you want to create your backups on backup devices. For example, you can overwrite outdated backups, or you can append new backups to the backup media. For more information, see Managing Backups.

Performing a backup operation has minimal effect on running transactions, so backup operations can be run during normal operations.

**Note** Creating or deleting database files is not possible when the database or transaction log is being backed up. If you attempt to create or delete a database file while a backup operation is in progress, the create or delete will fail. If you attempt to start a backup operation while a database file is being created or deleted, the backup operation will wait until the create or delete is completed or the backup operation times out.

## See Also

Selecting a Recovery Model

Analyzing Availability and Recovery Requirements

Planning for Disaster Recovery

# Administering SQL Server

## Database Backups

A database backup creates a duplicate of the data that is in the database when the backup completes. This is a single operation, usually scheduled at regular intervals. Database backups are self-contained.

You can re-create the entire database from a database backup in one step by restoring the database. The restore process overwrites the existing database or creates the database if it does not exist. The restored database will match the state of the database at the time the backup completed, minus any uncommitted transactions. Uncommitted transactions are rolled back when the database is recovered.

A database backup uses more storage space per backup than transaction log and differential database backups. Consequently, database backups need more time to complete the backup operation and so are typically created less frequently than differential database or transaction log backups. For more information, see Transaction Log Backups and Differential Database Backups.

### Restoring a Database Backup

Restoring a database backup re-creates the database and all of its associated files that were in the database when the backup was completed. However, any modifications made to the database after the backup was created are lost. To restore transactions made after the database backup was created, you must use transaction log backups or differential backups.

When restoring a database, Microsoft® SQL Server™:

1. Copies all of the data from the backup into the database. The rest of the database is created as empty space.

2. Rolls back any incomplete transactions in the database backup to ensure that the database is consistent.

To prevent overwriting a database unintentionally, the restore operation performs safety checks automatically. The restore operation fails if:

- The database name in the restore operation does not match the database name recorded in the backup set.

- The database named in the restore operation already exists on the server but is not the same database contained in the database backup. For example, the database names are the same, but each database was created differently.

- One or more files need to be created automatically by the restore operation, but the file names already exist.

These safety checks can be disabled if the intention is to overwrite another database. For more information, see RESTORE.

**Note** If you restore a database on a different instance of SQL Server than the one on which the backup was created, you may need to run **sp_change_users_login** to update user login information. For more information, see sp_change_users_login.

## Backing Up Full-Text Indexes

Backing up a database does not back up full-text index data in full-text catalogs. However, if full-text indexes have been defined for tables, the meta data is backed up when a database backup is created. After a database backup is restored, the full-text index catalogs can be re-created and repopulated. For more information, see Full-text Indexes.

## Estimating the Size of Your Database Backup

Before you implement a backup and restore strategy, you need to estimate how much disk space your database backup will use. During a database backup, the backup operation copies only the data in the database to the backup file. Because the database backup contains only the actual data in the database and not any unused space, the database backup is likely to be smaller than the database itself. You can estimate the size of the database backup by using the **sp_spaceused** system stored procedure. For more information, see sp_spaceused.

### To create a database backup

# Administering SQL Server

## Differential Database Backups

A differential database backup records only the data that has changed since the last database backup. You can make more frequent backups because differential database backups are smaller and faster than database backups. Making frequent backups decreases your risk of losing data.

**Note** If you have created any file backups since the last full database backup, those files will be scanned by Microsoft® SQL Server™ 2000 at the beginning of a differential database backup. This may cause some degradation of performance in the differential database backup. For more information, see Using File Backups.

You use differential database backups to restore the database to the point at which the differential database backup was completed. To recover to the exact point of failure, you must use transaction log backups. For more information, see Transaction Log Backups.

Consider using differential database backups when:

- Only a relatively small portion of the data in the database has changed since the last database backup. Differential database backups are particularly effective if the same data is modified many times.
- You are using the Simple Recovery model and want more frequent backups, but don't want to do frequent full database backups.
- You are using the Full or Bulk-Logged Recovery model and want to minimize the time it takes to roll forward transaction log backups when restoring a database.

A recommended process for implementing differential database backups is:

1. Create regular database backups.
2. Create a differential database backup periodically between database backups, such as every four hours or more for highly active systems.

3. If using Full or Bulk-Logged Recovery, create transaction log backups more frequently than differential database backups, such as every 30 minutes.

The sequence for restoring differential database backups is:

1. Restore the most recent database backup.

2. Restore the last differential database backup.

3. Apply all transaction log backups created after the last differential database backup was created if you use Full or Bulk-Logged Recovery.

### To create a differential database backup

# Administering SQL Server

# Transaction Log Backups

The transaction log is a serial record of all the transactions that have been performed against the database since the transaction log was last backed up. With transaction log backups, you can recover the database to a specific point in time (for example, prior to entering unwanted data), or to the point of failure.

When restoring a transaction log backup, Microsoft® SQL Server™ rolls forward all changes recorded in the transaction log. When SQL Server reaches the end of the transaction log, it has re-created the exact state of the database at the time the backup operation started. If the database is recovered, SQL Server then rolls back all transactions that were incomplete when the backup operation started.

Transaction log backups generally use fewer resources than database backups. As a result, you can create them more frequently than database backups. Frequent backups decrease your risk of losing data.

**Note** Sometimes a transaction log backup is larger than a database backup. For example, a database has a high transaction rate causing the transaction log to grow quickly. In this situation, create transaction log backups more frequently.

Transaction log backups are used only with the Full and Bulk-Logged Recovery models. For more information, see Using Recovery Models.

## Using Transaction Log Backups with Database Backups

Restoring a database using both database and transaction log backups works only if you have an unbroken sequence of transaction log backups after the last database or differential database backup. If a log backup is missing or damaged, you must create a database or differential database backup and start backing up the transaction logs again. Retain the previous transaction logs backups if you want to restore the database to a point in time within those backups.

The only time database or differential database backups must be synchronized with transaction log backups is when starting a sequence of transaction log backups. Every sequence of transaction log backups must be started by a database or differential database backup.

Usually, the only time that a new sequence of backups is started is when the database is backed up for the first time or a change in recovery model from Simple to Full or Bulk-Logged has occurred. For more information, see Switching Recovery Models.

## Truncating the Transaction Log

When SQL Server finishes backing up the transaction log, it automatically truncates the inactive portion of the transaction log. This inactive portion contains completed transactions and so is no longer used during the recovery process. Conversely, the active portion of the transaction log contains transactions that are still running and have not yet completed. SQL Server reuses this truncated, inactive space in the transaction log instead of allowing the transaction log to continue to grow and use more space.

> Although the transaction log may be truncated manually, it is strongly recommended that you do not do this, as it breaks the log backup chain. Until a full database backup is created, the database is not protected from media failure. Use manual log truncation only in very special circumstances, and create a full database backup as soon as practical.

The ending point of the inactive portion of the transaction log, and hence the truncation point, is the earliest of the following events:

- The most recent checkpoint.

- The start of the oldest active transaction, which is a transaction that has not yet been committed or rolled back.

  This represents the earliest point to which SQL Server would have to roll back transactions during recovery.

- The start of the oldest transaction that involves objects published for replication whose changes have not been replicated yet.

  This represents the earliest point that SQL Server still has to replicate.

## Conditions for Backing Up the Transaction Log

The transaction log cannot be backed up during a full database backup or a differential database backup. However, the transaction log can be backed up while a file backup is running.

Do not back up the transaction log:

- Until a database or file backup has been created because the transaction log contains the changes made to the database after the last backup was created. For more information, see Using File Backups.

- If the transaction log has been explicitly truncated, unless a database or differential database backup is created after the transaction log truncation occurs.

## Restoring Transaction Log Backups

It is not possible to apply a transaction log backup:

- Unless the database or differential database backup preceding the transaction log backup is restored first.

- Unless all preceding transaction logs created since the database or differential database was backed up are applied first.

  If a previous transaction log backup is lost or damaged, you can restore only transaction logs up to the last backup before the missing transaction log.

- If the database has already recovered and all outstanding transactions have been either rolled back or rolled forward.

  When applying transaction log backups, the database must not be recovered until the final transaction log has been applied. If you allow recovery to take place when applying one of the intermediate transaction log backups, you cannot restore past that point without restarting the entire restore operation, starting with the database backup.

## Creating a Sequence of Transaction Log Backups

To create a set of backups, you typically make a database backup at periodic intervals, such as daily, and transaction log backups at shorter intervals, such as every 10 minutes. You must have at least one database backup, or a covering set of file backups, to make log backups useful. The interval between backups varies with the criticality of the data and the workload of the server. If your transaction log is damaged, you will lose work performed since the most recent log backup. This suggests frequent log backups for critical data, and highlights the importance of placing the log files on fault tolerant storage.

The sequence of transaction log backups is independent of the database backups. You make one sequence of transaction log backups, and then make periodic database backups that are used to start a restore operation. For example, assume the following sequence of events.

| Time | Event |
|---|---|
| 8:00 A.M. | Back up database |
| Noon | Back up transaction log |
| 4:00 P.M. | Back up transaction log |
| 6:00 P.M. | Back up database |
| 8:00 P.M. | Back up transaction log |
| 10:00 P.M. | Failure occurs |

The transaction log backup created at 8:00 P.M. contains transaction log records from 4:00 P.M. through 8:00 P.M., spanning the time when the database backup was created at 6:00 P.M. The sequence of transaction log backups is continuous from the initial database backup created at 8:00 A.M. to the last transaction log backup created at 8:00 P.M. The following procedures can be used to restore the database to its state at 10:00 P.M. (point of failure).

**Restore the database using the last database backup created.**

1. Create a backup of the currently active transaction log.

2. Restore the 6:00 P.M. database backup, and then apply the 8:00 P.M. and active transaction log backups.

   The restore process detects that the 8:00 P.M. transaction log backup contains transactions that have occurred prior to the last restored backup. Therefore, the restore operation scans down the transaction log to the point corresponding to when the 6:00 P.M. database backup completed and rolls forward only the completed transactions from that point on within the transaction log backup. This occurs again for the 10:00 P.M. transaction log backup.

**Restore the database using an earlier database backup (earlier than the most recent database backup created).**

1. Create a backup of the currently active transaction log.

2. Restore the 8:00 A.M. database backup, and then restore all four transaction log backups in sequence. Do not restore the 6:00 P.M. database backup. This rolls forward all completed transactions up to 10:00 P.M.

   This process will take longer than restoring the 6:00 P.M. database backup.

The second option points out the redundant security offered by a chain of transaction log backups that can be used to restore a database even if a database backup is lost. You can restore an earlier database backup, and then restore all of the transaction log backups created after the database backup was created.

**Note** It is important not to lose a transaction log backup. Consider making multiple copies of log backup sets. This can be accomplished by backing the log up to disk, then copying the disk file to another device, such as a separate disk or tape.

## Recovery and Transaction Logs

When you finish the restore operation and recover the database, all incomplete transactions are rolled back. This is required to restore the integrity of the database.

After this has been done, no more transaction log backups can be applied to the database. For example, a series of transaction log backups contain a long-running transaction. The start of the transaction is recorded in the first transaction log backup, but the end of the transaction is recorded in the second transaction log backup. There is no record of a commit or rollback operation in the first transaction log backup. Therefore, if a recovery operation runs when the first transaction log backup is applied, the long-running transaction is treated as incomplete. Data modifications recorded in the first transaction log backup for the transaction are rolled back. SQL Server does not allow the second transaction log backup to be applied after the recovery operation has run.

Therefore, when restoring transaction log backups, the database must not be recovered until the final transaction log has been applied. This prevents any transactions from being partially rolled back. The only time outstanding transactions need to be rolled back is at the end of the last restore operation.

# Administering SQL Server

## Backup Restrictions

In Microsoft® SQL Server™, backup operations can occur while the database is online and in use. However, some operations are not allowed during a database backup:

- Creating or deleting database files.

- The file truncation portion of a shrink operation on either the database (automatically or manually) or the database files. They will fail if a backup is running. You can perform the truncation after the backup completes. For more information, see Shrinking a Database.

If a backup is started when one of these operations is in progress, the backup waits for the operation to complete, up to the limit set by the session timeout. If a backup is in progress and one of these operations is attempted, the operation fails and the backup continues.

## Administering SQL Server

## Restoring a Database to a Prior State

At times, you may want to restore your database to an earlier point in time. For example, if an earlier transaction within a database changed some data incorrectly, you will need to restore the database to a point in time earlier than the incorrect data entry. To do this, recover the entire database to a point within a transaction log. You can recover a database to either a specific point in time within a transaction log or to a named mark that was previously inserted into the log.

### To create a transaction log backup

## Recovering to a Point In Time

You can recover to a point in time by recovering only the transactions that occurred before a specific point in time within a transaction log backup, rather than the entire backup. By viewing the header information of each transaction log backup or the information in the **backupset** table in **msdb**, you can quickly identify which backup contains the time to which you want to restore the database. You then need only apply transaction log backups up to that point.

You cannot skip specific transactions. This would compromise the integrity of the data in the database. Any transactions that occur after the transaction you want to undo might depend on the data modified by the undone transaction.

If you do not want to restore any modifications made to the database after a specific point in time:

- Restore the last database backup without recovering the database.
- Apply each transaction log backup in the same sequence in which they were created.
- Recover the database at the desired point in time within a transaction log backup.

This process also can be used to restore a database and transaction logs if some transaction log backups created after a point in time are missing or damaged.

### To restore to a point in time

# Recovering to a Named Transaction

Microsoft® SQL Server™ 2000 supports the insertion of named marks into the transaction log to allow recovery to that specific mark. Log marks are transactional and are inserted only if their associated transaction commits. As a result, marks can be tied to specific work, and you can recover to a point that includes or excludes this work.

Before inserting named marks into the transaction log, consider the following:

- Because transaction marks consume log space, use them only for transactions that play a significant role in the database recovery strategy.

- For each marked transaction that commits, a row is inserted in the **logmarkhistory** table in **msdb**.

- If a marked transaction spans multiple databases on the same database server or on different servers, the marks must be recorded in the logs of all the affected databases. For more information, see Backup and Recovery of Related Databases.

## Inserting Named Marks into a Transaction Log

To insert marks into the transaction logs, use the BEGIN TRANSACTION statement and the WITH MARK [*description*] clause. Because the name of the mark is the same as its transaction, a transaction name is required. The optional *description* is a textual description of the mark.

The transaction log records the mark name, description, database, user, datetime information, and the Log Sequence Number (LSN). To allow their reuse, the transaction names are not required to be unique. The datetime information is used along with the name to uniquely identify the mark.

## Recovering to a Mark

There are two ways to recover to a mark in the log:

- Use RESTORE LOG and the WITH STOPATMARK='*mark_name*' clause to roll forward to the mark and include the transaction that contains the mark.

- Use RESTORE LOG and the WITH STOPBEFOREMARK='*mark_name*' clause to roll forward to the mark and exclude the transaction that contains the mark.

The WITH STOPATMARK and WITH STOPBEFOREMARK clauses support an optional AFTER *datetime* clause. If AFTER *datetime* is omitted, recovery stops at the first mark with the specified name. If AFTER *datetime* is specified, recovery stops at the first mark with the specified name on or after *datetime*.

**Note** Recovering to a mark is subject to the same restrictions as point-in-time recovery. Specifically, recovering to a mark is disallowed during intervals in which the database is undergoing operations that are bulk-logged.

## Recovery Paths

A new recovery path is created if you recover a database to an earlier point in time and begin using the database from that point. This recovery path will contain new transactions that make it unique. If you need to restore the database again, it is not possible to combine the data from the two recovery paths. You must restore data along one path or the other.

**Note** Restoring a full database backup and recovering the database without using any other type of backup does not result in a new recovery path.

Examples of when a new recovery path is created include:

- Restoring a full database backup and a differential backup and recovering the database without applying existing transaction log backups.
- Recovering the database at the end of a differential backup other than the most recent differential backup.
- Recovering the database at the end of a transaction log backup other than the most recent transaction log backup.
- Recovering the database at a specific time or a marked transaction within a transaction log backup.

In the example above, a Full Database Backup and a sequence of four Log Backups are created. The database is then restored to the end of Log Backup 2 by restoring the Full Database Backup, Log Backup 1, and Log Backup 2. The database is recovered at this point, creating a new recovery path. The database is then used for a time, and two more transaction log backups, Log Backup 5 and Log Backup 6, are created. If you again restore the Full Database Backup and apply transaction log backups, you must follow one of the two recovery paths:

- Log Backup 1, Log Backup 2, Log Backup 3, and Log Backup 4

  -or-

- Log Backup 1, Log Backup 2, Log Backup 5, and Log Backup 6

The database can be recovered at any point in time along either path, but it is not possible to combine data from the two. For example, you cannot restore Log Backups 1 through 6 in sequence because Log Backups 3 and 4 contain data that is inconsistent with Log Backups 5 and 6.

# Administering SQL Server

## Partial Database Restore Operations

Application or user errors often affect an isolated portion of the database, such as a table. To support recovery from these events, Microsoft® SQL Server™ provides a mechanism to restore part of the database to another location so that the damaged or missing data can be copied back to the original database. For example, if an application erroneously dropped a table, you may want to restore only the part of the database that contained the table. Restoring log or differential backups can bring the table to a point prior to when the table was dropped. Then the content of the table can be extracted and reloaded into the original database.

Performing a partial restore operation is also useful when you are:

- Creating a subset of a database on another server for development or reporting purposes.

- Restoring archived data.

Partial restore operations work with database filegroups. The primary filegroup is always restored, along with the files that you specify and their corresponding filegroups. The result is a subset of the database. Filegroups that are not restored are marked as offline and are not accessible.

**Note** Because the primary file is restored, all catalogs (except full-text catalogs) are restored, even those associated with files that are not included in the restore operation.

Partial restore operations are accomplished with the PARTIAL clause of the RESTORE statement. You can also use the PARTIAL option when restoring a full database backup. Partial database restore of file backups is not supported.

### To perform a partial restore operation

1. Execute the RESTORE DATABASE statement using a full database backup, specifying:

   - The name of the database to restore. Specify a new name for the database, unless you are planning to overwrite the original database or are restoring the database on a different server.

- The backup device from which the database backup will be restored.

- The FILEGROUP clause for each file or filegroup to restore.

  **Note** If a file is specified, all of the files in its filegroup are also restored.

- The MOVE clause if you are restoring the files in a new location.

- The PARTIAL clause.

- The NORECOVERY clause, if there are transaction log or differential backups to be applied. Otherwise, specify RECOVERY.

2. Optionally, execute the RESTORE DATABASE statement to restore a differential database backup, specifying:

   - The name of the database to which the differential database backup will be applied.

   - The backup device where the differential database backup will be restored from.

   - The NORECOVERY clause, if you have transaction log backups to apply after the differential database backup is restored; otherwise specify the RECOVERY clause.

3. Execute the RESTORE LOG statement to apply each transaction log backup, specifying:

   - The name of the database to which the log is to be applied.

- The backup device from which the log backup will be restored.

- The NORECOVERY clause, if there are other log backups to be applied. Otherwise, specify RECOVERY.

## Examples

This example performs a partial restore operation in a database, named **mywind**. **mywind** is using the Full Recovery model. The database is created on two filegroups, **new_customers**, which contains the file **mywind_data_1**, and **sales**, which contains the file **mywind_data_2**:

```
CREATE DATABASE mywind
GO

ALTER DATABASE mywind ADD FILEGROUP new_customers
ALTER DATABASE mywind ADD FILEGROUP sales
GO

ALTER DATABASE mywind ADD FILE
   (NAME='mywind_data_1',
   FILENAME='g:\mw.dat1')
   TO FILEGROUP new_customers
ALTER DATABASE mywind
   ADD FILE
   (NAME='mywind_data_2',
   FILENAME='g:\mw.dat2')
   TO FILEGROUP sales
GO
```

A full database backup is performed. Then the **t1** table is created on **new_customers** and the **t2** table is created on **sales**. The transaction log is backed up:

```
BACKUP DATABASE mywind
   TO DISK ='g:\mywind.dmp'
   WITH INIT
GO

USE mywind
GO

CREATE TABLE t1 (id int) ON new_customers
CREATE TABLE t2 (id int) ON sales
GO

BACKUP LOG mywind TO DISK='g:\mywind.dmp'
WITH NOINIT
GO
```

At some point, it becomes necessary to restore the **t2** table on the **sales** filegroup. RESTORE FILELISTONLY lists the database files and the filegroups in which they reside. RESTORE HEADERONLY lists the contents of the backup medium:

```
RESTORE FILELISTONLY FROM DISK='g:\mywind.dmp'
GO
RESTORE HEADERONLY FROM DISK='g:\mywind.dmp'
GO
```

The RESTORE DATABASE statement restores the database under a different name and the **sales** filegroup using the WITH PARTIAL and NORECOVERY options. In addition, the primary file and filegroup (**mywind**), the log (**mywind_log**), and all files in the restored filegroup (in this example, **mywind_data_2** is the only file in **sales**) are moved to a new location. The log is then recovered:

```
RESTORE DATABASE mywind_part
```

```
   FILEGROUP = 'sales'
   FROM DISK='g:\mywind.dmp'
   WITH FILE=1,NORECOVERY,PARTIAL,
   MOVE 'mywind' TO 'g:\mw2.pri',
   MOVE 'mywind_log' TO 'g:\mw2.log',
   MOVE 'mywind_data_2' TO 'g:\mw2.dat2'
GO

RESTORE LOG mywind_part
   FROM DISK = 'g:\mywind.dmp'
   WITH FILE = 2,RECOVERY
GO
```

Notice that **t2** is accessible after the partial restore operation.

```
SELECT COUNT(*) FROM mywind_part..t2
```

Here is the result:

```
---------------
0
```

Notice that **t1** is not accessible after the partial restore operation.

```
SELECT COUNT(*) FROM mywind_part..t1
```

Here is the resulting message:

```
The query processor is unable to produce a plan because
the table 'mywind_part..t1' is marked OFFLINE.
```

### See Also

Recovering to a Point In Time

RESTORE

## Administering SQL Server

## Recovering a Database Without Restoring

Usually, you recover the database when you restore the last backup. It is also possible to recover the database without restoring a backup. This is necessary if:

- You did not recover the database as part of the last restore, but you now want to use the database.

- Your database is in standby mode, and you want to make it updatable without applying another log backup.

### To recover a database without restoring

# Administering SQL Server

## Restarting Interrupted Backup and Restore Operations

If a backup or restore operation is interrupted (for example, if the power fails), you can restart the backup or restore operation from the point at which it was interrupted. This can be useful if you restore large databases onto other servers as an automated process. If the automated process fails near the end of the restore operation, you can attempt to restart the restore operation from where it left off, rather than restoring the whole database from the beginning.

### To restart an interrupted backup operation

## Backup and Recovery of Related Databases

If you have two or more databases that must be logically consistent, you may need to implement special procedures to ensure the recoverability of these databases.

It is important to consider the recovery goals for the entire set of databases. In the worst case you need to consider how long it will take to recover all of the databases. To avoid excessive recovery with a large number of databases, you need to avoid sharing media at backup time, and you need sufficient hardware to restore the databases in parallel.

Three potential related database scenarios are:

- You experience media failure that affects one or more of the databases, but the transaction log(s) are not damaged. You want to recover to current time.

- One or more transaction logs are destroyed. You need to restore the set of databases to a consistent state at the time of your last log backup.

- You need to restore the entire set of databases to a mutually consistent state at some earlier point in time.

In all three of these cases, you must be using the Full Recovery model for these databases. For more information, see Full Recovery.

The first scenario does not require you to implement any special recovery procedures. To recover the damaged databases, back up the tail of the log, restore the damaged files or the database, and then roll forward using transaction log backups. The undamaged databases require no action.

The other two scenarios require you to use a special procedure to ensure recoverability: marking transactions in the databases.

### Marked Transaction Basics

You can mark transactions across related databases and use these marked transactions to recover related databases to the same transaction-consistent point in time. Accomplish this by placing distributed marks across all databases before backing up the log in any database. This will ensure that all log backups have a mark that will appear in all databases. Synchronized backups are not necessary. Instead, placing marks in the transaction log allows synchronization during restore. Use the Full Recovery model to ensure that all the marks will be valid.

**IMPORTANT** Related database recovery does not allow recovery to a specific point in time. Recovery of related databases can only be accomplished by recovering to a marked transaction.

An example of related database recovery is a bank that has a database containing checking account data and another database containing savings account data. The two databases are located on different servers, and there are transactions that transfer funds back and forth between checking accounts and savings accounts. When the databases are backed up while fund transfer transactions are active in the system, even if the databases are backed up at the same time, there is a good chance that some transfer transactions will have committed in one database but not the other. Marked transactions can be used to backup and later restore these databases to a point where the outcome of all transactions is the same in both of the restored databases.

For this example, the backup strategy would be:

1. Set the recovery model to Full for both databases.

2. Back up each database.

   Databases can be backed up in series or in parallel.

3. Prior to backing up the transaction log, run a marked transaction that spans each database.

4. Back up the transaction log on each database.

To restore the backup:

1. Restore each database backup.

2. Restore each log backup, stopping at the marked transaction.

3. Recover each database.

In the event of a media failure, if you want to recover all the databases to a marked transaction, you must determine the most recent marked transaction that is available in all of the transaction logs. This information is stored in the **logmarkhistory** table, which is in the **msdb** database, on all of the servers.

When you have determined the marked transaction to which you want to restore:

1. Identify the log backups for all related databases containing this mark.

2. Create transaction log backups on the undamaged databases as required.

3. Resolve hardware problems.

4. Restore and recover all related databases to the target mark.

## Creating Marked Transactions

The statement BEGIN TRAN *new_name* WITH MARK can be nested within an already existing transaction. Upon doing so, *new_name* becomes the mark name for the transaction, despite the name that the transaction may already have been given. Issuing a second, nested BEGIN TRAN...WITH MARK will result in a warning (not error) message:

```
Server: Msg 3920, Level 16, State 1, Line 2
WITH MARK option only applies to the first BEGIN TRAN WITH M
The option is ignored.
```

The transaction mark is only placed in the logs of databases that are updated by the marked transaction. In addition, the only databases that will contain the mark are those on the server where the BEGIN TRAN...WITH MARK statement was executed. The following example shows how to put a mark in multiple databases:

```
BEGIN TRAN T1
UPDATE db1.dbo.table1 set column1 = 2
BEGIN TRAN M2 WITH MARK
UPDATE db2.dbo.table1 set column1 = 2
UPDATE server2.db21.dbo.table1 set column1 = 2
SELECT * from db3.dbo.table1
COMMIT TRAN M2
UPDATE db4.dbo.table1 set column1 = 2
COMMIT TRAN T1
```

In this example the name of the mark is M2, and it will be placed in the logs of databases db1, db2, and db4. The mark is placed in the logs when the transaction commit log record is generated for the COMMIT TRAN T1 statement. db1 is marked even though the update was executed before the transaction was actually marked. db3 is not marked, despite having been accessed, because no update was made in db3. Also, even though db21 on another server was updated within the transaction, it will not be marked because no BEGIN TRAN...WITH MARK was actually executed by server2.

As indicated in the example, a transaction mark name is not automatically distributed to another server as the transaction spreads to the other server. In order to force the mark's spread to the other servers, a stored procedure must be written which contains a BEGIN TRAN *name* WITH MARK. That stored procedure must then be executed on the remote server under the scope of the transaction in the originating server. For example, consider a partitioned database that exists on multiple instances of Microsoft® SQL Server™. On each instance is a database named coyote. First, create stored procedure **sp_SetMark** in every database:

```
CREATE PROCEDURE sp_SetMark
@name nvarchar (128)
AS
BEGIN TRANSACTION @name WITH MARK
```

```
UPDATE coyote.dbo.Marks SET on = 1
COMMIT TRANSACTION
GO
```

Next, create stored procedure **sp_MarkAll** containing a transaction that will place a mark in every database. **sp_MarkAll** can be run from any of the instances:

```
CREATE PROCEDURE sp_MarkAll
@name nvarchar (128)
AS
BEGIN TRANSACTION
EXEC instance0.coyote.dbo.sp_SetMark @name
EXEC instance1.coyote.dbo.sp_SetMark @name
EXEC instance2.coyote.dbo.sp_SetMark @name
COMMIT TRANSACTION
GO
```

When a marked transaction is committed, the commit log record for each database in the marked transaction is placed in the log at a point where there are no in-doubt transactions in any of the logs. At this point, it is guaranteed that there are no transactions that appear as committed in one log, but not committed in another log. The following steps accomplish this during the commit of a marked transaction:

**Note** The commit of a distributed transaction is done in two phases: prepare and commit.

1. Prepare phase of a marking transaction will stall all new prepares and commits.

2. Only commits of already prepared transactions are allowed to continue.

3. Marking transaction then waits for all prepared transactions to drain (with time out).

4. Marked transaction is prepared and committed.

5. The stall of new prepares and commits is removed.

The stalls generated by marked transactions that span multiple databases can reduce the transaction processing performance of the server.

While rare in practice, it is possible for the commit of a distributed (cross-server) marked transaction to deadlock with other distributed transactions that are committing at the same time. When this happens, the marking transaction will be chosen as the deadlock victim and will be rolled back. When this error occurs, the application can retry the marked transaction. When multiple marked transactions attempt to commit concurrently, there is a higher probability of deadlock. Thus, running concurrent marked transactions is not recommended.

If the database is using log backups, and a log backup chain is active, log marks are traced in the **logmarkhistory** table:

- In the background after a transaction commits.

- One row per marked database, containing mark name, description, commit LSN, time.

- Time is computed before the commit record is generated.

- All entries for a distributed mark have the same time in a given **msdb** database.

- All times are before the timestamp in the commit log record.

## See Also

BEGIN TRANSACTION

RESTORE

Distributed Transactions Architecture

# Administering SQL Server

## Managing Backups

Manage your backups carefully to ensure that you can restore your system when needed. Each backup contains the descriptive text you provided when you created the backup, as well as expiration information. This information can be used to:

- Identify a backup.
- Determine when the backup can be safely overwritten.
- Identify all the backups on a backup medium, such as a tape, to determine which backup needs to be restored.

Additionally, the **msdb** database contains a complete history of all backup and restore operations on the server. SQL Server Enterprise Manager uses this information to suggest and execute a restore plan that can be used if a database needs to be restored. For example, if a database backup for a user database is created every night, and transaction log backups are created every hour during the day, this backup history information is stored in the **msdb** database. If the user database needs to be restored, SQL Server Enterprise Manager can use the history information stored in **msdb** to apply all the transaction log backups that relate to a specific database backup when the database backup is restored.

**Note** If the **msdb** database needs to be restored, any backup history information saved since the last backup of **msdb** was created is lost.

When working with backups:

- Maintain backups in a secure place, preferably at a site different from the site where the data resides.
- Keep older backups for a designated amount of time in case the most recent backup is damaged, destroyed, or lost.

- Establish a system for overwriting backups, reusing the oldest backups first.

- Use expiration dates on backups to prevent premature overwriting.

- Label backup media to prevent overwriting critical backups. This allows for easy identification of the data stored on the backup media or the specific backup set.

## See Also

Using Backup Media

Using Media Sets and Families

Viewing Information About Backups

# Administering SQL Server

## Backup Devices

When creating backups, you must select a backup device for the data to be backed up to. Microsoft® SQL Server™ 2000 can back up databases, transaction logs, and files to disk and tape devices.

### Disk Devices

Disk backup devices are files on hard disks or other disk storage media and are the same as regular operating system files. Referring to a disk backup device is the same as referring to any other operating system file. Disk backup devices can be defined on a local disk of a server or on a remote disk on a shared network resource, and they can be as large or as small as needed. The maximum file size is equivalent to the free disk space available on the disk.

If the backup is to be performed over the network to a disk on a remote computer, use the universal naming convention (UNC) name in the form \\Servername\Sharename\Path\File to specify the location of the file. As with writing files to the local hard disk, the appropriate permissions needed to read or write to the file on the remote disk must be granted to the user account used by SQL Server.

Because backing up data over a network can be subject to network errors, verify the backup operation after completion. For more information, see Verifying Backups.

**IMPORTANT** Backing up to a file on the same physical disk as the database is not recommended. If the disk device containing the database fails, there is no way to recover the database because the backup is located on the same failed disk.

### Tape Devices

Tape backup devices are used in the same way as disk devices, with the exception that:

- The tape device must be connected physically to the computer running an instance of SQL Server.

  Backing up to remote tape devices is not supported.

- If a tape backup device is filled during the backup operation, but more data still needs to be written, SQL Server prompts for a new tape and continues the backup operation.

**Note** Backups to tape devices cannot be performed on instances of SQL Server 2000 running on Microsoft Windows® 98.

To back up SQL Server (or Microsoft Windows NT® 4.0 or Windows 2000) data to tape, use a tape backup device or tape drive supported by Windows NT 4.0 or Windows 2000. Additionally, use only the recommended tapes for the specific tape drive (as suggested by the drive manufacturer). For more information about installing a tape drive, see the Windows NT 4.0 and Windows 2000 documentation.

## Physical and Logical Devices

SQL Server identifies backup devices using either a physical or logical device name.

A physical backup device is the name used by the operating system to identify the backup device, for example, C:\Backups\Accounting\Full.bak.

A logical backup device is an alias, or common name, used to identify the physical backup device. The logical device name is stored permanently in the system tables within SQL Server. The advantage of using a logical backup device is that it can be simpler to refer to than a physical device name. For example, a logical device name could be Accounting_Backup, but the physical device would be C:\Backups\Accounting\Full.bak.

When backing up or restoring a database, you can use either physical or logical backup device names interchangeably.

For example, execute the BACKUP statement with either the logical or physical device name:

```
-- Specify the logical backup device.
BACKUP DATABASE accounting
   TO Accounting_Backup
-- Or, specify the physical backup device.
BACKUP DATABASE accounting
```

```
   TO DISK = 'C:\Backups\Accounting\Full.Bak'
```

### To create a logical disk backup device

⊞ Transact-SQL

# Administering SQL Server

# Using Backup Media

The backup media is the actual physical storage used by the backup device to store the backup. Backup media can be either disk or tape.

For example, a backup device might be the file C:\Backups\Accounting\Full.bak. The backup media is the disk containing the file. Similarly for tape, a backup device might be the \\.\TAPE0 tape device on the local computer. The backup media are the physical tapes used to store the backup.

This section discusses the following aspects of working with backup media.

| Topic | Description |
|---|---|
| Using Media Sets and Families | Microsoft® SQL Server™ uses media sets, families, sequence numbers and other methods to properly organize backups and ensure correct media is being used for each backup and restore operation. |
| Initializing Backup Media | Before using the backup media for the first time, SQL Server must initialize, or format, the media and write a media header. |
| Password Protection | SQL Server 2000 allows backups to be protected with a password. Both the media and the backup itself can be password protected. |
| Overwriting Backup Media | SQL Server has safeguards to prevent you from accidentally overwriting media. Additionally, SQL Server can automatically overwrite backup sets that have reached a predefined expiration date. |
| Appending Backup Sets | New backup sets can be appended to existing media to make the best |

| | |
|---|---|
| | possible use of the available space. |
| Identifying the Backup Set to Restore | Backup sets are numbered so that users can specify which backup set on the media is to be restored. |

## See Also

BACKUP

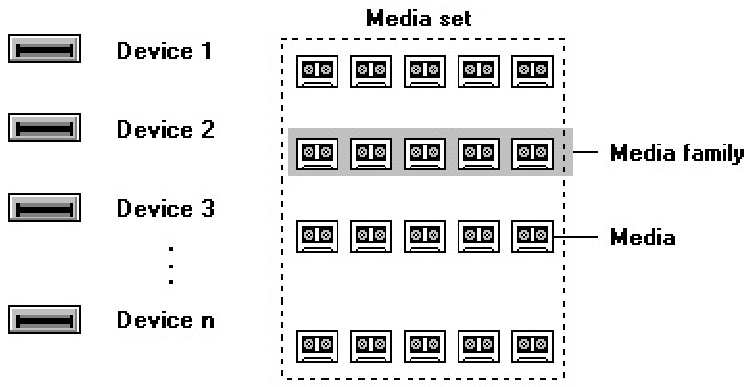
RESTORE

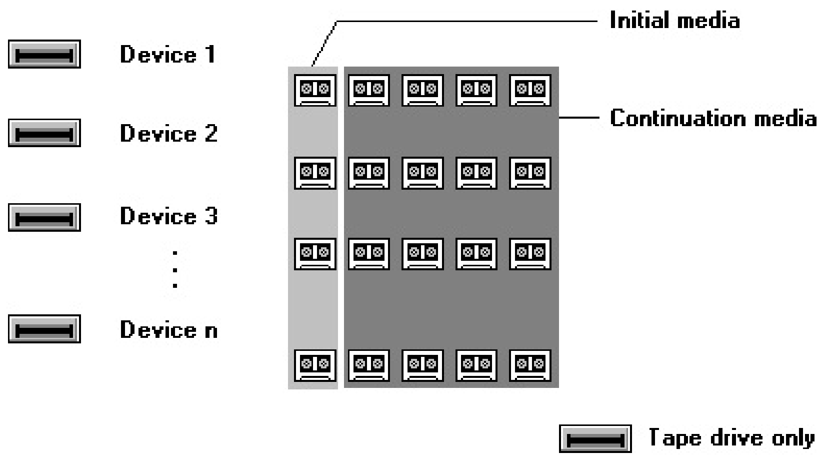
Using Media Sets and Families

## Using Media Sets and Families

A media set can contain one or more backup sets and describes all of the media used by those backup sets, regardless of the number of media or backup devices involved. For example, if four tape backup devices are used when creating a database backup, and five tapes per tape backup device are used to store the backup, the media set contains 20 tapes.

A media family describes all the media used by a single backup device for a single backup set. In the example earlier, there are four media families with each set of five tapes used by each tape backup device comprising one media family.





The initial media is the first media in a media family. If the initial media becomes full during the backup operation, more media is used until the backup operation is complete. All media in a media family except the initial media is described as continuation media.

**Note** Only tape backup devices use continuation media, allowing Microsoft® SQL Server™ to continue writing the backup after the initial tape is full.

To distinguish between each physical medium used within a media family, each medium is tagged with a sequence number to specify the order in which the media were used. The initial media is tagged with 1, the second media (the first continuation media) is tagged with 2, and so on. These sequence numbers are used when the backup set is restored to ensure that the operator restoring the backup mounts the correct media in the correct order. Additionally, media families within a media set are numbered sequentially.

When appending a backup set to a media set containing multiple media families, you must mount the last media in the family. If the last media is not mounted, SQL Server scans forward to the end of the media, requiring media to be changed until the last media in the family is mounted correctly.

Each SQL Server backup is stored on a media set, regardless of the number of backup devices used by the individual backup operation. Examples of media sets include:

- A single disk file.

- A single tape.

- A set of tapes written by one backup device. This set of tapes consists of a single media family (an initial media and one or more continuation media).

- A set of tapes written by four backup devices. Each set of tapes written by one backup device is the media family. Each media family contains an initial media and possibly one or more continuation media.

- A set of three disk files, used by one or more backup operations, with each backup operation using three backup devices.

When using multiple backup devices:

- The entire media set created by a backup operation must be used by all subsequent backup operations. For example, if a media set was created using two tape backup devices, all subsequent backup operations involving the same media set must use two backup devices.

- When restoring using tape devices, it is not necessary to use the same number of backup devices used by the media set when the backup was created. For example, restoring using fewer backup devices may be necessary when moving a database to another server, because the server may have fewer physical backup devices. You can restore media families in parallel. However, you must complete restoring an entire media family before starting another on a given tape device.

## See Also

Using Backup Media

## Initializing Backup Media

When creating a backup on a tape backup device for the first time, Microsoft® SQL Server™ needs to initialize the backup media before the backup can be created. Initializing media causes a media header to be written and deletes any existing media header, effectively deleting the previous contents of the tape. When initialized, previous information on the tape cannot be retrieved.

Initializing disk media involves only the backup device file(s) specified by the backup operation. Other files on the disk are unaffected. When using backup devices for the first time, SQL Server automatically creates the file(s) needed by the backup device(s) for the backup operation. Reinitializing disk backup devices overwrites the contents of the files used by the backup devices and writes a new media header.

### To initialize media for the first time when creating a backup

## Password Protection

Microsoft® SQL Server™ 2000 supports password protection for backup media and backup sets. Passwords are not required to perform backup operations, but they provide an added level of security. You can use them in addition to using SQL Server security roles. The use of password protection helps guard against:

- Unauthorized restoration of databases.
- Unauthorized appends to the media.
- Unintentional overwriting of the media.

**IMPORTANT** Password security does not prevent overwriting the media by formatting it or using it for a continuation volume. Additionally, specifying a password does not encrypt the data in any way.

Passwords can be used for either media sets or backup sets:

- Media set passwords protect all the data saved to that media. The media set password is set when the media header is written; it cannot be altered. If a password is defined for the media set, the password must be supplied to perform any append or restore operation.

  You will only be able to use the media for SQL Server backup and restore operations. Specifying a media set password prevents a Microsoft Windows NT® 4.0 or Windows® 2000 backup from being able to share the media.

- Backup set passwords protect only a particular backup set. Different backup set passwords can be used for each backup set on the media. A backup set password is set when the backup set is written to the media. If a password is defined for the backup set, the password must be supplied to perform any restore of that backup set.

## Overwriting Backup Media

By overwriting backups on media, the existing contents of the backup set are overwritten with the new backup and are no longer available. For disk backup media, only the files used by the backup device(s) specified in the backup operation are overwritten; other files on the disk are unaffected. When overwriting backups, the existing media header can be preserved, and the new backup is created as the first backup on the backup device. If there is no existing media header, a valid media header with an associated media name and media description is written automatically. If the existing media header is invalid, the backup operation terminates.

Backup media is not overwritten if either of the following conditions is met:

- The existing backups on the media have not expired.

  The expiration date specifies the date the backup expires and can be overwritten by another backup. You can specify the expiration date when a backup is created. By default, the expiration date is determined by the **media retention** option set with **sp_configure**.

- The media name, if provided, does not match the name on the backup media.

  The media name is a descriptive name used for easy identification of the media.

However, these checks can be explicitly skipped if you are sure you want to overwrite the existing media (for example, if you know that the backups on the tape are no longer needed).

If the backup media is password protected by Microsoft® Windows NT® 4.0 or Windows® 2000, Microsoft SQL Server™ does not write to the media. To overwrite media that is password protected, you need to reinitialize the media.

### To create a database backup

## Appending Backup Sets

Backups performed at different times from the same or different databases can be stored on the same media. Additionally, data other than Microsoft® SQL Server™ data can be stored on the same media, such as Microsoft Windows NT® 4.0 file backups. By appending a new backup set to existing media, the previous contents of the media remain intact, and the new backup is written after the end of the last backup on the media.

By default, SQL Server always appends new backups to media. Appending can occur only at the end of the media. For example, if a media contains five backup sets, it is not possible to skip the first three backup sets to overwrite the fourth backup set with a new backup set.

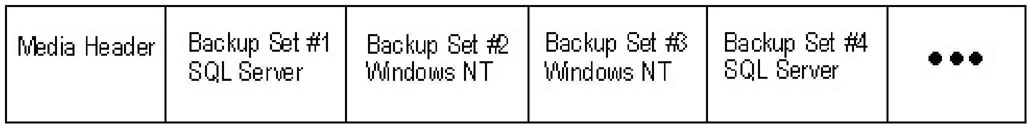
If you use BACKUP WITH NOREWIND for a tape backup, the tape will be left open at the end of the operation. This allows you to append further backups to the tape without rewinding the tape and then scanning forward again to find the last backup set. A list of currently open tapes can be found by querying the **sysopentapes** table in the **master** database.

Windows NT 4.0 and Microsoft Windows® 2000 backups and SQL Server backups are not interoperable. Though media can be shared between the two, a SQL Server backup cannot be used to backup Windows NT 4.0 data. You can use NTBackup to backup database files if an instance of SQL Server is not running. Do not rely on file-level backups using NTBackup if an instance of SQL Server is running.

### To append a new backup to existing media

## Identifying the Backup Set to Restore

Each backup set on media, including foreign backup sets such as Microsoft® Windows NT® 4.0 file backups, is numbered. This allows the backup set you want to restore to be referenced easily. For example, the following media contains four backup sets: two Microsoft SQL Server™ backups and two foreign backup sets (for example, Windows NT 4.0 files).



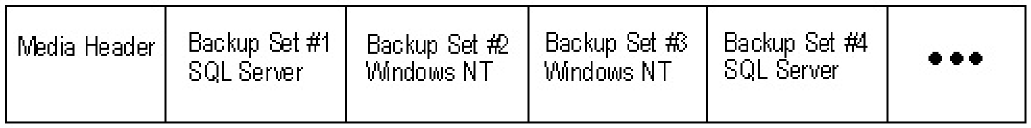
To restore a specific backup set, specify the position number of the backup set you want to restore. For example, to restore the second SQL Server backup set, the fourth backup set on the media, specify 4 as the backup set to restore.

**To restore a specific database backup**

# Administering SQL Server

## Backup Formats

All media used for a backup or restore operation use a standard backup format called Microsoft® Tape Format (MTF). MTF enables Microsoft SQL Server™ backups to coexist on the same media as backups that are not SQL Server backups (foreign backup sets), provided that the backups use MTF. For example, SQL Server backups can exist on the same media as Microsoft Windows NT® 4.0 and Windows® 2000 backups.



Integrating any backups supporting MTF onto a single tape reduces backup media storage requirements, costs, and administrative overhead because the same tape media can be used to store different backups from different applications.

All media begins with a media header describing the media. The media header is usually written one time and remains intact for the life of the media. This allows each piece of media to be tracked. The media header can contain a media name, the name given to the particular media, and is assigned by the first person using the media. Consistent use of media names helps identify the media and prevent errors.

### See Also

BACKUP

Using Backup Media

RESTORE HEADERONLY

RESTORE LABELONLY

# Administering SQL Server

## Viewing Information About Backups

After backups are created, you may need to view information about the backups, such as:

- A list of the database and transaction log files contained in a specific backup set.

- The backup header information for all backups on a particular backup media.

- The media header information for a particular backup medium.

### Listing Database and Transaction Log Files

Information displayed when listing the database and transaction log files in a backup includes the logical name, physical name, file type (database or log), filegroup membership, file size (in bytes), the maximum allowed file size, and the predefined file growth size (in bytes). This information is useful to determine the names of the files in a database backup before restoring the database backup when:

- You have lost a disk drive containing one or more of the files for a database.

  You can list the files in the database backup to determine which files were affected, and then restore those files onto a different drive when restoring the entire database, or restore just those files and apply any transaction log backups created since the database was backed up.

- You are restoring a database from one server onto another server, but the directory structure and drive mapping does not exist on the server.

  Listing the files in the backup allows you to determine which files are affected. For example, the backup contains a file that it needs to restore to the E:\ drive, but the destination server does not have an E:\ drive. The file needs to be relocated to another location, such as the C:\ drive, when the file is restored.

## Viewing Header Information

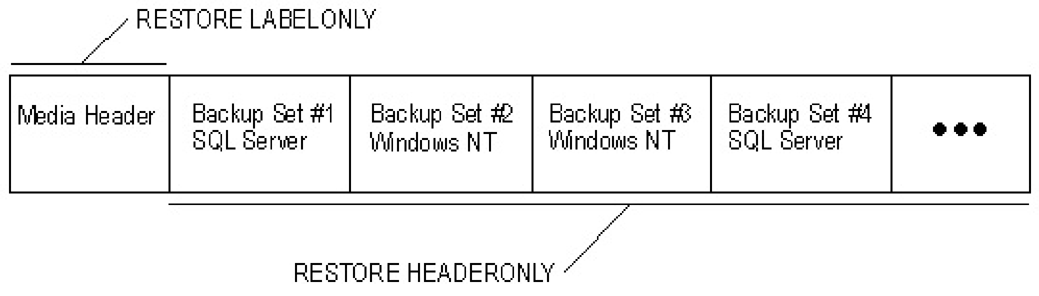
Viewing the backup header displays information about all Microsoft® SQL Server™ and foreign backup sets on the media. Information displayed includes the types of backup devices used, the types of backup (for example, database, transaction, file, or differential database), and backup start and stop date/time information. This information is useful when you need to determine which backup set on the tape to restore, or the backups that are contained on the media.

**Note** Viewing backup header information can take a long time for high-capacity tapes because the entire media needs to be scanned to display information about each backup on the media.

Viewing the media header displays information about the media itself, rather than the backups on the media. Media header information displayed includes the media name, description, name of the software that created the media header, and the date the media header was written. For more information about a detailed list of the header information displayed, see RESTORE LABELONLY.

**Note** Viewing the media header is quick because only the media header is read after it has been located one time at the beginning of the media.

The following chart provides an example of the differences between viewing backup header and media header information. In this example, restoring the backup header information for the tape media containing two SQL Server backups and two foreign (Microsoft Windows NT® 4.0 or Microsoft Windows® 2000) backups retrieves information for all backup sets on the media, requiring that the entire tape be scanned. However, restoring the media header requires only information from the single media header written at the beginning of the tape to be retrieved.



## To view the data and log files in a backup set

⊞ Transact-SQL

## Administering SQL Server

## Verifying Backups

Although not required, verifying a backup checks that the backup is intact physically, and that you can rely on your backup in the event you need to use it. Verifying a backup involves:

- Checking the backup set to ensure that all files have been written.
- Checking to ensure that the files in the backup are readable.

Verifying a backup does not check that the structure of the data contained within the backup set is correct. For example, although the backup set may have been written correctly, it may be possible for some type of database integrity problem to be present within the database files that comprise the backup set. To verify the structure of the data before creating a backup, you can perform database consistency checks. For more information about running database consistency checks, see Data Integrity Validation.

### To verify the backup set

⊞ Transact-SQL

# Administering SQL Server

## Backing Up and Restoring System Databases

The system databases need to be backed up just as user databases are backed up. This allows the system to be rebuilt in the event of system or database failure, for example, if a hard disk fails. It is important to have regular backups of the following system databases:

- **master**
- **msdb**
- **distribution** (when the server is configured as a replication Distributor)
- **model** (if modified)

**Note** It is not possible to back up the **tempdb** system database. **tempdb** is rebuilt each time an instance of Microsoft® SQL Server™ is started. When an instance of SQL Server is shut down, any data in **tempdb** is deleted permanently.

# Administering SQL Server

## Backing Up the master Database

The **master** database must be backed up. If **master** is damaged in some way, for example because of media failure, an instance of Microsoft® SQL Server™ may not be able to start. In this event, it is necessary to rebuild **master**, and then restore the database from a backup.

Consider backing up **master** after any statement or system procedure is executed that changes information in **master**, for example, changing a server-wide configuration option. If **master** is not backed up after it changes and then the backup is restored, any changes since the last backup are lost. For example, a user database is created after **master** is backed up and tables and data are added to the database. If **master** is then restored because of a hard disk failure, the user database will not be known to SQL Server because there are no entries in the restored **master** database for this new user database. In this case, if all database files comprising the user database still exist on the disk(s), the user database can be created by attaching the database files. For more information, see Attaching and Detaching Databases.

**Note** It is recommended that user objects not be created in **master**; otherwise **master** needs to be backed up more frequently. Additionally, user objects compete with the system objects for space.

The types of operations that cause master to be updated, and that require a backup to take place, include:

- Creating or deleting a user database.

  If a user database grows automatically to accommodate new data, this does not affect **master**. Deleting files and filegroups does not affect **master**.

- Adding logins or other login security-related operations.

  Database security operations, such as adding a user to a database, do not affect **master**.

- Changing any server-wide or database configuration options.

- Creating or removing logical backup devices.
- Configuring the server for distributed queries and remote procedure calls, such as adding linked servers or remote logins.

**Note** Only full database backups of **master** can be created.

### To create a database backup

## Administering SQL Server

## Restoring the master Database

If **master** is damaged in some way, for example due to media failure, an instance of Microsoft® SQL Server™ may not be able to start if the damage is severe. There are two methods to return **master** to a usable state:

- Restore from a current backup.

- Rebuild completely using the Rebuild Master utility.

IMPORTANT  Keep a current backup of **master**. Rebuilding **master** using the Rebuild Master utility causes all data stored previously in **master** to be lost permanently. SQL Server will still be able to access other databases.

If an instance of SQL Server can be started because **master** is accessible, and at least partly usable, it is possible to restore **master** from a full database backup. However, if an instance of SQL Server cannot be started because of severe damage to **master**, it is not possible to restore a backup of **master** immediately because an instance of SQL Server needs to be running to restore any database. The **master** database first needs to be rebuilt using the Rebuild Master utility, and the current database backup can be restored as normal.

## Restoring the master Database from a Current Backup

If there have been any changes to **master** after the database backup was created, those changes are lost when the backup is restored. Therefore, it is necessary to re-create those changes manually after restoring **master** from a backup by executing the statements necessary to re-create the missing changes. For example, if any Microsoft® SQL Server™ logins have been created after the backup was performed, those are lost when **master** is restored. Re-create the logins using SQL Server Enterprise Manager or the original scripts used to create the logins.

The **master** database can only be restored from a backup created on an instance of SQL Server 2000. Restore of **master** database backups which were made on SQL Server version 7.0 or earlier is not supported.

**Note** Any database users previously associated with logins that need to be re-created are orphaned because the login is lost. For information about associating an existing database user to a new SQL Server login, see sp_addlogin. For information about associating an existing database user with a Microsoft Windows NT® 4.0 or Windows® 2000 user, see sp_grantlogin.

If any user databases were created after **master** was backed up, those databases cannot be accessed once **master** is restored unless:

- The databases are restored from backups.

-or-

- The databases are reattached to SQL Server. It is recommended that you attach the databases to avoid restore time.

Attaching the database to SQL Server re-creates the system table entries needed and makes the database available in the same state it was before the **master** database was restored. It is not necessary to re-create the database first; the files can be attached without knowing how the database was created, as long as all the files comprising the database are attached.

It is necessary to restore a backup of the database only if the data and transaction log files of the database no longer exist or are unusable or damaged in some other way due to a media failure.

If any objects, logins, or databases, for example, have been deleted after **master** was backed up, those objects, logins, and databases should be deleted from **master**.

**IMPORTANT** If any databases no longer exist, but are referenced in a backup of **master** that is restored, SQL Server may report errors when it starts because it cannot find those databases any longer. Those databases should be dropped after the backup is restored.

After restoring **master**, the instance of SQL Server is stopped automatically. If you need to make further repairs and wish to prevent more than a single connection to the server, you should start the server in single user mode again. Otherwise, the server can be restarted normally. If you choose to restart the server in single-user mode, all SQL Server services (except SQL Server itself) and utilities, such as the SQL Server Agent, should be stopped because they may try to access the instance of SQL Server.

When **master** has been restored and any changes have been reapplied, back up **master** immediately.

### To start the default instance of SQL Server in single-user mode

## Rebuilding the master Database

The **master** database can be rebuilt using the Rebuild Master utility if:

- A current backup of **master** is not available.
- The backup cannot be restored because an instance of Microsoft® SQL Server™ cannot start due to severe damage to **master**.

When **master** has been rebuilt, a current backup of **master** can be restored or the user databases, backup devices, SQL Server logins, and so on can be re-created using SQL Server Enterprise Manager or the original scripts used to create those entries.

IMPORTANT The Rebuild Master utility rebuilds **master** completely. Because the **msdb** and **model** system databases are rebuilt as well, it will normally be necessary to restore backups of those databases.

The general steps required to rebuild **master** completely if no backup is available are:

- Run the Rebuild Master utility to rebuild the system databases.

  IMPORTANT The compact disc or shared network directory containing the SQL Server installation software is required to rebuild the **master** database.
- Re-create any necessary backup devices.
- Reimplement security operations.
- Restore **msdb** if necessary.
- Restore **model** if necessary.

- Restore **distribution** if necessary.

- Restore or attach user databases if necessary.

When **master** has been re-created and any changes have been reapplied, back up **master** immediately.

## To rebuild the master database

# Administering SQL Server

## Backing Up the model, msdb, and distribution Databases

The **model**, **msdb,** and **distribution** databases are backed up in the same way as user databases and should be backed up regularly if they are changed. These databases perform the following functions:

- The **model** database is the template used by Microsoft® SQL Server™ when creating other databases, such as **tempdb** or user databases. When a database is created, the entire contents of the **model** database, including database options, are copied to the new database.

- The **msdb** database is used by SQL Server, SQL Server Enterprise Manager, and SQL Server Agent to store data, including scheduling information and backup and restore history information.

  SQL Server automatically maintains a complete online backup and restore history in **msdb**. This information includes who performed the backup, at what time, and on which devices or files it is stored. This information is used by SQL Server Enterprise Manager to propose a plan for restoring a database and applying any transaction log backups. Backup events for all databases are recorded even if they were created with custom applications or third-party tools. For example, if you use a Microsoft Visual Basic® application that calls SQL-DMO objects to perform backup operations, the event is logged in the **msdb** system tables, the Microsoft Windows® application log, and SQL Server error log.

  If you use the backup and restore history information in **msdb** when recovering user databases, it is recommended that you use the Full Recovery model for **msdb**. Additionally, consider placing the **msdb** transaction log on fault tolerant storage.

- The **distribution** database is used by the replication components of SQL Server, such as the Distribution Agent, to store such data as transactions, snapshot jobs, synchronization status, and replication history information. Any server configured to participate either as a remote distribution server or as a combined Publisher/Distributor has a **distribution** database.

## Backup Considerations

It is important to back up **model**, **msdb**, or **distribution** after any operation that updates the database:

- If **model** is damaged in some way due to media failure, and there is no current backup available, any user-specific template information added to **model** is lost and needs to be re-created manually.

- If **msdb** is damaged, then any scheduling information used by the SQL Server Agent is lost and needs to be re-created manually. Backup and restore history information is also lost.

- If **distribution** is damaged, and there is no current backup available, any replication information used by the SQL Server replication utilities is lost and needs to be re-created manually. For this reason, consider using Full Recovery model for **distribution**.

All recovery models are supported for **model**, **msdb** and **distribution**.

## Modifying the model, msdb and distribution Databases

The **model**, **msdb** and **distribution** databases can be modified in the following ways:

- The **model** database is modified only by specific user changes.

- The **msdb** database is altered automatically by:
    - Scheduling tasks.

    - Storing Data Transformation Services (DTS) packages created with the DTS Import/Export Wizard to an instance of SQL Server.

    - Maintaining online backup and restore history.
    - Replication.
- The **distribution** database is altered automatically by:
    - The Replication Log Reader Agent utility.
    - The Replication Distribution Agent utility.
    - The Replication Snapshot Agent utility.
    - The Replication Merge Agent utility.

As with **master**, it is recommended that user objects not be created in **msdb** or **distribution**; otherwise **msdb** and **distribution** need to be backed up more frequently. Additionally, user objects compete with the system objects for space.

## See Also

Backing Up and Restoring Databases

Backing Up and Restoring Replication Databases

Configuring the SQLServerAgent Service

Replication Overview

System Tables

Using Recovery Models

# Administering SQL Server

## Restoring the model, msdb, and distribution Databases

The **model**, **msdb**, or **distribution** database may need to be restored from a backup when:

- The **master** database has been rebuilt using the **Rebuild master** command prompt utility.
- The **model**, **msdb**, or **distribution** database has been damaged, for example, due to media failure.
- The **model** has been modified. In this case, it is necessary to restore **model** from a backup when you rebuild **master** because the Rebuild Master utility deletes and re-creates **model.**

The **model** and **msdb** databases can only be restored from backups created on a Microsoft® SQL Server™ 2000 server. Restore of backups of these databases made on SQL Server version 7.0 or earlier is not supported.

If **msdb** contains scheduling or other data used by the system, it is necessary to restore **msdb** from a backup when you rebuild **master** because the utility deletes and re-creates **msdb.** This results in a loss of all scheduling information, as well as the backup and restore history. If **msdb** is not restored, and is not accessible, SQL Server Agent cannot access or initiate any previously scheduled tasks.

Meta Data Services uses **msdb** as the default repository database. An open connection between Meta Data Services and **msdb** will disrupt an **msdb** restore. To release the connection, restart Enterprise Manager and then restore **msdb**. Do not click the Meta Data Services node in Enterprise Manager until **msdb** is fully restored.

The **distribution** database is not rebuilt automatically when the Rebuild Master utility is used to rebuild **master**; therefore it is not necessary to restore **distribution** after rebuilding **master**. If the **distribution** database is still intact, **distribution** can be re-created automatically by attaching the database to SQL

Server. Alternatively, a backup of **distribution** can be restored instead.

However, if **distribution** is not re-created by restoring a backup or attaching the database, the SQL Server replication utilities will not run, preventing data replication. If the **distribution** database is used for replication by many Publishers, this can affect many systems.

You cannot restore a database that is being accessed by users. Therefore, when restoring **msdb**, SQL Server Agent should be stopped. If SQL Server Agent is running, it may access **msdb**. Similarly, when restoring **distribution**, the SQL Server replication utilities should be stopped. If the SQL Server replication utilities are running, they may access **distribution**.

Replication utilities that must be stopped are:

- The Replication Log Reader Agent utility.
- The Replication Distribution Agent utility.
- The Replication Snapshot Agent utility.
- The Replication Merge Agent utility.

## See Also

Attaching and Detaching Databases

Backing Up and Restoring Replication Databases

Configuring the SQLServerAgent Service

Replication Overview

Administering SQL Server

## Handling Large Mission-Critical Environments

Mission-critical environments often require that databases be available continuously, or for extended periods of time with minimal down-time for maintenance tasks. Therefore, the duration of unexpected situations, such as a hardware failure, that require databases to be restored needs to be kept as short as possible. Additionally, mission-critical databases are often large, requiring longer periods of time to back up and restore. Microsoft® SQL Server™ offers several methods for increasing the speed of backup and restore operations, thereby minimizing the effect on users during both operations.

The following practices will help:

- Use multiple backup devices simultaneously to allow backups to be written to all devices at the same time. Similarly, the backup can be restored from multiple devices at the same time.

- Use a combination of database, differential database, and transaction log backups to minimize the number of backups that need to be applied to bring the database to the point of failure.

- Use file and filegroup backups and transaction log backups, which allows only those files that contain the relevant data, rather than the entire database, to be backed up or restored.

- Use snapshot backups which reduce backup and restore time to a minimum. Snapshot backups are supported by third party vendors. For more information, see Snapshot Backups.

# Administering SQL Server

## Using Multiple Media or Devices

Multiple backup devices can be used for backup and restore operations. This allows Microsoft® SQL Server™ to use parallel I/O to increase the speed of backup and restore operations because each backup device can be written to or read from at the same time as other backup devices. For enterprises with large databases, using many backup devices can greatly reduce the time taken for backup and restore operations. SQL Server supports a maximum of 64 backup devices for a single backup operation.

However, all backup devices used in a single backup (and consequently restore) operation must be of the same type (disk or tape). For example, to back up the **sales_db** database daily using database and differential database backups to tape, only multiple tape drives can be used.

**Note** Tape backup devices must be attached to the server physically. It is not possible to use tape backup devices on remote computers.

Creating and restoring backups using multiple backup devices is the same as creating and restoring backups using a single device. The only difference is that all backup devices involved in the operation, not just one, are specified. For example, if a database backup is to be created using three tape backup devices such as \\.\TAPE0, \\.\TAPE1, and \\.\TAPE2, each of the tape devices needs to be specified as part of the backup operation, although fewer tape backup devices can be used when restoring the backup later.

When creating a backup using multiple backup devices on removable media, each backup media does not need to be the same size, have the same amount of storage available, or operate at the same speed. If one backup media used by a backup device runs out of space while a backup is being created, SQL Server stops writing to the backup device and prompts for new media to continue writing to that backup device. While waiting for new media to be inserted into the backup device, the backup operation continues writing data to any other backup devices involved in the backup operation, as long as the backup media used by these devices has space available.

For example, three tape backup devices of equal speed are used to store a database backup. The first two tape media are 10 gigabytes (GB) in size, but the third is only 5 GB in size. If the **sales** database, which is 20 GB in size, is backed up to all three tape backup devices simultaneously, the backup operation will stop writing to the third backup device and prompt for a new tape when 5 GB has been written to the tape. However, the backup operation continues writing data to the other two backup devices. When the tape media on the third backup device is replaced with a new tape, the backup operation continues writing data to the third backup device.

Several internal synchronization points occur when a database backup is written to multiple backup devices. The most important synchronization point occurs when all the data in the database has been backed up, and the transaction log is about to be backed up. All backup devices used in the backup operation must not be blocked during these synchronization points; otherwise, the entire backup operation is blocked until all backup media is available. For example, three tape backup devices are used to store a database backup, and the second tape backup device is blocked, waiting for the existing tape to be replaced because the space on the tape has been exhausted. If a synchronization point occurs, the entire backup operation will stop until the tape in the second backup device is replaced.

**IMPORTANT** When using multiple backup devices to perform backup operations, the backup media involved can be used only for SQL Server backup operations. For more information, see Using Backup Media.

## See Also

Using Backup Media

Optimizing Backup and Restore Performance

# Administering SQL Server

## Reducing Recovery Time

Using database, differential database, and transaction log backups together can reduce the amount of time it takes to restore a database back to any point in time after the database backup was created. Additionally, creating both differential database and transaction log backups can increase the robustness of a backup in the event that either a transaction log backup or differential database backup becomes unavailable, for example, due to media failure.

Typical backup procedures using database, differential database, and transaction log backups create database backups at longer intervals, differential database backups at medium intervals, and transaction log backups at shorter intervals. For example, create database backups weekly, differential database backups one or more times per day, and transaction log backups every ten minutes.

If a database needs to be recovered to the point of failure, for example, due to a system failure:

1. Back up the currently active transaction log. This operation will fail if the transaction log has been damaged.

2. Restore the last database backup created.

3. Restore the last differential backup created since the database backup was created.

4. Apply all transaction log backups, in sequence, created after the last differential backup was created, finishing with the transaction log backup created in Step 1.

**Note** If the active transaction log cannot be backed up, it is possible to restore the database only to the point when the last transaction log backup was created. Changes made to the database since the last transaction log backup are lost and must be redone manually.

By using differential database and transaction log backups together to restore a database to the point of failure, the time taken to restore a database is reduced because only the transaction log backups created since the last differential database backup was created need to be applied. If a differential database backup was not created, then all the transaction log backups created since the database was backed up need to be applied.

For example, a mission-critical database system requires that a database backup is created each night at midnight, a differential database backup is created on the hour, Monday through Saturday, and transaction log backups are created every 10 minutes throughout the day. If the database needs to be restored to its state at 5:19 A.M. on Wednesday:

1. Restore the database backup created on Tuesday night.

2. Restore the differential database backup created at 5:00 A.M. on Wednesday.

3. Apply the transaction log backup created at 5:10 A.M. on Wednesday.

4. Apply the transaction log backup created at 5:20 A.M. on Wednesday, specifying that the recovery process only applies transactions that occurred before 5:19 A.M.

Alternatively, if the database needs to be restored to its state at 3:04 A.M. on Thursday, but the differential database backup created at 3:00 A.M. on Thursday is unavailable:

1. Restore the database backup created on Wednesday night.

2. Restore the differential database backup created at 2:00 A.M. on Thursday.

3. Apply all the transaction log backups created from 2:10 A.M. to 3:00 A.M. on Thursday.

4. Apply the transaction log backup created at 3:10 A.M. on Thursday, specifying that the recovery process only applies transactions that occurred before 3:04 A.M.

### To create a database backup

# Administering SQL Server

## Using File Backups

The files in a database can be backed up and restored individually. Doing this can increase the speed of recovery by allowing you to restore only damaged files without restoring the rest of the database. For example, if a database is comprised of several files physically located on different disks and one disk fails, only the file on the failed disk needs to be restored.

File backup and restore operations must be used in conjunction with transaction log backups. For this reason, file backups can only be used with the Full Recovery and Bulk-Logged Recovery models. For more information on recovery models, see Selecting a Recovery Model.

File backups offer these advantages:

- Recovery from isolated media failures is faster. The damaged file or files can be quickly restored.

- File and transaction log backups can be created simultaneously, allowing you to maintain regular log backup schedules.

- File backups allow greater flexibility in scheduling and media handling. For example, for very large databases, full database backups can become unmanageable.

  This flexibility also is useful for large databases that contain data with varying update characteristics.

To maximize these advantages, consider your data layout and usage patterns. It is recommended that you:

- Back up frequently modified data often.

- Back up infrequently modified data less often.

- Back up read-only data once.

**Note** When restoring a file backup, you must roll forward the transaction log to ensure the file is consistent with the rest of the database. To avoid needing to roll forward many transaction log backups on files that are backed up rarely, use file differential backups. For more information, see File Differential Backups.

File and filegroup backups are functionally equivalent. A filegroup backup is a single backup of all files in the filegroup and is equivalent to explicitly listing all files in the filegroup when creating the backup. Files in a filegroup backup can be restored individually or as a group.

Only one file backup operation can occur at a time. You can backup multiple files in a single operation, but this may extend your recovery time if you only need to restore a single file, because the entire backup will be read to locate that file.

A complete set of file backups, together with backups of the transaction log covering the time that the file backups were created, is the equivalent of a database backup.

**Note** Individual files can be restored from a database backup. This means that you can use database and transaction log backups as your backup procedure, and still be able to restore individual files. However, it will take longer to locate and restore a file from a database backup than a file backup.

The primary disadvantage of file backups as compared to database backups is the additional administrative complexity. Care must be taken to maintain a full set of file backups and covering log backups. A media failure can render an entire database unrecoverable if there is no backup of the damaged file.

When creating file backups, the transaction log is not captured by the backup operation. Transaction log backups must be created after a file backup is created. After restoring files, you must bring the database to a consistent state by restoring the transaction log backups created since the file backups were created.

- Recovery time can be reduced through the use of file differential backups. For more information, see File Differential Backups.

## Restoring File Backups

After restoring files, you must restore the transaction log backups created since the file backups were created to bring the database to a consistent state. The transaction log backup can be rolled forward quickly, because only the changes that apply to the restored files are applied.

To restore a damaged file or files from file backups:

1. Back up the active transaction log. If you cannot do this because the log has been damaged, you must restore the entire database.

2. Restore each damaged file from the most recent backup of that file.

3. Restore transaction log backups in sequence, starting with the backup that covers the oldest of the restored files.

4. Restore the backup of the active transaction log created in step 1.

5. Recover the database.

**Important** Microsoft® SQL Server™ requires that files be recovered to a state consistent with the rest of the database. It is not possible to stop the recovery of individual files early. For this reason, you must always back up the active transaction log prior to restoring a file backup. If the transaction log is damaged or if you wish to recover the entire database to a specific point in time, you must restore the entire set of file backups before you apply transaction log backups. To minimize the risk of transaction log damage, locate the transaction log on fault tolerant storage.

The procedure for restoring the entire database is similar. The only difference is that all files are restored. File backups can also be used to restore the database to an earlier point in time. To do this, you must restore a complete set of file backups, then restore transaction log backups in sequence to reach the desired time. You can stop at a time or a marked transaction.

For more information on point-in-time recovery, see Restoring a Database to a Prior State.

### To back up files and filegroups

## File Differential Backups

You can create a file differential backup to back up only the data changed since the last file backup. File differential backups can dramatically reduce recovery time by reducing the amount of transaction log that must be restored. In Microsoft® SQL Server™ 2000, file differential backups can be extremely fast because SQL Server 2000 tracks changes made since the file was last backed up. Therefore, the file is not scanned.

Consider file differential backups if:

- You are backing up some files much less frequently than others.
- Your files are large and the data is updated infrequently, or the same data is updated repeatedly.
- You have backed up a read-only file. A recent file differential backup will eliminate the need to apply many log backups to recover the file.

File differential backups can be used only in conjunction with file backups and are only supported by the Full Recovery and Bulk-Logged Recovery models. For more information, see Using File Backups and Selecting a Recovery Model.

To restore a damaged file or files from file backups and file differential backups:

1. Back up the active transaction log. If you cannot do this because the log has been damaged, you must restore the entire database.
2. Restore each damaged file from the most recent file backup of that file.
3. Restore the most recent file differential backup for each file restored in Step 2.

4. Restore transaction log backups in sequence, starting with the backup that covers the oldest of the restored files.

5. Restore the backup of the active transaction log created in Step 1.

6. Recover the database.

The procedure for restoring the entire database is similar. The only difference is that all files are restored, and you can recover to a specific point in time or a named transaction.

Information about available backups is contained in **msdb**. If **msdb** is unavailable, this information can be obtained from the backup itself.

It is not recommended to use both database differential and file differential backups on the same database.

## Administering SQL Server

## Snapshot Backups

Microsoft® SQL Server™ 2000 supports snapshot backup and restore technologies in conjunction with independent hardware and software vendors. Snapshot backups minimize or eliminate the use of server resources to accomplish the backup. This is especially beneficial for moderate to very large databases in which availability is extremely important. The primary benefits of this technology are:

- A backup can be created in a very short time, typically measured in seconds, with little or no impact on the server.
- Restore can be accomplished from a disk backup just as quickly.
- Backup to tape can be accomplished by another host with no impact on the production system.
- A copy of a production database can be created instantly for reporting or testing.

Snapshot backups can be created for an entire database or individual files. They are functionally equivalent to conventional full database and file backups and can be rolled forward using conventional, differential and log backups. Like other backups, snapshot backups and restores are tracked in **msdb**.

The snapshot backup and restore functionality is accomplished in cooperation with third party hardware and/or software vendors. These vendors use features of SQL Server 2000 designed for this purpose. The underlying backup technology creates an instantaneous copy of the data being backed up. This is typically accomplished by splitting a mirrored set of disks or creating a copy of a disk block when it is written, preserving the original. At restore time, the original is made available immediately and synchronizing the underlying disks is done in the background, resulting in almost instantaneous restores.

For more information, see the SQL Server page at the Microsoft Web site. In addition, you can contact your enterprise storage and/or backup software vendor.

# Administering SQL Server

## Copying Databases to Other Servers

Creating database backups allows you to copy data from one computer to another. The copied database can be used for testing, checking consistency, developing software, running reports, or possibly making databases available to remote branch operations. By copying a database from one computer to another, it is possible to reduce resource contention because processing is offloaded to other computers. Copied databases restored onto separate computers are often used for read-only operations.

**Note** With Microsoft® SQL Server™ 2000, the sort order and code page of the database being copied is no longer a concern. SQL Server now handles multiple collations.

A database can also be copied to another computer to act as a standby server. The database and the transaction logs are copied to another computer periodically, which can be brought online if the primary computer fails for some reason. The level of synchronization between the primary computer and the standby server is determined by how often regular backups of the primary computer are created and then applied to the standby server. For more information, see Using Standby Servers.

**Note** It is possible to back up and restore databases between computers running an instance of SQL Server on Microsoft Windows NT® 4.0, Microsoft Windows® 2000, and Windows 98.

Other methods for copying data between multiple instances of SQL Server include using:

- The Data Transformation Services (DTS) Import/Export Wizard to copy and modify data between any ODBC, OLE DB, or text data source and an instance of SQL Server.

- The **bcp** utility to copy data between an instance of SQL Server and a data file, using native, character, or Unicode mode.

- The INSERT statement, which uses a distributed query as the select list to extract data from another data source.

- The Copy Database Wizard to copy or move databases and associated meta data between servers.

## See Also

bcp Utility

DTS Import/Export Wizard

Using Standby Servers

Using the Copy Database Wizard

# Administering SQL Server

## Copying Databases

The general steps required to copy a database to another computer are:

1. Back up the database from the source computer running an instance of Microsoft® SQL Server™.

2. Create backup devices, if desired, at the destination computer running an instance of SQL Server.

3. Restore the database backup to the destination computer. It is not necessary to create the files or the database before restoring the backup.

### Re-creating Database Files

Restoring a database automatically creates the files needed by the database backup to restore the backup into. The database files (hence the database) do not need to be created before restoring a backup. By default, the files created by SQL Server during the restoration process use the same name and path as the backup files from the original database on the source computer. Therefore, it is useful to know in advance the files that are created automatically by the restore operation, because:

- The file names may already exist on the computer, causing an error.

- The directory structure or drive mapping may not exist on the computer.

  For example, the backup contains a file that it needs to restore to drive E, but the destination computer does not have a drive E.

- If the database files are allowed to be replaced, any existing database and files with the same names as those in the backup are overwritten, unless those files belong to a different database.

## Moving the Database Files

If the files within the database backup cannot be restored onto the destination computer because of the reasons mentioned earlier, it is necessary to move the files to a new location as they are being restored. For example:

- It may be necessary to restore some of the database files in the backup to a different drive because of capacity considerations. This is likely to be a common occurrence because most computers within an organization do not have the same number and size of disk drives or identical software configurations.

- It may be necessary to create a copy of an existing database on the same computer for testing purposes. In this case, the database files for the original database already exist, so different file names need to be specified when the database copy is created during the restore operation.

## Changing the Database Name

The name of the database can be changed as it is restored to the destination computer, without having to restore the database first and then change the name manually. For example, it may be necessary to change the database name from **Sales** to **SalesCopy** to indicate that this is a copy of a database.

The database name explicitly supplied when restoring a database is used automatically as the new database name. Because the database name does not already exist, a new one is created using the files in the backup.

## Database Ownership

When a database is restored onto another computer, the SQL Server login or Windows NT® 4.0 or Windows® 2000 user who initiates the restore operation becomes the owner of the new database automatically. When the database is restored, the system administrator or the new database owner can change database ownership. To prevent unauthorized restores of a database, use media or backup set passwords. For more information, see Password Protection.

## Restoring Full-Text Index Data

If the database being copied contains tables that have been defined for full-text indexing, then the destination computer must also have Full-Text Search installed and the MSSearch Service started before the full-text catalogs can be re-created and repopulated.

Because the meta data for the full-text index definitions is stored in the system tables of a database, it is useful to know in advance whether any of the full-text catalogs on the source computer resided on drives and directories other than the default. These directories or drive mappings may not exist on the destination computer and must be created first. To view the locations of the full-text catalog(s) on the source computer, execute the **sp_help_fulltext_catalogs** system stored procedure. The PATH column value is the location where the full-text catalog will be re-created on the destination computer. If the PATH column value of the result set is NULL, then this denotes the default full-text catalog location.

### To view the data and log files in a backup set

⊞ Transact-SQL

## Administering SQL Server

# Copying Databases from Earlier Versions of SQL Server

In Microsoft® SQL Server™ 2000, you can restore a database backup created using SQL Server version 7.0. You can also use the Copy Database Wizard to copy databases from SQL Server 7.0. For more information, see Using the Copy Database Wizard.

However, backups of the **master**, **model** and **msdb** created using SQL Server 7.0 cannot be restored by SQL Server 2000. Also, it is not possible to restore a database backup created using SQL Server version 6.5 or earlier. Database backups created using SQL Server 6.5 or earlier are in a format incompatible with SQL Server 2000.

You can, however, convert a database created using SQL Server 6.5 or earlier to SQL Server 2000 by doing one of the following:

- Upgrading to SQL Server 2000.

  Any databases are upgraded automatically. New backups from the upgraded computer running SQL Server can now be restored into another computer running an instance of SQL Server 2000.

- Using the Data Transformation Services (DTS) Import/Export Wizard to copy data between multiple instances of SQL Server.

- Using the **bcp** utility to copy data from a computer running an instance of SQL Server 6.5 or earlier to a data file, and then copy the data from the data file into a computer running an instance of SQL Server 2000.

## See Also

DTS Import/Export Wizard

Importing and Exporting Data

Upgrading Databases from SQL Server 6.5 (Upgrade Wizard)

## Administering SQL Server

## Using the Copy Database Wizard

The Copy Database Wizard allows you to copy or move databases between servers. You can move and copy databases between different instances of Microsoft® SQL Server™ 2000, and you can upgrade databases from SQL Server version 7.0 to SQL Server 2000. For more information, see Database Upgrade from SQL Server 7.0 (Copy Database Wizard).

### To upgrade databases online using the Copy Database Wizard

# Administering SQL Server

## Managing Servers

Microsoft® SQL Server™ server management comprises a wide variety of administration tasks, including:

- Registering servers and assigning passwords.
- Reconfiguring network connectivity.
- Configuring linked servers, which allows you to execute distributed queries and distributed transactions on OLE DB data sources across the enterprise.
- Configuring remote servers, which allows you to use one instance of SQL Server to execute a stored procedure residing on another instance of SQL Server.
- Configuring standby servers.
- Setting server configuration options.
- Managing SQL Server messages.
- Setting the polling intervals.

In most cases, you do not need to reconfigure the server. The default settings for the server components, configured during SQL Server Setup, allow you to run SQL Server immediately after it is installed. However, server management is necessary in those situations where you want to add new servers, set up special server configurations, change the network connections, or set server configuration options to improve SQL Server performance.

# Administering SQL Server

## Registering Servers

You must register a local or remote server before you can administer and manage it by using SQL Server Enterprise Manager. When you register a server, you must specify:

- The name of the server.
- The type of security used to log on to the server.
- Your login name and password, if appropriate.
- The name of the group where you want the server to be listed after it is registered.

You also can optionally display the Microsoft® SQL Server™ state in the console, start an instance of SQL Server automatically, or show system databases and system objects. The first two options are selected by default when you register a server.

When you run SQL Server Enterprise Manager for the first time, it automatically registers all instances of a local SQL Server. However, if you have one instance of SQL Server registered, and then install more instances of SQL Server, only the original instance of SQL Server will be registered. You can launch the Register Server Wizard or use the **Registered SQL Server Properties** dialog box to register additional servers. The **Registered SQL Server Properties** dialog box is populated with the names of all local instances of SQL Server.

If you have difficulty connecting to the remote server, you can use the Client Network Utility to configure access to the server.

### To register a server

# Administering SQL Server

## Creating Server Groups

You can create a server group within SQL Server Enterprise Manager and place your server within the server group. Server groups provide a convenient way to organize a large number of servers into a few manageable groups.

### To create server groups

# Administering SQL Server

## Accessing Server Registration Options

Microsoft® SQL Server™ allows you to maintain shared or private registry information. Shared registry information allows multiple users to use the same configuration from the same local computer or from a central computer. Alternatively, private registry information prevents others from gaining access to your configuration.

You can access server registration information in two ways:

- Remotely, on a central computer referred to as the central store.

  Remotely accessing the server registration information stored on a central computer allows different client computers to view shared registry information. For example, the system administrator registers servers x, y, and z to view server activity. With a central registration store, the system administrator can view all servers from any client.

  **Note** To read server registration information from a remote server, you must have servers registered and disable the **Store User Independent** option on the remote computer.

- Locally, accessing either private or shared information.

  From a local computer, the system administrator can make changes to the central store and configure SQL Server Enterprise Manager to save registration information without enabling the **Store User Independent** option. The **Store User Independent** option allows all users to share registration information. When this option is disabled, the central store maintains private registration information for each user.

### To set up a central store for server registration information

# Administering SQL Server

## Assigning an sa Password

When you install Microsoft® SQL Server™, SQL Server Setup does not assign a password to the **sa** login. Assign a password to **sa** after a server is installed.

Assign an **sa** password if the server security is set for Mixed Mode. If the server is set for Windows Authentication Mode, an **sa** password is not necessary, because **sa** is a SQL Server login.

**IMPORTANT** If you cannot provide the correct **sa** password, you must reinstall SQL Server.

The first time you log in to an instance of SQL Server, use **sa** as your login identification and no password. After you log in, change the **sa** password to prevent other users from using the **sa** permissions.

**Note** Before the **sa** password can be changed, the server must be registered to use SQL Server Enterprise Manager.

### To assign the sa password on a newly installed server

Administering SQL Server

# Managing AWE Memory

Microsoft® SQL Server™ 2000 uses the Microsoft Windows® 2000 Address Windowing Extensions (AWE) API to support very large memory sizes. SQL Server 2000 can use as much memory as Windows 2000 Advanced Server or Windows 2000 Datacenter Server allows. For more information about the AWE API, search on "awe memory" in the MSDN® Online Microsoft Web site.

**Note** This feature is available only in the SQL Server 2000 Enterprise and Developer editions.

## Using AWE Memory

To use AWE memory, you must run the SQL Server 2000 database engine under a Windows 2000 account that has been assigned the Windows 2000 **lock pages in memory** privilege.

SQL Server Setup will automatically grant the MSSQLServer service account permission to use the **Lock Page in Memory** option. If you are starting an instance of SQL Server 2000 from the command prompt using **sqlservr.exe**, you must manually assign this permission to the interactive user's account using the Windows 2000 Group Policy utility (**gpedit.msc**), or SQL Server will be unable to use AWE memory when not running as a service.

**To enable the Lock Page in Memory option**

# Administering SQL Server

## Configuring Network Connections

Server management includes reconfiguring the Microsoft® SQL Server™ server network connections. Most of the time, you do not need to change the server network connections. Only reconfigure the server connections if you need to:

- Configure an instance of SQL Server to listen on a particular network protocol.
- Use a proxy server to connect to an instance of SQL Server.
- Use a firewall system to isolate the network containing the instance of SQL Server from the rest of the Internet.

# Administering SQL Server

## Net-Libraries and Network Protocols

A matching pair of Microsoft® SQL Server™ Net-Libraries must be installed on a client and server computer to support a particular network protocol (for example, client TCP/IP Sockets Net-Library and server TCP/IP Sockets Net-Library). Some Net-Libraries, such as Named Pipes and Multiprotocol, support several network protocols.

All of the SQL Server client and server Net-Libraries are installed by SQL Server Setup. By default, during setup:

- Named Pipes and TCP/IP Sockets listen on Microsoft Windows NT® 4.0 or Windows® 2000 servers.

- TCP/IP and Shared Memory listen on Microsoft Windows 98 servers. (Shared Memory is a Net-Library used only for client/server connections on the same computer. You do not need to configure the Shared Memory Net-Library.)

After the network connections are installed and configured, SQL Server can listen on any combination of the server Net-Libraries simultaneously.

The correct network protocols should already be installed on the client and server. Network protocols are typically installed during Windows setup; they are not part of SQL Server Setup. A SQL Server Net-Library will not work unless its corresponding network protocol is already installed on both the client and server.

### Activating Server Net-Libraries after Setup

If you have installed SQL Server and want to change your server Net-Libraries, start SQL Server Network Utility. This application allows you to activate, deactivate, and reconfigure server Net-Libraries to listen for clients on their corresponding network protocols.

### Windows 98 Servers and Named Pipes

When running on Windows 98, SQL Server does not support the server Named

Pipes Net-Library. If you are using a Windows 98 server to run SQL Server, either the default Net-Library for the client must be changed to TCP/IP Sockets or Multiprotocol, or a new configuration entry must be created on the client that uses one of those Net-Libraries.

## Configuring Clients

After activating the appropriate server Net-Library for a network protocol, you must configure any clients accessing the server through that network protocol. Use Client Network Utility to:

- Set up a new configuration entry to connect to that specific server.

- Change the default Net-Library used by the client to support the Net-Library you just configured on the server; however, the client Net-Library you select becomes the default Net-Library for all connections from that client.

## See Also

Client Network Utility

Communication Components

Configuring Client Network Connections

Managing Clients

Client Net-Libraries and Network Protocols

SQL Server Network Utility

# Administering SQL Server

## SQL Server Network Utility

In most cases, you do not need to reconfigure Microsoft® SQL Server™ to listen on additional server Net-Libraries. However, if your server uses a network protocol on which SQL Server, by default, is not listening (for example, if your server is using NWLink IPX/SPX), and the SQL Server server Net-Library for that protocol is not activated to listen for SQL Server clients, you must use SQL Server Network Utility.

Although no server Net-Library configuration actions are necessary to enable SQL Server applications to connect to any instance of SQL Server, you can do the following:

- Manage the server Net-Library properties for each instance of SQL Server on a database computer.

- Enable the server protocols on which the instance of SQL Server will listen. For example, enable the protocol for VIA (Virtual Interface Architecture). This protocol provides highly reliable and efficient data transfer, when used with specific hardware. For VIA to work, you must use the supported hardware (Giganet). VIA is not available for systems running Microsoft Windows® 98. For more information about VIA, see VIA Clients.

- Disable a server protocol that is no longer needed.

- Specify or change the network address on which each enabled protocol will listen.

  When you are entering network addresses manually on a computer running multiple instances of SQL Server, you must not duplicate network addresses between instances. You can specify a comma-separated list of port addresses for the TCP/IP protocol. If you specify a list of port addresses, the instance of SQL Server will listen on those ports on each IP address available on the computer running the instance.

If the instance is running on a SQL Server 2000 failover cluster, it will listen on those ports on each IP address selected for SQL Server during SQL Server setup.

- Enable the Secure Sockets Layer (SSL) encryption for all of the enabled server protocols. The encryption is turned on or off for the entire enabled server protocols and you cannot specify encryption for a specific protocol. For more information about SSL encryption, see Net-Library Encryption.

  To use SSL encryption, you must install a certificate using the fully qualified domain name of the computer running the instance of SQL Server 2000. For more information about certificates, see the Windows 2000 documentation.

- Enable a WinSock proxy. For more information about setting up a proxy server, see Connections to SQL Server Through Proxy Server.

SQL Server Network Utility automatically detects if the instance of SQL Server you specify is on a failover cluster. If the instance is on a failover cluster, all of the information you specify for the instance is replicated to all nodes automatically. However, if you want to use encryption with a failover cluster, you must install the server certificate with the fully qualified DNS name of the virtual server on all nodes in the failover cluster. For example, if you have a two-node cluster, with nodes named test1.redmond.corp.microsoft.com and test2.redmond.corp.microsoft.com and a virtual SQL Server "Virtsql", you need to get a certificate for "virtsql.redmond.corp.microsoft.com" and install the certificate on both nodes. You can then check the **Force protocol encryption** check box on the Server Network Utility to configure your failover cluster for encryption.

Use Client Network Utility to configure the corresponding client Net-Libraries to any server Net-Libraries you activate.

## To start the SQL Server Network Utility

# Administering SQL Server

## Connections to SQL Server Through Proxy Server

You can connect to an instance of Microsoft® SQL Server™ through Microsoft Proxy Server, a stand-alone program that provides secured access to data. Thus, you can prevent unauthorized users from connecting to your private network. This keeps your sensitive data secure by controlling all the permissions and accesses to the listening port. Microsoft Proxy Server is integrated with Microsoft Windows® 2000 Server user authentication. You can block access to restricted sites by ranges of IP addresses, domains, or individual users so you can ensure that your users are using their Internet permissions appropriately.

For more information about Local Address Table (LAT) configuration in the context of remote listen and accept calls, see the Microsoft Proxy Server documentation.

### To connect to an instance of SQL Server through Microsoft Proxy Server

# Administering SQL Server

# Connections to SQL Server Over the Internet

You can connect to an instance of Microsoft® SQL Server™ over the Internet using SQL Query Analyzer or a client application based on ODBC or DB-Library.

To share data over the Internet, the client and server must be connected to the Internet. In addition, you must use the TCP/IP or Multiprotocol Net-Libraries. If you use the Multiprotocol Net-Library, ensure that TCP/IP support is enabled. If the server is registered with Domain Name System (DNS), you can connect using its registered name.

Although this connection is less secure than a Microsoft Proxy Server connection, using a firewall or an encrypted connection will help keep sensitive data secure.

## Using a Firewall System with SQL Server

Many companies use a firewall system to isolate their networks from unplanned access from the Internet. A firewall can be used to restrict Internet applications access to your network by forwarding only requests targeted at specific TCP/IP addresses in the local network. Requests for all other network addresses are blocked by the firewall. You can allow Internet applications to access an instance of SQL Server in the local network by configuring the firewall to forward network requests that specify the network address of the instance of SQL Server.

To work effectively with a firewall, you must ensure that the instance of SQL Server always listens on the network address that the firewall is configured to forward. The TCP/IP network addresses for SQL Server are comprised of two parts: an IP address associated with one or more network cards in a computer, and a TCP port address specific to an instance of SQL Server. Default instances of SQL Server use TCP port 1433 by default. Named instances, however, dynamically assign an unused TCP port number the first time the instance is started. The named instance can also dynamically change it's TCP port address on a subsequent startup if the original TCP port number is being used by another application. SQL Server only dynamically changes to an unused TCP port if the port it is currently listening on was dynamically selected. That is, if the port was statically selected (manually), SQL Server will display an error and continue to listen on other ports. It is unlikely another application would attempt to use 1433 since that port is registered as a well-known address for SQL Server.

When using a named instance of SQL Server with a firewall, use the Server Network Utility to configure the named instance to listen on a specific TCP port. You must pick a TCP port that is not used by another application running on the same computer or cluster. For a list of well-known ports registered for use by various applications, see http://www.ise.edu/in-notes/iana/assignments/port-numbers.

Have the network administrator configure the firewall to forward the IP address and TCP port the instance of SQL Server is listening on (using either 1433 for a default instance, or the TCP port you configured a named instance to listen on). Also configure the firewall to forward requests for UDP port 1434 on the same IP address. SQL Server 2000 uses UDP port 1434 to establish communications links from applications.

For example, consider a computer running one default instance and two named instances of SQL Server. The computer is configured such that the network addresses that the three instances listen on all have the same IP address. The default instance would listen on TCP port 1433, one named instance could be assigned TCP port 1434, and the other named instance TCP port 1954. You would then configure the firewall to forward network requests for UDP port 1434 and TCP ports 1433, 1434, and 1954 on that IP address.

## Establishing an Encrypted Connection

If you want users to be able to establish an encrypted connection to an instance of SQL Server, you can do so by enabling encryption for the Multiprotocol Net-Library.

### To enable encryption after SQL Server has been installed

# Administering SQL Server

# Configuring Linked Servers

A linked server configuration allows Microsoft® SQL Server™ to execute commands against OLE DB data sources on different servers. Linked servers offer these advantages:

- Remote server access.
- The ability to issue distributed queries, updates, commands, and transactions on heterogeneous data sources across the enterprise.
- The ability to address diverse data sources similarly.

## Linked Server Components

A linked server definition specifies an OLE DB provider and an OLE DB data source.

An OLE DB provider is a dynamic-link library (DLL) that manages and interacts with a specific data source. An OLE DB data source identifies the specific database accessible through OLE DB. Although data sources queried through linked server definitions are usually databases, OLE DB providers exist for a wide variety of files and file formats, including text files, spreadsheet data, and the results of full-text content searches. The following table shows examples of the most common OLE DB providers and data sources for SQL Server.

| **OLE DB provider** | **OLE DB data source** |
|---|---|
| Microsoft OLE DB Provider for SQL Server | Instance of SQL Server (in the form servername\instancename) and database, such as **pubs** or **Northwind** |
| Microsoft OLE DB Provider for Jet | Path name of .mdb database file |
| Microsoft OLE DB Provider for ODBC | ODBC data source name (pointing to a particular database) |
| Microsoft OLE DB Provider for Oracle | SQL*Net alias that points to an Oracle database |

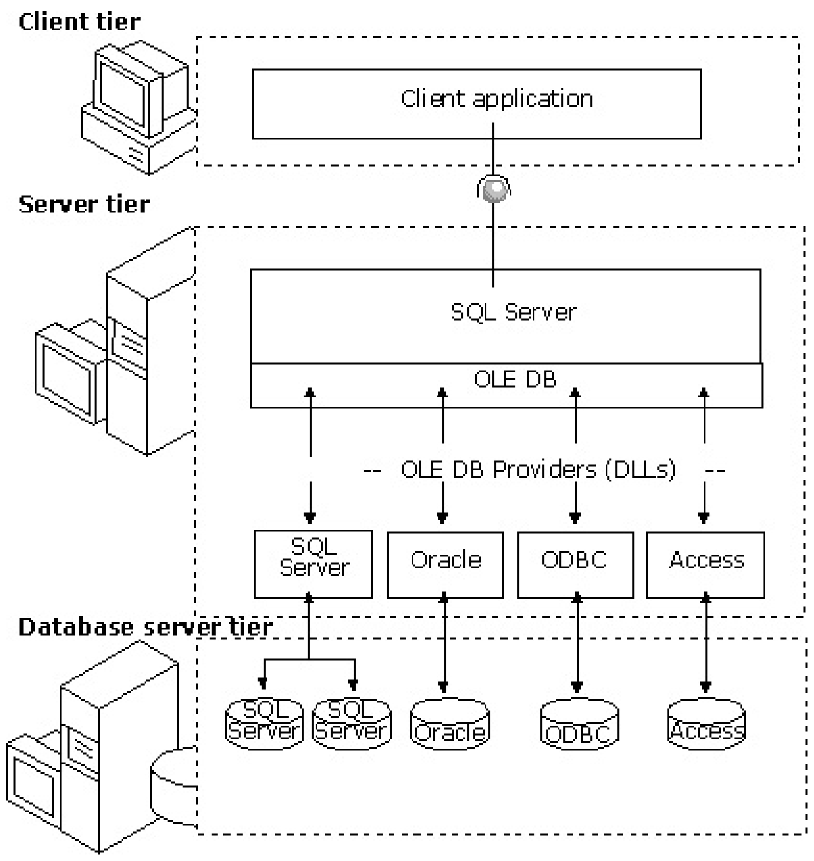
|   |   |
|---|---|
| Microsoft OLE DB Provider for Indexing Service | Content files on which property searches or full-text searches can be run |

**Note** SQL Server has been tested only against the Microsoft OLE DB Provider for SQL Server, Microsoft OLE DB Provider for Jet, Microsoft OLE DB Provider for Oracle, Microsoft OLE DB Provider for Indexing Service, and the Microsoft OLE DB Provider for ODBC. However, SQL Server distributed queries are designed to work with any OLE DB provider that implements the requisite OLE DB interfaces.

For a data source to return data through a linked server, the OLE DB provider (DLL) for that data source must be present on the same server as SQL Server.

## Linked Server Details

This illustration shows the basics of how a linked server configuration functions.



Typically, linked servers are used to handle distributed queries. When a client application executes a distributed query through a linked server, SQL Server breaks down the command and sends rowset requests to OLE DB. The rowset request may be in the form of executing a query against the provider or opening a base table from the provider.

## Managing a Linked Server Definition

When setting up a linked server, register the connection information and data source information with SQL Server. After registration is accomplished, that data source can always be referred to with a single logical name.

You can create or delete a linked server definition with stored procedures or through SQL Server Enterprise Manager.

- With stored procedures:
  - Create a linked server definition using **sp_addlinkedserver**. To view information about the linked servers defined in a given instance of SQL Server, use **sp_linkedservers**. For more information, see sp_addlinkedserver and sp_linkedservers.

  - Delete a linked server definition using **sp_dropserver**. You can also use this stored procedure to remove a remote server. For more information, see sp_dropserver.
- With SQL Server Enterprise Manager:
  - Create a linked server definition using the SQL Server Enterprise Manager console tree and the **Linked Servers** node (under the **Security** folder). Define the name, provider properties, server options, and security options for the linked server. For more information about the various ways a linked server can be set up for different OLE DB data sources and the parameter values to be used, see sp_addlinkedserver.

  - Edit a linked server definition by right-clicking the linked server and clicking **Properties**.

- Delete a linked server definition by right-clicking the linked server and clicking **Delete**.

When executing a distributed query against a linked server, include a fully qualified, four-part table name for each data source to query. This four-part name should be in the form *linked_server_name.catalog.schema.object_name*. For more information, see Distributed Queries.

## See Also

Identifying a Data Source Using a Linked Server Name

OLE DB Providers Tested with SQL Server

Using Transactions with Distributed Queries

# Administering SQL Server

## Establishing Security for Linked Servers

During a linked server connection (for example, when processing a distributed query), the sending server provides a login name and password to connect to the receiving server on its behalf. For this connection to work, create a login mapping between the linked servers using Microsoft® SQL Server™ stored procedures.

Linked server login mappings can be added using **sp_addlinkedsrvlogin** and removed using **sp_droplinkedsrvlogin**. A linked server login mapping establishes a remote login and remote password for a given linked server and local login. When SQL Server connects to a linked server in order to execute a distributed query or a stored procedure, it looks for any login mappings for the current login that is executing the query of the procedure. If there is one, it sends the corresponding remote login and password while connecting to the linked server.

Consider a mapping for a linked server, S1, that has been set up from a local login, U1, to remote login, U2, using a remote password of "my_pwd". When local login U1 executes a distributed query that accesses a table stored in linked server S1, U2 and "my_pwd" are passed as the user ID and password when SQL Server connects to the linked server S1.

For example, a mapping for a linked server, S1, has been set up for a local login, U1, to remote login, U2, using a remote password of "my pwd". When local login U1 executes a distributed query that accesses a table stored in linked server S1, U2 and "my pwd" are passed as the user ID and password when SQL Server connects to the linked server S1.

The default mapping for a linked server configuration is to emulate the current security credentials of the login. This type of mapping is known as self mapping. When a linked server is added using **sp_addlinkedserver**, a default self mapping is added for all local logins.

If security account delegation is not available on the client or sending server, or the linked server/provider does not recognize Windows Authentication Mode, then self mapping will not work for Windows Authenticated logins. Therefore, you need to set up a local login mapping from a Windows Authenticated login to a specific login on the linked server. In this case, the remote login will be a SQL Server Authenticated login if the linked server is an instance of SQL Server.

If security account delegation is available and the linked server supports Windows Authentication, then the self mapping for the Windows Authenticated logins will be supported. For more information about security account delegation, see Security Account Delegation.

Distributed queries are subject to the permissions granted to the remote login by the linked server on the remote table. While processing a distributed query, SQL Server does not perform any permission validation at compilation time. Any permission violations are detected at query execution time as reported by the provider.

### To add a linked server login

⊞ Transact-SQL

# Administering SQL Server

# Configuring OLE DB Providers for Distributed Queries

Microsoft® SQL Server™ provides a number of advanced options for managing distributed queries. Some of the options are managed at the provider level in the Microsoft Windows® 2000 registry, and others are managed at the linked server level through **sp_serveroption**. Configuring these options should be undertaken only by experienced system administrators in the interests of maximizing the performance of distributed queries against linked servers.

## OLE DB Provider Options

The OLE DB provider options for managing distributed queries are set using SQL Server Enterprise Manager. In the left pane of SQL Server Enterprise Manager, right-click a linked server definition that uses the OLE DB provider for which you want to set the properties. On the **General** tab, click **Options**, and then set the properties.

| Provider option | Description |
|---|---|
| **DynamicParameters** | If nonzero, indicates that the provider allows '?' parameter marker syntax for parameterized queries. Set this option only if the provider supports the **ICommandWithParameters** interface and supports a '?' as the parameter marker. Setting this option allows SQL Server to execute parameterized queries against the provider. The ability to execute parameterized queries against the provider can result in better performance for certain queries. |
| **NestedQueries** | If nonzero, indicates that the provider allows nested SELECT statements in the FROM clause. Setting this option allows SQL Server to delegate certain queries to the provider that require nesting SELECT statements in the FROM clause. |
| **LevelZeroOnly** | If nonzero, only level 0 OLE DB interfaces are |

| | |
|---|---|
| | invoked against the provider. |
| **AllowInProcess** | If nonzero, SQL Server allows the provider to be instantiated as an in-process server. When this option is not set in the registry, the default behavior is to instantiate the provider outside the SQL Server process. Instantiating the provider outside the SQL Server process protects the SQL Server process from errors in the provider. When the provider is instantiated outside the SQL Server process, updates or inserts referencing long columns (**text**, **ntext**, or **image**) are not allowed. |
| **NonTransactedUpdates** | If nonzero, SQL Server allows updates, even if **ITransactionLocal** is not available. If this option is enabled, updates against the provider are not recoverable, because the provider does not support transactions. |
| **IndexAsAccessPath** | If nonzero, SQL Server attempts to use indexes of the provider to fetch data. By default, indexes are used only for meta data and are never opened. |
| **DisallowAdhocAccess** | If a nonzero value is set, SQL Server does not allow ad hoc access through the **OpenRowset()** and **OpenDataSource()** functions against the OLE DB provider. When this option is not set, the default behavior is to allow **OpenRowset** and **OpenDataSource**. |

These options operate at the provider level. When the options are set for a provider, the settings apply to all linked server definitions using the same OLE DB provider.

Setting either **DynamicParameters** or **NestedQueries** to nonzero values allows SQL Server to send queries requiring this syntax to the OLE DB provider for remote query execution. These two options should be set only if the provider supports their syntax.

## Linked Server Options

Several options for managing distributed queries are available at the linked server level through **sp_serveroption**. The server level options (in contrast to provider level options) only affect the behavior against the specified linked server.

The following table describes the various linked server options.

| Linked server options | Description |
|---|---|
| **use remote collation** | If set to **true**, SQL Server will use the collation information of character columns from the linked server. If the linked server is an instance of SQL Server, then the collation information is derived automatically from the SQL Server OLE-DB provider interface. If the linked server is not an instance of SQL Server, then SQL Server will use the collation set in the **collation name** option.<br><br>If set to **false**, SQL Server will interpret character data from the specified linked server in the default collation of the instance of a local SQL Server. |
| **collation name** | This specifies the collation to be used for character data from the linked server if **use remote collation** is set to **true**. This option is ignored if **use remote collation** is set to **false**, or if the linked server is an instance of SQL Server. |
| **connection timeout** | This specifies the time-out value (in seconds) to be used when SQL Server attempts to make a connection to the linked server. If this option is not set, the current value set for the global configuration option **remote login timeout** is used as the default. |
| **lazy schema validation** | If this option is set to **false** (the default value), SQL Server checks for schema changes that have occurred since compilation in remote tables. This occurs before query execution begins. If there is a change in the schema, SQL Server recompiles the query with the new schema. |

| | |
|---|---|
| | If this option is set to **true**, the checking of the schema of remote tables is delayed until execution. This can cause a distributed query to fail with an error, if the schema of a remote table has changed between the time the query was compiled and executed.<br><br>You may want to set this option to **true** when distributed, partitioned views are being used against a linked SQL Server. A given table participating in the partitioned view may not be actually used in a given execution of a query against the view, so deferring the schema validation can be useful to improve performance. |

## See Also

Establishing Security for Linked Servers

sp_addlinkedserver

sp_serveroption

# Administering SQL Server

# Configuring Remote Servers

A remote server configuration allows a client connected to one instance of Microsoft® SQL Server™ to execute a stored procedure on another instance of SQL Server without establishing another connection. The server to which the client is connected accepts the client request and sends the request to the remote server on behalf of the client. The remote server processes the request and returns any results to the original server, which in turn passes those results to the client.

If you want to set up a server configuration in order to execute stored procedures on another server and do not have existing remote server configurations, use linked servers instead of remote servers. Both stored procedures and distributed queries are allowed against linked servers; however, only stored procedures are allowed against remote servers.

**Note** Support for remote servers is provided for backward compatibility only. New applications that must execute stored procedures against remote instances of SQL Server should use linked servers instead.

## Remote Server Details

Remote servers are set up in pairs. To set up a pair of remote servers, configure both servers to recognize each other as remote servers. Then, verify that configuration options are set properly for both servers so that each instance of SQL Server allows remote users to execute procedure calls. Check the configuration options in the **Server Properties** dialog box on both the local and the remote servers.

In most cases, you should not need to set configuration options for remote servers; the defaults set on both local and remote computers by SQL Server Setup allow for remote server connections.

For remote server access to work, the **remote access** configuration option, which controls logins from remote servers, must be set to 1 (the default setting) on both the local and remote computers. If the setting for either server's **remote access** option has been changed, you must reset the option (for one or both servers) back to 1 to allow remote access. This can be accomplished through either SQL

Server Enterprise Manager or the Transact-SQL **sp_configure** statement.

From the local server, you can disable a remote server configuration to prevent user access to that server.

### To set up a remote server

## Administering SQL Server

## Establishing Security for Remote Servers

Setting up security for executing remote procedure calls (RPC) against a remote server involves setting up login mappings in the remote server and possibly in the local server running an instance of Microsoft® SQL Server™.

**Note** Support for remote servers is provided for backward compatibility only. New applications that must execute stored procedures against remote instances of SQL Server should use linked servers instead.

### Setting Up the Remote Server

Remote login mappings must be set up on the remote server. Using these mappings, the remote server maps the incoming login for an RPC connection from a given server to a local login. Remote login mappings can be set up using the **sp_addremotelogin** stored procedure on the remote server.

### Setting Up the Local Server

In SQL Server 2000, create remote server connections for remote server logins created by Windows Authentication by:

- Setting up a local login mapping on a local server that defines what login and password are used by an instance of SQL Server when it makes an RPC connection to a remote server.

  For logins created by Windows Authentication, you must create a mapping to a login name and password. This login name and password must match the incoming login and password expected by the remote server.

- Using the **sp_addlinkedsrvlogin** stored procedure to create local login mappings.

  **Note** For logins created by SQL Server Authentication, it is not necessary to create any local login mappings for executing a stored procedure against a remote server.

## Remote Server Security Example

Consider two SQL Server installations, *serverSend* and *serverReceive*. *serverReceive* is configured to map an incoming login from *serverSend*, called Sales_Mary, to a SQL Server authenticated login in *serverReceive*, called Alice. Another incoming login from *serverSend*, called Joe, is mapped to a SQL Server Athenticated login in *serverReceive*, called Joe.

The following Transact-SQL code can be executed to configure *serverSend* to perform RPCs against *serverReceive*:

```
--Create remote server entry for RPCs from serverSend.
EXEC sp_addserver 'serverSend'
GO

--Create remote login mapping for login 'Sales_Mary' from serverSend
--to Alice.
EXEC sp_addremotelogin 'serverSend', 'Alice', 'Sales_Mary'
GO

--Set trusted option on for this mapping to disable password checking
--for Sales_Mary from serverSend.
EXEC sp_remoteoption 'serverSend', 'Alice', 'Sales_Mary', trusted, tru
GO

--Create remote login mapping for login Joe from serverReceive to san
--assumes same password for Joe in both servers.
EXEC sp_addremotelogin 'serverSend', 'Joe', 'Joe'
GO
```

On *serverSend*, a local login mapping is created for a Windows Authenticated login Sales\Mary to a login Sales_Mary. No local mapping is necessary for Joe, as the default is to use the same login name and password, and *serverReceive* has a mapping for Joe:

```
--Create a remote server entry for RPCs from serverReceive.
```

```
EXEC sp_addserver 'serverReceive'
GO

--Create a local login mapping for the Windows Authenticated login.
--Sales\Mary to Sales_Mary.
EXEC sp_addlinkedsrvlogin 'serverReceive', false, 'Sales\Mary',
   'Sales_Mary,' NULL
GO
```

## See Also

Configuring Remote Servers

sp_addremotelogin

sysremotelogins

Administering SQL Server

# Viewing Local or Remote Server Properties

You can review server attributes for local or remote servers (such as the Microsoft® SQL Server™ version number, type and number of processors in the computer, and the operating system version) in one convenient location. From the local server, you can view databases, files, logins, and tools for a remote server.

### To view server properties

# Administering SQL Server

## Using Standby Servers

A standby server is a second server that can be brought online if the primary production server fails. The standby server contains a copy of the databases on the primary server. A standby server can also be used when a primary server becomes unavailable due to scheduled maintenance. For example, if the primary server needs a hardware or software upgrade, the standby server can be used.

A standby server allows users to continue working with databases if the primary server becomes unavailable. When the primary server becomes available again, any changes to the standby server's copies of databases must be restored back to the primary server. Otherwise, those changes are lost. When users start using the primary server again, its databases should be backed up and restored on the standby server again.

Implementing a standby server involves these phases:

- Creating the database and ongoing transaction log backups on the primary server.
- Setting up and maintaining the standby server by backing up the database on the primary server and restoring them on the standby server.
- Bringing the standby server online if the primary server fails.

  **Important** All user processes must log in to the standby server and restart any tasks they were performing when the primary server became unavailable. User processes are not switched automatically to the standby server and transactions are not maintained between the primary server and the standby server. If the primary server is taken off the network or renamed manually, and the standby server is renamed, then the standby server will have a network name and address different from the server the users were using previously.

Periodically, transaction log backups from the databases on the primary server are applied on the standby to ensure that the standby remains synchronized with the primary server. In the event of the primary server failing, or even if just a single database fails, the databases on the standby server are made available to user processes. Any user processes that cannot access the primary server should use the standby server instead.

A standby server configuration is not the same as the virtual server configuration used in Microsoft® SQL Server™ 2000 failover clustering. A standby server contains a second copy of the SQL Server databases. In a virtual server configuration, a single copy of the databases, loaded on a shared cluster disk, is shared by the primary and secondary physical servers that underlie the virtual server.

## Creating the Backups on the Primary Server

On the primary server:

1. Create a full database backup of each database to be duplicated. For more information, see Database Backups.

2. Periodically, create a transaction log backup of each database to be duplicated. For more information, see Transaction Log Backups.

   The frequency of transaction log backups created on the primary server depends on the volume of transaction changes of the production server database. If the transaction frequency is high, it may be useful to back up the transaction log frequently to minimize the potential loss of data in the event of failure.

   IMPORTANT  When restoring a copy of **master** from a production server to a standby server, you cannot back up the transaction log of **master**. Only a database backup and restore of **master** is possible.

## Setting Up and Maintaining the Standby Server

A standby server is set up and maintained as follows:

1. Restore the database backups from the primary server onto the standby server in standby mode, specifying an undo file (one undo file per database).

When a database or transaction log is restored in standby mode, recovery needs to roll back any uncommitted transactions so that the database can be left in a logically consistent state and used, if necessary, for read-only purposes. Pages in the database affected by the uncommitted, rolled back transactions are modified. This undoes the changes originally performed by the uncommitted transactions. The undo file is used to save the contents of these pages before recovery modifies them to prevent the changes performed by the uncommitted transactions from being lost. Before a subsequent transaction log backup is next applied to the database, the uncommitted transactions that were previously rolled back by recovery must be reapplied first. The saved changes in the undo file are reapplied to the database, and then the next transaction log is applied.

**Note** There must be enough disk space for the undo file to grow so that it can contain all the distinct pages from the database that were modified by rolling back uncommitted transactions.

2. Periodically, apply each subsequent transaction log, created on the primary server, to the databases on the standby server. Apply each transaction log in standby mode, specifying the same undo file used when previously restoring the database.

   The frequency of transaction log backups applied to the standby server depends on the frequency of transaction log backups of the primary production server database. Frequently applying the transaction log reduces the work required to bring the standby server online in the event of a production system failure.

In standby mode, the database is available for read-only operations, such as database queries that do not attempt to modify the database. This allows the database to be used for decision-support queries or DBCC checks.

## Bringing the Standby Server Online

When the primary server initially becomes unavailable, it is unlikely that all the databases on the standby server are in complete synchronization. Some transaction log backups created on the primary server may not have been applied to the standby server yet. Additionally, some changes to the databases on the primary server are likely to have occurred since the transaction log on those databases were last backed up, especially in heavily used systems. Before the users use the standby copies, it is possible to synchronize the primary databases with the standby copies and bring the standby server online by:

1. Applying to the standby server in sequence any transaction log backups created on the primary server that have not yet been applied.

2. Creating a backup of the active transaction log on the primary server and applying the backup to the database on the standby server. The backup of the active transaction log when applied to the standby server allows users to work with an exact copy of the primary database as it was immediately prior to failure (although any noncommitted transactions will have been permanently lost). For more information, see Transaction Log Backups.

   If the primary server is undamaged, as in the case of planned maintenance or upgrades, you can back up the active transaction log with NORECOVERY. This will leave the database in the restoring state and allow you to update the primary server with transaction log backups from the secondary server. Then you can switch back to the primary server without creating a complete database backup of the secondary server. For more information, see BACKUP.

3. Recover the databases on the standby server. This recovers the databases without creating an undo file, making the database available for users to modify.

A standby server can contain backups of databases from several instances of SQL Server. For example, a department could have five servers, each running a mission-critical database system. Rather than having five separate standby servers, a single standby server can be used. The database backups from the five primary systems can be loaded onto the single backup system, reducing the number of resources required and saving money. It is unlikely that more than one primary system would fail at the same time. Additionally, the standby server can be of higher specification than the primary servers to cover the remote chance that more than one primary system is unavailable at a given time.

### To set up, maintain, and bring online a standby server

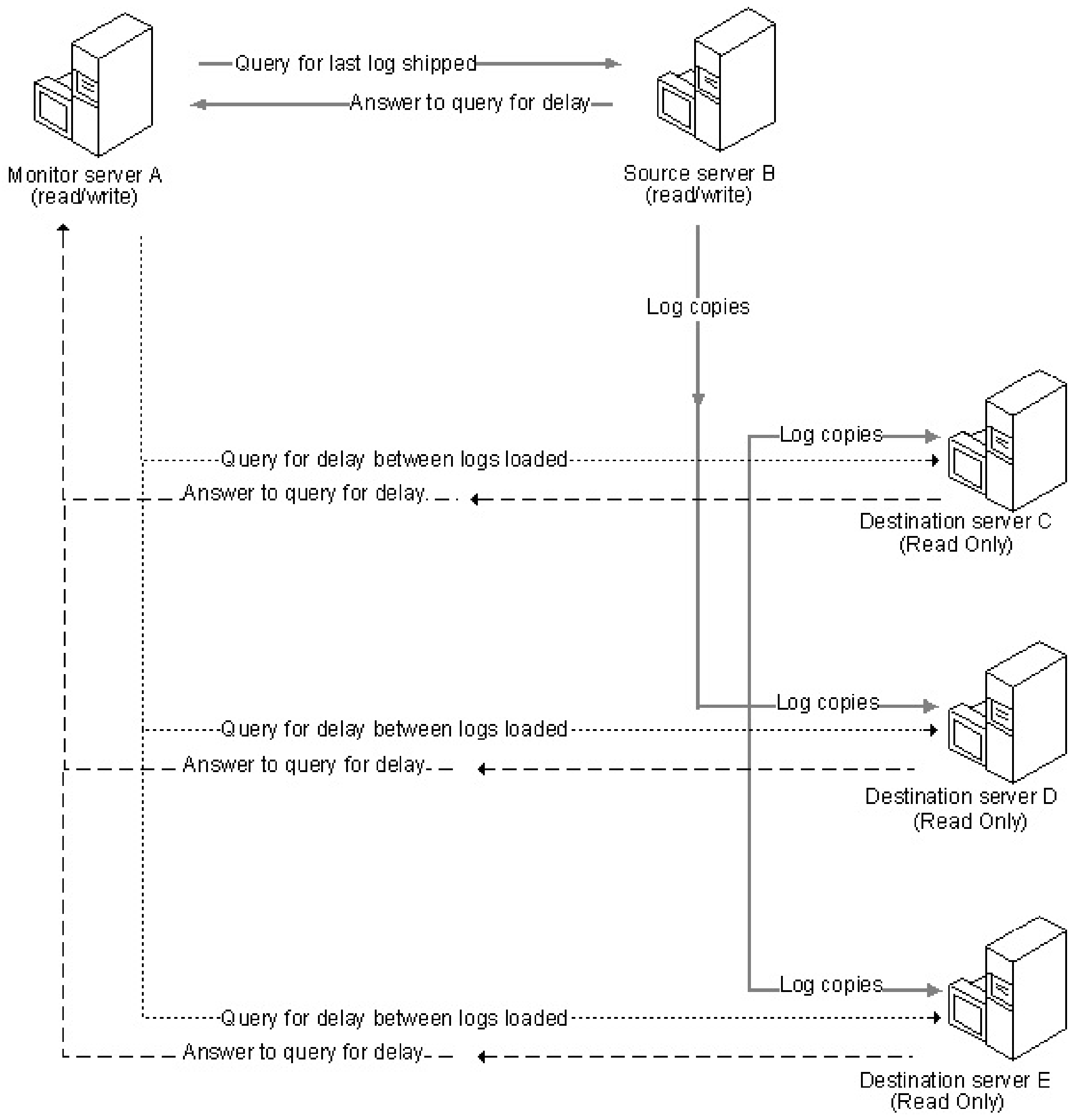
# Administering SQL Server

## Log Shipping

In Microsoft® SQL Server™ 2000 Enterprise Edition, you can use log shipping to feed transaction logs from one database to another on a constant basis. Continually backing up the transaction logs from a source database and then copying and restoring the logs to a destination database keeps the destination database synchronized with the source database. This allows you to have a backup server and also provides a way to offload query processing from the main computer (the source server) to read-only destination servers.

### Log Shipping Model

The illustration shows the log shipping model.


Query for last log shipped
Answer to query for delay
Monitor server A (read/write)
Source server B (read/write)
Log copies
Log copies
Query for delay between logs loaded
Answer to query for delay.
Destination server C (Read Only)
Log copies
Query for delay between logs loaded
Answer to query for delay.
Destination server D (Read Only)
Log copies
Query for delay between logs loaded
Answer to query for delay.
Destination server E (Read Only)

In this example, an enterprise has five servers: server **A**, server **B**, server **C**, server **D**, and server **E**. Server **B** is the source server, the server on which log backups and restores are performed and copied. Server **C**, server **D**, and server **E** contain the destination databases on which the log backups from server **B** are restored, keeping these servers in synchronization with server **B**. Server **A** is the monitor server on which the enterprise-level monitoring of log shipping occurs. Each destination or source server is maintained by only one monitor server. The Database Maintenance Plan Wizard is used to define an appropriate delay between the time server **B** backs up the log backup and the time server **C**, server **D**, and server **E** must restore the log backup. If more time elapses than defined, then server **A** generates an alert using SQL Server Agent. This alert can aid in troubleshooting the reason the destination server has failed to restore the backups.

Do not use the monitor server as the source server, because the monitor server maintains critical information regarding the log shipping system. The monitor server should be regularly backed up. Keeping the monitor server independent is also better for performance, because monitoring adds unnecessary overhead. Also, as a source server supporting a production workload, it is most likely to fail, which would disrupt the monitoring. The source and destination servers can be on the same computer. However, in this case, SQL Server 2000 failover clustering may provide better results. For more information, see Failover Clustering.

## Configuring Log Shipping with the Database Maintenance Plan Wizard

To easily configure log shipping, use the Database Maintenance Plan Wizard. With this wizard, you can:

- Define how often the logs are generated, the time between a backup and a restore operation, and when a destination server is out of synchronization with a source server.

- Register any new servers.

- Create the source databases on all destination servers. When adding a destination database through the Database Maintenance Plan Wizard, you have the option of creating the databases on the destination server or using existing databases. Any existing databases must be in standby mode before you can configure them for log shipping.

- Specify which destination servers might assume the role of the source server.

- Set a restore delay. This delay defines how old a transaction log must be before it is restored. If something goes wrong on the source server, this delay provides an extra time before the corrupted log is restored onto the destination server.

- Create a schedule that sets the backup schedule.

Before using the Database Maintenance Plan Wizard, consider the following:

- The user configuring log shipping must be a member of the **sysadmin** server role in order to have permission to modify the database to log ship.

- You can configure log shipping only on one database at a time. If you select more than one database, the log shipping option on the wizard is disabled.

- The login used to start the MSSQLServer and SQLServerAgent services must have access to the log shipping plan jobs, the source server, and the destination server.

- When you use the Database Maintenance Plan Wizard to configure log shipping, you can log ship only to disks. The backup-to-tape option is not available.

## Configuring Log Shipping Manually

SQL Server 2000 supports manual log shipping from a SQL Server version 7.0 Service Pack 2 (SP2) transaction log if the pending upgrade option is enabled on the computer running SP2.

To enable this option, execute the following code:

EXEC sp_dboption '*database name*', 'pending upgrade', 'true'

However, when you are restoring the database after log shipping, you can recover only with the NORECOVERY option.

**Note** When you manually configure log shipping between a computer running

SP2 and a computer running an instance of SQL Server 2000, you cannot use SQL Server replication.

For more information, see the SP2 documentation.

### To configure log shipping

## Modifying Log Shipping

After log shipping has been configured, it is possible to add, delete, or edit the destination servers. For example, you can change the transaction log destination, specify if you want to create a new database on the destination server, or use an existing database. If you choose to create a new database, you must specify a database name and file directories for the data and logs.

### To add or edit a destination server

## Monitoring Log Shipping

After you have configured log shipping, use the monitor server to view information about the status of all the log shipping servers.

The monitor server provides you with all of the details of log shipping, such as:

- When the source server was last backed up, and when the destination servers last copied and restored the backup files.
- Information about the backup failure alert.
- Information detailing alert generation suppression.

Using the monitor server, you can also:

- Edit the alert generation suppression information for both the source and destination servers. Alert suppression would be used to suppress an alert during specific times and dates in the event of a backup failure.
- Change the role of a server from a destination server to a source server (if the destination server was configured to assume this role).

### To view the status of servers configured for log shipping

## Administering SQL Server

## Concurrent Administrative Operations

This table illustrates the administrative tasks that are or are not allowed to run at the same time.

**Note** File shrink operations spend most processing time reallocating pages into areas retained after the shrink has completed, and then attempt to change the file size only as the last step. File shrink operations can be started while a backup is running, provided the backup finishes before the file shrink attempts to change the size of the files.

# Administering SQL Server

## Managing SQL Server Messages

Microsoft® SQL Server™ provides tools for managing server messages, allowing administrators to:

- Search for specific error messages based on filters such as message text, error number, severity level, whether the message is user-defined, and whether the message is logged.

- Create new messages.

- Edit user-defined messages.

- Delete user-defined messages.

**To add a new SQL Server message**

# Administering SQL Server

# SQL Mail

SQL Mail provides a way to receive e-mail messages generated by Microsoft® SQL Server™. Messages can be triggered to provide you with the status of a job or a warning caused by an alert. SQL Mail can include a result set in a reply to e-mail messages that contain queries. SQL Mail allows SQL Server to send and receive e-mail by establishing a client connection with a mail server.

SQL Server uses two services to handle mail. MSSQLServer processes mail for all of the mail stored procedures. SQLServerAgent does not use SQL Mail to send e-mail. Instead, SQLServerAgent uses its own mail capabilities that are configured and operated separately from SQL Mail.

The SQL Server Agent mail features will be referred to as SQLAgentMail to distinguish it from the SQL Mail features provided by MSSQLServer. SQL Mail establishes an extended MAPI connection with a mail host, while SQLAgentMail establishes an extended MAPI connection on its own. Both SQL Mail and SQLAgentMail can connect with Microsoft Exchange Server, Microsoft Windows NT® Mail, or a Post Office Protocol 3 (POP3) server.

SQL Mail requires a post office connection, a mail store (mailbox), a mail profile, and the Windows NT 4.0 or Microsoft Windows® 2000 domain user account used to log in to an instance of SQL Server. SQL Mail consists of a number of stored procedures, which are used by SQL Server to process e-mail messages that are received in the designated SQL Mail account mailbox or to reply to e-mail messages generated by the stored procedure **xp_sendmail**. Using SQL Mail extended stored procedures, messages can be sent from either a trigger or a stored procedure. SQL Mail stored procedures can manipulate data, process queries received by e-mail and return the result set by creating a reply e-mail.

## Processing an E-mail Request Received by SQL Server

To process e-mail automatically, you must create a regularly scheduled job that uses the stored procedure, **sp_processmail**. **sp_processmail** checks your SQL Mail mail profile and then checks your mailbox for mail. **sp_processmail** uses **xp_sendmail** to execute query requests contained in the text of the e-mail and then returns the result set to the original sender and any additional recipients. For example, a supplier may be allowed to execute a stored procedure that produces current inventory levels for all materials supplied by the organization.

## SQLAgentMail

SQLAgentMail can use its own domain account and mail profile that is different from the one set up for SQL Mail. With SQL Server, you can configure SQLAgentMail e-mail messages to be sent when:

- An alert is triggered.

  Alerts can be configured to send e-mail notification of specific events that occur without implementing SQL Mail. For example, alerts can be configured to notify an operator of a particular database event that may need immediate action.

  For more information about configuring alerts, see Defining Alerts.

- A scheduled task, such as a database backup or replication event, succeeds or fails.

E-mail messages can be sent to a list of recipients informing them of the status of scheduled jobs for possible user action. You can expand the capabilities of jobs to include sending a result set by e-mail to a list of recipients. For example, a monthly inventory report could send SQLAgentMail notification to the designated operators and the result set to the purchasing manager and supplier.

# Administering SQL Server

## Configuring SQL Mail

SQL Mail must run using a mail profile created in the same domain account that is used to start an instance of Microsoft® SQL Server™. Under the **Support Services** folder in SQL Server Enterprise Manager, you can see a graphical depiction of the SQL Mail Service and determine if the service is running. You can start SQL Mail automatically by clicking **Autostart SQL Mail when SQL Server Starts** on the **General** tab of the **SQL Mail Configuration** dialog box. After SQL Mail starts, you can use the stored procedures to send and receive mail.

### To configure a mail profile

# Administering SQL Server

# Configuring Mail Profiles

SQL Mail and SQLAgentMail can use the same or different mail profile. If necessary, each mail profile can be configured within its own domain account.

## Configuring a SQL Mail Profile

When configured, mail profiles are specific to the Microsoft® Windows NT® 4.0 or Windows® 2000 user domain account that is activated when a user logs on to Windows NT 4.0 or Windows 2000 successfully. SQL Mail must have a mail profile created in the same user domain account or context that is used to start an instance of Microsoft SQL Server™. When a mail stored procedure is executed, SQL Mail looks for the defined mail profile in the domain account that triggered it.

If you plan to use mail stored procedures you must:

- Have a mail server that is extended MAPI-compliant.

- Configure a mail profile for MSSQLServer to use to connect to your mail server.

**To configure a mail profile**

# Administering SQL Server

## Using SQL Mail Stored Procedures

SQL Mail contains a number of stored procedures, which allow you to develop triggers, applications, and other stored procedures. The stored procedures can then be used to manipulate mail, run queries, return a result set to a list of recipients, or reply to an e-mail containing a simple query or stored procedure.

The following table provides a brief description of the extended procedures and how they can be used.

| SQL Mail procedures | Function |
|---|---|
| xp_startmail | Starts a mail client session. The mail client session must be started prior to using any of the other mail stored procedures. |
| xp_stopmail | Closes a Microsoft® SQL Server™ mail client session. |
| xp_findnextmsg | Used with **sp_processmail** in order to process mail in the SQL Mail inbox by accepting a message ID for input and returning the message ID for output. |
| xp_readmail | Used by **sp_processmail** to read a mail message from the SQL Mail inbox. |
| xp_deletemail | Used by **sp_processmail** to delete a message from the SQL Mail inbox. |
| xp_sendmail | Used by **sp_processmail** or as part of a stored procedure or trigger. Can be used with alerts. Sends a message and a query result set attachment to the specified recipients. |
| sp_processmail | Uses extended stored procedures (**xp_findnextmessage**, **xp_readmail**, and **xp_deletemail**) to process incoming mail messages (expected to be a single query only) and uses **xp_sendmail** to return the result set to the message sender. **sp_processmail** must be set up as a regularly scheduled job to check for mail received in the SQL Mail inbox. |

## To use SQL Mail

Administering SQL Server

# Setting Configuration Options

You can manage and optimize Microsoft® SQL Server™ resources through configuration options by using SQL Server Enterprise Manager or the **sp_configure** system stored procedure. The most commonly used server configuration options are available through SQL Server Enterprise Manager; all configuration options are accessible through **sp_configure**. Consider the effects on your system carefully before setting these options.

**Important** Advanced options are those that should be changed only by a experienced system administrator or certified SQL Server technician.

## Using the sp_configure System Stored Procedure

When using **sp_configure,** you must run either RECONFIGURE or RECONFIGURE WITH OVERRIDE after setting a configuration option. The RECONFIGURE WITH OVERRIDE statement is usually reserved for configuration options that should be used with extreme caution (for example, setting the **allow updates** option to 1 allows users to update fields in system tables). However, RECONFIGURE WITH OVERRIDE works for all configuration options, and you can use it in place of RECONFIGURE.

The following is an example of a script you would use with **sp_configure** to change the **fill factor** option from its default setting to a value of 100:

```
sp_configure 'fill factor', 100
GO
RECONFIGURE
GO
```

## Categories of Configuration Options

Configuration options either take effect either:

- Immediately after setting the option and issuing the RECONFIGURE (or in some cases, RECONFIGURE WITH OVERRIDE) statement.

  -or-

- After doing these actions and stopping and restarting an instance of SQL Server.

To configure an advanced option with **sp_configure**, you must first run **sp_configure** with the **show advanced options** option set to 1, and then run RECONFIGURE:

```
sp_configure 'show advanced options', 1
GO
RECONFIGURE
GO
sp_configure 'cursor threshold', 0
GO
RECONFIGURE
GO
```

In the previous example, reconfiguring the **cursor threshold** option to a new value takes place immediately. If you run **sp_configure** again, the new value for **resource timeout** appears in the configuration options **run_value** column.

Some options require a server stop and restart before the new configuration value takes effect. For example, you cannot configure the **affinity mask** option until you set **show advanced options** to 1, run RECONFIGURE, and stop and restart the server. If you set the new value and run **sp_configure** before stopping and restarting the server, the new value appears in the configuration options **config_value** column, but not in the **run_value** column. After stopping and restarting the server, the new value appears in the **run_value** column.

If you use SQL Server Enterprise Manager to change a configuration option, and the configuration option requires a server stop and restart to take effect, SQL Server displays a dialog box asking if you want to stop and restart the server.

Self-configuring options are those that SQL Server adjusts according to the needs of the system. In most cases, this eliminates the need for setting the values manually. Examples include the **min server memory** and **max server memory** options, and the **user connections** option.

## Configuration Options Table

The following table lists all available configuration options, the range of possible settings, and default values. Letter codes next to a configuration option indicate:

- Advanced options (those that should be changed only by a certified SQL Server technician, and require setting **show advanced options** to 1), marked with "A."

- Options requiring a server restart to take effect, marked with "RR."

- Self-configuring options (those that SQL Server self-configures, depending on the needs of the system), marked with "SC."

| Configuration option | Minimum | Maximum | Default |
|---|---|---|---|
| affinity mask (A, RR) | 0 | 2147483647 | 0 |
| allow updates | 0 | 1 | 0 |
| awe enabled (A, RR) | 0 | 1 | 0 |
| c2 audit mode (A, RR) | 0 | 1 | 0 |
| cost threshold for parallelism (A) | 0 | 32767 | 5 |
| cursor threshold (A) | –1 | 2147483647 | -1 |
| default full-text language (A) | 0 | 2147483647 | 1033 |
| default language | 0 | 9999 | 0 |
| fill factor (A, RR) | 0 | 100 | 0 |
| index create memory (A, SC) | 704 | 2147483647 | 0 |
| lightweight pooling (A, RR) | 0 | 1 | 0 |
| locks (A, RR, SC) | 5000 | 2147483647 | 0 |
| max degree of parallelism (A) | 0 | 32 | 0 |
| max server memory (A, SC) | 4 | 2147483647 | 2147483647 |
| max text repl size | 0 | 2147483647 | 65536 |
| max worker threads (A, RR) | 32 | 32767 | 255 |
| media retention (A, RR) | 0 | 365 | 0 |
| min memory per query (A) | 512 | 2147483647 | 1024 |
| min server memory (A, SC) | 0 | 2147483647 | 0 |
| Using Nested Triggers | 0 | 1 | 1 |

| | | | |
|---|---|---|---|
| network packet size (A) | 512 | 65536 | 4096 |
| open objects (A, RR, SC) | 0 | 2147483647 | 0 |
| priority boost (A, RR) | 0 | 1 | 0 |
| query governor cost limit (A) | 0 | 2147483647 | 0 |
| query wait (A) | -1 | 2147483647 | -1 |
| recovery interval (A, SC) | 0 | 32767 | 0 |
| remote access (RR) | 0 | 1 | 1 |
| remote login timeout | 0 | 2147483647 | 20 |
| remote proc trans | 0 | 1 | 0 |
| remote query timeout | 0 | 2147483647 | 600 |
| scan for startup procs (A, RR) | 0 | 1 | 0 |
| set working set size (A, RR) | 0 | 1 | 0 |
| show advanced options | 0 | 1 | 0 |
| two digit year cutoff | 1753 | 9999 | 2049 |
| user connections (A, RR, SC) | 0 | 32767 | 0 |
| user options | 0 | 32767 | 0 |

## See Also

sp_configure

Using Options in SQL Server

# Administering SQL Server

## affinity mask Option

In Microsoft® Windows NT® 4.0 and Windows® 2000, an activity (thread) in a process can migrate from processor to processor, with each migration reloading the processor cache. Under heavy system loads, specifying which processor should run a specific thread can improve performance by reducing the number of times the processor cache is reloaded. The association between a processor and a thread is called processor affinity.

Use the **affinity mask** option to increase performance on symmetric multiprocessor (SMP) systems (with more than four microprocessors) operating under heavy load. You can associate a thread with a specific processor and specify which processors Microsoft SQL Server™ will use. You can exclude SQL Server activity from processors given specific workload assignments by the Windows NT 4.0 or Windows 2000 operating system.

If you set a bit representing a processor to 1, that processor is selected for thread assignment. When you set **affinity mask** to 0 (the default), the Windows NT 4.0 or Windows 2000 scheduling algorithms set the thread's affinity. When you set **affinity mask** to any nonzero value, SQL Server affinity interprets the value as a bit mask that specifies those processors eligible for selection. Excluding SQL Server threads from running on particular processors helps evaluate the system's handling of processes specific to Windows NT 4.0 or Windows 2000. For example, you can use **affinity mask** to evaluate whether an additional network interface card (NIC) increases performance or assess NIC performance with increasing loads.

Because using SQL Server processor affinity is a specialized operation, it is recommended that SQL Server processor affinity be used only when necessary. In most cases, the Windows NT 4.0 or Windows 2000 default affinity provides the best performance.

Before you change the setting of **affinity mask**, keep in mind that Windows NT 4.0 and Windows 2000 assign deferred process call (DPC) activity associated with NICs to the highest numbered processor in the system. In systems with more than one installed and active NIC, each additional card's activity is assigned to the next highest numbered processor. For example, an eight-processor system with two NICs has DPCs for each NIC assigned to processor 7 and to processor 6.

**Note** You can use System Monitor (Performance Monitor in Windows NT 4.0) to view and analyze individual processor usage.

For example, if processors 1, 2, and 5 are selected as available with bits 1, 2, and 5 set to 1 and bits 0, 3, 4, 6, and 7 set to 0, a hexadecimal value of 0x26 or the decimal equivalent of 38 is specified. Number the bits from the right to left. The rightmost bit is bit 0. Set bits 1, 2, and 5 (the third, fifth, and sixth bits) to 1. The number calculated from setting the specified bits is binary 00100110, which is decimal 38 or hexadecimal 0x26.

These are **affinity mask** values for an eight-processor system.

| Decimal value | Binary bit mask | Allow SQL Server threads on processors |
|---|---|---|
| 1 | 00000001 | 0 |
| 3 | 00000011 | 0 and 1 |
| 7 | 00000111 | 0, 1, and 2 |
| 15 | 00001111 | 0, 1, 2, and 3 |
| 31 | 00011111 | 0, 1, 2, 3, and 4 |
| 63 | 00111111 | 0, 1, 2, 3, 4, and 5 |
| 127 | 01111111 | 0, 1, 2, 3, 4, 5, and 6 (isolates SQL Server activity from DPC processor only) |

**affinity mask** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **affinity mask** only when **show advanced options** is set to 1. The setting takes effect after stopping and restarting the server.

### To configure the affinity mask

# Administering SQL Server

## allow updates Option

Use the **allow updates** option to specify whether direct updates can be made to system tables. By default, **allow updates** is disabled (set to 0), so users cannot update system tables through ad hoc updates. Users can update system tables using system stored procedures only. When **allow updates** is disabled, updates are not allowed, even if you have the appropriate permissions (assigned using the GRANT statement).

When **allow updates** is enabled (set to 1), any user who has appropriate permissions can update system tables directly with ad hoc updates and can create stored procedures that update system tables.

**Caution** Updating fields in system tables can prevent an instance of Microsoft® SQL Server™ from running or can cause data loss. If you create stored procedures while the **allow updates** option is enabled, those stored procedures always have the ability to update system tables even after you disable **allow updates**. On production systems, you should not enable allow updates except under the direction of Microsoft Product Support Services.

Because system tables are critical to the operation of SQL Server, enable **allow updates** only in tightly controlled situations. Prevent other users from accessing SQL Server while you are directly updating system tables by restarting an instance of SQL Server from the command prompt with **sqlservr -m**. This command starts an instance of SQL Server in single-user mode and enables **allow updates**. For more information, see Starting SQL Server with Minimal Configuration.

If you set **allow updates** to 1 using the **sp_configure** system stored procedure, you must use the RECONFIGURE WITH OVERRIDE statement. This setting takes effect immediately (without a server stop and restart).

### To set the allow updates option

## Administering SQL Server

# awe enabled Option

In Microsoft® SQL Server™ 2000, you can use the Microsoft Windows® 2000 Address Windowing Extensions (AWE) API to support up to a maximum of 64 gigabytes (GB) of physical memory. The specific amount of memory you can use depends on hardware configuration and operating system support.

**Note** This feature is available only in the SQL Server 2000 Enterprise and Developer editions.

## Enabling AWE

To enable AWE, set **awe enabled** to 1. SQL Server will reserve almost all available memory, leaving 128 megabytes (MB) or less, unless a value has been specified for **max server memory**.

If the option has been successfully enabled, the message "Address Windowing Extension enabled" is printed in the SQL Server error log when the instance of SQL Server 2000 is started.

**awe enabled** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **awe enabled** only when **show advanced options** is set to 1. You must restart the instance of SQL Server 2000 for changes to take effect.

## Disabling AWE

To disable AWE, set **awe enabled** to 0. This setting is the default. The AWE API is not used. SQL Server 2000 operates in a normal dynamic memory allocation mode and is limited to 3 GB of physical memory.

## Usage Considerations

Before enabling AWE, consider the following:

- When **awe enabled** is set to 1, instances of SQL Server 2000 do not dynamically manage the size of the address space. SQL Server will reserve and lock almost all available memory (or the value of **max**

**server memory** if the option has been set) when the server is started. It is strongly recommended that you set a value for the **max server memory** option each time you enable AWE. Otherwise other applications or instances of SQL Server 2000 will have less than 128 MB of physical memory in which to run.

- If the total available memory is less than 3 GB, the instance of SQL Server 2000 will be started in non-AWE mode even if **awe enabled** is set to 1. In this situation, you do not need to manage AWE memory because dynamic memory allocation is used automatically.

- You can determine the amount of memory you can safely allocate to instances of SQL Server 2000 by identifying how much memory is available after all other applications to be used on the computer have been started.

  Use the SQL Server Performance Monitor **Total Server Memory (KB) counter** to determine how much memory is allocated by the instance of SQL Server running in AWE mode. Configure the max server memory option to leave some additional memory free to allow for the varying needs of other applications and Windows 2000. For more information, see Monitoring Memory Usage.

IMPORTANT Using the **awe enabled** option and the **max server memory** setting can have a performance impact on other applications or on SQL Server running in a multi-instance or cluster environment. For more information about using AWE memory, see Managing AWE Memory.

## Example

The following example shows how to enable AWE and configure a limit of 6 GB for **max server memory**:

```
sp_configure 'show advanced options', 1
RECONFIGURE
GO
sp_configure 'awe enabled', 1
```

```
RECONFIGURE
GO
sp_configure 'max server memory', 6144
RECONFIGURE
GO
```

## See Also

Memory Architecture

SQL Server: Buffer Manager Object

RECONFIGURE

Setting Configuration Options

sp_configure

# Administering SQL Server

# c2 audit mode Option

In Microsoft® SQL Server™ 2000, use the **c2 audit mode** option to review both successful and unsuccessful attempts to access statements and objects. With this information, you can document system activity and look for security policy violations.

C2 auditing tracks C2 audit events and records them to a file in the \mssql\data directory for default instances of SQL Server 2000, or the \mssql$instancename\data directory for named instances of SQL Server 2000. If the file reaches a size limit of 200 megabytes (MB), C2 auditing will start a new file, close the old file, and write all new audit records to the new file. This process will continue until SQL Server is shut down or auditing is turned off.

## Enabling and Disabling C2 Auditing

Before enabling and disabling C2 auditing, consider the following:

- You must be a member of the **sysadmin** role to enable or disable C2 auditing.

- **c2 audit mode** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **c2 audit mode** only when **show advanced options** is set to 1.

- If the audit directory fills up, the instance of SQL Server will be stopped. You can restart the instance of SQL Server if auditing is not set to start up automatically. But if auditing is set to start up automatically, you must free up disk space for the audit log before you can restart the instance of SQL Server.

  Alternatively, you can restart the instance with the **–f** flag, which will bypass all auditing. This is useful if you want to disable auditing until you can free up additional disk space or in an emergency situation where you do not have enough disk space to allocate the 200 MB audit file.

To enable C2 auditing, set the **c2 audit mode** option to 1. This setting establishes the C2 audit trace and turns on the option to fail the server should the server be unable to write to the audit file for any reason. After setting the option to 1, restart the server to begin C2 audit tracing. To stop C2 audit tracing, set **c2 audit mode** to 0.

**Important** If all audit counters are turned on for all objects, there could be a significant performance impact on the server.

## See Also

RECONFIGURE

Setting Configuration Options

sp_configure

# Administering SQL Server

# cost threshold for parallelism Option

Use the **cost threshold for parallelism** option to specify the threshold where Microsoft® SQL Server™ creates and executes parallel plans. SQL Server creates and executes a parallel plan for a query only when the estimated cost to execute a serial plan for the same query is higher than the value set in **cost threshold for parallelism**. The cost refers to an estimated elapsed time in seconds required to execute the serial plan on a specific hardware configuration. Only set **cost threshold for parallelism** on symmetric multiprocessors (SMP).

Longer queries usually benefit from parallel plans; the performance advantage negates the additional time required to initialize, synchronize, and terminate the plan. The **cost threshold for parallelism** option is actively used when a mix of short and longer queries is executed. The short queries execute serial plans while the longer queries use parallel plans. The value of **cost threshold for parallelism** determines which queries are considered short, thus executing only serial plans.

In certain cases, a parallel plan may be chosen even though the query's cost plan is less than the current **cost threshold for parallelism** value. This is because the decision to use a parallel or serial plan, with respect to **cost threshold for parallelism**, is based on a cost estimate provided before the full optimization is complete.

The **cost threshold for parallelism** option can be set to any value from 0 through 32767. The default value is 5.

If your computer has only one processor, if only a single CPU is available to SQL Server because of the **affinity mask** configuration value, or if the **max degree of parallelism** option is set to 1, SQL Server ignores **cost threshold for parallelism**.

**cost threshold for parallelism** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **cost threshold for parallelism** only when **show advanced options** is set to 1. The setting takes effect immediately (without a server stop and restart).

**To configure the cost threshold for parallelism**

# Administering SQL Server

# cursor threshold Option

Use the **cursor threshold** option to specify the number of rows in the cursor set at which cursor keysets are generated asynchronously. If you set **cursor threshold** to -1, all keysets are generated synchronously, which benefits small cursor sets. If you set **cursor threshold** to 0, all cursor keysets are generated asynchronously. With other values, the query optimizer compares the number of expected rows in the cursor set and builds the keyset asynchronously if it exceeds the number set in **cursor threshold**. Do not set **cursor threshold** too low because small result sets are better built synchronously.

When cursors generate a keyset for a result set, the query optimizer estimates the number of rows that will be returned for that result set. If the query optimizer estimates that the number of returned rows is greater than this threshold, the cursor is generated asynchronously, allowing the user to fetch rows from the cursor while the cursor continues to be populated. Otherwise, the cursor is generated synchronously, and the query waits until all rows are returned.

The accuracy of the query optimizer to determine an estimate for the number of rows in a keyset depends on the currency of the statistics for each of the tables in the cursor.

**cursor threshold** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **cursor threshold** only when **show advanced options** is set to 1. The setting takes effect immediately (without a server stop and restart).

### To set the cursor threshold option

⊞ Transact-SQL

⊞ SQL-DMO

## See Also

RECONFIGURE

Setting Configuration Options

sp_configure

## UPDATE STATISTICS

## Administering SQL Server

# default full-text language Option

Use the **default full-text language** option to specify a default language value for full-text indexed columns. Linguistic analysis is performed on all data that is full-text indexed and is dependant on the language of the data. The default value of this option is the language of the server.

The value of the **default full-text language** option is used when no language has been specified for a column by using **sp_fulltext_column**. If a value is specified for which a linguistic analysis package is not available, neutral is used. Neutral should be used when the column contains data in multiple languages, or if the language being used is not supported.

The following linguistic analysis packages are part of Microsoft® SQL Server™ 2000.

| Language | Setting |
|---|---|
| Chinese Simplified | 2052 |
| Chinese Traditional | 1028 |
| Dutch | 1043 |
| English UK | 2057 |
| English US | 1033 |
| French | 1036 |
| German | 1031 |
| Italian | 1040 |
| Japanese | 1041 |
| Korean | 1042 |
| Neutral | 0 |
| Spanish Modern | 3082 |
| Swedish Default | 1053 |

It is possible for additional languages to be added (for example, independent software vendors may provide additional languages).

The **default full-text language** option replaces the **language neutral full-text** option in SQL Server version 7.0. When upgrading from SQL Server 7.0, the

**default full-text language** value is set based on the values of SQL Server 7.0 configuration options **Unicode locale id** and **language neutral full-text**. This is done to allow compatibility with SQL Server 7.0 applications.

**default full-text language** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **default full-text language** only when **show advanced options** is set to 1.

### To set the default full-text language option

⊞ Transact-SQL

⊞ SQL-DMO

## See Also

RECONFIGURE

Setting Configuration Options

sp_configure

sp_fulltext_column

# Administering SQL Server

## default language Option

Use the **default language** option to specify the default language for all newly created logins. To set **default language**, specify the **langid** value of the desired language, as listed in the **syslanguages** table. For more information, see syslanguages.

The default language for a login can be overridden by using **sp_addlogin** or **sp_defaultlanguage**. The default language for a session is the language for that session's login, unless overridden on a per-session basis using the ODBC or OLEDB APIs.

**Note** The language for a session can be changed during the session through **SET LANGUAGE**. For more information, see SET LANGUAGE.

For information about what the language for a session determines, see SQL Server Language Support.

### To set the default language

# Administering SQL Server

## fill factor Option

Use the **fill factor** option to specify how full Microsoft® SQL Server™ should make each page when it creates a new index using existing data. The **fill factor** percentage affects performance because SQL Server must take time to split pages when they fill up.

The **fill factor** percentage is used only at the time the index is created. The pages are not maintained at any particular level of fullness.

The default for **fill factor** is 0; valid values range from 0 through 100. A **fill factor** value of 0 does not mean that pages are 0 percent full. It is treated similarly to a **fill factor** value of 100 in that SQL Server creates clustered indexes with full data pages and nonclustered indexes with full leaf pages. It is different from 100 in that SQL Server leaves some space within the upper level of the index tree. There is seldom a reason to change the default **fill factor** value because you can override it with the CREATE INDEX statement.

Small **fill factor** values cause SQL Server to create new indexes with pages that are not full. For example, a **fill factor** value of 10 is a reasonable choice if you are creating an index on a table that you know contains only a small portion of the data that it will eventually hold. Smaller **fill factor** values cause each index to take more storage space, allowing room for subsequent insertions without requiring page splits.

If you set **fill factor** to 100, SQL Server creates both clustered and nonclustered indexes with each page 100 percent full. Setting **fill factor** to 100 is suitable only for read-only tables, to which additional data is never added.

**fill factor** is an advanced option. If you will be using the **sp_configure** system stored procedure to change the setting, you can change **fill factor** only when **show advanced options** is set to 1. The setting takes effect after stopping and restarting the server.

**To set a fixed fill factor**

# Administering SQL Server

# index create memory Option

Use the **index create memory** option to control the amount of memory used by index creation sorts. The **index create memory** option is self-configuring and should work in most cases without requiring adjustment. However, if you experience difficulties creating indexes, consider increasing the value of this option from its run value. Query sorts are controlled through the **min memory per query** option.

The default value for this option is 0 (self-configuring).

**index create memory** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **index create memory** only when **show advanced options** is set to 1. The setting takes effect immediately (without a server stop and restart).

### To set the index create memory option

⊞ Transact-SQL

⊞ SQL-DMO

## See Also

RECONFIGURE

Server Memory Options

Setting Configuration Options

sp_configure

## Administering SQL Server

## lightweight pooling Option

Use the **lightweight pooling** option to provide a means of reducing the system overhead associated with the excessive context switching sometimes seen in symmetric multiprocessor (SMP) environments. When excessive context switching is present, **lightweight pooling** may provide better throughput by performing the context switching inline, thus helping to reduce user/kernel ring transitions.

Setting **lightweight pooling** to 1 causes Microsoft® SQL Server™ to switch to fiber mode scheduling. The default value for this option is 0.

**lightweight pooling** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **lightweight pooling** only when **show advanced options** is set to 1. The setting takes effect after stopping and restarting the server.

### To set the lightweight pooling option

⊞ Transact-SQL

⊞ SQL-DMO

### See Also

RECONFIGURE

Setting Configuration Options

sp_configure

SQL Server Task Scheduling

Using the lightweight pooling Options

# Administering SQL Server

## locks Option

Use the **locks** option to set the maximum number of available locks, thereby limiting the amount of memory Microsoft® SQL Server™ uses for locks. The default setting is 0, which allows SQL Server to allocate and deallocate locks dynamically based on changing system requirements.

When the server is started with **locks** set to 0, the lock manager allocates two percent of the memory allocated to SQL Server to an initial pool of lock structures. As the pool of locks is exhausted, additional locks are allocated. The dynamic lock pool does not allocate more than 40 percent of the memory allocated to SQL Server.

Generally, if more memory is required for locks than is available in current memory, and more server memory is available (the **max server memory** threshold has not been reached), SQL Server allocates memory dynamically to satisfy the request for locks. However, if allocating that memory would cause paging at the operating system level (for example, if another application was running on the same computer as an instance of SQL Server and using that memory), more lock space is not allocated.

Allowing SQL Server to use locks dynamically is the recommended configuration. However, you can set **locks** and override SQL Server's ability to allocate lock resources dynamically. Increase this value if SQL Server displays a message that you have exceeded the number of available locks. Because each lock consumes memory (96 bytes per lock), increasing this value can require increasing the amount of memory dedicated to the server.

**locks** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **locks** only when **show advanced options** is set to 1. The setting takes effect after stopping and restarting the server.

### To set the locks option

⊞ Transact-SQL

⊞ SQL-DMO

## See Also

Locking

RECONFIGURE

Setting Configuration Options

sp_configure

# Administering SQL Server

## max degree of parallelism Option

Use the **max degree of parallelism** option to limit the number of processors (a maximum of 32) to use in parallel plan execution. The default value is 0, which uses the actual number of available CPUs. Set **max degree of parallelism** to 1 to suppress parallel plan generation. Set the value to a number greater than 1 to restrict the maximum number of processors used by a single query execution. If a value greater than the number of available CPUs is specified, the actual number of available CPUs is used.

**Note** If the **affinity mask** option is not set to the default, it may restrict the number of CPUs available to Microsoft® SQL Server™ on a symmetric multiprocessor (SMP) systems.

Change **max degree of parallelism** rarely for servers running on an SMP computer. If your computer has only one processor, the **max degree of parallelism** value is ignored.

**max degree of parallelism** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **max degree of parallelism** only when **show advanced options** is set to 1. The setting takes effect immediately (without a server stop and restart).

In addition to queries, this option also controls the parallelism of DBCC CHECKTABLE, DBCC CHECKDB, and DBCC CHECKFILEGROUP. Parallel checking can be overridden by using trace flag 2528. For more information, see Trace Flags.

### To set the max degree of parallelism option

# Administering SQL Server

# max text repl size Option

Use the **max text repl size** option to specify the maximum size (in bytes) of **text** and **image** data that can be added to a replicated column in a single INSERT, UPDATE, WRITETEXT, or UPDATETEXT statement.

The setting takes effect immediately (without a server stop and restart).

### To set the max text repl size option

⊞ Transact-SQL

⊞ SQL-DMO

## See Also

INSERT

RECONFIGURE

Replication Overview

Setting Configuration Options

sp_configure

UPDATE

UPDATETEXT

WRITETEXT

# Administering SQL Server

## max worker threads Option

Use the **max worker threads** option to configure the number of worker threads available to Microsoft® SQL Server™ processes. SQL Server uses the native thread services of the Microsoft Windows NT® 4.0 or Windows® 2000 operating system so that one or more threads support each network that SQL Server supports simultaneously; another thread handles database checkpoints; and a pool of threads handles all users.

Thread pooling helps optimize performance when large numbers of clients are connected to the server. Usually, a separate operating system thread is created for each client connection to consume fewer system resources. However, with hundreds of connections to the server, using a thread-per-connection can consume large amounts of system resources. **max worker threads** enables SQL Server to create a pool of worker threads to service a larger number of client connections, which improves performance.

The default setting for **max worker threads** (255) is best for most systems. However, depending on your system configuration, setting **max worker threads** to a smaller value sometimes improves performance.

When the actual number of user connections is less than the amount set in **max worker threads**, one thread handles each connection. However, if the actual number of connections exceeds the amount set in **max worker threads**, SQL Server pools the worker threads so that the next available worker thread can handle the request.

When the maximum number of worker threads is reached, SQL Server returns the following message:

```
The working thread limit of 255 has been reached.
```

Because Windows 98 does not support thread pooling, the option has no effect on those systems.

**max worker threads** is an advanced option. If you will be using the **sp_configure** system stored procedure to change the setting, you can change **max worker threads** only when **show advanced options** is set to 1. The setting takes effect immediately (without a server stop and restart).

### To configure the maximum number of worker threads

# Administering SQL Server

## media retention Option

Use the **media retention** option to provide a system-wide default for the length of time to retain each backup medium after it has been used for a database or transaction log backup. **media retention** helps protect backups from being overwritten until the specified number of days has elapsed. When you set **media retention**, you do not have to specify the length of time to retain system backups each time you perform a backup. The default is 0 days. If you use the backup medium before the set number of days has passed, Microsoft® SQL Server™ issues a warning message. SQL Server does not issue a warning unless you change the default.

This option can be overridden by using the RETAINDAYS clause of the BACKUP statement.

**media retention** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **media retention** only when **show advanced options** is set to 1. The setting takes effect after stopping and restarting the server.

**To set the backup retention duration**

# Administering SQL Server

## min memory per query Option

Use the **min memory per query** option to specify the minimum amount of memory (in kilobytes) that will be allocated for the execution of a query. For example, if **min memory per query** is set to 2048 kilobytes (KB), the query is guaranteed to get at least that much total memory. You can set **min memory per query** to any value from 512 through 2147483647 KB (2 gigabytes). The default is 1024 KB.

The Microsoft® SQL Server™ 2000 query processor attempts to determine the optimal amount of memory to allocate to a query. The **min memory per query** option lets the administrator specify the minimum amount of memory any single query will receive. Queries will generally receive more memory than this if they have hash and sort operations on a large volume of data. Increasing the value of **min memory per query** may improve performance for some small to medium sized queries, but could lead to increased contention for memory resources. **min memory per query** includes memory allocated for sorting and replaces the **sort pages** option in SQL Server version 7.0 or earlier.

**min memory per query** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **min memory per query** only when **show advanced options** is set to 1. The setting takes effect immediately (without a server stop and restart).

**To set minimum query memory**

# Administering SQL Server

## nested triggers Option

Use the **nested triggers** option to control whether a trigger can cascade (perform an action that initiates another trigger that initiates another trigger, and so on). When **nested triggers** is set to 0, triggers cannot cascade. When **nested triggers** is set to 1 (the default), triggers can cascade to as many as 32 levels.

The setting takes effect immediately (without a server stop and restart).

### To set the nested triggers option

# Administering SQL Server

## network packet size Option

Use the **network packet size** option to set the packet size (in bytes) used across the entire network. Packets are the fixed-size chunks of data that transfer requests and results between clients and servers. The default packet size set by Microsoft® SQL Server™ is 4096 bytes. If an application does bulk copy operations, or sends or receives large amounts of **text** or **image** data, a packet size larger than the default may improve efficiency because it results in fewer network reads and writes. If an application sends and receives small amounts of information, you can set the packet size to 512 bytes, which is sufficient for most data transfers.

**Note** Do not change the packet size unless you are certain that it will improve performance. For most applications, the default packet size is best.

On systems using differing network protocols, set **network packet size** to the size for the most common protocol used. **network packet size** improves network performance when network protocols support larger packets. Client applications can override this value.

You can also call OLE DB, ODBC, and DB-Library functions to change the packet size.

**network packet size** is an advanced option. If you will be using the **sp_configure** system stored procedure to change the setting, you can change **network packet size** only when **show advanced options** is set to 1. All connections created after this setting is changed receive the new value.

**To configure packet size**

# Administering SQL Server

## open objects Option

Use the **open objects** option to set the maximum number of database objects that can be open at one time on an instance of Microsoft® SQL Server™. Database objects are those objects defined in the **sysobjects** table: tables, views, rules, stored procedures, defaults, and triggers.

**open objects** is a dynamic self-configuring option by default (when the value is set to 0). In other words, SQL Server sets this value depending on the current needs of the system. In most cases, you should not need to change this value.

Consider increasing the value set in **open objects** if SQL Server displays a message that you have exceeded the number of open objects. Because open objects consume memory, increasing this value takes memory from other SQL Server uses and makes it necessary to increase the amount of memory dedicated to the server. The default is to allow SQL Server to set and increase **open objects** as needed.

At server startup, SQL Server builds a pool of descriptor data structures in memory that are used to describe database objects as they are referenced. The number of descriptors built is equal to the number set in **open objects**. The first time a database object is referenced, SQL Server takes one of the descriptors from the free pool of descriptor data and allocates it to the specific object. If multiple tasks reference the same object at the same time, it is still considered one open object.

For example, two tasks issue the following command at the same time:

```
UPDATE table_a SET cola = @variable
```

There is only one descriptor allocated to **table_a**, which is considered one open object. However, if **table_a** has an update trigger, then a second descriptor is allocated to the trigger, counting as a second open object.

Each allocated descriptor has a use counter that indicates how many concurrent queries are referencing the object it defines. The use count is increased by one at the start of a query, and decreased by one by the end of the query. In the previous example, the **table_a** descriptor would have a use count of 2 until the two queries finish; it then decreases to 0.

After the free pool of descriptors has been used, SQL Server starts reusing inactive descriptors when it needs to allocate a new descriptor. An inactive descriptor is one whose use count is 1. The first time SQL Server has to reuse a descriptor, it issues this message in the error log:

```
Warning: OPEN OBJECTS parameter may be too low;
attempt was made to free up descriptors in localdes().
Run sp_configure to increase parameter value.
```

SQL Server repeats this message after each 1,000 times it has to reuse a descriptor. If you notice that these messages are being issued frequently in the error log, set **open objects** to a higher value.

**open objects** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **open objects** only when **show advanced options** is set to 1. The setting takes effect after stopping and restarting the server.

### To set the open objects option

⊞ Transact-SQL

⊞ SQL-DMO

## See Also

RECONFIGURE

Setting Configuration Options

sp_configure

# Administering SQL Server

## priority boost Option

Use the **priority boost** option to specify whether Microsoft® SQL Server™ should run at a higher Microsoft Windows NT® 4.0 or Windows® 2000 scheduling priority than other processes on the same computer. If you set this option to 1, SQL Server runs at a priority base of 13 in the Windows NT 4.0 or Windows 2000 scheduler. The default is 0, which is a priority base of 7.

**priority boost** should be used only on a computer dedicated to SQL Server, and with a symmetric multiprocessor (SMP) configuration.

**Caution** Boosting the priority too high may drain resources from essential operating system and network functions, resulting in problems shutting down SQL Server or using other Windows NT 4.0 or Windows 2000 tasks on the server.

In some circumstances, setting **priority boost** to anything other than the default can cause the following communication error to be logged in the SQL Server error log:

```
Error: 17824, Severity: 10, State: 0 Unable to write to ListenOn
connection '<servername>', loginname '<login ID>', hostname '<hostna
OS Error: 64, The specified network name is no longer available.
```

Error 17824 indicates that SQL Server encountered connection problems while attempting to write to a client. These communication problems may be caused by network problems, if the client has stopped responding, or if the client has been restarted. However, error 17824 does not always indicate a network problem. Check **priority boost** and make sure that the option is set to the default. Deviating from the default may cause error 17824.

**priority boost** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **priority boost** only when **show advanced options** is set to 1. The setting takes effect after stopping and restarting the server.

**To set the priority boost option**

# Administering SQL Server

# query governor cost limit Option

Use the **query governor cost limit** option to specify an upper limit for the time in which a query can run. Query cost refers to the estimated elapsed time, in seconds, required to execute a query on a specific hardware configuration.

If you specify a nonzero, nonnegative value, the query governor disallows execution of any query that has an estimated cost exceeding that value. Specifying 0 (the default) for this option turns off the query governor. In this case, all queries are allowed to run.

If you use **sp_configure** to change the value of **query governor cost limit**, the changed value is server-wide. To change the value on a per connection basis, use the SET QUERY_GOVERNOR_COST_LIMIT statement.

**query governor cost limit** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **query governor cost limit** only when **show advanced options** is set to 1. The setting takes effect immediately (without a server stop and restart).

## To set the query governor cost limit option

# Administering SQL Server

# query wait Option

In Microsoft® SQL Server™, memory-intensive queries, such as those involving sorting and hashing, are queued when there is not enough memory available to run the query. The query times out after a set amount of time calculated by SQL Server (25 times the estimated cost of the query) or the time amount specified by the non-negative value of the query wait.

Use the **query wait** option to specify the time in seconds (from 0 through 2147483647) that a query waits for resources before timing out. If the default value of -1 is used, or if –1 is specified, then the time-out is calculated as 25 times of the estimated query cost.

**IMPORTANT** A transaction containing the waiting query may hold locks while the query waits for memory. In rare situations, it is possible for an undetectable deadlock to occur. Decreasing the query wait time lowers the probability of such deadlocks. Eventually, a waiting query will be terminated and the transaction locks released. However, increasing the maximum wait time may increase the amount of time for the query to be terminated. Changes to this option are not recommended.

**query wait** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **query wait** only when **show advanced options** is set to 1. The setting takes effect immediately (without a server stop and restart).

### To set the query wait option

⊞ Transact-SQL

⊞ SQL-DMO

## See Also

RECONFIGURE

Setting Configuration Options

sp_configure

## Thread and Fiber Execution

## Administering SQL Server

## recovery interval Option

Use the **recovery interval** option to set the maximum number of minutes per database that Microsoft® SQL Server™ needs to recover databases. Each time an instance of SQL Server starts, it recovers each database, rolling back transactions that did not commit and rolling forward transactions that did commit but whose changes were not yet written to disk when an instance of SQL Server stopped. This configuration option sets an upper limit on the time it should take to recover each database. The default is 0, indicating automatic configuration by SQL Server. In practice, this means a recovery time of less than one minute and a checkpoint approximately every one minute for active databases.

**recovery interval** controls when SQL Server issues a checkpoint in each database. Checkpoints are done on a per database basis. At a checkpoint, SQL Server ensures all log information and all modified pages are flushed from memory to disk. This limits the time needed for recovery by limiting the number of transactions rolled forward to ensure they are on disk. No modifications done before the checkpoint need to be rolled forward because they have been flushed to disk at the checkpoint.

**recovery interval** does not affect the time it takes to undo long-running transactions. For example, if a long-running transaction has taken two hours to perform updates before the server became disabled, the actual recovery will take considerably longer than the **recovery interval** value to roll back the long transaction.

SQL Server estimates how many data modifications it can roll forward in the recovery time interval. SQL Server typically issues a checkpoint in a database when the number of data modifications made in the database after the last checkpoint reaches the number SQL Server estimates it can roll forward in the recovery time interval. Sometimes SQL Server will issue the checkpoint when the log becomes 70 percent full, if that is less than the estimated number. For more information, see Checkpoints and the Active Portion of the Log.

The frequency of checkpoints in each database depends on the amount of data modifications made, not on any time-based measure. A database used primarily for read-only operations will not have many checkpoints. A transaction database will have frequent checkpoints.

Keep **recovery interval** set at 0 (self-configuring) unless you notice that checkpoints are impairing performance because they are occurring too frequently. If this is the case, try increasing the value in small increments.

**recovery interval** is an advanced option. If you will be using the **sp_configure** system stored procedure to change the setting, you can change **recovery interval** only when **show advanced options** is set to 1. The setting takes effect immediately (without a server stop and restart).

### To set the recovery interval

# Administering SQL Server

## remote access Option

Use the **remote access** option to control logins from remote servers running instances of Microsoft® SQL Server™. **remote access** is used with remote stored procedures. Set **remote access** to 1 (default) to allow logins from remote servers. Set the option to 0 to secure a local server and prevent access from a remote server.

The setting takes effect after stopping and restarting the server.

### To set remote server access

Administering SQL Server

# remote login timeout Option

Use the **remote login timeout** option to specify the number of seconds to wait before returning from a failed remote login attempt. For example, if you are attempting to log in to a remote server and that server is down, **remote login timeout** ensures that you do not have to wait indefinitely before your computer ceases its attempts to log in.

**remote login timeout** affects connections to OLE DB providers made for heterogeneous queries. The default setting for **remote login timeout** is 20 seconds. A value of 0 allows for an infinite wait.

The setting takes effect immediately (without a server stop and restart).

### To set the remote login timeout option

⊞ Transact-SQL

⊞ SQL-DMO

## See Also

RECONFIGURE

Setting Configuration Options

sp_configure

# Administering SQL Server

# remote proc trans Option

Use the **remote proc trans** option to protect the actions of a server-to-server procedure through a Microsoft® Distributed Transaction Coordinator (MS DTC) transaction. Set **remote proc trans** to 1 to provide an MS DTC-coordinated distributed transaction that protects the ACID properties of transactions. Sessions begun after setting this option to 1 inherit the configuration setting as their default.

The setting takes effect immediately (without a server stop and restart).

For more information about ACID properties, see Transactions.

For more information about MS DTS, see the Microsoft Distributed Transaction Coordinator documentation.

## To enforce distributed transactions for remote procedures

# Administering SQL Server

## remote query timeout Option

Use the **remote query timeout** option to specify the number of seconds that must elapse when processing a remote operation before Microsoft® SQL Server™ assumes the command failed or took too much time to perform (times out). The default is 600, which allows a ten minute wait.

For heterogeneous queries, **remote query timeout** specifies the number of seconds (initialized in the command object using the DBPROP_COMMANDTIMEOUT rowset property) that a remote provider should wait for result sets before the query times out. This value is also used to set DBPROP_GENERALTIMEOUT if supported by the remote provider. This will cause any other operations to time out after the specified number of seconds.

For remote stored procedures, **remote query timeout** specifies the number of seconds that must elapse after sending a remote "EXEC sp" before the remote stored procedure times out.

The setting takes effect immediately (without a server stop and restart).

**To set a time limit for remote queries**

# Administering SQL Server

# scan for startup procs Option

Use the **scan for startup procs** option to scan for automatic execution of stored procedures at Microsoft® SQL Server™ startup time. If this option is set to 1, SQL Server scans for and executes all automatically executed stored procedures defined on the server. The default value for **scan for startup procs** is 0 (do not scan).

The value for this option can be set using **sp_configure**; however, it will be set automatically if you use **sp_procoption**, which is used to mark or unmark stored procedures as automatically executed (autoprocs). When **sp_procoption** is used to mark the first stored procedure as an autoproc, this option is set automatically to a value of 1. When **sp_procoption** is used to unmark the last stored procedure as an autoproc, this option is automatically set to a value of 0. If you use **sp_procoption** to mark and unmark autoprocs, and always unmark autoprocs before dropping them, there is no need to set this option manually.

**scan for startup procs** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **scan for startup procs** only when **show advanced options** is set to 1. The setting takes effect after stopping and restarting the server.

### To set the scan for startup procs option

⊞ Transact-SQL

⊞ SQL-DMO

## See Also

RECONFIGURE

Setting Configuration Options

sp_configure

sp_procoption

# Administering SQL Server

## Server Memory Options

Use the two server memory options, **min server memory** and **max server memory**, to reconfigure the amount of memory (in megabytes) in the buffer pool used by an instance of Microsoft® SQL Server™.

By default, SQL Server can change its memory requirements dynamically based on available system resources. The default setting for **min server memory** is 0, and the default setting for **max server memory** is 2147483647. The minimum amount of memory you can specify for **max server memory** is 4 megabytes (MB).

When SQL Server is using memory dynamically, it queries the system periodically to determine the amount of free physical memory available. SQL Server grows or shrinks the buffer cache to keep free physical memory between 4 MB and 10 MB depending on server activity. This prevents Microsoft Windows NT® 4.0 or Windows® 2000 from paging. If there is less memory free, SQL Server releases memory to Windows NT 4.0 or Windows 2000 that usually goes on the free list. If there is more memory free, SQL Server recommits memory to the buffer cache. SQL Server adds memory to the buffer cache only when its workload requires more memory; a server at rest does not grow its buffer cache.

Allowing SQL Server to use memory dynamically is the recommended configuration; however, you can set the memory options manually and override SQL Server's ability to use memory dynamically. Before you set the amount of memory for SQL Server, determine the appropriate memory setting by subtracting from the total physical memory the memory required for Windows NT 4.0 or Windows 2000 and any other instances of SQL Server (and other system uses, if the computer is not wholly dedicated to SQL Server). This is the maximum amount of memory you can assign to SQL Server.

**Note** If you have installed and are running the Full-Text Search support (Microsoft Search service, also known as MSSearch), then you must set the **max server memory** option manually to leave enough memory for the MSSearch service to run. The **max server memory** setting must be adjusted in conjunction with the Windows NT 4.0 virtual memory size such that the virtual memory remaining for Full-Text Search is 1.5 times the physical memory (excluding the virtual memory requirements of the other services on the computer). Configure the SQL Server **max server memory** option so that there is sufficient virtual memory left to satisfy this Full-Text Search memory requirement. Total virtual memory - (SQL Server maximum virtual memory + virtual memory requirements of other services) >= 1.5 times the physical memory.

## Setting the Memory Options Manually

There are two principal methods for setting the SQL Server memory options manually:

- In the first method, set **min server memory** and **max server memory** to the same value. This value corresponds to the fixed amount of memory to allocate to SQL Server.

- In the second method, set **min server memory** and **max server memory** to span a range of memory values. This is useful in situations where system or database administrators want to configure an instance of SQL Server in conjunction with the memory requirements of other applications running on the same computer.

Use **min server memory** to guarantee a minimum amount of memory to an instance of SQL Server. SQL Server will not immediately allocate the amount of memory specified in **min server memory** on startup. However, after memory usage has reached this value due to client load, SQL Server cannot free memory from the allocated buffer pool unless the value of **min server memory** is reduced.

**Note** SQL Server is not guaranteed to allocate the amount of memory specified in **min server memory**. If the load on the server never necessitates the allocation of the amount of memory specified in **min server memory**, then SQL Server will run with less memory.

Use **max server memory** to prevent SQL Server from using more than the specified amount of memory, thus leaving remaining memory available to start other applications quickly. SQL Server does not immediately allocate the memory specified in **max server memory** on startup. Memory usage is increased as needed by SQL Server until reaching the value specified in **max**

**server memory**. SQL Server cannot exceed this memory usage unless the value of **max server memory** is raised.

**IMPORTANT** Instances of SQL Server 2000 running in Address Windowing Extensions (AWE) memory mode do allocate all the full amount of memory specified in **max server memory** on server startup. For more information about AWE memory, see Managing AWE Memory.

There is a short delay between the start of a new application and the time SQL Server releases memory. Using **max server memory** prevents this delay and may give better performance to the other application. Only set **min server memory** if the start time of new applications sharing the same server as SQL Server shows up as a problem. It is better to let SQL Server use all of the available memory.

If you set the memory options manually, be sure to set them appropriately for servers used in replication. If the server is a remote Distributor or a combined Publisher/Distributor, you must assign it at least 16 MB of memory.

Ideally, you want to allocate as much memory as possible to SQL Server without causing the system to swap pages to disk. The threshold varies depending on your system. For example, on a 32-MB system, 16 MB might be appropriate for SQL Server; on a 64-MB system, 48 MB might be appropriate.

**Note** As you increase the amount of SQL Server memory, ensure that there is sufficient disk space to grow the operating system's virtual memory support file (Pagefile.sys) to accommodate additional memory. For more information about the virtual memory support file, see the Windows NT 4.0 and Windows 2000 documentation.

The amount of memory specified must be sufficient for the SQL Server static memory needs (kernel overhead, open objects, locks, and so on), as well as for the data cache (also called buffer cache).

Use statistics from System Monitor (Performance Monitor in Windows NT 4.0) to help you adjust the memory value if necessary. Change this value only when you add or remove memory, or when you change how you use your system.

## Virtual Memory Manager

Windows NT 4.0 and Windows 2000 provide a 4-gigabyte (GB) virtual address space at any time, the lower 2 GB of which is private per process and available for application use. The upper 2 GB is reserved for system use. Windows NT Server, Enterprise Edition provides a 4-GB virtual address space for each Microsoft Win32® application, the lower 3 GB of which is private per process and available for application use. The upper 1 GB is reserved for system use.

The 4-GB address space is mapped to the available physical memory by Windows NT Virtual Memory Manager (VMM). The available physical memory can be up to 4 GB, depending on hardware platform support.

A Win32 application such as SQL Server perceives only virtual or logical addresses, not physical addresses. How much physical memory an application uses at a given time (the working set) is determined by available physical memory and the VMM. The application cannot control memory residency directly.

Virtual address systems such as Windows NT 4.0 or Windows 2000 allow the over-committing of physical memory, such that the ratio of virtual to physical memory exceeds 1:1. As a result, larger programs can run on computers with a variety of physical memory configurations. However, using significantly more virtual memory than the combined average working sets of all the processes results in poor performance.

SQL Server can lock memory as a working set. Because memory is locked, you can receive out of memory errors when running other applications. If out-of-memory errors occur, you may have too much memory assigned to SQL Server. The **set working set size** option (set with **sp_configure** or SQL Server Enterprise Manager) can disable the locking of memory as a working set. By default, **set working set size** is disabled.

Configuring SQL Server manually for more virtual memory than there is physical memory can result in poor performance. Also, the Windows NT 4.0 or Windows 2000 operating system memory requirement must be considered (about 12 MB, with some variation depending on application overhead). System overhead requirements can grow as SQL Server parameters are configured upward and Windows NT 4.0 or Windows 2000 needs more resident memory to support additional threads, page tables, and so on. Allowing SQL Server to use memory dynamically helps to avoid memory-related performance problems.

**min server memory** and **max server memory** are advanced options. If you are using the **sp_configure** system stored procedure to change these settings, you can change them only when **show advanced options** is set to 1. These settings take effect immediately (without a server stop and restart).

### To set a fixed amount of memory

# Administering SQL Server

## set working set size Option

Use the **set working set size** option to reserve physical memory space for Microsoft® SQL Server™ that is equal to the server memory setting. The server memory setting is configured automatically by SQL Server based on workload and available resources. It will vary dynamically between **min server memory** and **max server memory**. Setting **set working set size** means Microsoft® Windows NT® 4.0 or Windows® 2000 do not swap out SQL Server pages even if they can be used more readily by another process when SQL Server is idle.

Do not set **set working set size** if you are allowing SQL Server to use memory dynamically. Before setting **set working set size** to 1, set both **min server memory** and **max server memory** to the same value, the amount of memory you want SQL Server to use.

**set working set size** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **set working set size** only when **show advanced options** is set to 1. The setting takes effect after stopping and restarting the server.

**To set the working set size option**

## Administering SQL Server

## show advanced options Option

Use the **show advanced options** option to display the **sp_configure** system stored procedure advanced options. When you set **show advanced options** to 1, you can list the advanced options by using **sp_configure**. The default is 0.

The setting takes effect immediately (without a server stop and restart).

### To set the show advanced options option

⊞ Transact-SQL

⊞ SQL-DMO

### See Also

RECONFIGURE

Setting Configuration Options

sp_configure

## Administering SQL Server

## two digit year cutoff Option

Use the **two digit year cutoff** option to specify an integer from 1753 to 9999 that represents the cutoff year for interpreting two-digit years as four-digit years.

A two-digit year that is less than or equal to the last two digits of the cutoff year is in the same century as the cutoff year. A two-digit year that is greater than the last two digits of the cutoff year is in the century that precedes the cutoff year. For example, if **two digit year cutoff** is 2049 (the default), the two-digit year 49 is interpreted as 2049 and the two-digit year 50 is interpreted as 1950.

**Note** Microsoft® SQL Server™ uses 2049 as the cutoff year for interpreting dates; OLE Automation objects use 2030. You can use the **two digit year cutoff** option to provide consistency in date values between SQL Server and client applications. However, to avoid ambiguity with dates, use four-digit years in your data.

**To set the two digit year cutoff option**

# Administering SQL Server

## user connections Option

Use the **user connections** option to specify the maximum number of simultaneous user connections allowed on Microsoft® SQL Server™. The actual number of user connections allowed also depends on the version of SQL Server you are using and the limits of your application(s) and hardware. SQL Server allows a maximum of 32,767 user connections.

Because **user connections** is dynamic (self-configuring option), SQL Server adjusts the maximum number of user connections automatically as needed, up to the maximum value allowable. For example, if only 10 users are logged in, 10 user connection objects are allocated. In most cases, you should not need to change the value for this option.

You can use SQL Query Analyzer and the following Transact-SQL statement to determine the maximum number of user connections that your system allows:

```
SELECT @@MAX_CONNECTIONS
```

**user connections** helps avoid overloading the server with too many concurrent connections. You can estimate the number of connections based on system and user requirements. For example, on a system with many users, each user would not usually require a unique connection. Connections can be shared among users. Users who are running OLE DB applications need a connection for each open connection object, users who are running ODBC applications need a connection for each active connection handle in the application, and users who are running DB-Library applications need one connection for each process started that calls the DB-Library **dbopen** function.

**IMPORTANT** If you must use this option, do not set the value too high because each connection takes approximately 40 kilobytes (KB) of overhead regardless of whether the connection is being used. If you exceed the maximum number of user connections, you receive an error message and are not able to connect until another connection becomes available.

**user connections** is an advanced option. If you are using the **sp_configure** system stored procedure to change the setting, you can change **user connections** only when **show advanced options** is set to 1. The setting takes effect after stopping and restarting the server.

### To set user connections

# Administering SQL Server

## user options Option

Use the **user options** option to specify global defaults for all users. A list of default query processing options is established for the duration of a user's work session. **user options** allows you to change the default values of the SET options (if the server's default settings are not appropriate). A user can override these defaults by using the SET statement. You can configure **user options** dynamically for new logins. After you change the setting of **user options**, new logins use the new setting; current logins are not affected.

| Value | Configuration | Description |
|---|---|---|
| 1 | DISABLE_DEF_CNST_CHK | Controls interim or deferred constraint checking. |
| 2 | IMPLICIT_TRANSACTIONS | Controls whether a transaction is started implicitly when a statement is executed. |
| 4 | CURSOR_CLOSE_ON_COMMIT | Controls behavior of cursors after a commit operation has been performed. |
| 8 | ANSI_WARNINGS | Controls truncation and NULL in aggregate warnings. |
| 16 | ANSI_PADDING | Controls padding of fixed-length variables. |
| 32 | ANSI_NULLS | Controls NULL handling when using equality operators. |
| 64 | ARITHABORT | Terminates a query when an overflow or divide-by-zero error occurs during query execution. |
| 128 | ARITHIGNORE | Returns NULL when an overflow or divide-by-zero error occurs during a query. |
| 256 | QUOTED_IDENTIFIER | Differentiates between single and double quotation marks when evaluating an expression. |

| | | |
|---|---|---|
| 512 | NOCOUNT | Turns off the message returned at the end of each statement that states how many rows were affected. |
| 1024 | ANSI_NULL_DFLT_ON | Alters the session's behavior to use ANSI compatibility for nullability. New columns defined without explicit nullability are defined to allow nulls. |
| 2048 | ANSI_NULL_DFLT_OFF | Alters the session's behavior not to use ANSI compatibility for nullability. New columns defined without explicit nullability are defined not to allow nulls. |
| 4096 | CONCAT_NULL_YIELDS_NULL | Returns NULL when concatenating a NULL value with a string. |
| 8192 | NUMERIC_ROUNDABORT | Generates an error when a loss of precision occurs in an expression. |
| 16384 | XACT_ABORT | Rolls back a transaction if a Transact- SQL statement raises a run-time error. |

**Note** Not all configuration values for **user options** are compatible with each other. For example, ANSI_NULL_DFLT_ON cannot be enabled when ANSI_NULL_DFLT_OFF is enabled.

The bit positions in **user options** are identical to those in @@OPTIONS. Each connection has its own @@OPTIONS function, which represents the configuration environment. When logging in to Microsoft® SQL Server™, a user receives a default environment that assigns the current **user options** value to the @@OPTIONS. Executing SET statements for **user options** affects the corresponding value in the session's @@OPTIONS.

All connections created after this setting is changed receive the new value.

## To configure user options

# Administering SQL Server

## Managing Clients

A client is a front-end application that uses the services provided by a server. The computer that hosts the application is referred to as the client computer. Client software enables computers to connect to an instance of Microsoft® SQL Server™ on a network.

SQL Server clients can include applications of various types, such as:

- OLE DB consumers.

  These applications use the Microsoft OLE DB Provider for SQL Server or the Microsoft OLE DB Provider for ODBC to connect to an instance of SQL Server. The OLE DB providers serve as intermediaries between SQL Server and client applications that consume SQL Server data as OLE DB rowsets.

- ODBC applications.

  These include client utilities installed with SQL Server, such as SQL Server Enterprise Manager and SQL Query Analyzer, as well as other applications that use the SQL Server ODBC driver to connect to an instance of SQL Server.

- DB-Library clients.

  These include the SQL Server **isql** command prompt utility and clients written to DB-Library.

Regardless of the type of application, managing a client consists mainly of configuring its connection with the server components of SQL Server. Depending on the requirements of your site, client management can range from little more than entering the name of the server computer to building a library of custom configuration entries to accommodate a diverse multiserver environment.

### Simple Client Management

For the majority of clients, the default network configuration installed during SQL Server Setup can be used without modification. For those clients to be able to connect, you need only supply the network name of the server running one or more instances of SQL Server. For ODBC clients, you may need to provide the client with the ODBC data source name and know how to configure an ODBC data source.

## Advanced Client Management

Advanced users can create and save individual network protocol configurations. This is useful in situations where SQL Server clients are connecting to multiple servers running different network protocols, or where unique site considerations, such as nonstandard port addresses, are used.

## Before Configuring a Client

Before configuring a SQL Server client:

- You must install a matching pair of SQL Server Net-Libraries on the client and server. By default, all of the SQL Server client Net-Libraries and server Net-Libraries are installed automatically during the Setup program. Each pair of Net-Libraries supports a particular network protocol (for example, the client TCP/IP Sockets Net-Library and server TCP/IP Sockets Net-Library support TCP/IP). Some SQL Server server Net-Libraries (such as NW Link IPX/SPX) should be activated to listen for clients, either during or after setup, using the SQL Server Network Utility.

- You must install the correct network protocols on the client and server. Network protocols are typically installed during Microsoft Windows® Setup; they are not part of SQL Server Setup or configuration. A SQL Server Net-Library will not work unless its corresponding network protocol is installed already on both the client and server.

## Client Management Tools

The following tools are used to manage most types of SQL Server clients:

- Client Network Utility lets you change the default network protocols, and create and save entries that define how to connect to specified servers.

The application is installed as part of the standard SQL Server client setup. SQL Server Client Network Registration creates registry entries for the client network protocol configurations and default network protocol. You do not use the application to install either the SQL Server Net-Libraries or the network protocols.

- The Setup program and SQL Server Network Library Configuration let you select and activate server Net-Libraries (all the client and server Net-Libraries are installed during setup).

  Activating a server Net-Library allows SQL Server to listen for clients on the corresponding network protocol. The actual network protocols are installed as part of Windows Setup (or through Networks in Control Panel).

- The ODBC Data Source Administrator (available through ODBC in Control Panel) lets you configure ODBC data sources on computers running the Microsoft Windows NT® 4.0, Windows 2000, Windows 95, or Windows 98 operating system.

## See Also

SQL Server Network Utility

# Administering SQL Server

# Client Net-Libraries and Network Protocols

Microsoft® SQL Server™ uses a dynamic-link library (DLL) called a Net-Library to communicate with a particular network protocol. A matching pair of Net-Libraries must be active on client and server computers to support the desired network protocol. For example, to enable a client application to communicate with a specific instance of SQL Server across TCP/IP, the client TCP/IP Sockets Net-Library (DBNETLIB.dll) must be configured to connect to that server on the client computer, and the server TCP/IP Sockets Net-Library (SSNETLIB.dll) must be listening on the server computer.

By themselves, a pair of Net-Libraries cannot support a client/server connection. Both the client and server also must be running a protocol stack supporting the Net-Libraries. For example, if the server TCP/IP Sockets Net-Library is listening on the server computer, and the client TCP/IP Sockets Net-Library is configured to connect to that server on the client computer, the client can only connect to the server if a TCP/IP protocol stack is installed on both computers.

## Multiple Network Protocol Support

The Named Pipes and Multiprotocol Net-Libraries both support multiple network protocols (NW Link IPX/SPX, NetBEUI, and TCP/IP), and will select automatically any supported network protocol that is available. Using either of these Net-Libraries is useful if the client must connect to multiple servers running different network protocols, and you do not want to create and manage configuration entries for each server-network protocol combination.

## Net-Library Setup and Defaults

The client Net-Libraries are installed during SQL Server Setup. You define which client Net-Libraries are used to connect to particular instances of SQL Server using the Client Network Utility. You can specify a default Net-Library for all connections and also define the use of specific Net-Libraries for connecting to specific instances of SQL Server. TCP/IP is the default protocol on clients running the Microsoft Windows NT® 4.0, Windows® 2000, Windows 95, or Windows 98 operating system.

SQL Server can be listening simultaneously on any combination of server Net-Libraries. Use SQL Server Network Library Configuration during or after the Setup program to choose the server Net-Libraries to be activated.

For computers running Windows NT 4.0 or Windows 2000, the default server Net-Libraries are:

- TCP/IP Sockets.
- Named Pipes.

For computers running Windows 98, the default server Net-Libraries are:

- TCP/IP Sockets.
- Shared Memory.

When you install SQL Server client utilities on a workstation, SQL Server Setup installs TCP/IP as the default client protocol.

If most of the servers to which you will be connecting are not configured to support the current default client protocol, you can change the default to another protocol.

For more information about the SQL Server Net-Libraries and the network protocols they support, see Communication Components.

SQL Server 2000 can use the Secure Sockets Layer (SSL) to encrypt all data transmitted between an application computer and an instance of SQL Server on a database computer. Both the client and the server computers must have the proper certificates installed for SSL encryption to function.

Because the Shared Memory Net-Library is used only for intra-computer communications, it is inherently secure and does not need encryption. For more information, see Net-Library Encryption.

# Administering SQL Server

## Configuring Client Network Connections

Each instance of Microsoft® SQL Server™ 2000 listens on a unique set of network addresses so that applications can connect to different instances. SQL Server 2000 clients do not need any specific configuration to connect to an instance of SQL Server 2000. The SQL Server 2000 client components query a computer running one or more instances of SQL Server 2000 to determine the Net-Libraries and network addresses for each instance. The client components then choose a supported Net-Library and address for the connection automatically, without requiring any configuration work on the client. The only information the application must supply is the computer name and instance name.

A SQL Server 2000 default instance listens on the same network addresses as SQL Server version 7.0 or earlier, so applications using earlier versions of the client connectivity components can continue to connect to the default instance with no change. However, named instances of SQL Server 2000 listen on alternative network addresses, and client computers using earlier versions of the client connectivity components must be set up to connect to the alternative addresses.

By default, on computers running Windows NT 4.0 and Windows 2000, an instance of SQL Server listens on the server TCP/IP Sockets and Named Pipes Net-Libraries. On computers running Windows 98, an instance of SQL Server listens on the server TCP/IP Sockets and Multiprotocol Net-Libraries. If the connection is local on a computer (client and server on the same computer), an instance of SQL Server listens on the server Shared Memory Net-Library.

For information about compatibility issues with earlier versions of the client network utility, see SQL Server 2000 and SQL Server version 7.0.

Administering SQL Server

## Configuring Client Net-Libraries

The Client Network Utility is installed as part of Microsoft® SQL Server™ client setup. The application consists of several tabs and dialog boxes in which you can:

- Create client connections to specified servers and save them as configuration entries, which consist of a server alias, a client Net-Library, and any relevant connection parameters, such as a pipe file name or port number. Any saved entry can be used when you want to reconfigure a client connection.

- Change the default client Net-Library.

- Display information about the SQL Server client Net-Libraries currently installed on the system.

- Display the DB-Library version currently installed on the system, and set defaults for DB-Library options.

  **IMPORTANT** The Client Network Utility creates registry entries for the server alias configurations and default client Net-Library. The application does not install either the SQL Server client Net-Libraries or the network protocols. The SQL Server client Net-Libraries are installed during SQL Server Setup. The network protocols are installed as part of Microsoft Windows® Setup (or through Networks in Control Panel). A particular network protocol may not available as part of Windows Setup. For more information about installing these network protocols, see the vendor documentation.

### Viewing Network Library Information

If the Client Network Utility does not display a Net-Library version number and you are using a Net-Library provided by Microsoft, then one of the following problems may have occurred:

- You are using a version that is no longer supported.
- The Client Network Utility cannot find the library in the path.
- A component required by the selected default network library cannot be found.

### To display network library file and version information

# Administering SQL Server

## Setting Up Client Configuration Entries

By default, clients running on the Microsoft® Windows NT® 4.0, Microsoft Windows® 2000, Windows 95, or Windows 98 operating system use the client TCP/IP protocol. You may need to connect using an alternate Net-Library if:

- You need to add a specific client configuration for communicating with a specific server.
- The server with which you want to communicate is configured to listen on another port.

In either case, you must create a configuration entry on the client.

The Client Network Utility lets you configure any of the following network protocols to communicate with a specific server:

- Named Pipes
- TCP/IP Sockets
- Multiprotocol
- NWLink IPX/SPX
- AppleTalk
- Banyan VINES
- Other (for network protocols supplied by a third party)

The Net-Libraries for a protocol must be installed before you can set up a configuration. If the client Net-Library for a network protocol is not installed, it will not be listed on the **Network Libraries** tab. You can also set up configurations for network protocols supplied by a third party, using the **Others** option in the **Add Network Library Configuration** dialog box.

**IMPORTANT** For a client to connect to an instance of Microsoft SQL Server™, it must use a protocol that matches one of the protocols listening on the server. For example, if the client tries to connect to an instance of SQL Server using TCP/IP, and the server has only the NWLink IPX/SPX protocol installed, the client will not be able to establish a connection. In that case, you must use the SQL Server Network Utility on the server to activate the server NWLink IPX/SPX protocol, and SQL Server Network Utility on the client to configure the client NWLink IPX/SPX protocol to connect to that server. Both the client and the server must be running the same network protocol.

Client configuration information is used by SQL Server in the following manner:

- If the server name matches a server specified in the **Server alias configurations** list, then the client connects using the protocol and associated parameters of that configuration.

- If the server name does not match a server specified in the **Server alias configurations** list, then the default protocol is used.

- If no default protocol has been defined, then TCP/IP is used.

### To add a network library configuration

## TCP/IP Sockets Clients

Microsoft® SQL Server™ supports client communication with the TCP/IP network protocol using standard Microsoft Windows® sockets.

**IMPORTANT** The TCP/IP Sockets Net-Libraries have been tested extensively on supported platforms for connecting to instances of SQL Server. If you have purchased a non-TCP/IP Sockets network protocol from a third-party vendor and want to use it to connect to SQL Server, the connection should work if the protocol properly supports TCP/IP Sockets. However, the use of third-party TCP/IP protocols on these platforms is not guaranteed. You can test to see if your sockets are functioning by using the **ping** command from a command prompt.

### Simplified System Administration Using DHCP and WINS

Microsoft Windows NT® version 3.5 or later provides easy administration of large TCP/IP networks by offering the Dynamic Host Configuration Protocol (DHCP) service for automatic TCP/IP configuration, and the Windows Internet Name Service (WINS) for dynamic mapping of network names and addresses. This enables users to operate in large-scale TCP/IP networking environments with little administrative support.

If your network has a DHCP service and WINS, you can use SQL Server instance names to specify a connection to a server. If your network does not have these services, then you should specify the server using the IP address.

**To configure a client to use TCP/IP (Client Network Utility)**

## Named Pipes Clients

Named Pipes clients usually connect using the server instance name on the default pipe. However, if a server is set up to listen on an alternate pipe, the client must also be configured to communicate to that pipe.

### To alias a client to an alternate pipe

# Multiprotocol Clients

The Multiprotocol selection has two key features:

- Automatic selection of an available network protocol to communicate with an instance of Microsoft® SQL Server™.

  This is convenient when you want to connect to multiple servers running different network protocols but do not want to reconfigure the client connection for each server. If the client and server Net-Libraries for TCP/IP Sockets, NWLink IPX/SPX, or Named Pipes are installed on the client and server, the Multiprotocol Net-Library will automatically choose the first available network protocol to establish a connection.

- Client encryption.

  You can enforce encryption over the Multiprotocol Net-Library on clients running on the Microsoft Windows NT® 4.0, Windows® 2000, Windows 95, or Windows 98 operating system to prevent others from intercepting and viewing sensitive data.

The Multiprotocol Net-Library takes advantage of the remote procedure call (RPC) facility of Windows NT 4.0 and Windows 2000, which provides Windows Authentication. For the Multiprotocol Net-Library, clients determine the server address using the server name.

## Usage Considerations

Before using the Multiprotocol Net-Library, consider the following:

- The Multiprotocol Net-Library does not support named instances of SQL Server 2000. You can use the Multiprotocol Net-Library to connect to the default instance of SQL Server on a computer, but you cannot connect to any named instances.

- The Multiprotocol Net-Library does not support server enumeration.

From applications that can list servers by calling **dbserverenum**, you cannot identify servers running an instance of SQL Server and listening on the Multiprotocol Net-Library.

## Multiprotocol Name Resolution

Using the RPC run time, which is called by the Multiprotocol Net-Library, clients can connect to servers using a variety of other protocols. When establishing a connection, the Multiprotocol Net-Library passes the computer name to the RPC run time, which determines the available network protocols and attempts to use each one until a connection is established. Only NWLink IPX/SPX, TCP/IP Sockets, and Named Pipes are tested and supported.

To accomplish the computer name to node connection, the RPC run time uses a naming service compatible with the network protocol used (WINS for TCP/IP, SAP for NWLink IPX/SPX, and Net BIOS broadcasts for Named Pipes). Only the computer name should be specified, because a local RPC database is used to resolve the names over the supported protocols.

## Client Encryption

You can enforce encryption over the Multiprotocol Net-Library on a per-client basis. Only this client's communications are encrypted. Other clients using the Multiprotocol Net-Library that do not have this parameter set do not use encryption.

**To configure a client to use the Multiprotocol Net-Library**

## NetWare Link IPX/SPX Clients

You can configure Microsoft® SQL Server™ clients to communicate with instances of SQL Server by using the NW Link IPX/SPX Compatible Transport, the native protocol of Novell NetWare networks.

The Client Network Utility provides two specification methods for creating an NW Link IPX/SPX network protocol configuration:

- By service name and port number
- By network address, port number, and network number

Consult your network administrator for this information before setting up the configuration.

**To configure a client to use the NWLink IPX/SPX network library**

## AppleTalk ADSP Clients

Microsoft® SQL Server™ can communicate with clients using the AppleTalk ADSP network protocol.

The AppleTalk Net-Library does not support server enumeration. From applications that can list servers by calling **dbserverenum**, you cannot identify instances of SQL Server listening on the AppleTalk Net-Library.

If you experience difficulties establishing connections from clients through AppleTalk, review the error messages listed in the Microsoft Windows® application log and the SQL Server error log, and verify that the AppleTalk Net-Library is loaded correctly.

If the AppleTalk Net-Library is loaded on the server correctly, you see a message in the Windows application log or the SQL Server error log similar to the following:

```
Using 'SSMSADSN.DLL' version '6.00.0.0' to listen on 'servicename'
```

### To configure a client to use the AppleTalk network library

## Banyan VINES Clients

Microsoft® SQL Server™ supports Banyan VINES Sequenced Packet Protocol (SPP) across the Banyan VINES IP network protocol.

Clients running the Microsoft Windows NT® 4.0 or Windows® 2000 operating system require Banyan VINES client software version 5.56(2) or later.

A StreetTalk language PC-based service name has the form: *servicename@group@org* where *servicename* is the StreetTalk PC-based service name used by the server.

**Note**  The service name used by a server must first be created using the MSERVICE program included with Banyan VINES software.

If a client application gives a partial StreetTalk name (a name that does not include the group and organization) as the server name, the VINES SPP Net-Library uses standard VINES services to complete the rest of the StreetTalk name with your login defaults. If no PC-based service or nickname matching the server name is found within the user's own group and organization, the VINES SPP Net-Library looks for a special group named MSSQL within your organization. This allows network administrators to define a group of SQL Servers that are accessible with one-part names from all groups in the same organization.

You can override the default name MSSQL modifying the Banyan VINES properties using the Client Network Utility.

The VINES SPP Net-Library files do not support the use of StreetTalk names that contain embedded spaces, and they use one VINES IP socket per server and one SPP connection per database connection.

**To configure a client to use the Banyan VINES network library**

## VIA Clients

Microsoft® SQL Server™ 2000 introduces new Net-Libraries to be used for highly reliable, fast, efficient data transfer between servers in the same data center. These new Net-Libraries contain functionality for different hardware sets based around the Virtual Interface Architecture (VIA). SQL Server 2000 comes with Net-Libraries to support hardware from Giganet.

**To configure a client to use the VIA network library**

## Other Network Protocol Clients

You can create network configurations for network protocols not listed in the **Add Network Library Configuration** dialog box. To do so, use the **Others** option. To use this option, you must have already installed the client and server Net-Libraries supporting the network protocol.

Use the **Others** option when the client communicates with a server that is listening on a protocol supplied by a third party, such as NLSPY32.

To use this option, you must know:

- The name of the DLL for the network library supplied by the third party.
- Any required parameters and their format.

### To configure a client to use a nonstandard network library

## Administering SQL Server

## Configuring ODBC Data Sources

An ODBC application uses a data source to connect to an instance of Microsoft® SQL Server™. A data source is a stored definition that records:

- The ODBC driver to use for connections specifying the data source.
- The information used by the ODBC driver to connect to a source of data.
- Driver-specific options to be used for the connection. For example, a SQL Server ODBC data source can record the SQL-92 options to use, or whether the drivers should record performance statistics.

Each ODBC data source on a client has a unique data source name (DSN). An ODBC data source for the SQL Server ODBC driver includes all the information used to connect to an instance of SQL Server, plus any essential options.

# Administering SQL Server

## Using the ODBC Data Source Administrator

To configure Microsoft® SQL Server™ ODBC data sources, use the ODBC Data Source Administrator. For more information, see Miscellaneous Utilities.

Using the ODBC Data Source Administrator, you can:

- Display version information for the SQL Server ODBC driver currently installed on the system.
- Add, change, and remove data sources for the SQL Server ODBC driver.

The ODBC Data Source Administrator can create tabs for user, system, and file data sources.

User data sources are specific to the Microsoft Windows NT® 4.0, Windows® 2000, Windows 95, or Windows 98 account that is in effect when they are created. They are not visible to any other login account. They are not always visible to applications running as a service on a computer running Windows NT 4.0 or Windows 2000.

System data sources are visible to all login accounts on a client. They are always visible to applications running as a service on a computer running Windows NT 4.0 or Windows 2000.

File data sources were added with ODBC version 3.0. File data sources are not stored in the system registry; they are stored in a file on the client.

After you choose the type of data source, the ODBC Data Source Administrator starts the SQL Server DSN Configuration Wizard, which guides you through the process of adding an ODBC data source.

### To check the ODBC SQL Server driver version

# Administering SQL Server

## Using ODBC API Functions

You can add an ODBC data source to connect to an instance of Microsoft® SQL Server™ by writing one of the following ODBC API functions into your application.

**SQLConfigDataSource**

User or system data sources can be created by an ODBC application that calls the **SQLConfigDataSource** function with the *fRequest* parameter set to either ODBC_ADD_DSN or ODBC_ADD_SYS_DSN.

**SQLWriteFileDSN**

A file data source can be created by an ODBC application that calls the **SQLWriteFileDSN** function.

**SQLDriverConnect**

If an application specifies the SAVEFILE keyword in the connect string of a successful call to **SQLDriverConnect**, a file data source is created using the information specified in the **SQLDriverConnect** connect string.

**SQLCreateDataSource**

An ODBC application can call the function **SQLCreateDataSource** to display an ODBC dialog box that guides a user through creating a data source.

Data sources that reference the SQL Server ODBC driver contain driver-specific information and options. When a data source is created with either **SQLConfigDataSource** or **SQLWriteFileDSN**, all of the driver-specific information is supplied through keyword-value pairs in a character string passed to the function. When a data source is created using the **ODBC Data Source Administrator** or **SQLCreateDataSource** dialog boxes, the SQL Server DSN Configuration Wizard is invoked to help you specify the driver-specific information.

### See Also

SQLConfigDataSource

SQLDriverConnect

SQLPrepare

# Administering SQL Server

## Adding or Deleting an ODBC Data Source

You must add an ODBC data source to connect your ODBC client to an instance of Microsoft® SQL Server™. You can use the ODBC Data Source Administrator in Control Panel and the SQL Server DSN Configuration Wizard to accomplish this. You can also add a data source programmatically using one of several ODBC API functions; however, these methods are recommended only for advanced users.

ODBC data sources can be deleted in several ways:

- Using the ODBC Data Source Administrator utility in Control Panel,
- Calling **SQLConfigDataSource** with the *fRequest* parameter set to either SQL_REMOVE_DSN or SQL_REMOVE_SYS_DSN.
- Deleting the file containing the data source.

### To add a data source

# Administering SQL Server

# Configuring OLE DB Clients

Configuring OLE DB clients to connect to an instance of Microsoft® SQL Server™ requires making the server name and connection information available to the client (or OLE DB consumer) through an OLE DB provider. SQL Server connections through OLE DB are generally made using either:

- Microsoft OLE DB Provider for SQL Server (SQLOLEDB).
- Microsoft OLE DB Provider for ODBC.

## Connecting SQLOLEDB Clients

SQLOLEDB, the SQL Server native OLE DB provider, exposes interfaces to consumers who want access to data on one or more instances of SQL Server. Using SQLOLEDB allows you to develop an OLE DB consumer optimized for SQL Server databases. Unlike the Microsoft OLE DB Provider for ODBC, which can access data from a number of OLE DB-compliant ODBC applications, you can only use SQLOLEDB with SQL Server. You cannot use the information in an ODBC SQL Server data source name (DSN) to make a connection.

When setting up clients through the Microsoft OLE DB Provider for SQL Server, the client should provide the necessary connection attributes, and either prompt for connection data or supply that data from an OLE DB data source saved in a persisted file.

## Connecting OLE DB Provider for ODBC Clients

Using the Microsoft OLE DB Provider for ODBC allows you to use a single OLE DB provider to connect to multiple ODBC data sources, including SQL Server. However, connecting to SQL Server clients with this provider has more administrative overhead than using the native Microsoft OLE DB Provider for SQL Server.

Usually, when connecting to an instance of SQL Server using the Microsoft OLE DB Provider for ODBC, the information you need is created through the ODBC

Data Source Administrator and saved in a SQL Server ODBC DSN (as either a user, system, or file DSN). Therefore, you can code your application to use a SQL Server DSN to make a connection.

## See Also

Programming OLE DB SQL Server Applications

# Administering SQL Server

# DB-Library Options

The Client Network Utility includes file information on the DB-Library installed on your computer, and options for setting DB-Library preferences. Options include:

- **DB-Library information**

  Includes the file name, version, date, and size of the currently installed DB-Library. This type of information is useful if you have a technical support issue with DB-Library.

- **Automatic ANSI to OEM conversion**

  Enables DB-Library to convert characters from OEM to ANSI when communicating with an instance of Microsoft® SQL Server™, and from ANSI to OEM when communicating with the client from an instance of SQL Server. By default, the **Automatic ANSI to OEM conversion** option is selected for clients running Microsoft Windows NT® 4.0, Windows® 2000, Windows 95, or Windows 98. For more information, see Using the DB-Library Automatic ANSI to OEM Conversion Option.

- **Use international settings**

  Enables DB-Library to get date, time, and currency formats from the system rather than using hard-coded parameters or parameters specified in Sqlcommn.loc. By default, the **Use international settings** option is selected for clients running Windows NT 4.0, Windows 2000, Windows 95, or Windows 98.

### To set DB-Library conversion preferences

## Administering SQL Server

## Using the DB-Library Automatic ANSI to OEM Conversion Option

When enabled, the **Automatic ANSI to OEM conversion** option converts a character set when communicating from:

- ANSI clients to OEM servers.
- OEM clients to ANSI servers.

This option is enabled by default. When the option is disabled, conversion of characters is disabled for all connections.

If the client code page is different from the code page on the instance of SQL Server, then the character set should be converted. Microsoft Windows NT®, Microsoft Windows® 2000, Windows 95, and Windows 98 have both an ANSI and an OEM character set, which are set during installation. For U.S. English, Windows NT, Windows 2000, Windows 95, and Windows 98 use the default ANSI character set, code page 1252, and the default OEM character set, code page 437. Windows 3.*x* runs as an extension to MS-DOS, and has only the default ANSI character set of code page 1252. The Windows NT 4.0 and Windows 2000 Console is internally Unicode, which behaves like an OEM character set.

Any clients running Windows NT 4.0, Windows 2000, Windows 95, or Windows 98 are considered ANSI clients. Console-based applications, such as the **isql** utility, are considered OEM clients.

A server with the default code page of 12*xx*, such as 1252, is considered to be an ANSI server; with any other code page, it is considered to be an OEM server (for example, code page 850 or 437).

Although default code page values exist for both ANSI and OEM, the client's current operating system code page determines conversion values when characters are translated.

## Checking the Validity of Saved Data

You can use the Transact-SQL string function ASCII(*char_expr*) to reveal a character saved in a Microsoft® SQL Server™ database. You can also use the ASCII(*column_name*) function to reveal the ASCII value for a particular column in the database.

Another way to reveal the code page is to set **Automatic ANSI to OEM conversion** to OFF and query the data from SQL Query Analyzer.

For example, assume you have saved the character "±" on a server using the OEM code page 437. If you select this data from SQL Query Analyzer (which is using ANSI code page 1252) when **Automatic ANSI to OEM conversion** is on, you see the "±" character. The OEM 437 "±" (which has an ASCII value of 241) has been converted to the ANSI 1252 "±" (ASCII 177). However, if you select the data from SQL Query Analyzer when **Automatic ANSI to OEM conversion** is off, you see the "ñ" character (ASCII 241). The OEM 437 ASCII value of 241 has been directly replaced by the ANSI 1252 ASCII value of 241.

| Automatic ANSI to OEM conversion | OEM code page 437 | ANSI code page 1252 |
|---|---|---|
| **ON** | ± (ASCII 241) | ± (ASCII 177) |
| **OFF** | ± (ASCII 241) | ñ (ASCII 241) |

## Code Page Incompatibilities

When a character in one code page is unavailable on another and conversion occurs, the character is converted to its closest equivalent character in the other code page. For example, ASCII 156 (œ) in code page 1252 is converted to ASCII 111 (o) in code page 437 because this is the most similar character in the code page 437. When you convert this ANSI character back to code page 1252, the result is ASCII 111 (o) because ASCII 111 (o) exists in both code pages. The original 1252 character (œ) is lost. This means that incorrect data is saved in the database if the character exists in one code page but not the other, and **Automatic ANSI to OEM conversion** is turned on.

| Conversion | ANSI code page 1252 | OEM code page 437 |
|---|---|---|
| ANSI to OEM | œ (ASCII 156) | Does not exist. Substitutes o (ASCII 111). |
| OEM to ANSI | o (ASCII 111) | o (ASCII 111). |

When you save data on a server with a code page different from the code page that is used by the clients, be sure to test the data for accuracy. If possible, choose characters that convert easily between ANSI and OEM.

## Administering SQL Server

# Automating Administrative Tasks

Automated administration is the programmed response to a predictable administrative responsibility or server event. By using automated administration, you can free time to perform administrative tasks that lack predictable or programmable responses and require creativity.

For example, if you want to back up all the company servers every weekday after hours, you can create a job to perform this task. Schedule the job to run at the required time. If the job encounters a problem, SQL Server Agent can record the event and page you.

If you are running multiple instances of Microsoft® SQL Server™, use multiserver administration to automate tasks. For more information, see Multiserver Administration.

To automate administration:

- Establish which administrative responsibilities or server events occur regularly and can be administered programmatically.
- Define a set of jobs, alerts and operators by using SQL Server Enterprise Manager, Transact-SQL scripts, or SQL-DMO objects. For more information, see Creating Jobs.
- Run the SQL Server Agent service.

## Automatic Administration Components

Jobs, alerts, and operators are the three main components of automatic administration.

## Jobs

A job is a specified series of operations performed sequentially by SQL Server Agent. Use jobs to define an administrative task that can be executed one or more times and monitored for success or failure each time it executes. Execute jobs:

- On one local server or on multiple remote servers.
- According to one or more schedules.
- By one or more alerts.

For more information, see Creating Jobs.

## Alerts

An alert signals the designated operator that an event has occurred. For example, an event can be a job starting or system resources reaching a threshold. You define the conditions under which an alert is generated. You also define which of the following actions the alert takes:

- Notify one or more operators.
- Forward the event to another server.
- Execute a job.

For more information, see Defining Alerts.

## Operators

An operator is an individual responsible for the maintenance of one or more instances of SQL Server. In some enterprises, operator responsibilities are assigned to one individual. In larger enterprises with multiple servers, many individuals share operator responsibilities.

Operators are notified of alerts in one or more of the following ways:

- E-mail

  You can define the e-mail alias of an operator as the alias for a group of individuals. In this way, all members of that alias are notified at the same time.

- Pager (through e-mail)

- **net send**

For more information, see Defining Operators.

# Administering SQL Server

## Multiserver Administration

Multiserver administration is the process of automating administration across multiple instances of Microsoft® SQL Server™.

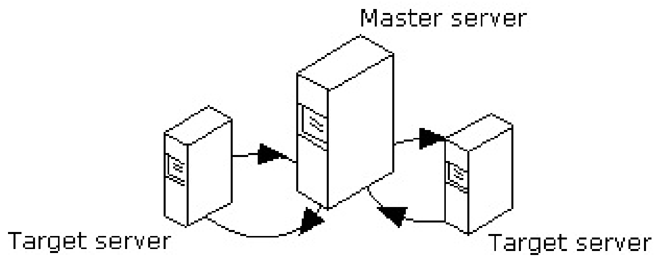
Use multiserver administration if you:

- Manage two or more servers.

- Schedule information flows between enterprise servers for data warehousing.

With multiserver administration, you must have at least one master server and at least one target server. A master server distributes jobs to and receives events from target servers. A master server stores the central copy of job definitions for jobs run on target servers. Target servers connect periodically to their master server to update their list of jobs to perform. If a new job exists, the target server downloads the job and disconnects from the master server. After the target server completes the job, it reconnects to the master server and reports the status of the job.

Multiserver Administration Configuration



For example, if you administer departmental servers across a large corporation, you can define:

- One backup job with job steps.

- Operators to notify in case of failure.

- An execution schedule.

Write this backup job one time on the master server and then enlist each departmental server as a target server. In this way, all the departmental servers run the same backup job even though you defined it only one time.

Multiserver administration features are intended for members of the **sysadmin** role. However, a member of the **sysadmin** role on the target server cannot edit the operations performed on the target server by the master server. This security measure prevents job steps from being accidently deleted and operations on the target server from being interrupted.

## Creating a Multiserver Environment

To create a multiserver environment, use the Make Master Server Wizard. The wizard takes you through the following steps:

- Checking the security settings for the SQL Server Agent service and the SQL Server service on all servers that will become target servers.

  It is recommended that both services be running in Microsoft Windows NT® 4.0 or Windows® 2000 domain accounts.

- Creating a master server operator (MSXOperator) on the master server.

  The MSXOperator is the only operator that can receive notifications for multiserver jobs.

- Starting the SQL Server Agent service on the master server.

- Enlisting one or more servers as target servers.

If you have a large number of target servers, it is recommended that you define your master server on a nonproduction server, so production is not slowed by target server traffic. If you also forward events to this server, you can centralize administration on one server. For more information, see Managing Events.

When creating a multiserver environment, consider the following:

- Each target server reports to only one master server. You must defect a target server from one master before you can enlist it into a different one.

- The master and target servers must be running on the Windows NT 4.0 or Windows 2000 operating system.

- When changing the name of a target server, you must defect it before changing the name and reenlist it after the change.

- If you want to dismantle a multiserver configuration, you must defect all the target servers from the master server.

## To make a master server

# Administering SQL Server

## Configuring the SQLServerAgent Service

SQLServerAgent is a Microsoft® Windows NT® 4.0 or Windows® 2000 service that executes jobs, monitors Microsoft SQL Server™, and fires alerts. SQLServerAgent is the service that allows you to automate some administrative tasks. As such, you must start the SQLServerAgent service before your local or multiserver administrative tasks can run automatically. SQL Server Agent is also supported on the Microsoft Windows 98 operating system, but SQL Server Agent cannot be used with Windows Authentication when run on Windows 98.

You can specify some configuration options for SQL Server Agent during SQL Server installation. The full set of configuration options is available from within SQL Server Enterprise Manager only.

**Note** You can click SQL Server Agent in the console tree of SQL Server Enterprise Manager to administer jobs, operators, alerts, and the SQL Server Agent service.

### See Also

Security Levels

Starting SQL Server Manually

## Administering SQL Server

## Starting SQLServerAgent Service

The service startup account defines the Microsoft® Windows NT® 4.0 or Windows® 2000 account in which the SQLServerAgent service runs. This information defines the network permissions of the SQLServerAgent service. These are the available options:

- **System account**

  The system account is the built-in local system administrator account. It is a member of the **Administrators** group on the local computer, and is therefore a member of the **sysadmin** role within Microsoft SQL Server™.

  Use **System account** if your jobs require resources from the local system only.

- **This account**

  **This account** enables you to specify in which Windows NT 4.0 or Windows 2000 domain account SQLServerAgent runs. The domain account that you specify must be a member of the **sysadmin** role on the local instance of SQL Server.

  Use **This account** if:

  - You want to forward events to the application logs of other computers running on the Windows NT 4.0 or Windows 2000 operating system.

  - Your jobs require resources across the network, including replication resources.

  - You want to notify operators through e-mail or pagers.

    **Note** If the Microsoft Exchange or Microsoft Outlook® client is configured to deliver mail to a personal folder that is password-protected, SQL Server Agent cannot start its mail session. To avoid this, remove the password protection from the .pst file.

If you are running SQLServerAgent in an account other than a Windows NT 4.0 or Windows 2000 domain account, the following will occur:

- CmdExec and ActiveScripting steps of jobs owned by nonsysadmins will fail.
- The autorestart features in SQLServerAgent will not work.
- On-idle job schedules will not allow the job to run.

For best results, use a Windows NT 4.0 or Windows 2000 domain account that has sufficient permissions across the domain to access information necessary for SQL Server Agent job execution. You can change the SQLServerAgent service account to a non-Windows NT 4.0 administrator account. However, the Windows NT 4.0 account must be a member of the **sysadmin** fixed server role to run SQL Server Agent.

## To set the service startup account for SQL Server Agent

# Administering SQL Server

## Connecting to SQL Server

Two methods define how the SQL Server Agent service connects to an instance of a local Microsoft® SQL Server™. Regardless of the method you select, the account must have system administrator permissions within SQL Server.

- **Use Windows Authentication**

  This method forces the SQL Server Agent service to connect to an instance of SQL Server using the Microsoft Windows NT® 4.0 or Windows® 2000 domain account you defined as the service startup account.

- **Use SQL Server Authentication**

  This method forces the SQL Server Agent service to connect to an instance of SQL Server using a SQL Server authenticated login. Only logins that are members of the **sysadmin** role are available.

  Select **Use SQL Server Authentication** if you are running SQLServerAgent on a server that is not running on the Windows NT 4.0 or Windows 2000 operating system.

Both options allow you to set a time limit for logins. If the SQLServerAgent service requires more time to connect to the local instance of SQL Server than the duration you have specified, the login session will time out. You can specify a value from 5 through 45 seconds for the login time-out.

**To set the SQL Server connection**

# Administering SQL Server

## Specifying a SQL Server Alias

By default, SQL Server Agent connects to an instance of Microsoft® SQL Server™ over named pipes using dynamic server names that require no additional client configuration.

You must specify a server connection alias only when:

- You are using a nondefault network transport to connect to an instance of SQL Server.

- You are connecting to an instance of SQL Server that listens on an alternate named pipe.

### To set a SQL Server alias

# Administering SQL Server

## Using the SQL Server Agent Error Log

SQL Server Agent creates an error log that, by default, records warnings and errors. The following types of messages are displayed in the SQL Server Agent error log:

- Warning messages that provide information about potential problems, such as, "Job test was deleted while it was executing."

- Error messages that usually require intervention by a system administrator to resolve, such as, "Unable to start mail session." Error messages can be sent to a specific user or computer by network popup.

SQL Server maintains up to nine SQL Server Agent error logs. Each archived error log has an extension indicating the relative age of the error log. For example, an extension of .1 indicates the newest archived error log and an extension of .9 indicates the oldest archived error log.

By default, execution trace messages are not written to the SQL Server Agent error log, because they can fill it, thereby reducing your ability to select and analyze more difficult errors. As the SQL Server Agent error log adds an additional processing load to the server, consider what value you attain by capturing execution trace messages to this error log. Generally, it is best to capture all messages only when you are debugging a specific problem.

When SQL Server Agent is stopped, you can modify the location of the error log. When the Microsoft® SQL Server™ error log is empty, it cannot be viewed.

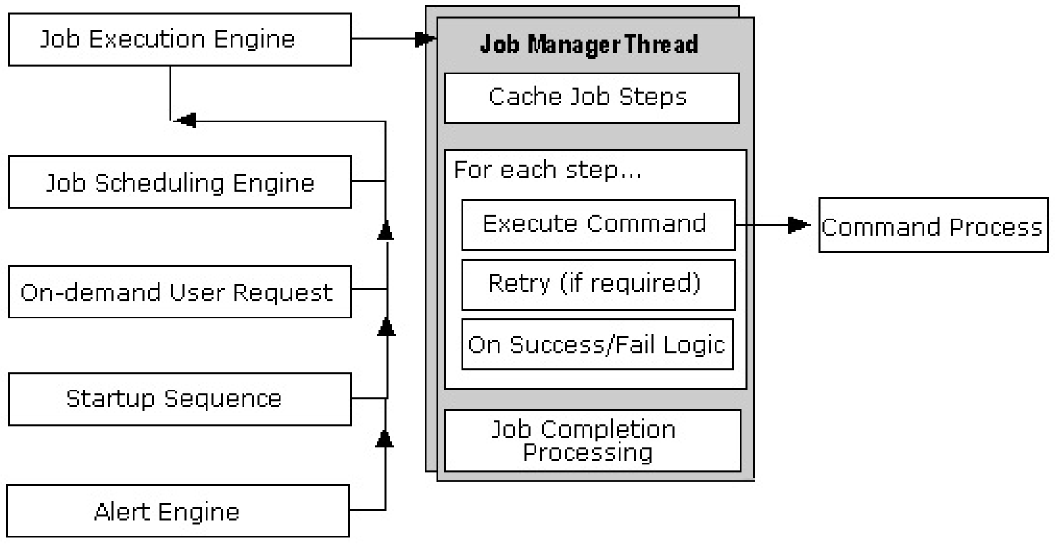
**To view the SQL Server Agent error log**

# Administering SQL Server

## Implementing Jobs

Using SQL Server Agent jobs, you can automate administrative tasks and run them on a recurring basis. You can run a job manually or schedule it to run in response to schedules and alerts.

This illustration shows the job execution and job step processing that occurs when a job is run by SQL Server Agent.



Jobs can be written to run on the local instance of Microsoft® SQL Server™ or on multiple servers. To run jobs on multiple servers, you must set up at least one master server and one or more target servers.

Anyone can create a job, but a job can be edited only by its owner or members of the **sysadmin** role.

## Administering SQL Server

## Creating Jobs

A job is a specified series of operations performed sequentially by SQL Server Agent. A job can perform a wide range of activities, including running Transact-SQL scripts, command line applications, and Microsoft® ActiveX® scripts. Jobs can be created to run tasks that are often repeated or schedulable, and they can automatically notify users of job status by generating alerts.

### To create a job

# Administering SQL Server

## Creating Job Steps

A job step is an action that the job takes on a database or a server. Every job must have at least one job step. Job steps can be operating system commands, Transact-SQL statements, Microsoft® ActiveX® scripts, or replication tasks.

### CmdExec Job Steps

CmdExec job steps are operating system commands or executable programs ending with .bat, .cmd, .com, or .exe.

When you create a CmdExec job step, you must specify:

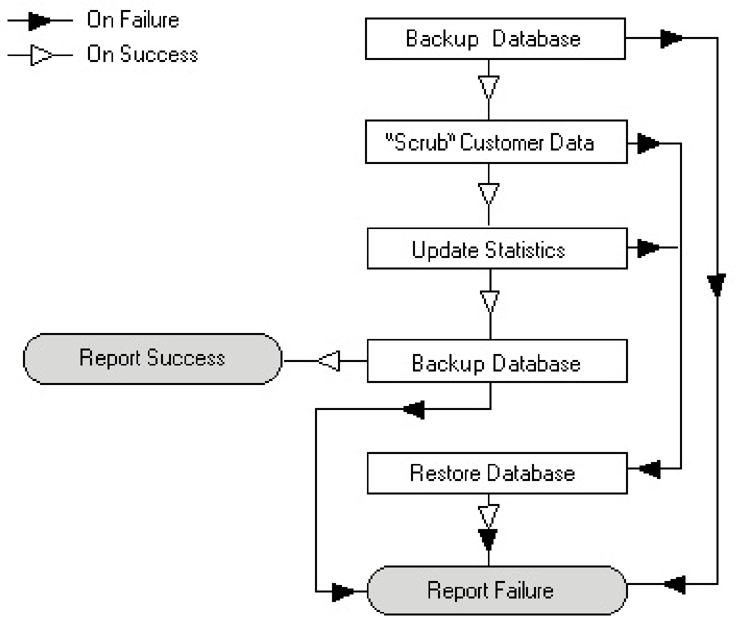
- The process exit code returned if the command was successful.
- The **CmdExec** command (for example, C:\Program Files\Microsoft SQL Server\80\Tools\Binn\Osql.exe\E\Q "sp_who").
- A full path to all executables.

**To create a CmdExec job step**

# Administering SQL Server

## Handling Multiple Job Steps

If your job has more than one job step, you must impose an order of execution on the job steps. This is called control-of-flow. You can add new job steps and rearrange the flow of job steps at any time. The changes take effect the next time the job is run. This illustration shows a control-of-flow for a database backup job.



You define a control-of-flow action for the success and failure of each job step. You must specify the action to be taken when a job step succeeds and when a job step fails. You can also define the number of and interval between retry attempts for failed job steps.

Job steps must be atomic. A job cannot pass Boolean values, data, or numeric values between job steps. You can pass values from one Transact-SQL job step to another by using permanent tables or global temporary tables. You can pass values from one CmdExec job step to another by using files.

**Note** If you create looping job steps (job step 1 is followed by job step 2, then job step 2 returns to job step 1), a warning message appears when the job is created using SQL Server Enterprise Manager.

SQL Server Agent records job and job step execution information in the job history.

### To set job step success or failure flow

## Administering SQL Server

## Scheduling Jobs

Scheduling your administrative jobs is one way to automate administrative tasks. You can schedule local jobs or multiserver jobs. You can define a job to run:

- Whenever SQL Server Agent starts.
- Whenever CPU utilization of the computer is at a level you have defined as idle.
- One time, at a specific date and time.
- On a recurring schedule.
- In response to an alert.

You can also execute a job manually; scheduling jobs is optional.

**Note** Only one instance of the job can be run at a time. If you execute a job manually while it is running as scheduled, SQL Server Agent refuses the request.

All jobs are enabled by default. To prevent a job from running according to its schedule, you must disable the schedule. The job can still execute in response to an alert or when a user runs the job manually.

SQL Server Agent automatically disables schedules that are no longer current. If you edit the schedule after it has been disabled by SQL Server Agent, you must explicitly reenable it. Schedules are disabled if:

- They are defined to run one time, at a specific date and time, and that time has passed.
- They are defined to run on a recurring schedule, and the end date has passed.

## CPU Idle Schedules

To maximize CPU resources, you can define a CPU idle condition for SQL Server Agent. SQL Server Agent uses the CPU idle condition setting to determine the most advantageous time to execute jobs.

For example, you can schedule a daily backup job to occur during CPU idle time and slow production periods.

Before you define jobs to execute during CPU idle time, determine how much CPU the job requires. You can use SQL Profiler or System Monitor (Performance Monitor in Windows NT 4.0) to monitor server traffic and collect statistics. You can use the information you gather to set the CPU idle time percentage.

Define the CPU idle condition as a percentage below which the average CPU usage must remain for a specified time. Next, set the amount of time. When this time has been exceeded, SQL Server Agent starts all jobs that have a CPU idle time schedule.

**To schedule a job**

# Administering SQL Server

## Specifying Job Responses

You can define job responses to occur after a job completes. Typical job responses include:

- Notifying the operator by using e-mail, electronic paging, or a **net send** message.

  Use one of these job responses if the operator must perform a follow-up action. For example, if a backup job completes successfully, the operator must be notified to remove the backup tape and store it in a safe location.

- Writing an event message to the Microsoft® Windows® application log.

  You can use this response only for failed jobs.

- Automatically deleting the job.

  Use this job response if you are certain that you will not need to rerun this job.

### To notify an operator of job status

Administering SQL Server

## Running Jobs

You may need to execute a job often, but not regularly. In such cases, you can write a job once and execute it manually as needed. You can also execute jobs manually that have been assigned a schedule. For example, even though you have scheduled a **master** database backup job to occur in the evening, you may want to back up the database immediately after making changes to the system tables.

If a job has started according to its schedule, you cannot start another instance of that job on the same server until the scheduled job has completed. In multiserver environments, every target server can run one instance of the same job simultaneously.

You can disable a job if you do not want it to run. You can also stop a job while it is executing. In most cases, when you issue a stop command, the current job step is canceled and any retry logic is ignored. Some job steps, such as long-running Transact-SQL statements (BACKUP) or some DBCC commands, may not respond quickly to stop requests. When you stop a job, a job-canceled entry is recorded in the job history.

### Multiserver Job Processing

You can run a multiserver job on one or more target servers. Each target server connects periodically to the master server, downloads an actual copy of any new jobs assigned to the target server, then disconnects from the master server. The target server stores a complete copy of the job locally, then reconnects to the master server to upload the job outcome status.

**Note** If the master server is inaccessible when the target server attempts to upload job status, the job status is spooled until the master server is accessible again.

**To start a job**

# Administering SQL Server

## Modifying and Viewing Jobs

After you have created a job, you can view the job definition. After you have executed a job, you can view its history. If the requirements of a job change, you can modify the job so that it performs differently.

**Note** The job must have been executed at least one time for there to be a job history. You can limit the total size and the size per job of the job history log.

You can modify:

- Response options.
- Schedules.
- Job steps.
- Ownership.
- Job category.
- Target servers (multiserver jobs only).

If you make changes to multiserver job definitions outside of SQL Server Enterprise Manager, you must post the changes to the download list so that target servers can download the updated job again. To ensure that target servers have the most current job definitions, post an INSERT instruction after you update the multiserver job:

```
EXECUTE sp_post_msx_operation 'INSERT', 'JOB', '<job id>'
```

You must notify the target servers manually that the job has been modified using the above command after you finish modifying the schedules and steps of a multiserver job using any of the following procedures:

- sp_add_jobstep
- sp_update_jobstep
- sp_delete_jobstep
- sp_add_jobschedule
- sp_update_jobschedule
- sp_delete_jobschedule

  **Note** It is not necessary to call **sp_post_msx_operation** after you call **sp_update_job** or **sp_delete_job**, because these stored procedures post the required changes to the download list automatically.

### To view a job

# Administering SQL Server

## Scripting Jobs Using Transact-SQL

You can generate Transact-SQL scripts to create the jobs that you have defined. With job scripting, you can:

- Control versions of job creation source code.
- Migrate jobs from test into production.
- Script alerts and operators.

It is also possible to create a script on a computer running an instance of Microsoft® SQL Server™ 2000 that can be run on a computer running an instance of SQL Server version 7.0. If you choose to create a SQL Server 7.0 compatible script, certain SQL Server 2000 features are ignored, such as:

- Column level collation.
- User-defined functions, extended properties.
- INSTEAD OF triggers on tables and views.
- Indexes on views (indexed views).
- Indexes on computed columns.
- Descending indexes.
- Reference permissions on views.

  This option is only available on an instance of SQL Server 2000.

### To script jobs using Transact-SQL

# Administering SQL Server

## Responding to Events

Microsoft® SQL Server™ events are written to the Microsoft Windows® application log. You can define an alert on one or more events to specify how SQL Server Agent should respond.

SQL Server Agent monitors the Windows application log for SQL Server events. When events that you have defined for action occur, SQL Server Agent responds automatically according to your specifications. For example, if an event of severity 17 occurs, you can specify that an operator be notified immediately.

Automated event response is called alerting. When an event occurs, SQL Server Agent compares the event details against the alerts defined by the SQL Server administrator. If it finds a match, SQL Server Agent performs the defined response.

You can define alerts to:

- Notify operators.
- Execute a job.
- Forward the event to the Windows application log on another server.

# Administering SQL Server

## Defining Operators

The primary attributes of an operator are name and contact information. It is recommended that you define operators before you define alerts. You must set up one or more of the following in order to notify an operator:

- For e-mail, a MAPI-1-compliant e-mail client.

  SQL Server Agent requires a valid mail profile in order to send e-mail. Examples of MAPI-1 clients include Microsoft® Outlook® and Microsoft Exchange client.

- For paging, third-party pager-to-e-mail software and/or hardware.

  You need these to use the pager notification features.

- To use **net send** notifications, you must be running the Microsoft Windows NT® 4.0 or Windows® 2000 operating system.

### Naming an Operator

Every operator must have a name. Operator names must be unique and can be no longer than 128 characters.

### Providing Contact Information

An operator's contact information defines how the operator is notified. Operators can be notified by e-mail, pager, or **net send**:

- E-mail Notification

  SQL Server Agent establishes its own mail session using the mail profile information supplied in the **SQL Agent Properties** dialog box.

- Pager Notification

  Paging is implemented using e-mail. To set up pager notification, you must install on the mail server software that processes inbound mail and converts it to a pager message. The software can take one of several approaches, including:

- Forwarding the mail to a remote mail server at the pager provider's site.

  The pager provider must offer this service, although the software required is generally available as part of the local mail system. For more information, see the pager documentation.

- Routing the mail by way of the Internet to a mail server at the pager provider's site.

  This is a variation on the first approach.

- Processing the inbound mail and dial using an attached modem.

  This software is proprietary to pager service providers. The software acts as a mail client that periodically processes its inbox either by interpreting all or part of the e-mail address information as a pager number, or by matching the e-mail name to a pager number in a translation table.

If all of the operators share a pager provider, you can use SQL Server Enterprise Manager to specify any special e-mail formatting required by the pager-to-e-mail system. The special formatting can be a prefix or a suffix:

- Subject line

- Cc line

- To line

  **Note** If you are using a low-capacity alphanumeric paging system (for example, limited to 64 characters per page), you can shorten the text sent by excluding the error text from the pager notification.

- **net send**

  The **net send** notification method specifies the recipient (computer or user) of a network message. This method is not supported on the

Windows 98 operating system.

## Designating a Fail-Safe Operator

The fail-safe operator is notified about an alert after all pager notifications to the designated operators have failed. For example, if you have defined three operators for pager notifications and none of the designated operators can be paged, the fail-safe operator is notified.

The fail-safe operator is notified when:

- The operator(s) responsible for the alert could not be paged.

  Reasons for this include incorrect pager addresses and off-duty operators.

- SQL Server Agent cannot access system tables in the **msdb** database.

  The **sysnotifications** system table specifies operator responsibilities for alerts.

Because the fail-safe operator is a safety feature, you cannot delete the operator assigned to fail-safe duty without reassigning fail-safe duty to another operator or deleting the fail-safe assignment.

**To create an operator**

# Administering SQL Server

## Modifying and Viewing Operators

Because operators are individuals with changing responsibilities and job schedules, you may need to update operator information. After an operator has been created, you can:

- View an operator's information.

  You can view the alerts for which the operator is responsible. You can view the dates of the most recent attempts by SQL Server Agent to notify the operator.

- Edit an operator's information.

  You can edit the notification addresses, pager on-duty schedule, assigned alerts, and notification methods.

- Change an operator's availability.

  By default, operators are available to receive notifications (enabled) as soon as they are defined. You can specify the operator as unavailable to receive notifications (disabled) when you create it or at any time thereafter.

  For example, if an individual who is assigned operator responsibilities goes on vacation, you can disable the operator. The alerts assigned to that operator and the notification methods for those alerts have not changed; only the operator's availability to respond to alerts has been changed. When the individual returns from vacation, you do not need to redefine the operator; rather, you reenable the operator.

- Delete an operator.

  You can delete an operator when the individual no longer has operator responsibilities. When you delete an operator, you also delete all of the operator's alert notifications. You cannot remove an operator that has been assigned to be the fail-safe operator. You first must remove the fail-safe duty from the operator or reassign the fail-safe duty to another operator before you can delete the operator.

## To view information about an operator

Administering SQL Server

## Alerting Operators

You can choose the operators to be notified in response to an alert. You can assign rotating responsibilities for operator notification by using pagers. For example, if one or more defined alerts occur on Monday, Wednesday, or Friday, Mary is notified. If those alerts occur on Tuesday, Thursday, or Saturday, Joe is notified. If Mary or Joe cannot be notified on the respective day, or if the alert occurs on Sunday, the fail-safe operator is notified.

You notify operators using one or more of these methods:

- E-mail
- Pager
- **net send**

The SQLServerAgent service uses a mail session that is exclusive to SQL Server Agent activities. If you are using a SQL Mail session for the MSSQLServer service, it is recommended that SQL Server Agent and Microsoft® SQL Server™ use the same Microsoft Windows NT® 4.0 or Windows® 2000 domain user account. This allows both mail sessions to use the same mail profile. If SQL Server Agent and SQL Server use separate domain user accounts, you must configure a mail profile for each service.

**To define the response to an alert**

# Administering SQL Server

## Defining Alerts

Errors and messages, or events, are generated by Microsoft® SQL Server™ and entered into the Microsoft Windows® application log. SQL Server Agent reads the application log and compares events to alerts that you have defined. When SQL Server Agent finds a match, it fires an alert.

By default, the following SQL Server events are logged in the Windows application log:

- Severity 19 or higher **sysmessages** errors.

  You can use **sp_altermessage** to designate specific **sysmessages** errors as "always logged" to log error messages with a severity lower than 19.

- Any RAISERROR statement invoked by using the WITH LOG syntax.

  RAISERROR WITH LOG is the recommended way to write to the Windows application log from an instance of SQL Server.

- Any application logged by using **xp_logevent**.

  **Note** Make sure that the Windows application log is of sufficient size to avoid losing SQL Server event information.

Alerts must be defined before notifications can be sent. The primary attributes of an alert are name and event or performance condition specification.

### Naming an Alert

Every alert must have a name. Alert names must be unique and can be no longer than 128 characters.

### Selecting an Event

You can specify an alert to occur in response to one or more events. You specify the set of events to trigger an alert according to:

- Error number.

SQL Server Agent fires an alert when a specific error occurs.

- Severity level.

  SQL Server Agent fires an alert when any error of the specific severity occurs.

- Database.

  Specifies a database in which the event occurred if you want to restrict the alert.

- Event text.

  Specifies a text string in the event message if you want to restrict the alert.

## Selecting a Performance Condition

You can specify a performance condition to monitor by firing an alert when the performance threshold is reached. To set a performance condition you must define the following:

- Object.

  The area of SQL Server performance to be monitored.

- Counter.

  The attribute with the area to be monitored. Performance data is sampled periodically, which can lead to a small delay (a few seconds) between the threshold being reached and the performance alert firing.

- Instance.

  The specific instance (if any) of the attribute to be monitored.

- Alert if counter/value.

  The behavior the counter or counter instance must exhibit for the alert to fire.

## Creating a User-defined Event Message

You can create user-defined event messages if you have special event tracking needs that are not addressed by standard SQL Server event messages. User-defined event messages generate error numbers greater than 50,000. Additionally, you can assign them a severity level.

User-defined event messages must be unique and have a unique error number. They can each have a unique language.

**Note** When using SQL Server Enterprise Manager, you should select the **Write to Windows NT application event log** option. By default, user-defined messages with severities less than 19 are not sent to the Windows application log when they occur and therefore do not trigger SQL Server Agent alerts.

If you administer a multiple language SQL Server environment, create user-defined messages in each of the languages you support. For example, if you are creating a new event message that will be used on both an English and a German server, use the same event number for both, but assign a different language for each.

### To create an alert using an error number

# Administering SQL Server

## Modifying and Viewing Alerts

After you create an alert, you can:

- View the alert's information.

  You can view the characteristics of an alert, the date of the most recent occurrence of and response to the alert, as well as the number of times the alert has occurred since the count was last reset.

- Modify the alert's information.

  You can add new operators, reset the number of times an alert has occurred, disable an alert, or change a database.

- Delete the alert.

  You can delete an alert that is no longer needed. When you delete an alert, you also delete all of the alert's operator notifications.

### To view information about an alert

# Administering SQL Server

# Copying Operators or Alerts to Other Servers

You can generate a Transact-SQL script to create one or all of the operators or alerts that you have defined. If the same group of operators is responsible for responding to the same alerts on other servers, you can save time by automatically scripting all the predefined operators and alerts, and then copying to those servers.

### To script operators using Transact-SQL

Administering SQL Server

# Managing Events

You can forward all Microsoft® SQL Server™ event messages that meet or exceed a specific error severity level to one instance of SQL Server. The forwarding server is a dedicated server that can also be a master server. You can use event-forwarding to enable centralized alert management for a group of servers. In this way, you can reduce the workload on heavily used servers.

In a multiserver environment, it is recommended that you designate the master server as the alerts management server.

## Advantages

Advantages of setting up an alerts management server include:

- Centralization.

  Centralized control and a consolidated view of the events of several instances of SQL Server is possible from a single server.

- Scalability.

  Many physical servers can be administered as one logical server. You can add or remove servers to this physical server group as needed.

- Efficiency.

  Configuration time is reduced, because you need to define alerts and operators only once on one server.

## Disadvantages

Disadvantages of setting up an alerts management server include:

- Increased traffic.

  Forwarding events to an alerts management server can increase network traffic, although this can be moderated by restricting event-forwarding to severity events only above a designated level.

- Single point of failure.

- Server load.

  Handling alerts for the forwarded events causes an increased processing load at the alerts-forwarding server.

## Guidelines

When configuring event forwarding, follow these guidelines:

- Avoid running critical or heavily used applications on the alerts-forwarding server.

- Avoid configuring many servers to share the same forwarding server. If congestion results, reduce the number of servers that use a particular alerts management server.

  The servers that are registered within SQL Server Enterprise Manager constitute the list of servers available to be chosen by that server as the alerts-forwarding server.

- Define alerts that require a server-specific response on the local instance of SQL Server instead of forwarding them.

  The alerts-forwarding server views all the servers forwarding to it as a logical whole. For example, an alerts-forwarding server responds in the same way to a 605 event from server A and a 605 event from server B.

- After configuring your alert system, periodically check the Microsoft Windows® application log for SQL Server Agent events.

  Failure conditions encountered by the alerts engine are written to the local Windows application log with a source name of SQL Server Agent. For example, if SQL Server Agent cannot send an e-mail notification as it has been defined, an event is logged in the application log.

If a locally defined alert is disabled and an event occurs that would have caused the alert to fire, the event is forwarded to the alerts-forwarding server (if it satisfies the alert forwarding condition). This allows local overrides (alerts defined locally that are also defined at the alerts forwarding server) to be turned off and on as needed by the user at the local site. You can also request that events always be forwarded, even if they are handled by local alerts.

### To designate an events forwarding server

Administering SQL Server

## Monitoring the Environment

SQL Server Agent monitors itself and the Microsoft® SQL Server™ service.

### Self-Monitoring

SQL Server Agent starts the **xp_sqlagent_monitor** extended stored procedure (SQL Server Agent Monitor) to monitor the SQLServerAgent service to ensure that it is available to execute scheduled jobs, raise alerts, and notify operators. If the SQLServerAgent service terminates unexpectedly, the SQL Server Agent Monitor restarts the service.

### Restarting the SQL Server Service

SQL Server Agent can restart the local instance of SQL Server if it has terminated for reasons other than a typical shutdown. Automatic restart is enabled by default. SQL Server Agent restarts the instance of SQL Server when it detects abnormal termination. This allows an alert to be set on this event.

**Note** If you are using SQL Server 2000 failover clustering, you must ensure auto-restart is disabled in order for failover clustering to work.

**To set job execution shutdown**

# Administering SQL Server

## Managing Security

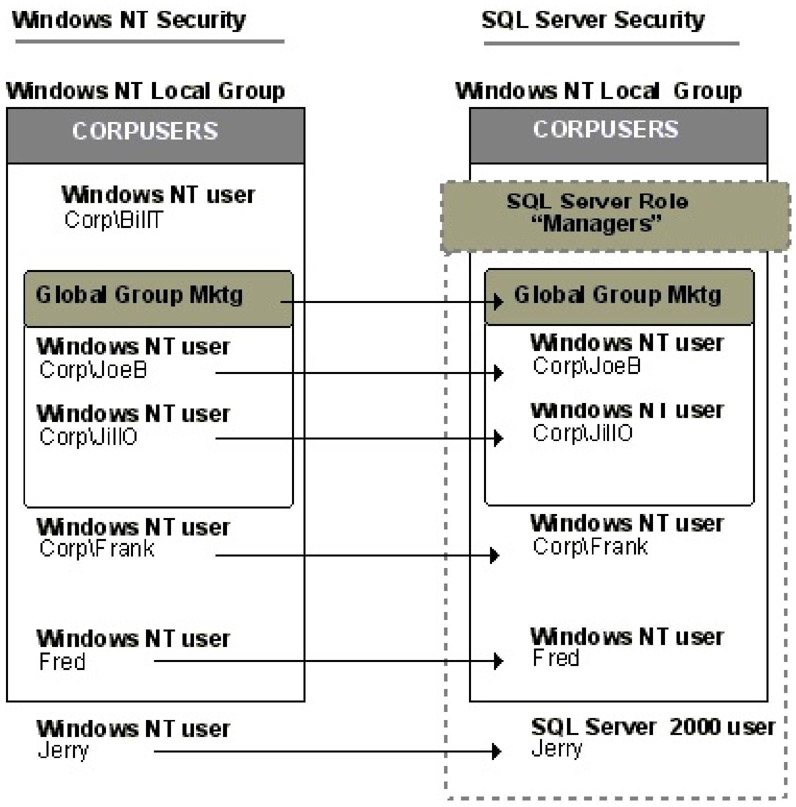
A database must have a solid security system to control which activities can be performed and which information can be viewed and modified. A solid security system ensures the protection of data, regardless of how users gain access to the database.

This section describes the security tools built into Microsoft® SQL Server™ 2000 and includes information about:

Administering SQL Server

## Security Architecture

The architecture of a security system is based on users and groups of users. The following illustration shows how users and local and global groups in Microsoft® Windows NT® 4.0 and Windows® 2000 can map to security accounts in Microsoft SQL Server™, and how SQL Server can handle security accounts independently of the accounts in Windows NT 4.0 and Windows 2000.



The **CORPUSERS** local group contains two users and a global group, **Mktg**, which also contains two users. SQL Server allows Windows NT 4.0 and Windows 2000 local and global groups to be used directly to organize its user accounts. Additionally, the Windows NT 4.0 users **Fred** and **Jerry**, not part of a Windows NT 4.0 group, can be added to an instance of SQL Server either directly as a Windows NT 4.0 user (**Fred** for example), or as a SQL Server user (**Jerry**).

SQL Server extends this model further with the use of roles. Roles are groups of users organized for administrative purposes, like Windows NT 4.0 or Windows 2000 groups, but are created in SQL Server when an equivalent Windows NT 4.0

or Windows 2000 group does not exist. For example, the **Managers** role contains the Windows NT 4.0 **Mktg** global group and the Windows NT 4.0 users **Frank** and **Fred**.

SQL Server also provides security at the application level through the use of individual database application roles.

For more information, see the Windows NT 4.0 or Windows 2000 documentation.

## See Also

Creating Security Accounts

## Administering SQL Server

## Planning Security

A security plan identifies which users can see which data and perform which activities in the database. To developing a security plan:

1. List all the items and activities in the database that must be controlled through security.

2. Identify the individuals and groups in the company.

3. Cross-reference the two lists to identify which users can see which sets of data and perform which activities in the database.

### See Also

Single Person Security Example

Small Company Security Example

Corporate Environment Security Example

# Administering SQL Server

## Single Person Security Example

In the simplest possible security system, a single person is responsible for all aspects of the database and will be its sole user. This hypothetical user (Tom Brown in London) must be able to:

- Create the database and its tables.
- Write programs that interface with the data.
- Load and maintain data.
- Produce reports.

The users-to-activity map for this example lists the single user and the activities he needs to perform.

| User account | Activity |
|---|---|
| **LONDON\tombrown** | All database access |

The first step in creating a security system is to add a Microsoft® SQL Server™ login for **LONDON\tombrown**, allowing him access to SQL Server. Because the predefined **sysadmin** role contains all permissions necessary for this user, the **LONDON\tombrown** SQL Server login should be added as a member of the **sysadmin** role. When **LONDON\tombrown** connects to an instance of SQL Server, SQL Server calls back to Microsoft Windows NT® 4.0 or Windows® 2000 to authenticate the connection. If it is validated, the connection is accepted, and he is allowed to perform activities based on the permissions associated with the **sysadmin** role.

If Tom Brown did not have a Windows NT 4.0 or Windows 2000 login, he could be given a SQL Server login. In this case, an instance of SQL Server would need to be running under Mixed Mode, which allows users to log in under Windows NT 4.0, Windows 2000, or SQL Server logins. A login named **tombro** could be added to SQL Server independent of the Windows NT 4.0 or Windows 2000 login, and **tombro** could then be added to the **sysadmin** role. When the user logs into Windows NT 4.0 or Windows 2000 and attempts to connect to an instance of SQL Server, he must specify the **tombro** login name and password that SQL Server knows.

# Administering SQL Server

## Small Company Security Example

In a moderately complex security system, multiple people perform various tasks in the database. For example, a database administrator is responsible for the database environment: creating the database, tables, and security accounts, performing backups, and tuning the database. Two developers are responsible for writing client applications to provide an interface to the data. Managers prepare information reports from the database and so need access to all available data. The administrative staff performs customer and sales data entry and must be able to view all data.

The users-to-activity map for this example is slightly more complicated than a single user database.

| User account | Activity |
|---|---|
| **LONDON\joetuck** | All database access. |
| **LONDON\marysmith**, **LONDON\billb** | Full access to data and the ability to create procedures. |
| **LONDON\managers** | Full access to all data. |
| **LONDON\admins** | Full access to customer data and sales. Read-only access for all other data. |

The first step in installing the security for this example is to add login rights for **LONDON\joetuck**. Then, because the **LONDON\joetuck** login requires full access, the next step is to add this user to the **sysadmin** role.

Login rights should be added for the developers, too. One way to do this is to grant individual developers (**LONDON\marysmith** and **LONDON\billb**) permissions to access data. But if another developer (or another 10 developers) joined the project, separate permissions would have to be added to each new person, a time-consuming task. A better solution is to add a SQL Server database role named **Developers**, granting permissions to access data and creating procedures to the role. When **LONDON\marysmith** and **LONDON\billb**, or accounts for other new developers, are added to the **Developers** role, their user accounts get the permissions granted to the role.

Roles are only applicable at the database level. That is, roles solve the problem of controlling database user access. Instead of individually granting database access to 10 developers, you can create a role, add the 10 developers to it, and grant the role database access.

Finally, login rights must be added to SQL Server for **LONDON\managers** and **LONDON\admins**. When a manager connects, she is recognized as a member of an existing Microsoft Windows NT® 4.0 and Windows® 2000 group and allowed to perform activities based on the permissions granted to that group. The same is true for **LONDON\admins**.

## Administering SQL Server

## Corporate Environment Security Example

In a large corporate security system, there is a complex web of users who perform specialized, exclusive tasks.

A single person is responsible for all aspects of the database application. A few people are responsible for creating databases and tables, but they must not be allowed to see sensitive personnel information about their coworkers (or even themselves). An evening team backs up data, but these workers need not see the data, nor create tables and databases. The Personnel department must have access to general employee information, and a few select individuals in this department are the only people in the company with access to confidential and sensitive employee information. Also, customer service employees need to see but not change product specifications in response to customer inquiries.

The users-to-activity map for this example is fairly complex.

| User account | Activity |
|---|---|
| **LONDON\annej** | All database access |
| **LONDON\dbadmins** | Create databases |
| **LONDON\dboperations** | Perform evening backups |
| **LONDON\personnel** | Full access to general employee data |
| **LONDON\mikebo**, **LONDON\marym**, **LONDON\billsm** | Full access to confidential data |
| **LONDON\custservice** | Read-only access to product information |

The **LONDON\annej** user account must be granted login rights to Microsoft® SQL Server™ and added to the **sysadmin** role because the **sysadmin** role has full permissions across the server. The **LONDON\dbadmins** Microsoft Windows NT® 4.0 and Windows® 2000 group user account must be added in SQL Server and granted permission to create databases. The **LONDON\operations** Windows NT 4.0 group should be added also and granted only the BACKUP DATABASE permissions to allow them to perform backups.

The **LONDON\personnel** Windows NT 4.0 and Windows 2000 group should be added and granted the permissions to see only the nonsensitive columns in the

**employees** table, as well as the permissions to see other tables.

The users **LONDON\mikebo**, **LONDON\marym**, and **LONDON\billsm** are members of the **LONDON\personnel** Windows NT 4.0 group, so they already have the permissions necessary to do most of their work. However, they also need special access to the sensitive employee information columns. To meet this need, create a database role called **PersonnelSecure** in SQL Server and grant the permissions required to see the sensitive employee information. Individual users get the special permissions in SQL Server when added to the role. Or, add the special permissions to their user accounts directly.

The final step is to add an account for the **LONDON\custservice** Windows NT 4.0 group in SQL Server, and grant it permission to see product information.

# Administering SQL Server

## Security Levels

A user passes through two stages of security when working in Microsoft® SQL Server™: authentication and authorization (permissions validation).The authentication stage identifies the user using a login account and verifies only the ability to connect to an instance of SQL Server. If authentication is successful, the user connects to an instance of SQL Server. The user then needs permissions to access databases on the server, which is done by granting access to an account in each database, mapped to the user login. The permissions validation stage controls the activities the user is allowed to perform in the SQL Server database.

# Administering SQL Server

## Authentication Modes

Microsoft® SQL Server™ can operate in one of two security (authentication) modes:

- Windows Authentication Mode (Windows Authentication)

  Windows Authentication mode allows a user to connect through a Microsoft Windows NT® 4.0 or Windows® 2000 user account.

- Mixed Mode (Windows Authentication and SQL Server Authentication)

  Mixed Mode allows users to connect to an instance of SQL Server using either Windows Authentication or SQL Server Authentication. Users who connect through a Windows NT 4.0 or Windows 2000 user account can make use of trusted connections in either Windows Authentication Mode or Mixed Mode.

  SQL Server Authentication is provided for backward compatibility. For example, if you create a single Windows 2000 group and add all necessary users to that group you will need to grant the Windows 2000 group login rights to SQL Server and access to any necessary databases.

### Windows Authentication

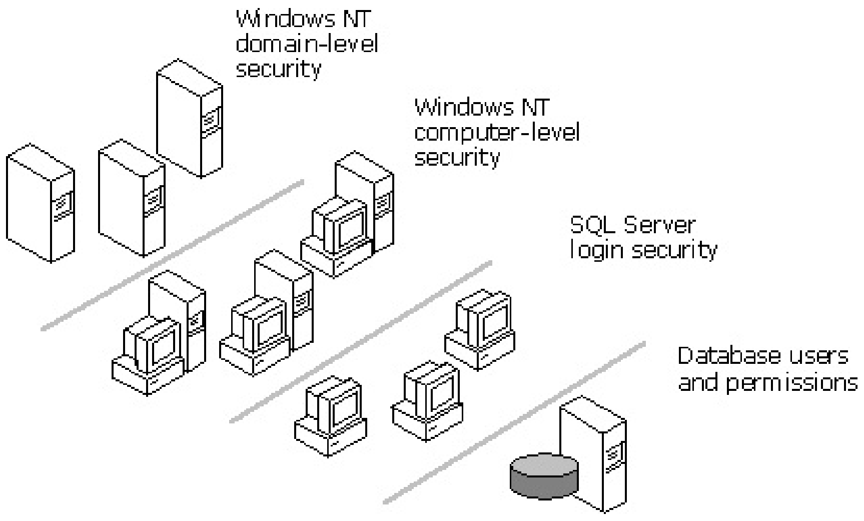
When a user connects through a Windows NT 4.0 or Windows 2000 user account, SQL Server revalidates the account name and password by calling back to Windows NT 4.0 or Windows 2000 for the information.

SQL Server achieves login security integration with Windows NT 4.0 or Windows 2000 by using the security attributes of a network user to control login access. A user's network security attributes are established at network login time and are validated by a Windows domain controller. When a network user tries to connect, SQL Server uses Windows-based facilities to determine the validated network user name. SQL Server then verifies that the person is who they say they are, and then permits or denies login access based on that network user name alone, without requiring a separate login name and password.

Login security integration operates over any supported network protocol in SQL

Server.

**Note** If a user attempts to connect to an instance of SQL Server providing a blank login name, SQL Server uses Windows Authentication. Additionally, if a user attempts to connect to an instance of SQL Server configured for Windows Authentication Mode by using a specific login, the login is ignored and Windows Authentication is used.


Windows NT domain-level security
Windows NT computer-level security
SQL Server login security
Database users and permissions

Windows Authentication has certain benefits over SQL Server Authentication, primarily due to its integration with the Windows NT 4.0 and Windows 2000 security system. Windows NT 4.0 and Windows 2000 security provides more features, such as secure validation and encryption of passwords, auditing, password expiration, minimum password length, and account lockout after multiple invalid login requests.

Because Windows NT 4.0 and Windows 2000 users and groups are maintained only by Windows NT 4.0 or Windows 2000, SQL Server reads information about a user's membership in groups when the user connects. If changes are made to the accessibility rights of a connected user, the changes become effective the next time the user connects to an instance of SQL Server or logs on to Windows NT 4.0 or Windows 2000 (depending on the type of change).

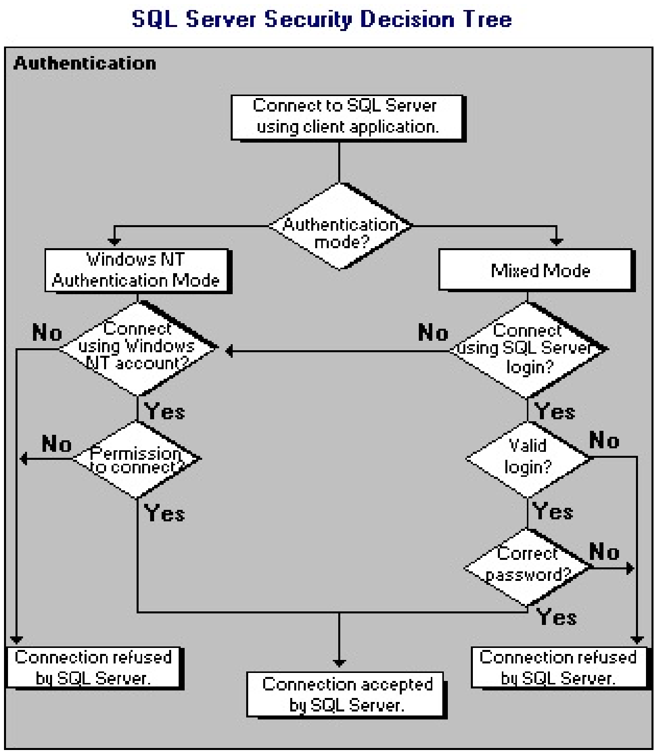
**Note** Windows Authentication Mode is not available when an instance of SQL Server is running on Windows 98 or Microsoft Windows Millennium Edition.

## SQL Server Authentication

When a user connects with a specified login name and password from a nontrusted connection, SQL Server performs the authentication itself by checking to see if a SQL Server login account has been set up and if the specified password matches the one previously recorded. If SQL Server does not have a login account set, authentication fails and the user receives an error message.

SQL Server Authentication is provided for backward compatibility because applications written for SQL Server version 7.0 or earlier may require the use of SQL Server logins and passwords. Additionally, SQL Server Authentication is required when an instance of SQL Server is running on Windows 98 because Windows Authentication Mode is not supported on Windows 98. Therefore, SQL Server uses Mixed Mode when running on Windows 98 (but supports only SQL Server Authentication).

Application developers and database users may prefer SQL Server Authentication because they are familiar with the login and password functionality. SQL Server Authentication may also be required for connections with clients other than Windows NT 4.0 and Windows 2000 clients.


SQL Server Security Decision Tree
Authentication
Connect to SQL Server using client application.
Authentication mode?
Windows NT Authentication Mode
Mixed Mode
Connect using Windows NT account?
Connect using SQL Server login?
No
Yes
Permission to connect?
Valid login?
Correct password?
Connection refused by SQL Server.
Connection accepted by SQL Server.
Connection refused by SQL Server.

**Note** When connecting to an instance of SQL Server running on Windows NT 4.0 or Windows 2000 using Named Pipes, the user must have permission to connect to the Windows NT Named Pipes IPC, \\<computername>**\IPC$**. If the user does not have permission to connect, it is not possible to connect to an instance of SQL Server using Named Pipes unless either the Windows NT 4.0 or Windows 2000 **guest** account on the computer is enabled (disabled by default), or the permission "access this computer from the network" is granted to their user account.

## To set up Windows Authentication Mode security

Administering SQL Server

## Security Account Delegation

Security account delegation is the ability to connect to multiple servers, and with each server change, to retain the authentication credentials of the original client. For example, if a user (**LONDON\joetuck**) connects to ServerA, which then connects to ServerB, ServerB knows that the connection security identity is **LONDON\joetuck**.

To use delegation, all servers that you are connecting to must be running Microsoft® Windows® 2000, with Kerberos support enabled, and you must be using Microsoft Active Directory™, the directory service for Windows 2000. The following options in Active Directory must be specified as follows in order for delegation to work:

- The **Account is sensitive and cannot be delegated** check box must not be selected for the user requesting delegation.

- The **Account is trusted for delegation** check box must be selected for the service account of SQL Server.

- The **Computer is trusted for delegation** check box must be selected for the server running an instance of Microsoft SQL Server™.

To use security account delegation, SQL Server must have:

- A Service Principal Name (SPN) assigned by the Windows 2000 account domain administrator.

  The SPN must be assigned to the service account of the SQL Server service on that particular computer. Delegation enforces mutual authentication. The SPN proves that SQL Server is verified on the particular server, at the particular socket address, by the Windows 2000 account domain administrator. You can have your domain administrator establish an SPN for SQL Server with the **setspn** utility through the Windows 2000 Resource Kit.

  To create an SPN for SQL Server, enter the following code at a command prompt:

```
setspn -A MSSQLSvc/Host:port serviceaccount
```

For example:

```
setspn -A MSSQLSvc/server1.redmond.microsoft.com sqlacco
```

For more information about the **setspn** utility, see the Windows 2000 documentation.

Before enabling delegation, consider the following:

- You must be using TCP/IP. You cannot use Named Pipes, because the SPN targets a particular TCP/IP socket. If you are using multiple ports, you must have a SPN for each port.

- You can also enable delegation by running under the **LocalSystem** account. SQL Server will self-register at service startup and automatically register the SPN. This option is easier than enabling delegation using a domain user account. However, when SQL Server shuts down, the SPNs will be unregistered for the **LocalSystem** account.

  **Note** If you change service accounts in SQL Server, you need to delete any previous SPNs and create new ones.

## Adding an SPN to SQL Server

To add an SPN on an instance of SQL Server named "myserver.microsoft.com", for an instance listening on port 1433, using service account MYDOMAIN\sqlsvc, run the following at a command prompt:

```
setspn -A MSSQLSvc/myserver.microsoft.com:1433 sqlsvc
```

You cannot use the Netbios name. You must use the fully qualified DNS name. You cannot specify the domain qualifier for the service account. You must use only the account name.

To change and use the **LocalSystem** account, enter the following code at a command prompt to delete the previously registered SPN :

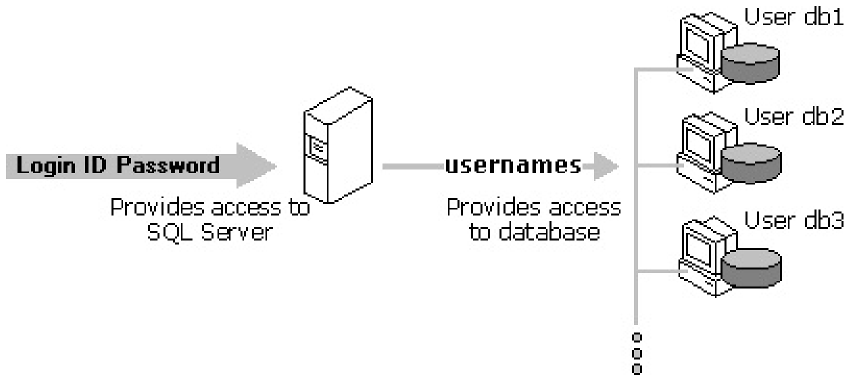
```
setspn -D MSSQLSvc/myserver.microsoft.com:1433 sqlsvc
```

For more information about security account delegation, see the Windows 2000 documentation.

# Administering SQL Server

## Permissions Validation

After a user has been authenticated and allowed to log in to an instance of Microsoft® SQL Server™, a separate user account is required in each database the user must access. Requiring a user account in each database prevents users from connecting to an instance of SQL Server and accessing all the databases on a server. For example, if a server contains a **personnel** database and a **recruiting** database, users who should be able to access the **recruiting** database but not the **personnel** database would have a user account created only in the **recruiting** database.



The user account in each database is used to apply security permissions for the objects (for example, tables, views, and stored procedures) in that database. This user account can be mapped from Microsoft Windows NT® 4.0 and Windows® 2000 user accounts, Windows NT 4.0 and Windows 2000 groups in which the user is a member, or SQL Server login accounts. If there is no account mapped directly, the user may be allowed to work in a database under the **guest** account, if one exists. The activities a user is allowed to perform are controlled by the permissions applied to the user account from which they gained access to a database.

SQL Server accepts commands after a user gains access to a database. All activities a user performs in a database are communicated to SQL Server through Transact-SQL statements. When an instance of SQL Server receives a Transact-SQL statement, it ensures the user has permission to execute the statement in the database. If the user does not have permission to execute a statement or access an object used by the statement, SQL Server returns a permissions error.

## Administering SQL Server

## Hierarchical Security

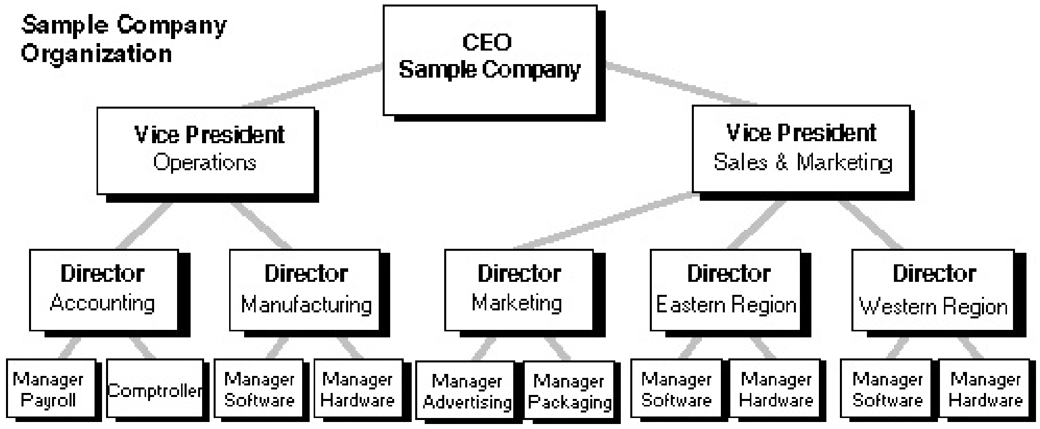
The security environment in Microsoft® SQL Server™ is stored, managed, and enforced through a hierarchical system of users. To simplify the administration of many users, SQL Server uses groups and roles:

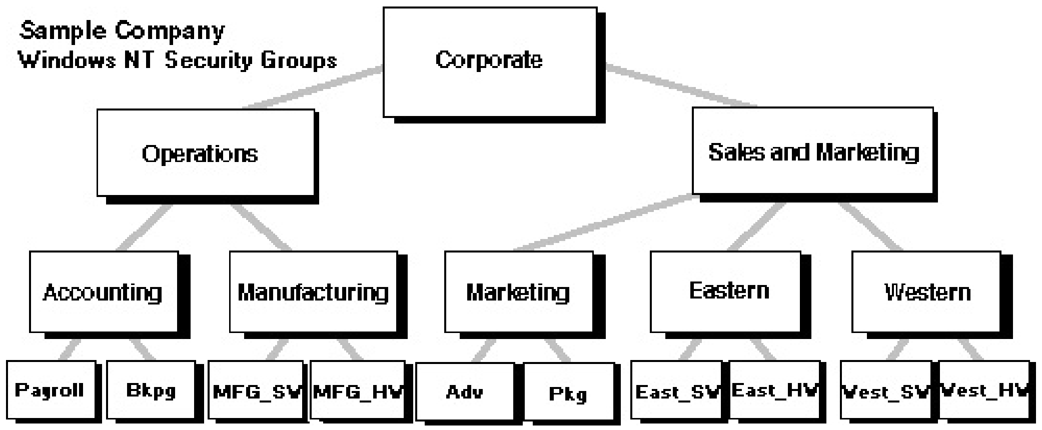
- A group is an administrative unit within Microsoft Windows NT® 4.0 and Windows® 2000 that contains Windows NT 4.0 and Windows 2000 users or other groups.

- A role is an administrative unit within SQL Server that contains SQL Server logins, Windows NT 4.0 and Windows 2000 logins, groups, or other roles.

Arranging users into groups and roles makes it easier to grant or deny permissions to many users at once. The security settings defined for a group are applied to all members of that group. When a group is a member of a higher-level group, all members of the group inherit the security settings of the higher-level group, in addition to the security settings defined for the group itself or user accounts.

The organizational chart of the security system often corresponds to the organizational chart of a company.





These two organizational charts are largely compatible, but there is one common rule for a company's organizational hierarchy that does not apply to the security model: an individual reports only to one manager. This rule implies that an employee can fall into only a single branch of the hierarchical model, as shown in the diagram.

The requirements of a database security system go beyond this one-manager limitation; employees belong to security groups that do not fall within the strict organizational plan of the company. For example, administrative staff exists in every branch of the company and require security permissions regardless of their organizational branch. To support this broader model, the security system in Windows NT 4.0, Windows 2000, and SQL Server allows groups to be defined across a hierarchy. An **Administrative** group can be created to contain administrative employees for every branch of the company from the **Corporate** group to the **Payroll** group.

This hierarchical system of security groups simplifies management of security settings. It allows security settings to be applied collectively to all group members, without having to be defined redundantly for each person. The hierarchical model also accommodates security settings applied only to a single user.

# Administering SQL Server

## Creating Security Accounts

Each user must gain access to an instance of Microsoft® SQL Server™ through a login account that establishes the user's ability to connect (authentication). This login then has to be mapped to a SQL Server user account, which is used to control activities performed in the database (permissions validation). Therefore, a single login is mapped to one user account created in each database the login is accessing. If no user account exists in a database, the user cannot access the database even though the user may be able to connect to an instance of SQL Server.

The login is created in Microsoft Windows NT® 4.0 or Windows® 2000 rather than in SQL Server. This login is then granted permission to connect to an instance of SQL Server. The login is granted access within SQL Server.

## Administering SQL Server

## Security Rules

Microsoft® SQL Server™ logins, users, roles, and passwords can contain from 1 through 128 characters, including letters, symbols, and digits, (for example **Andrew-Fuller**, **Margaret Peacock**, or **13&#57abc)**. Therefore, Microsoft Windows NT® 4.0, Microsoft Windows® 2000, or Microsoft Windows 98 user names can be used as SQL Server logins.

However, because logins, user names, roles, and passwords are often used in Transact-SQL statements, certain symbols must be delimited with double quotation marks ("), or square brackets ([ ]). Use delimiters in Transact-SQL statements when the SQL Server login, user, role, or password:

- Contains, or begins with, a space character.
- Begins with the **$** or **@** character.

  **Note** It is not necessary to specify delimiters when entering logins, users, roles, and passwords into the text boxes of the SQL Server graphical client tools, such as SQL Server Enterprise Manager.

Additionally, a SQL Server login, user, or role cannot:

- Contain a backslash (\) character, unless referring to an existing Windows NT 4.0 or Windows 2000 user or group. The backslash separates the Windows NT 4.0 or Windows 2000 computer or domain name from the user name.
- Already exist in the current database (or **master**, for logins only).
- Be NULL, or an empty string ("").

### See Also

Delimited Identifiers

## Administering SQL Server

## Adding a Windows User or Group

Microsoft® Windows NT® 4.0 and Windows® 2000 accounts (users or groups) must be granted permissions to connect to an instance of Microsoft SQL Server™ before they can access a database. If all members of a Windows NT 4.0 or Windows 2000 group will be connecting to an instance of SQL Server, you can grant permission to the group as a whole. Managing group permissions is much easier than managing permissions for individual users. If the group should not be granted permission collectively, grant permission to connect to an instance of SQL Server for each individual Windows NT 4.0 or Windows 2000 user.

### Users

When granting a Windows NT 4.0 or Windows 2000 user access to connect to an instance of SQL Server, specify the Windows NT 4.0 or Windows 2000 domain or computer name to which the user belongs, followed by a backslash, and then the user. For example, to grant access to the Windows NT 4.0 or Windows 2000 user **Andrew**, in the Windows NT 4.0 or Windows 2000 domain **LONDON**, specify **LONDON\Andrew** as the user name.

### Local and Global Groups

There are several types of Windows NT 4.0 and Windows 2000 groups, including global and local:

- Global groups contain user accounts from the Windows NT 4.0 or Windows 2000 domain in which they are created. Global groups cannot contain other groups or users from other domains and cannot be created on a computer running Microsoft Windows NT 4.0 Workstation or Microsoft Windows 2000 Professional.

- Local groups can contain user accounts and global groups from the domain in which they are created and in any trusted domain. Local groups cannot contain other local groups.

Additionally, Windows NT 4.0 and Windows 2000 have predefined, built-in local groups (for example, **Administrators**, **Users**, and **Guests)**.

When granting a Windows NT 4.0 or Windows 2000 local or global group access to connect to an instance of SQL Server, specify the domain or computer name the group is defined on, followed by a backslash, and then the group name. For example, to grant access to a global group called **SQL_Users**, in the **LONDON** domain, specify **LONDON\SQL_Users** as the group name.

To grant access to a Windows NT 4.0 or Windows 2000 built-in, local group, specify BUILTIN instead of the domain or computer name. To grant access to the built-in Windows NT 4.0 and Windows 2000 local group **Administrators**, specify **BUILTIN\Administrators** as the group name.

For more information about these accounts, see the Windows NT 4.0 and Windows 2000 documentation.

### To grant a Windows user or group login access to SQL Server

⊞ Transact-SQL

## Granting a Windows User or Group Access to a Database

To obtain access to a Microsoft® SQL Server™ database, a Microsoft Windows NT® 4.0 and Windows® 2000 user or group must have a corresponding user account in each database they need to access. Additionally, permissions must be applied to this user account.

Although possible, it is not necessary to add an individual user account in a database for each Windows NT 4.0 and Windows 2000 user in a Windows NT 4.0 and Windows 2000 group whose members all perform the same activities. Accounts can be added for groups rather than for each individual member. When the group members need to work in a database, they are granted access through their membership in the Windows NT 4.0 and Windows 2000 group; there is not a specific account for individual users within the group. For example, a Windows NT 4.0 and Windows 2000 group **London\Managers** contains the Windows NT 4.0 and Windows 2000 user **London\JoeB**. The SQL Server system administrator grants login access only to **London\Managers**. The owner of database **Accounts** grants only **London\Managers** permission to access **Accounts**. Although **London\JoeB** does not have explicit permission granted to connect to an instance of SQL Server or to access **Accounts**, he can connect to the instance of SQL Server and access **Accounts** due to his membership in **London\Managers**.

Add individual Windows NT 4.0 and Windows 2000 users to a database only if the user performs activities different from other members of any Windows NT 4.0 or Windows 2000 group (for example, special database administrative duties).

**Note** Users who are granted access to an instance of SQL Server through their memberships in a Windows NT 4.0 or Windows 2000 group do not have entries for their individual Windows NT 4.0 or Windows 2000 user accounts in the system tables. However, an entry is created for their individual user accounts if they create objects, such as a table or a stored procedure, in a SQL Server database.

## To grant a Windows user or group access to a database

⊞ Transact-SQL

# Administering SQL Server

## Adding a SQL Server Login

Add Microsoft® SQL Server™ login accounts that allow a connection by means of a specified login name and password, rather than through a Microsoft Windows NT® 4.0 or Windows® 2000 user or group account, if:

- SQL Server is configured to operate in Mixed Mode.
- An instance of SQL Server is running on Microsoft Windows 98.

Adding SQL Server logins is required:

- For compatibility with applications containing data imported from other databases vendors.
- For applications designed to work with general users who do not have Windows NT 4.0 or Windows 2000 accounts.
- To connect to an instance of SQL Server running on Windows 98 because Windows Authentication is not available on Windows 98.

### To add a SQL Server login

⊞ Transact-SQL

## System Administrator (sa) Login

System administrator (**sa**) is a special login provided for backward compatibility. By default, it is assigned to the **sysadmin** fixed server role and cannot be changed. Although **sa** is a built-in administrator login, do not use it routinely. Instead, make system administrators members of the **sysadmin** fixed server role, and have them log on using their own logins. Use **sa** only when there is no other way to log in to an instance of Microsoft® SQL Server™ (for example, when other system administrators are unavailable or have forgotten their passwords).

**Note** When SQL Server is installed, SQL Server Setup prompts you to change the **sa** login password if you request Mixed Mode authentication. It is recommended that the password be assigned immediately to prevent unauthorized access to an instance of SQL Server using the **sa** login.

### See Also

Assigning an sa Password

## Granting a SQL Server Login Access to a Database

Add a Microsoft® SQL Server™ user account to each database for each SQL Server login that requires access to the database. If a user is not created in the database, the SQL Server login cannot access the database.

To grant a SQL Server login access to a database, the SQL Server login must already exist. Furthermore, SQL Server logins must be granted access to a database one at a time.

### To grant a SQL Server login access to a database

⊞ Transact-SQL

# Administering SQL Server

## Database Owner (dbo)

The **dbo** is a user that has implied permissions to perform all activities in the database. Any member of the **sysadmin** fixed server role who uses a database is mapped to the special user inside each database called **dbo**. Also, any object created by any member of the **sysadmin** fixed server role belongs to **dbo** automatically.

For example, if user **Andrew** is a member of the **sysadmin** fixed server role and creates a table **T1**, **T1** belongs to **dbo** and is qualified as **dbo.T1**, not as **Andrew.T1**. Conversely, if **Andrew** is not a member of the **sysadmin** fixed server role but is a member only of the **db_owner** fixed database role and creates a table **T1**, **T1** belongs to **Andrew** and is qualified as **Andrew.T1**. The table belongs to **Andrew** because he did not qualify the table as **dbo.T1**.

The **dbo** user cannot be deleted and is always present in every database.

Only objects created by members of the **sysadmin** fixed server role (or by the **dbo** user) belong to **dbo**. Objects created by any other user who is not also a member of the **sysadmin** fixed server role (including members of the **db_owner** fixed database role):

- Belong to the user creating the object, not **dbo**.

- Are qualified with the name of the user who created the object.

### See Also

Delimited Identifiers

sp_changedbowner

# Administering SQL Server

## Database Object Owner

A user who creates a database object (a table, index, view, trigger, function, or stored procedure) is called a database object owner. Permission to create database objects must be given by the database owner or system administrator. However, after these permissions are granted, a database object owner can create an object and grant other users permission to use that object.

Database object owners have no special login IDs or passwords. The creator of a database object is granted all permissions implicitly but must give explicit permissions to other users before they can access the object.

### Referencing database objects

When users access an object created by another user, the object should be qualified with the name of the object owner; otherwise, Microsoft® SQL Server™ may not know which object to use because there could be many objects of the same name owned by different users. If an object is not qualified with the object owner when it is referenced (for example, **my_table** instead of **owner.my_table**), SQL Server looks for an object in the database in the following order:

1. Owned by the current user.

2. Owned by **dbo**.

If the object is not found, an error is returned.

For example, user **John** is a member of the **db_owner** fixed database role, but not the **sysadmin** fixed server role, and creates table **T1**. All users, except **John**, who want to access **T1** must qualify **T1** with the user name **John**. If **T1** is not qualified with the user name **John**, SQL Server first looks for a table named **T1** owned by the current user and then owned by **dbo**. If the current user and **dbo** do not own a table named **T1**, an error is returned. If the current user or **dbo** owns another table named **T1**, the other table named **T1**, rather than **John.T1**, is used.

If a database object owner must be removed from a database, the owned objects must be dropped first or their ownership transferred to another user.

**Note** SQL Server allows a role or Microsoft Windows NT® 4.0 or Windows® 2000 group to be specified as the owner of an object. For example, to create the table **group_table** owned by the Windows NT 4.0 or Windows 2000 group **LONDON\Users**, specify **[LONDON\Users].group_table** as the qualified table name. All members of the **LONDON\Users** group have database object owner permissions on **group_table**.

## See Also

Delimited Identifiers

sp_changeobjectowner

## Administering SQL Server

# guest User

The **guest** user account allows a login without a user account to access a database. A login assumes the identity of the **guest** user when both of the following conditions are met:

- The login has access to an instance of Microsoft® SQL Server™ but does not have access to the database through his or her own user account.

- The database contains a **guest** user account.

Permissions can be applied to the **guest** user as if it were any other user account. The **guest** user can be deleted and added to all databases except **master** and **tempdb**, where it must always exist. By default, a **guest** user account does not exist in newly created databases.

For example, to add a **guest** user account to a database named **Accounts**, run the following code in SQL Query Analyzer:

```
USE Accounts
GO
EXECUTE sp_grantdbaccess guest
```

### To grant a SQL Server login access to a database

⊞ Transact-SQL

# Administering SQL Server

## Creating User-Defined SQL Server Database Roles

Create Microsoft® SQL Server™ database roles when a group of users needs to perform a specified set of activities in SQL Server and one of the following is true:

- There is no applicable Microsoft Windows NT® 4.0 or Windows® 2000 group.
- You do not have permissions to manage Windows NT 4.0 or Windows 2000 user accounts.

**Note** Avoid deep levels of nested roles because this can affect performance.

For example, a company may form a Charity Event Committee involving employees from different departments and from several different levels in the organization. These employees need access to a special project table in the database. There is no existing Windows NT 4.0 or Windows 2000 group that includes just these employees, and there is no other reason to create one in Windows NT 4.0 or Windows 2000. A custom SQL Server database role, **CharityEvent**, can be created for this project and individual Windows NT 4.0 and Windows 2000 users added to the database role. When permissions are applied, the users in the database role gain table access. Permissions for other database activities are not affected, and the **CharityEvent** users are the only ones who can work with the project table.

SQL Server roles exist within a database and cannot span more than one database.

The advantages of using database roles include:

- For any user, more than one database role can be active at any time.
- SQL Server roles can contain Windows NT 4.0 or Windows 2000 groups and users and SQL Server users and other roles, provided that all users, groups, and roles exist in the current database.

- A user can belong to more than one role in the same database.

- A scalable model is provided for setting up the correct level of security within a database.

**Note** A database role is owned by either the user explicitly specified as the owner when the role is created, or the user who created the role when no owner is specified. The owner of the role determines who can be added or removed from the role. However, because a role is not a database object, multiple roles of the same name in the same database owned by different users cannot be created.

### To create a SQL Server database role

⊞ Transact-SQL

# Administering SQL Server

# Adding a Member to a SQL Server Database Role

When you add a new user account in Microsoft® SQL Server™ or change the permissions of an existing user, you can add the user to a SQL Server database role rather than applying permissions directly to the account. Roles can simplify security administration in databases with a large number of users or with a complex security system.

SQL Server users, Microsoft Windows NT® 4.0 or Windows® 2000 users and groups, and other SQL Server database roles all can be added as a member of a role. Because a role is restricted to a single database and cannot be added from one database to another, you can add users, groups, and roles known only to that database.

**Note** When you add a Windows NT 4.0 or Windows 2000 login without a user account in the database to a SQL Server database role, SQL Server creates a user account in the database automatically, even if that Windows NT 4.0 or Windows 2000 login cannot otherwise access the database.

A user account can be a member of any number of roles within the same database and can hold permissions appropriate to each role. For example, a SQL Server user can be a member of the **admin** role and the **users** role for the same database, with each role granting different permissions. The permission on an object granted to a member of more than one role are the cumulative permissions of the roles. However, a denied permission in one role has precedence over the same permission granted in another role. For example, the **admin** role may grant access to a table, whereas the **users** role denies access to the same table. A member of both roles is denied access to the table because denied access is more restrictive and has precedence.

Users to be added to a user-defined database role must already have permission to access the database containing the user-defined role.

## To add a member to a SQL Server database role

⊞ Transact-SQL

# Administering SQL Server

# Adding a Member to a Predefined Role

The security mechanism in Microsoft® SQL Server™ includes several predefined roles with implied permissions that cannot be granted to other user accounts. If you have users who require these permissions, you must add their accounts to these predefined roles. The two types of predefined roles are fixed server and fixed database.

## Fixed Server Roles

Fixed server roles, which cannot be created, are defined at the server level and exist outside of individual databases. To add a user to a fixed server role, the user must have a SQL Server or Microsoft Windows NT® 4.0 or Windows® 2000 login account. Any member of a fixed server role can add other logins.

**IMPORTANT** Windows NT 4.0 or Windows 2000 users who are members of the **BUILTIN\Administrators** group are members of the **sysadmin** fixed server role automatically.

The following table describes the fixed server roles.

| Fixed server role | Description |
|---|---|
| **sysadmin** | Performs any activity in SQL Server. The permissions of this role span all of the other fixed server roles. |
| **serveradmin** | Configures server-wide settings. |
| **setupadmin** | Adds and removes linked servers, and executes some system stored procedures, such as **sp_serveroption**. |
| **securityadmin** | Manages server logins. |
| **processadmin** | Manages processes running in an instance of SQL Server. |
| **dbcreator** | Creates and alters databases. |
| **diskadmin** | Manages disk files. |
| **bulkadmin** | Executes the BULK INSERT statement. |

The **securityadmin** has permission to execute the **sp_password** stored procedure for all users other than members of the **sysadmin** role.

The **bulkadmin** fixed server role has permission to execute BULK INSERT statements. Members of the **bulkadmin** role can add other logins to the role, as all members of any given fixed server role can do. However, due to the security implications associated with executing the BULK INSERT statement (the BULK INSERT statement requires read access to any data on the network and machine the server is running on), it may not be desirable for members of the **bulkadmin** role to grant permission to others. The **bulkadmin** role provides members of the **sysadmin** fixed server role with a method to delegate tasks requiring execution of the BULK INSERT statement, without granting users **sysadmin** rights. Members of the **bulkadmin** role are allowed to execute the BULK INSERT statement, but they still must have the INSERT permission on the table on which you wish to insert data.

### To add a member to a fixed server role

⊞ Transact-SQL

## public Role

The **public** role is a special database role to which every database user belongs. The **public** role:

- Captures all default permissions for users in a database.
- Cannot have users, groups, or roles assigned to it because they belong to the role by default.
- Is contained in every database, including **master**, **msdb**, **tempdb**, **model**, and all user databases.
- Cannot be dropped.

# Administering SQL Server

## Using the Create Login Wizard

Although the steps required to grant login access to Microsoft® SQL Server™ and a database can be performed separately, the Create Login Wizard can simplify the process. The Create Login Wizard allows you to:

- Choose which authentication mode to use to connect to an instance of SQL Server (Windows Authentication Mode or Mixed Mode).

- Add a Microsoft Windows NT® 4.0, Windows® 2000 or SQL Server login.

- Add a Windows NT 4.0, Windows 2000 or SQL Server user to a fixed server role.

- Add a Windows NT 4.0, Windows 2000 or SQL Server user to one or more databases, thereby granting the user access to those databases.

### To grant SQL Server login access to a user by using the Create Login Wizard

# Administering SQL Server

## Managing Security Accounts

After security accounts have been added to Microsoft® SQL Server™, you can modify them as business needs change. This usually involves viewing, modifying, and removing the security accounts in the database to fit the needs of your business.

# Administering SQL Server

# Viewing Logins

View Microsoft® SQL Server™ logins to determine if a user or Microsoft Windows NT® 4.0 or Windows® 2000 group has permission to connect to an instance of SQL Server, and to identify which databases the login can access. Also, view a login before removing it to see which database users must be removed; it is not possible to remove a login without first removing the associated users.

You can view:

- Users in each database associated with the login.
- Default database and language the login uses when the user first connects to an instance of SQL Server.
- Windows NT 4.0 or Windows 2000 security identifier (SID).

**Note** It is not possible to view the password of any login unless the password is NULL. Passwords are encrypted when stored in SQL Server.

### To view a SQL Server login or Windows user or group

⊞ Transact-SQL

# Administering SQL Server

## Modifying Logins

After a login has been created, it may be necessary to change the password, default database, or default language. For example, a user may forget her password, want to change the password for security reasons, need to use a different database on a regular basis, or need to see messages in a different language.

**Note** If a user forgets a password, a member of the **sysadmin** or **securityadmin** fixed server role can change the password without knowing the original password. A user cannot change a password if he has forgotten it. Members of the **securityadmin** role cannot change the password of members of the **sysadmin** role.

### To change the password of a SQL Server login

⊞ Transact-SQL

## Administering SQL Server

## Removing Logins and Users

The process of deactivating security accounts (for example, when an employee leaves a company) is similar to the process of adding a new user. Update the security mechanism in Microsoft® Windows NT® 4.0 or Windows® 2000 by first removing the user's Windows NT 4.0 or Windows 2000 user account. If the user has a Microsoft SQL Server™ user account, removed it from SQL Server along with any SQL Server database roles specifically defined for that user. Finally, remove any SQL Server login.

Removing a SQL Server user or Windows NT 4.0 or Windows 2000 user or group from a SQL Server database automatically removes the permissions defined for the user or group and prevents that user from using the database under the old security account. The permissions do not have to be removed separately. However, it is not possible to remove a user from SQL Server if that user currently owns objects (tables, procedures, or views) within a database. If the user owns objects, then either drop those objects before removing the user or transfer ownership to another existing user by using the **sp_changeobjectowner** system stored procedure.

Removing a user does not remove a login automatically, so it does not prevent the user from connecting to an instance of SQL Server. After being removed, the user can log in to the databases only through the **guest** account and perform activities under those permissions. To prevent a user from connecting to an instance of SQL Server, remove his or her login.

If a linked server login is set up but is no longer required, remove it to prevent unauthorized access to the linked server and to keep the security system as simple as possible.

### To remove a user or group from a database

⊞ Transact-SQL

## Administering SQL Server

# Denying Login Access to Windows Accounts

When a Microsoft® Windows NT® 4.0 or Windows® 2000 user belongs to a Windows NT 4.0 or Windows 2000 group that has a login account in Microsoft SQL Server™, the user is allowed to connect through the group login. However, there may be times when such users or groups need to be prevented from connecting to an instance of SQL Server. You can deny login access to any Windows NT 4.0 or Windows 2000 user or group. Users cannot connect to an instance of SQL Server if their user account, or any group in which they are a member, has been denied login access.

## To deny login access to a Windows user or group

⊞ Transact-SQL

# Administering SQL Server

# Viewing Roles

When creating and using a database, you may need to find information about a Microsoft® SQL Server™ database role or a fixed server role. For example, you may need to see which roles exist in the current database, or list the fixed server roles.

### To view the roles defined in the current database

⊞ Transact-SQL

## Administering SQL Server

## Viewing and Modifying Role Memberships

While using a database, you may need to list the members of a database role or fixed server role. Or, when a Microsoft® SQL Server™ user no longer needs the permissions from a user-defined, fixed database or server role of which she is a member, you may want to remove the user from the role to keep the security system as simple as possible.

### To view the members of a database role

⊞ Transact-SQL

# Administering SQL Server

## Removing a SQL Server Database Role

The changing security requirements of a database can render a Microsoft® SQL Server™ database role obsolete. Remove roles when you have removed all users and are certain that the role and its permissions will not be required in the future. Empty roles can be saved if the permissions may be required for a new user. However, from an administrative perspective, it is much easier to work with a security system that is not cluttered with unnecessary security roles. SQL Server operates faster with a simpler security system, although it is will not be a problem unless there are an extremely large number of roles.

**Note** It is not possible to remove fixed server roles or fixed database roles.

### To remove a SQL Server role

⊞ Transact-SQL

Administering SQL Server

# Viewing Database Users

Viewing a Microsoft® SQL Server™ user account in a database shows:

- The roles of which the user is a member.
- The SQL Server login associated with the user.
- The default database.

Use this information to understand how the user fits into the security system of the database.

### To view a database user

⊞ Transact-SQL

Administering SQL Server

## Managing Permissions

When users connect to an instance of Microsoft® SQL Server™, the activities they can perform are determined by the permissions granted to:

- Their security accounts.
- The Microsoft Windows NT® 4.0 or Windows® 2000 groups or role hierarchies to which their security accounts belong.

The user must have the appropriate permissions to perform any activity that involves changing the database definition or accessing data.

Managing permissions includes granting or revoking user rights to:

- Work with data and execute procedures (object permissions).
- Create a database or an item in the database (statement permissions).
- Utilize permissions granted to predefined roles (implied permissions).

### Object Permissions

Working with data or executing a procedure requires a class of permissions known as object permissions*:*

- SELECT, INSERT, UPDATE, and DELETE statement permissions, which can be applied to the entire table and view.
- SELECT and UPDATE statement permissions, which can be selectively applied to individual columns of a table or view.
- SELECT permissions, which may be applied to user-defined functions.

- INSERT and DELETE statement permissions, which affect the entire row, and therefore can be applied only to the table and view and not to individual columns.

- EXECUTE statement permissions, which affect stored procedures and functions.

## Statement Permissions

Activities involved in creating a database or an item in a database, such as a table or stored procedure, require a different class of permissions called statement permissions. For example, if a user must be able to create a table within a database, then grant the CREATE TABLE statement permission to the user. Statement permissions, such as CREATE DATABASE, are applied to the statement itself, rather than to a specific object defined in the database.

Statement permissions are:

- BACKUP DATABASE
- BACKUP LOG
- CREATE DATABASE
- CREATE DEFAULT
- CREATE FUNCTION
- CREATE PROCEDURE
- CREATE RULE
- CREATE TABLE

- CREATE VIEW

## Implied Permissions

Implied permissions control those activities that can be performed only by members of predefined system roles or owners of database objects. For example, a member of the **sysadmin** fixed server role inherits automatically full permission to do or see anything in a SQL Server installation.

Database object owners also have implied permissions that allow them to perform all activities with the object they own. For example, a user who owns a table can view, add, or delete data, alter the table definition, or control permissions that allow other users to work with the table.

## See Also

BACKUP DATABASE

BACKUP LOG

CREATE DATABASE

CREATE DEFAULT

CREATE FUNCTION

CREATE PROCEDURE

CREATE RULE

CREATE TABLE

CREATE VIEW

## Administering SQL Server

## Granting Permissions

Grant statement and object permissions that allow a user account to:

- Perform activities or work with data in the current database.

- Restrict them from activities or information not part of their intended function.

  For example, you may be inclined to grant SELECT object permission on the **payroll** table to all members of the **personnel** role, allowing all members of **personnel** to view **payroll**. Months later, you may overhear members of **personnel** discussing management salaries, information not meant to be seen by all **personnel** members. In this situation, grant SELECT access to **personnel** for all columns in **payroll** except the **salary** column.

**Note** It is possible to grant permissions only to user accounts in the current database, for objects in the current database. If a user needs permissions to objects in another database, create the user account in the other database, or grant the user account access to the other database, as well as the current database. System stored procedures are the exception because EXECUTE permissions are already granted to the **public** role, which allows everyone to execute them. However, after EXECUTE has been issued, the system stored procedures check the user's role membership. If the user is not a member of the appropriate fixed server or database role necessary to run the stored procedure, the stored procedure will not continue.

### To allow access by granting permissions (on an object)

⊞ Transact-SQL

# Administering SQL Server

## Denying Permissions

Microsoft® SQL Server™ allows Microsoft Windows NT® 4.0 or Windows® 2000 users and groups, SQL Server users, and SQL Server database roles to be members of other roles. This results in a hierarchical security system that allows permissions to be applied through several levels of roles and members. But there may be times when you want to limit the permissions of a user or role. Denying permissions on a user account:

- Removes permission granted previously to the user, group, or role.
- Deactivates permission inherited from another role(s).
- Ensures that a user, group, or role will not inherit permission from a higher level group or role in the future.

For example, you may need to provide all tenured employees in your company with access to several tables in a database, with the exception of a few new employees scattered throughout the organization who you want to prevent from seeing the **CorporateSecrets** table.

Create a role for each department in the company and add all employees to their department role. Then create a company-wide **Corporate** role, to which you add each of the individual department roles and grant permissions to view the tables. At this point, every employee in the company can see all the tables because each inherits permission from the **Corporate** role through his department roles.

To selectively prevent employees from seeing **CorporateSecrets**, create a **Nonsecure** role, and add the individual employees who should not see the table. When you deny permission to view **CorporateSecrets** to **Nonsecure**, this access is removed from all members of **Nonsecure**, while the rest of the employees in the company are not affected.

You also can deny permissions to an individual user. In the previous example, a nonemployee may have a Windows NT 4.0 or Windows 2000 account while working on a short-term project in the database. You can deny the permissions to see **CorporateSecrets** to his individual user account without creating a SQL

Server database role for the purpose.

**Note** You can deny permissions to user accounts only in the current database, for objects in the current database.

## To prevent access by denying permissions (on an object)

⊞ Transact-SQL

## Administering SQL Server

## Revoking Permissions

You can revoke a permission that has been granted or denied previously. Revoking is similar to denying in that both remove a granted permission at the same level. However, although revoking a permission removes a granted permission, it does not prevent the user, group, or role from inheriting a granted permission from a higher level. Therefore, if you revoke permission for a user to view a table, you do not necessarily prevent the user from viewing the table because permission to view the table was granted to a role to which he belongs.

For example, removing SELECT access on the **Employees** table from the **HumanResources** role revokes permission so that **HumanResources** can no longer use the table. If **HumanResources** is a member of the **Administration** role. If you later grant SELECT permission on **Employees** to **Administration**, members of **HumanResources** can see the table through their membership in **Administration**. However, if you deny permission to **HumanResources**, the permission is not inherited if later granted to **Administration** because the deny permission cannot be undone by a permission at a different level.

Similarly, it is also possible to remove a previously denied permission by revoking the deny for the permission. However, if a user has other denied permissions at the group or role level, then the user still is denied access.

**Note** You can revoke permissions to user accounts only in the current database, for objects in the current database.

### To revoke permissions on an object

⊞ Transact-SQL

## Administering SQL Server

## Resolving Permission Conflicts

The permissions granted to a group or role are inherited by members of that group or role. Although a user may have permission granted or revoked at one level, conflicting permissions at a higher level (for example, due to role membership) can prevent or allow a user access to an object.

### Deny

A denied permission always takes precedence. Denied permission at any level (user, group, or role) denies the permission on the object regardless of existing granted or revoked permissions for that user. For example, if user **John**, who as a member of the **sales** role is granted SELECT permissions on the **customer** table, is explicitly denied SELECT permissions on the **customer** table, he can no longer access it. Similarly, if the **sales** role is denied access to **customer**, but **John** is granted access, he is denied access.

**Note** Microsoft® SQL Server™ always processes denied permissions first. If you deny permissions to **public**, you prevent anyone from accessing an object, including the issuer of the DENY statement.

### Revoke

A revoked permission removes only the granted or denied permission at the level revoked (user, group, or role). The same permission granted or denied at another level such as a group or role containing the user, group, or role still applies. For example, if the **sales** role is granted SELECT permissions on the **customer** table, and **John** (a member of **sales**) is explicitly revoked SELECT permissions on the **customer** table, he still can access the table because of his membership in the **sales** role. To prevent **John** from accessing the **customer** table, do one of the following:

- Revoke permission (assuming no other permissions have been granted elsewhere).

- Deny permission to the **sales** role (preventing all members of **sales** from accessing the table).

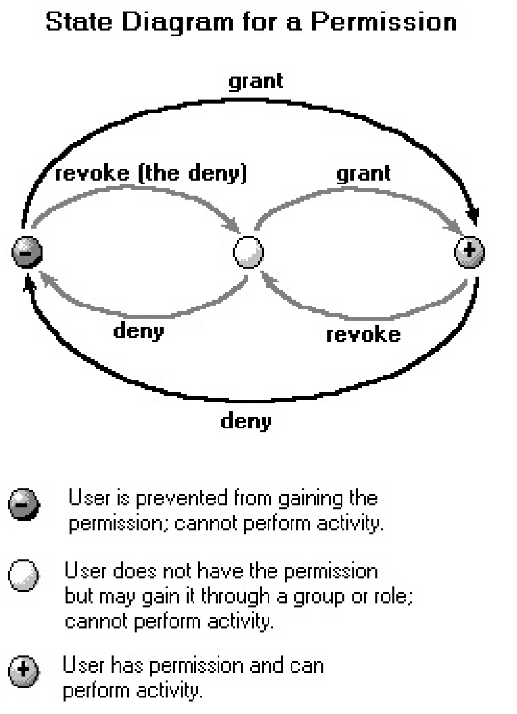
- Explicitly deny **John** SELECT permissions on **customer**.

## Grant

A granted permission removes the denied or revoked permission at the level granted (user, group, or role). The same permission denied at another level such as group or role containing the user still applies. However, although the same permission revoked at another level still applies, it does not prevent the user from accessing the object. For example, if **John** is already explicitly denied access to **customer**, has his access to **sales**, revoked, and then is explicitly granted access to **customer**, he now can access **customer** because the deny is removed. The revoke permission for **sales** joined with the granted permission for **John** gives **John** a granted permission overall.

Therefore, a user receives the union of all the permissions granted, denied, or revoked on an object, with any denied permissions taking precedence over the same permissions granted or revoked at another level.

The following diagram shows how the three permission management activities affect the state of a permission for a user account.


State Diagram for a Permission
grant
revoke (the deny)
grant
deny
revoke
deny
User is prevented from gaining the permission; cannot perform activity.
User does not have the permission but may gain it through a group or role; cannot perform activity.
User has permission and can perform activity.

## Database Access vs. Object Access

As an example of a permission conflict, a Microsoft Windows NT® 4.0 user **LONDON\joe** belongs to the **LONDON\clerks** and **LONDON\secretaries** Windows NT 4.0 groups. **LONDON\joe** can log in to an instance of SQL Server because the **LONDON\clerks** group has been granted permissions to connect to an instance of SQL Server. Additionally, **LONDON\joe** can access the **secrets** database because the **LONDON\secretaries** group has been granted permissions to access the database.

**Note** At this point there is no specific entry in the SQL Server system tables, **sysusers** and **sysxlogins**, for **LONDON\joe**. These system tables contain only entries for the **LONDON\clerks** and **LONDON\secretaries** groups.

**LONDON\joe** creates a table, **joetable**, in the **secrets** database. At this point, a new entry is created in the **sysusers** table for **LONDON\joe** specifying him as the object owner but not granting him database access. If **LONDON\joe** is dropped from the **LONDON\secretaries** group, he can no longer access the **secrets** database, although he owns an object, **joetable**, in the database.

## See Also

Adding a Windows NT User or Group

## Permissions for User-Defined Functions

Functions are subroutines made up of one or more Transact-SQL statements that can be used to encapsulate code for reuse. Microsoft® SQL Server™ 2000 allows users to create their own user-defined functions.

User-defined functions are managed through the following statements:

- CREATE FUNCTION, which creates a user-defined function.
- ALTER FUNCTION, which modifies user-defined functions.
- DROP FUNCTION, which drops user-defined functions.

Each fully qualified user-defined function name (*database_name*.*owner_name*.*function_name*) must be unique.

You must have been granted CREATE FUNCTION permissions to create, alter, or drop user-defined functions. Users other than the owner must be granted EXECUTE permission on a function (if the function is scalar-valued) before they can use it in a Transact-SQL statement. If the function is table-valued, the user must have SELECT permissions on the function before referencing it. If a CREATE TABLE or ALTER TABLE statement references a user-defined function in a CHECK constraint, a DEFAULT clause, or a computed column, the table owner must also own the function. If the function is being schema-bound, you must have REFERENCE permission on tables, views, and functions referenced by the function.

REFERENCE permissions can be granted through the GRANT statement to views and user-defined functions in addition to tables.

### See Also

User-Defined Functions

# Administering SQL Server

## Using Ownership Chains

Views and stored procedures provide a secondary method of giving users access to data and the ability to perform activities. They provide users with access to underlying items in the database and bypass the permissions defined directly for specific objects and statements.

Views can depend on other views or tables. Procedures can depend on other procedures, views, or tables. These dependencies can be thought of as an ownership chain. Ownership chains only apply to SELECT, INSERT, UPDATE, and DELETE statements.

Typically, the owner of a view also owns the underlying objects (other views or tables), and the owner of a stored procedure often owns all the referenced procedures, tables, and views. Also, views and underlying objects are usually all in the same database, as are stored procedures and all the objects referenced. When temporary objects are created within a stored procedure, they are owned by the procedure owner and not by the user currently executing the procedure.

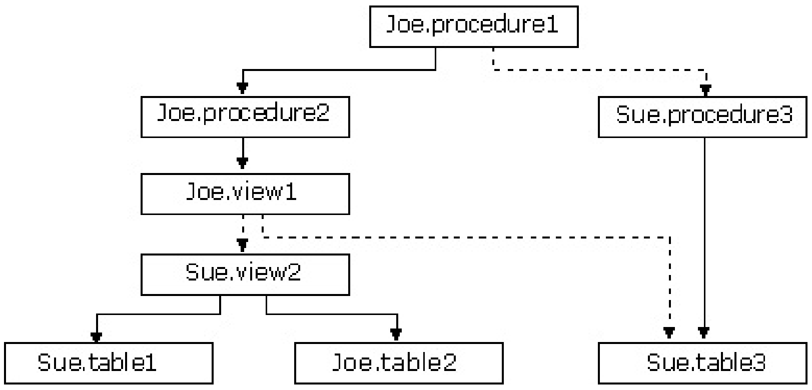
When a user accesses a view, Microsoft® SQL Server™ does not check permissions on any of the view's underlying objects if these objects and the view are all owned by the same user, and if the view and all its underlying objects are in the same database. If the same user owns a stored procedure and all the views or tables it references, and if the procedure and objects are all in the same database, SQL Server checks only the permissions on the procedure.

If the ownership chain of a procedure or view is broken (not all the objects in the chain are owned by the same user), SQL Server checks permissions on each object in the chain whose next lower link is owned by a different user. In this way, SQL Server allows the owner of the original data to retain control over its accessibility.

Usually, a user who creates a view has to grant permissions only on that view. For example, **Mary** has created a view called **auview1** on the **authors** table, which she also owns. If **Mary** grants **Sue** permission to use **auview1**, SQL Server allows **Sue** access to it without checking permissions on **authors**.

A user who creates a view or stored procedure that depends on an object owned by another user must be aware that any permissions he or she grants depend on the permissions allowed by the other owner.

For example, **Joe** creates a procedure called **procedure1**, which depends on **procedure2** (also owned by **Joe**), and **procedure3** (owned by **Sue**). These procedures in turn depend on other tables and views owned by **Joe** and **Sue**.



**Joe** grants **Mary** permission to use **procedure1**. SQL Server checks the permissions on **procedure1**, **procedure3**, **view2**, **table2**, and **table3** to check that **Mary** is allowed to use them.

## Using Views as Security Mechanisms

Views can serve as security mechanisms by restricting the data available to users. Some data can be accessible to users for query and modification, while the rest of the table or database is invisible and inaccessible. Permission to access the subset of data in a view must be granted, denied, or revoked, regardless of the set of permissions in force on the underlying table(s).

For example, the **salary** column in a table contains confidential employee information, but the rest of the columns contain information that should be available to all users. You can define a view that includes all of the columns in the table with the exception of the sensitive **salary** column. As long as table and view have the same owner, granting SELECT permissions on the view allows the user to see nonconfidential columns in the view without having any permissions on the table itself.

By defining different views and granting permissions selectively on them, users, groups, or roles can be restricted to different subsets of data. For example:

- Access can be restricted to a subset of the rows of a base table. For example, define a view that contains only rows for business and psychology books and keep information about other types of books hidden from users.

- Access can be restricted to a subset of the columns of a base table. For example, define a view that contains all the rows of the **titles** table but omits the **royalty** and **advance** columns because this information is sensitive.

- Access can be restricted to a row-and-column subset of a base table.

- Access can be restricted to the rows that qualify for a join of more than one base table. For example, define a view that joins the **titles**, **authors**, and **titleauthor** tables to display the names of authors and books they have written. This view hides personal data about the authors, and financial information about the books.

- Access can be restricted to a statistical summary of data in a base table. For example, define a view that contains only the average price of each type of book.

- Access can be restricted to a subset of another view or of some combination of views and base tables.

## Permissions and ALTER VIEW

Use the ALTER VIEW Transact-SQL statement to change the definition of a view without having to drop the view and reapply permissions. Any permissions applied to a column in the view are based on the column name defined in the view, rather than the underlying column in the table. Therefore, changing the definition of the view with ALTER VIEW by using the same column name but a different underlying column in a table results in the same permissions for the new column. This example assumes the user **Fred** exists in the **pubs** database:

```
USE pubs
GO
CREATE VIEW v1 AS SELECT title_id, title FROM titles
GO
GRANT SELECT(title_id) ON v1 TO Fred
GO
ALTER VIEW v1 AS SELECT qty AS 'title_id' FROM sales
GO
```

Although the view is altered so that the **title_id** column name refers to the **qty** column in the **sales** table, rather than the **title_id** column in the **titles** table, the SELECT permissions granted to **Fred** on the **title_id** column name still apply.

## See Also

ALTER VIEW

CREATE VIEW

## Using Stored Procedures as Security Mechanisms

Stored procedures, commonly used as an interface to perform complex activities, can be used to customize security permissions in much the same way as views.

For example, in an archiving scenario, stored procedures can copy data older than a specified interval into an archive table and then delete it from the primary table. Permissions can be used to prevent users from deleting the rows from the primary table directly or from inserting rows into the archive table without deleting them from the primary table. You can create a procedure to ensure that both of these activities are performed together, and then grant users permissions to execute the procedure.

### See Also

CREATE PROCEDURE

# Administering SQL Server

## Advanced Security Topics

The security topics presented here go beyond the basic use of security in Microsoft® SQL Server™ and provide more detail for specialized applications.

# Administering SQL Server

## Establishing Application Security and Application Roles

The security system in Microsoft® SQL Server™ is implemented at the lowest level: the database itself. This is the best method for controlling user activities regardless of the application used to communicate with SQL Server. However, sometimes security controls must be customized to accommodate the special requirements of an individual application, especially when dealing with complex databases and databases with large tables.

Additionally, you may want users to be restricted to accessing data only through a specific application (for example using SQL Query Analyzer or Microsoft Excel) or to be prevented from accessing data directly. Restricting user access in this way prohibits users from connecting to an instance of SQL Server using an application such as SQL Query Analyzer and executing a poorly written query, which can negatively affect the performance of the whole server.

SQL Server accommodates these needs through the use of application roles. Application roles are different than standard roles in that:

- Application roles contain no members.

  Microsoft Windows NT® 4.0 or Windows® 2000 groups, users, and roles cannot be added to application roles; the permissions of the application role are gained when the application role is activated for the user's connection through a specific application or applications. A user's association with an application role is due to his ability to run an application that activates the role, rather than his being a member of the role.

- Application roles are inactive by default and require a password to be activated.

- Application roles bypass standard permissions.

  When an application role is activated for a connection by the application, the connection permanently loses all permissions applied to the login, user account, or other groups or database roles in all databases for the duration of the connection. The connection gains the permissions associated with the application role for the database in which the application role exists. Because application roles are applicable only to the database in which they exist, the connection can gain access to another database only through permissions granted to the **guest** user account in the other database. Therefore, if the **guest** user account does not exist in a database, the connection cannot gain access to that database. If the guest user account does exist in the database but permissions to access an object are not explicitly granted to **guest**, the connection cannot access that object, regardless of who created the object. The permissions the user gained from the application role remain in effect until the connection logs out of an instance of SQL Server.

To ensure that all the functions of the application can be performed, a connection must lose default permissions applied to the login and user account or other groups or database roles in all databases for the duration of the connection and gain the permissions associated with the application role. For example, if a user is usually denied access to a table that the application must access, then the denied access should be revoked so the user can use the application successfully. Application roles overcome any conflicts with user's default permissions by temporarily suspending the user's default permissions and assigning them only the permissions of the application role.

Application roles allow the application, rather than SQL Server, to take over the responsibility of user authentication. However, because SQL Server still must authenticate the application when it accesses databases, the application must provide a password because there is no other way to authenticate an application.

If ad hoc access to a database is not required, users and Windows NT 4.0 or Windows 2000 groups do not need to be granted any permissions because all permissions can be assigned by the applications they use to access the database. In such an environment, standardizing on one system-wide password assigned to an application role is possible, assuming access to the applications is secure.

There are several options for managing application role passwords without hard-coding them into applications. For example, an encrypted key stored in the registry (or a SQL Server database), for which only the application has the decryption code, can be used. The application reads the key, decrypts it, and uses the value to set the application role. Using the Multiprotocol Net-Library, the network packet containing the password can also be encrypted. Additionally, the password can be encrypted, before being sent to an instance of SQL Server, when the role is activated.

When an application user connects to an instance of SQL Server using Windows Authentication Mode, an application role can be used to set the permissions the Windows NT 4.0 or Windows 2000 user has in a database when using the application. This method allows Windows NT 4.0 or Windows 2000 auditing of the user account and control over user permissions, while she uses the application, to be easily maintained.

If SQL Server Authentication is used and auditing user access in the database is not required, it can be easier for the application to connect to an instance of SQL Server using a predefined SQL Server login. For example, an order entry application authenticates users running the application itself, and then connects to an instance of SQL Server using the same **OrderEntry** login. All connections use the same login, and relevant permissions are granted to this login.

**Note** Application roles work with both authentication modes.

## Example

As an example of application role usage, a user **Sue** runs a sales application that requires SELECT, UPDATE, and INSERT permissions on the **Products** and **Orders** tables in database **Sales** to work, but she should not have any SELECT, INSERT, or UPDATE permissions when accessing the **Products** or **Orders** tables using SQL Query Analyzer or any other tool. To ensure this, create one user-database role that denies SELECT, INSERT, or UPDATE permissions on the **Products** and **Orders** tables, and add **Sue** as a member of that database role. Then create an application role in the **Sales** database with SELECT, INSERT, and UPDATE permissions on the **Products** and **Orders** tables. When the application runs, it provides the password to activate the application role by using **sp_setapprole**, and gains the permissions to access the **Products** and **Orders** tables. If **Sue** tries to log in to an instance of SQL Server using any tool except the application, she will not be able to access the **Products** or **Orders** tables.

## To create an application role

⊞ Transact-SQL

# Administering SQL Server

## Allowing Other Accounts to Grant Object Permissions

When you grant an object permission to a user account in a database, you can optionally specify the WITH GRANT OPTION clause, which allows the user account to grant that object permission to others. A user account can be a Microsoft® Windows NT® 4.0 or Windows® 2000 user or group or a Microsoft SQL Server™ user or role.

For example, if you use the WITH GRANT OPTION clause when you grant permissions on the **salaries** table to the user **user_a**, **user_a** is able to grant the same permissions on the table to any other user account in the database. For groups and roles, if you grant permissions on the **salaries** table to role **role_a** specifying the WITH GRANT OPTION clause, each member of **role_a** can grant the object permission to any other user account, provided that the AS clause of the GRANT statement is specified. For more information, see GRANT.

**IMPORTANT** When you use the WITH GRANT OPTION clause, you have no future control over which security accounts will receive that permission.

When you revoke a permission granted using the WITH GRANT OPTION clause, specify the CASCADE clause to have the permissions revoked from the user account as well as any other accounts that received the permission from the initial account.

For example, you have granted a permission specifying WITH GRANT OPTION to the user **user_a**. **User_a** granted the permission specifying WITH GRANT OPTION to the user **user_b**, and **user_b** granted the permission to the user **user_c**. **User_a** has left the company, but SQL Server does not allow you to remove a user account if it has granted a permission specifying the WITH GRANT OPTION clause to another account. Specifying the WITH GRANT OPTION clause has created a chain from **user_a** through **user_b** to **user_c**. You cannot remove the account for **user_a** until the permissions are revoked for **user_b** and **user_c**. When you revoke the permission from **user_a** and specify the CASCADE option, the permission is removed from the **user_a**, **user_b**, and **user_c** accounts. You then may remove the **user_a** account.

## Administering SQL Server

## Creating SQL Server File Permissions

Microsoft® SQL Server™ must create and access files in order to store databases, database backups, error logs, and so on. This SQL Server process must run in the context of a security account with the necessary permissions to create and access these files, whether these files exist on the local computer or a network drive on a remote computer. The security account SQL Server uses depends on the method used to start the instance of SQL Server. If an instance of SQL Server is started:

- As a service on Microsoft Windows NT® 4.0 or Windows® 2000 using the Service Control Manager, SQL Server uses the security account assigned to the SQL Server service.

- At the command prompt, independent of the Service Control Manager, SQL Server uses the security account of the logged on user.

- In Microsoft Windows 98 and Microsoft Windows Millennium Edition, SQL Server uses the security account of the logged on user.

The security account used by SQL Server requires full access permissions to the file system to create, read, write, delete, and execute files. For example, using the NTFS file system, the security account used by SQL Server requires authority to create files with NTFS Full Control permission.

To prevent unauthorized access to the files used by SQL Server, adjust the permissions on the files directly to allow only the security account used by SQL Server access to the files.

**Note** If SQL Server uses the Windows NT 4.0 and Windows 2000 **LocalSystem** built-in security account, file permissions must be granted to the **SYSTEM** account of the local computer running an instance of SQL Server.

### Securing the Windows NT Registry

SQL Server Setup removes write permissions from the

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSSQLServer\Providers key in the Windows 2000 registry for users who are not SQL Server system administrators. This prevents nonadministrator users from setting the provider options for linked server definitions when using SQL Server Enterprise Manager.

## See Also

Setting up Windows Services Accounts

Starting SQL Server

## Administering SQL Server

## Using Encryption Methods

Encryption is a method for keeping sensitive information confidential by changing data into an unreadable form. Encryption ensures that data remains secure by keeping the information hidden from everyone, even if the encrypted data is viewed directly. Decryption is the process of changing encrypted data back into its original form so it can be viewed by authorized users.

Microsoft® SQL Server™ encrypts or can encrypt:

- Login and application role passwords stored in SQL Server.
- Any data sent between the client and the server as network packets.
- Stored procedure definitions.
- User-defined function definitions.
- View definitions.
- Trigger definitions.
- Default definitions.
- Rule definitions.

**Note** If you are running Microsoft Windows® 2000 and want to use the Windows 2000 Encrypted File System to encrypt any SQL Server files, you must unencrypt the files before you can change the SQL Server service accounts. If you do not unencrypt the files and then reset the SQL Server service accounts, you cannot unencrypt the files.

## Login and Application Role Passwords

Login and application role passwords stored in the SQL Server system tables are always encrypted. This prevents users, including system administrators, from viewing any passwords, including their own. Additionally, application role passwords can be encrypted when the application role is activated before they are sent over the network.

**Note** Using the **sp_addlogin** system stored procedure, SQL Server logins can be added without encrypting the password, if required. However, this is not recommended unless the passwords are already encrypted because they are being imported from another instance of SQL Server.

## Data in Network Packets

SQL Server allows data sent between the client and the server to be encrypted. This ensures that any application or user intercepting the data packets on the network cannot view confidential or sensitive data (for example, passwords sent across the network as a user logs into an instance of SQL Server). SQL Server can use the Secure Sockets Layer (SSL) to encrypt all data transmitted between an application computer and an instance of SQL Server. The SSL encryption is performed within the Super Socket Net-Library (Dbnetlib.dll and Ssnetlib.dll) and applies to all inter-computer protocols supported by SQL Server 2000. Enabling encryption slows the performance of the Net-Libraries. Encryption forces the following actions in addition to all of the work for an unencrypted connection:

- An extra network round trip is required at connect time.
- All packets sent from the application to the instance of SQL Server must be encrypted by the client Net-Library and decrypted by the server Net-Library.
- All packets sent from the instance of SQL Server to the application must be encrypted by the server Net-Library and decrypted by the client Net-Library.

Shared memory Net-Library communications are inherently secure without the need for encryption. The shared memory Net-Library does not participates in inter-computer communications. The area of memory shared between the application process and the database engine process cannot be accessed from any other Windows process.

For compatibility with earlier versions of SQL Server, the Multiprotocol Net-Library continues to support its own encryption. This encryption is specified independently of the SSL encryption and is implemented by calling the Windows RPC encryption API. It does not require the use of certificates. The level of RPC encryption, 40-bit or 128-bit, depends on the version of the Windows operating system that is running on the application and database computers. The Multiprotocol Net-Library is not supported by named instances. For more information about SSL, see Net-Library Encryption.

## Configuring a Multiprotocol Alias

When you configure a multiprotocol alias, enable encryption. This encryption feature applies only to the Multiprotocol Net-Library. This encryption feature is offered only for compatibility with existing applications. SQL Server clients should use the SSL encryption specified on the **General** tab in the **Enable protocol encryption** check box of the Client Network Utility. For more information on the Client Network Utility, see Configuring Client Net-Libraries.

### To start the Client Network Utility

# Administering SQL Server

# Revealing SQL Server on a Network

When you install Microsoft® SQL Server™, SQL Server Setup makes an entry in the Microsoft Windows® 2000 registry that enables Named Pipes clients to see SQL Server in a server enumeration box in SQL Query Analyzer. SQL Server automatically announces itself as a service over Named Pipes to make it easier to locate servers running an instance of SQL Server. However, if you are using Active Directory™, the directory service included in Windows 2000, this functionality is no longer necessary.

Stop SQL Server from announcing itself over Named Pipes by running the NET CONFIG SERVER command with the switch as /HIDDEN:YES. You can reveal the server at any time.

### To reveal or cancel the announcement of SQL Server on a network

# Administering SQL Server

# Scripting Data Access Controls in Internet Explorer

Microsoft® SQL Server™ ships with several data access controls:

- SQL Namespace (SQL-NS)
- SQL Distribution control (replication)
- SQL Merge control (replication)

These controls are signed and marked "safe for initialization and scripting" and can be used in Microsoft Internet Explorer 5 or later.

Before deploying controls that can connect to data sources, you should thoroughly understand the security implications. When you use any of the SQL Server controls, the primary security concern is the ability to run under the authorized user's account through a Windows Authentication login to an instance of SQL Server. A Web page with a scripted control runs with the network identity of the user browsing the page. If the data source connection is based on the connected user's network identity (using Windows Authentication login), the control can access any data that the user browsing the page can access. If a Web page using the control is sent to a user, the control has the permissions of the user browsing the Web page. The control can then read or make changes to databases without the user's knowledge.

To prevent unauthorized access or changes to a database, all the data access controls that are marked as "safe for scripting" take into account security zones settings when being loaded in Internet Explorer version 4.0 or later. If a control is not marked safe for scripting, it can run a script inside of Internet Explorer only at the **Low** security mode of Internet Explorer, and even then only after the user responded to a message stating that a script will be run. Another way to deal with the issue is to remove the user's ability to use a Windows Authenticated login.

Internet Explorer 4.0 does not provide an explicit security option for data access. Therefore, all the controls marked safe for scripting allow, prompt, or disallow scripting based on the security zone being used. The following table shows the

Internet Explorer 4.0 settings.

| Security zone | Internet Explorer 4.0 notification |
|---|---|
| **Local computer zone** | Controls can be initialized or scripted regardless of data source or scripts. |
| **Local intranet zone** | User is warned of potential safety violation prior to loading the page. User can accept or reject initialization or scripting. |
| **Trusted sites zone** | Controls can be initialized or scripted regardless of data source or scripts. |
| **Internet zone** | User is warned of potential safety violation prior to loading the page. User can accept or reject initialization or scripting. |
| **Restricted sites zone** | Scripting errors occur if user attempts to view page and execute script. |

In contrast to Internet Explorer 4.0, Internet Explorer 5 supports an explicit security option for data access called "Access data sources across domains." This option can be customized, and the setting of this action is used to determine how the controls behave when they are run in Internet Explorer 5. The default settings in Internet Explorer 5 are the same as the programmed settings in Internet Explorer 4.0.

As with all security concerns, you must take specific actions to safeguard your system. SQL Server is protected from security problems only if users with the ability to use Windows Authenticated logins configure the security settings correctly, and answer all security prompts correctly.

**Note** These general steps to safeguard your system apply to any scripting host, including Microsoft Excel spreadsheets or Microsoft Word documents. Users who have the ability to use Windows Authenticated logins should always enable the macro warning feature or similar security setting of an application to detect and prevent any attacks on data.

## See Also

Developing SQL-DMO Applications

Programming SQL-NS Applications

# Administering SQL Server

# Auditing SQL Server Activity

Microsoft® SQL Server™ 2000 provides auditing as a way to trace and record activity that has happened on each instance of SQL Server (for example, successful and failed logins). SQL Server 2000 also provides an interface, SQL Profiler, for managing audit records. Auditing can only be enabled or modified by members of the **sysadmin** fixed security role, and every modification of an audit is an auditable event.

There are two type of auditing:

- Auditing, which provides some level of auditing but does not require the same number of policies as C2 auditing.

- C2 auditing, which requires that you follow very specific security policies. For more information about C2 auditing, see C2 Auditing.

Both types of auditing can be done by using SQL Profiler.

## Using SQL Profiler

SQL Profiler provides the user interface for auditing events. There are several categories of events that can be audited using SQL Profiler, such as:

- End user activity (all SQL commands, logout/login, enabling of application roles).

- DBA activity (DDL, other than grant/revoke/deny and security events, Configuration (DB or server).

- Security events (grant/revoke/deny, login user/role add/remove/configure).

- Utility events (backup/restore/bulk insert/BCP/DBCC commands.

- Server events (shutdown, pause, start).
- Audit events (add audit, modify audit, stop audit).

For more information about what categories of events can be monitored, see Security Audit Event Category.

It is possible to audit the following aspects of SQL Server through SQL Profiler:

- Date and time of event.
- User who caused the event to occur.
- Type of event.
- Success or failure of the event.
- The origin of the request (for example, the Microsoft Windows NT® 4.0 computer name).
- The name of the object accessed.
- Text of the SQL statement (passwords replaced with ****).
- If you are a member of the **sysadmin** or **securityadmin** fixed server role and you reset your own password by using **sp_password** with all three arguments specified ('*old_password*', '*new_password*', '*login*'), the audit record will reflect that you are changing someone else's password.

Auditing can have a significant performance impact. If all audit counters are turned on for all objects, the performance impact could be high. It is necessary to evaluate how many events need to be audited compared to the resulting performance impact. Audit trail analysis can be costly, so it is recommended that audit activity be run on a server separate from the production server.

**Note** If SQL Server is started with the **-f** flag, auditing will not run.

## See Also

Monitoring with SQL Profiler

# Administering SQL Server

## Using Audit Logs

SQL Profiler system stored procedures support file rollover. The maximum file size for the audit log is fixed at 200 megabytes (MB). When the audit log file reaches 200 MB, a new file will be created and the old file handle will be closed. If the directory fills up (for example, if the disk quota for the user of the service account has filled up or the disk is full), then the instance of Microsoft® SQL Server™ is stopped. The system administrator needs to either free up disk space for the audit log before restarting the instance of SQL Server or restart the instance of SQL Server (if auditing is not configured to start automatically).

Use file rollover to prevent the audit trace from failing because the audit log filled up. However, SQL Server will not shut down unless the user specifically requested this feature when they created the trace. An audit failure produces an entry in the Microsoft Windows® event log and the SQL Server error log.

It is strongly recommended that during SQL Server Setup you create a new directory to contain your audit files. \mssql\audit is the suggested path. If you are running SQL Server on a named instance, the suggested path is MSSQL$Instance\audit.

# Administering SQL Server

## C2 Auditing

C2 auditing is necessary if you are running a C2 certified system. A C2 certified system meets a government standard that defines the security level. To have a C2 certified Microsoft® SQL Server™, you must configure SQL Server in the evaluated C2 configuration. For more information about C2 certification, see the C2 Administrator's and User's Security Guide.

# Administering SQL Server

## Monitoring Server Performance and Activity

Microsoft® SQL Server™ 2000 provides a variety of tools that can be used to monitor the performance of an instance of SQL Server and the user activity that occurs in databases. Monitoring allows you to determine whether your database application is working efficiently and as expected, even as your application, database, and environment change. For example, as more concurrent users use a database application, the load on SQL Server can increase. By monitoring, you can determine whether the current instance of SQL Server or system configuration must be changed to handle the increased workload, or whether the increased load is having no significant effect on performance.

To monitor an application, an instance of SQL Server, or the operating system environment (hardware and software):

- Determine your monitoring goals.

- Choose the appropriate tool for the type of monitoring you will perform.

- Use the tool to monitor SQL Server or the system environment and analyze the captured data.

- Identify the events to monitor.

  The events determine which activities are monitored and captured. Your selection of events to monitor will depend on what is being monitored and why. For example, when monitoring disk activity, it is not necessary to monitor SQL Server locks.

- Determine the event data to capture.

  The event data describes each instance of an event as it occurs. For example, when monitoring lock events, you can capture data describing the tables, users, and connections affected by the lock event. The following explains the process involved in capturing event data and putting it to use.

- Apply filters to limit the event data collected.

  Limiting the event data allows the system to focus on the events pertinent to the monitoring scenario. For example, if you want to monitor slow queries, you can use a filter to monitor only those queries issued by the application that take more than 30 seconds to execute against a particular database.

- Monitor (capture) events.

  Once enabled, active monitoring captures data from the specified application, instance of SQL Server, or operating system. For example, when disk activity is monitored using System Monitor (Performance Monitor in Microsoft Windows NT® 4.0), monitoring captures event data such as disk reads and writes and displays it to the screen.

- Save captured event data.

  Saving captured data allows you to analyze it at a later time or even replay it using SQL Profiler. Captured event data is saved to a file that can be loaded back into the tool that originally created the file for analysis. SQL Profiler allows event data to be saved to a SQL Server table. Saving captured event data is vital when creating a performance baseline. The performance baseline data is saved and used when comparing recently captured event data to determine whether performance is optimal.

- Create definition files containing the settings specified to capture the events.

  Definition files include specifications about the events themselves, event data, and filters that are used to capture data. These files can be used to monitor a specific set of events at a later time without redefining the events, event data, and filters. For example, if you want to monitor frequently the number of deadlocks and the users involved in those deadlocks, you can create a file defining those events, event data, and event filters; save the definition; and reapply the filter the next time you want to monitor deadlocks. SQL Profiler uses trace definition files for this purpose.

- Analyze captured event data.

  In order to be analyzed, the captured, saved event data is loaded into the application that captured the data. For example, a captured trace from SQL Profiler can be reloaded into SQL Profiler for viewing and analysis. Analyzing event data involves determining what is happening and why. This information allows you to make changes that can improve performance, such as adding more memory, changing indexes, correcting coding problems with Transact-SQL statements or stored procedures, and so on, depending on the type of analysis performed. For example, you can use the Index Tuning Wizard to analyze a captured trace from SQL Profiler automatically and make index recommendations based on the results.

- Replay captured event data.

  Only available in SQL Profiler, event replay allows you to establish a test copy of the database environment from which the data was captured and repeat the captured events as they occurred originally on the real system. You can replay them at the same speed as they originally occurred, as fast as possible (to stress the system), or more likely, one step at a time (to analyze the system after each event has occurred). By analyzing the exact events in a test environment, you can prevent detrimental effects on the production system.

Monitoring SQL Server allows you to:

- Determine whether it is possible to improve performance. For example, by monitoring the response times for frequently used queries, you can determine whether changes to the query or indexes on the tables are necessary.

- Evaluate user activity. For example, by monitoring users attempting to connect to an instance of SQL Server, you can determine whether security is set up adequately and test applications and development systems. For example, by monitoring SQL queries as they are executed, you can determine whether they are written correctly and producing the expected results.

- Troubleshoot any problems or debug application components, such as stored procedures.

## See Also

Index Tuning Wizard

Optimizing Database Performance Overview

Administering SQL Server

## Evaluating Performance

Optimal performance comes from minimal response times and maximum throughput as a result of efficient network traffic, disk I/O, and CPU time. This goal is achieved by analyzing thoroughly the application requirements, understanding the logical and physical structure of the data, and assessing and negotiating tradeoffs between conflicting uses of the database, such as online transaction processing (OLTP) versus decision support.

### Response Time vs. Throughput

Response time is measured as the length of time required for the first row of the result set to be returned to the user in the form of visual confirmation that a query is being processed.

Throughput is a measure of the total number of queries handled by the server during a given time.

As the number of users increases, so does the competition for a server's resources, which in turn causes response time to increase and overall throughput to decrease.

### Factors That Affect Performance

The following areas affect the performance of SQL Server:

- System resources (hardware)
- The Microsoft Windows NT® 4.0 and Windows® 2000 operating systems
- Database applications
- Client applications

- Network

Before these areas can be monitored, you must know what level of performance is reasonable given normal working conditions. To do this, establish a server performance baseline by monitoring Microsoft® SQL Server™ performance at regular intervals, even when no problems occur.

## Troubleshooting Problems

You can monitor the following areas to troubleshoot problems:

- SQL Server stored procedures or batches of SQL statements submitted by user applications.
- User activity, such as blocking locks or deadlocks.
- Hardware activity, such as disk usage

Problems can include:

- Application development errors involving incorrectly written Transact-SQL statements.
- Hardware errors, such as disk or network-related errors.
- Excessive blocking due to an incorrectly designed database.

SQL Profiler can be used to monitor and troubleshoot Transact-SQL and application-related problems. System Monitor (Performance Monitor in Windows NT 4.0) can be used to monitor hardware and other system-related problems.

## Establishing a Performance Baseline

To determine whether your Microsoft® SQL Server™ system is performing optimally, take performance measurements over time and establish a server performance baseline. Compare each new set of measurements with those taken earlier.

After you establish a server performance baseline, compare the baseline statistics to current server performance. Numbers far above or far below your baseline are candidates for further investigation. They may indicate areas in need of tuning or reconfiguration. For example, if the amount of time to execute a set of queries increases, examine the queries to determine if they can be rewritten or if column statistics or new indexes must be added.

At a minimum, use baseline measurements to determine:

- Peak and off-peak hours of operation.
- Production query or batch command response times.
- Database backup and restore completion times.

### See Also

sp_configure

# Identifying Bottlenecks

Bottlenecks are caused by excessive demand on a system resource, and they are present in every system, to varying degrees. By monitoring the Microsoft® SQL Server™ system for bottlenecks, you can determine whether changes can be made to the limiting component to make it perform at an optimal level.

Reasons that bottlenecks occur include:

- Insufficient resources, requiring additional or upgraded components.
- Resources of the same type that do not share workloads evenly (for example, one disk is being monopolized).
- Malfunctioning resources.
- Incorrectly configured resources.

## Analyzing Bottlenecks

When analyzing event data, low numbers can be just as meaningful as high numbers. If a number is lower than expected, it may indicate a problem in another area. For example:

- Some other component may be preventing the load from reaching this component.
- Network congestion may be preventing client requests from reaching the server.
- A bottleneck may be preventing client computers from accessing the server as frequently as expected.

- System Monitor (Performance Monitor in Microsoft Windows NT® 4.0) may be employed incorrectly. For example, if you have not turned on the disk counters, or you are looking at the wrong instance, the wrong counters, or at the wrong computer, event data numbers may appear inexplicably low.

A low number also can mean that the system is performing better than expected.

These are five key areas to monitor when tracking server performance and identifying bottlenecks.

| **Bottleneck candidate** | **Effects on the server** |
|---|---|
| Memory usage | Insufficient memory allocated or available to SQL Server will degrade performance. Data must be read from the disk continually rather than residing in the data cache. Windows NT 4.0 and Microsoft Windows® 2000 perform excessive paging by swapping data to and from the disk as the pages are needed. |
| CPU processor utilization | A constantly high CPU rate may indicate the need for a CPU upgrade or the addition of multiple processors. |
| Disk I/O performance | A slow disk I/O (disk reads and writes) will cause transaction throughput to degrade. |
| User connections | An improperly configured number of users can cause the system to run slowly or restrict the amount of memory otherwise available to SQL Server. |
| Blocking locks | A process may be forcing another process to wait, thereby slowing down or stopping the blocking process. |

## See Also

Monitoring CPU Use

Monitoring Disk Activity

Monitoring Memory Usage

SQL Server: General Statistics Object

SQL Server: Locks Object

## Determining User Activity

You can monitor individual user activity to pinpoint transactions that may be blocking other transactions or causing the performance of Microsoft® SQL Server™ to be slower than expected.

Monitoring user activity helps identify trends such as the types of transactions run by certain users, the number of inefficient ad hoc queries being run, and the types of transactions requiring the most resources.

To collect statistical information about users, use either SQL Profiler or System Monitor (Windows NT Performance Monitor in Windows NT® 4.0). Use the SQL Server Enterprise Manager Current Activity window to perform ad hoc monitoring of SQL Server, which allows you to determine user activity on the system.

### See Also

Monitoring with SQL Server Enterprise Manager

Sessions Event Category

SQL Server: General Statistics Object

# Administering SQL Server

# Choosing a Monitoring Tool

Microsoft® SQL Server™ provides a comprehensive set of tools for monitoring events in SQL Server. Your choice of tool will depend on the type of monitoring and the events to be monitored. For example, ad hoc monitoring to determine the number of users currently connected to an instance of SQL Server can be accomplished by using the **sp_who** system stored procedure, rather than creating a trace and using SQL Profiler.

## SQL Profiler

Enables you to monitor server and database activity (for example, number of deadlocks, fatal errors, tracing stored procedures and Transact-SQL statements, or login activity). You can capture SQL Profiler data to a SQL Server table or a file for later analysis, and also replay the events captured on SQL Server, step by step, to see exactly what happened. SQL Profiler tracks engine process events, such as the start of a batch or a transaction.

## System Monitor

Enables you to monitor server performance and activity using predefined objects and counters or user-defined counters to monitor events. System Monitor (Performance Monitor in Microsoft Windows NT® 4.0) collects counts rather than data about the events (for example, memory usage, number of active transactions, number of blocked locks, or CPU activity). You can set thresholds on specific counters to generate alerts that notify operators. System Monitor primarily tracks resource usage, such as the number of buffer manager page requests in use.

System Monitor works only on Microsoft Windows® 2000 and can monitor (remotely or locally) an instance of SQL Server on Windows NT 4.0 or Windows 2000 only.

## Current activity window (SQL Server Enterprise Manager)

Graphically displays information about processes running currently on an instance of SQL Server, blocked processes, locks, and user activity. This is useful for ad hoc views of current activity.

## Error Logs

Contain additional information about events in SQL Server than is available elsewhere. You can use the information in the error log to troubleshoot SQL Server-related problems. The Windows application event log provides an overall picture of events occurring on the Windows NT 4.0 and Windows 2000 system as a whole, as well as events in SQL Server, SQL Server Agent, and full-text search.

## sp_who

Reports snapshot information about current SQL Server users and processes, including the currently executing statement and whether the statement is blocked. This is a Transact-SQL alternative to viewing user activity in the current activity window in SQL Server Enterprise Manager.

## sp_lock

Reports snapshot information about locks, including the object ID, index ID, type of lock, and type or resource to which the lock applies. This is a Transact-SQL alternative to viewing lock activity in the current activity window in SQL Server Enterprise Manager.

## sp_spaceused

Displays an estimate of the current amount of disk space used by a table (or a whole database). This is a Transact-SQL alternative to viewing database usage in SQL Server Enterprise Manager.

## sp_monitor

Displays statistics, including CPU usage, I/O usage, and the amount of time idle since **sp_monitor** was last executed.

## DBCC statements

Enables you to check performance statistics and the logical and physical consistency of a database. For more information, see DBCC.

### Built-in functions

Display snapshot statistics about SQL Server activity since the server was started; these statistics are stored in predefined SQL Server counters. For example, @@CPU_BUSY contains the amount of time the CPU has been executing SQL Server code; @@CONNECTIONS contains the number of SQL Server connections or attempted connections; and @@PACKET_ERRORS contains the number of network packets occurring on SQL Server connections. For more information, see Functions.

### SQL Profiler stored procedures and functions

Use Transact-SQL stored procedures to gather SQL Profiler statistics. For more information, see System Stored Procedures.

### Trace flags

Display information about a specific activity within the server and are used to diagnose problems or performance issues (for example, deadlock chains). For more information, see Trace Flags.

### Simple Network Management Protocol (SNMP)

Simple Network Management Protocol (SNMP) is an application protocol that offers network management services. Using SNMP, you can monitor an instance of SQL Server across different platforms (for example, Windows NT 4.0, Windows 98, and UNIX). With SQL Server and the Microsoft SQL Server Management Information Base (MSSQL-MIB), you can use SNMP applications to monitor the status of SQL Server installations. You can monitor performance information, access databases, and view server and database configuration parameters.

The choice of a monitoring tool depends on the type of events and activity to be monitored.

| | SQL | System | Current activity | Transact- | Error |
|---|---|---|---|---|---|

| Event or activity | Profiler | Monitor | window | SQL | logs |
|---|---|---|---|---|---|
| Trend analysis | Yes | Yes | | | |
| Replaying captured events | Yes | | | | |
| Ad hoc monitoring | Yes | | Yes | Yes | Yes |
| Generating alerts | | Yes | | | |
| Graphical interface | Yes | Yes | Yes | | Yes |
| Using within custom application | Yes 1 | | | Yes | |

1 Using SQL Profiler system stored procedures.

The key difference between the two main monitoring tools, SQL Profiler and System Monitor, is that SQL Profiler monitors engine events while System Monitor monitors resource usage associated with server processes. For example, SQL Profiler can be used to monitor deadlocks events, including the users and objects involved in the deadlock. System Monitor can be used to monitor the total number of deadlocks occurring in a database or on a specific object.

Windows NT 4.0 and Windows 2000 also provides these monitoring tools:

- Task Manager

  Shows a synopsis of the processes and applications running on the system.

- Network Monitor Agent

  Assists in monitoring network traffic.

For more information about Windows NT 4.0 or Windows 2000 tools, see the Windows NT 4.0 or Windows 2000 documentation.

# Administering SQL Server

## Monitoring with SQL Profiler

SQL Profiler is a graphical tool that allows system administrators to monitor events in an instance of Microsoft® SQL Server™. You can capture and save data about each event to a file or SQL Server table to analyze later. For example, you can monitor a production environment to see which stored procedures are hampering performance by executing too slowly.

Use SQL Profiler to monitor only the events in which you are interested. If traces are becoming too large, you can filter them based on the information you want, so that only a subset of the event data is collected. Monitoring too many events adds overhead to the server and the monitoring process and can cause the trace file or trace table to grow very large, especially when the monitoring process takes place over a long period of time.

After you have traced events, SQL Profiler allows captured event data to be replayed against an instance of SQL Server, thereby effectively reexecuting the saved events as they occurred originally.

Use SQL Profiler to:

- Monitor the performance of an instance of SQL Server.
- Debug Transact-SQL statements and stored procedures.
- Identify slow-executing queries.
- Test SQL statements and stored procedures in the development phase of a project by single-stepping through statements to confirm that the code works as expected.
- Troubleshoot problems in SQL Server by capturing events on a production system and replaying them on a test system. This is useful for testing or debugging purposes and allows users to continue using the production system without interference.

- Audit and review activity that occurred on an instance of SQL Server. This allows a security administrator to review any of the auditing events, including the success and failure of a login attempt and the success and failure of permissions in accessing statements and objects.

SQL Profiler provides a graphical user interface to a set of stored procedures that can be used to monitor an instance of SQL Server. For example, it is possible to create your own application that uses SQL Profiler stored procedures to monitor SQL Server.

You must have at least 10 megabytes (MB) of free space to run SQL Profiler. If free space drops below 10 MB while you are using SQL Profiler, all SQL Profiler functions will stop.

## Starting SQL Profiler

SQL Profiler is started from the Microsoft® Windows NT® 4.0, Microsoft Windows® 2000 or Microsoft Windows 98 **Start** menu, or from SQL Server Enterprise Manager.

With Windows Authentication mode, the user account that runs SQL Profiler must be granted permission to connect to an instance of SQL Server. The login account also must be granted permissions to execute SQL Profiler stored procedures. For more information, see System Stored Procedures.

### To start SQL Profiler

Administering SQL Server

## SQL Profiler Keyboard Shortcuts

The following table shows the keyboard shortcuts available in SQL Profiler.

| | |
|---|---|
| **CTRL+Shift+Delete** | Clear a trace window |
| **CTRL+F4** | Close a trace window |
| **-** | Collapse a trace grouping |
| **CTRL+C** | Copy |
| **ALT+Delete** | Delete a trace |
| **+** | Expand a trace grouping |
| **CTRL+F** | Find |
| **F3** | Find the next item |
| **Shift+F3** | Find the previous item |
| **F1** | Display available help |
| **CTRL+N** | Open a new trace |
| **ALT+F7** | Replay the settings |
| **CTRL+F10** | Run to cursor |
| **F5** | Start a replay |
| **F11** | Step |
| **Shift+F5** | Stop a replay |
| **F9** | Toggle a breakpoint |

# Administering SQL Server

## SQL Profiler Terminology

To use SQL Profiler, you need to understand the terminology that describes the way the tool functions. For example, you create a template that defines the data you want to collect. You collect this data by running a trace on the events defined in the template. While the trace is running, the event classes and data columns that describe the event data are displayed in SQL Profiler.

### Template

A template defines the criteria for each event you want to monitor with SQL Profiler. For example, you can create a template, specifying which events, data columns, and filters to use. Then you can save the template and launch a trace with the current template settings. The trace data captured is based upon the options specified in the template. A template is not executed, and must be saved to a file with the .tdf extension.

### Trace

A trace captures data based upon the selected events, data columns, and filters. For example, you can create a template to monitor exception errors. To do this, you would select to trace the **Exception** event class, and the **Error**, **State**, and **Severity** data columns, which need to be collected for the trace results to provide meaningful data. After you save the template, you can then run it as a trace, and collect data on any **Exception** events that occur in the server. This trace data can be saved and then replayed at a later date, or used immediately for analysis.

#### Filter

- When you create a trace or template, you can define criteria to filter the data collected by the event. If traces are becoming too large, you can filter them based on the information you want, so that only a subset of the event data is collected. If a filter is not set, all events of the selected event classes are returned in the trace output. For example, you can limit the Microsoft® Windows® 2000 user names in the trace to specific users, reducing the output data to only those users in which you are interested.

## Event Category

An event category defines the way events are grouped. For example, all lock events classes are grouped within the Locks event category. However, event categories only exist within SQL Profiler. This term does not reflect the way engine events are grouped.

## Event

An event is an action generated within the Microsoft SQL Server™ engine. For example:

- The login connections, failures, and disconnections.
- The Transact-SQL SELECT, INSERT, UPDATE, and DELETE statements.
- The remote procedure call (RPC) batch status.
- The start or end of a stored procedure.
- The start or end of statements within stored procedures.
- The start or end of an SQL batch.
- An error written to the SQL Server error log.
- A lock acquired or released on a database object.
- An opened cursor.
- Security permissions checks.

All of the data that is generated as a result of an event is displayed in the trace in a single row. This row contains columns of data called event classes that describe the event in detail.

## Event Class

An event class is the column that describes the event that was produced by the server. The event class determines the type of data collected, and not all data columns are applicable to all event classes. Examples of event classes include:

- **SQL:BatchCompleted**, which indicates the completion of an SQL batch.

- The name of the computer on which the client is running.

- The ID of the object affected by the event, such as a table name.

- The SQL Server name of the user issuing the statement.

- The text of the Transact-SQL statement or stored procedure being executed.

- The time the event started and ended.

## Data Column

The data columns describe the data collected for each of the event classes captured in the trace. Because the event class determines the type of data collected, not all data columns are applicable to all event classes. For example, the **Binary Data** data column, when captured for the **Lock:Acquired** event class, contains the value of the locked page ID or row but has no value for the **Integer Data** event class. Default data columns are populated automatically for all event classes.

# Administering SQL Server

## SQL Profiler Scenarios

Typically, you use SQL Profiler to:

- Find the worst-performing queries

  For example, you can create a trace that captures events relating to **TSQL** and **Stored Procedure** event classes, specifically **RPC:Completed** and **SQL:BatchCompleted**. Include all data columns in the trace, group by **Duration**, and specify event criteria. For example, if you specify that the **Duration** of the event must be at least 1,000 milliseconds, you can eliminate short-running events from the trace. The **Duration** minimum value can be increased as required. If you want to monitor only one database at a time, specify a value for the **Database ID** event criteria.

- Identify the cause of a deadlock

  For example, you can create a trace that captures events relating to **TSQL** and **Stored Procedure** event classes (**RPC:Starting** and **SQL:BatchStarting**) and **Locks** event classes (**Lock:Deadlock** and **Lock:Deadlock Chain**). Include all data columns in the trace and group by **Event Class**. If you want to monitor only one database at a time, specify a value for the **Database ID** event criteria.

  To view the connections involved in a deadlock, do one of the following:

  - Open the trace containing the captured data, group the data by **ClientProcessID**, and expand both connections involved in the deadlock.

  - Save the captured data to a trace file and open the trace file twice to make the file visible in two separate SQL Profiler windows. Group the captured data by **ClientProcessID** and then expand the client process ID involved in the deadlock; each deadlocked connection is in a separate window. Tile the windows to view the events causing the deadlock.

- Monitor stored procedure performance

  For example, you can create a trace that captures events relating to **Stored** Procedures event classes (**SP:Completed**, **SP:Starting**, **SP:StmtCompleted** and **SP:StmtStarting**), and **TSQL** event classes (**SQL:BatchStarting** and **SQL:BatchCompleted**). Include all data columns in the trace and group by **ClientProcessID**. If you want to monitor only one database at a time, specify a value for the **Database ID** event criteria. Similarly, if you want to monitor only one stored procedure at a time, specify a value for the **Object ID** event criteria.

- Audit Microsoft® SQL Server™ activity

You can audit activity in SQL Server using SQL Profiler. For example, if the security administrator always needs to know who is logged in to the server, you can create a SQL Profiler trace that provides a complete view of users who have logged in or out of the server. This information can then be used for legal purposes to document activity and for technical purposes to track security policy violations.

To set up a SQL Profiler trace that tracks users who have logged in or out of the server, do the following:

1. Create a trace, selecting **Audit Login Event**.

2. To return the appropriate information, specify the following data columns:

   **EventClass** (selected by default)

   **EventSubClass**

   **LoginSID**

   **LoginName**

- Monitor Transact-SQL activity per user.

  You can create a trace that captures events relating to the **Sessions** event class, **ExistingConnection**, and TSQL event classes. Include all data columns in the trace, do not specify any event criteria, and group the captured events by **DBUserName**.

## See Also

Locks Event Category

Sessions Event Category

Stored Procedures Event Category

TSQL Event Category

# Administering SQL Server

# Monitoring with SQL Profiler Event Categories

In SQL Profiler, use event categories to monitor events in Microsoft® SQL Server™. Event categories contain event classes that have been grouped together within the SQL Profiler user interface. For more information, see SQL Profiler Terminology.

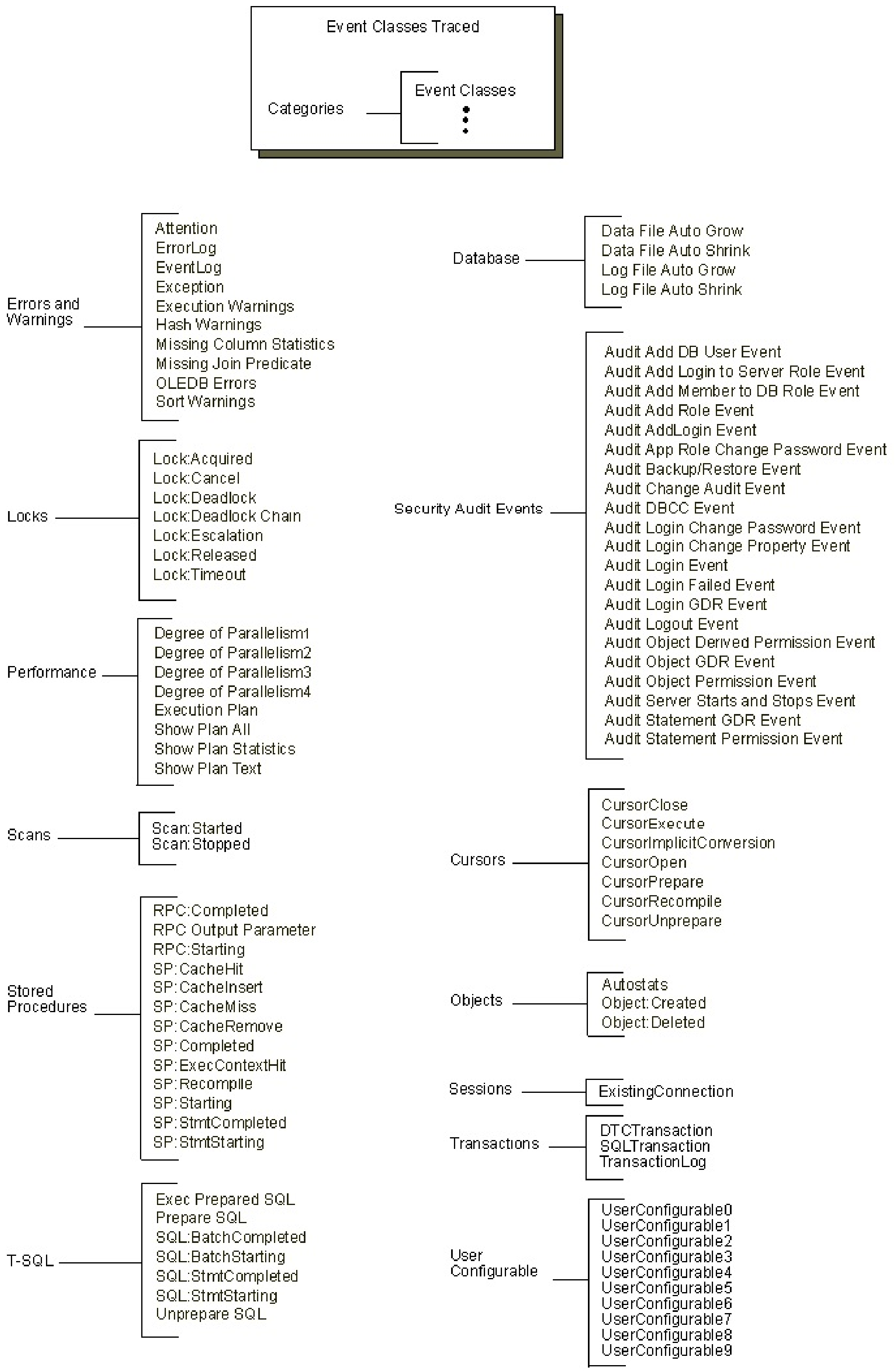
The following table describes the SQL Profiler event categories and their associated event classes.

| Event category | Description |
|---|---|
| Cursors | Collection of event classes produced by cursor operations. |
| Database | Collection of event classes produced when data or log files grow or shrink automatically. |
| Errors and Warnings | Collection of event classes produced when a SQL Server error or warning occurs (for example, an error during the compilation of a stored procedure or an exception in SQL Server). |
| Locks | Collection of event classes produced when a lock is acquired, cancelled, released, etc. |
| Objects | Collection of event classes produced when database objects are created, opened, closed, dropped, or deleted. |
| Performance | Collection of event classes produced when SQL data manipulation (DML) operators execute. |
| Scans | Collection tables and indexes are scanned. |
| Security Audit | Collection of event classes used to audit server activity. |
| Sessions | Collection of event classes produced by clients connecting to and disconnecting from an instance of SQL Server. |
| Stored Procedures | Collection of event classes produced by the execution of stored procedures. |
| Transactions | Collection of event classes produced by the execution |

| | |
|---|---|
| | of Microsoft Distributed Transaction Coordinator (MS DTC) transactions or by writing to the transaction log. |
| TSQL | Collection of event classes produced by the execution of Transact-SQL statements passed to an instance of SQL Server from the client. |
| User Configurable | Collection of user-configurable event classes. |

## SQL Profiler Event Classes

In SQL Profiler, event classes are rows that describe the events you are tracing. Within SQL Profiler, event classes are grouped into event categories. For example, all lock event classes are grouped within the **Locks** event category. For more information, see SQL Profiler Terminology.



- Errors and Warnings
  - Attention
  - ErrorLog
  - EventLog
  - Exception
  - Execution Warnings
  - Hash Warnings
  - Missing Column Statistics
  - Missing Join Predicate
  - OLEDB Errors
  - Sort Warnings
- Locks
  - Lock:Acquired
  - Lock:Cancel
  - Lock:Deadlock
  - Lock:Deadlock Chain
  - Lock:Escalation
  - Lock:Released
  - Lock:Timeout
- Performance
  - Degree of Parallelism1
  - Degree of Parallelism2
  - Degree of Parallelism3
  - Degree of Parallelism4
  - Execution Plan
  - Show Plan All
  - Show Plan Statistics
  - Show Plan Text
- Scans
  - Scan:Started
  - Scan:Stopped
- Stored Procedures
  - RPC:Completed
  - RPC Output Parameter
  - RPC:Starting
  - SP:CacheHit
  - SP:CacheInsert
  - SP:CacheMiss
  - SP:CacheRemove
  - SP:Completed
  - SP:ExecContextHit
  - SP:Recompile
  - SP:Starting
  - SP:StmtCompleted
  - SP:StmtStarting
- T-SQL
  - Exec Prepared SQL
  - Prepare SQL
  - SQL:BatchCompleted
  - SQL:BatchStarting
  - SQL:StmtCompleted
  - SQL:StmtStarting
  - Unprepare SQL
- Database
  - Data File Auto Grow
  - Data File Auto Shrink
  - Log File Auto Grow
  - Log File Auto Shrink
- Security Audit Events
  - Audit Add DB User Event
  - Audit Add Login to Server Role Event
  - Audit Add Member to DB Role Event
  - Audit Add Role Event
  - Audit AddLogin Event
  - Audit App Role Change Password Event
  - Audit Backup/Restore Event
  - Audit Change Audit Event
  - Audit DBCC Event
  - Audit Login Change Password Event
  - Audit Login Change Property Event
  - Audit Login Event
  - Audit Login Failed Event
  - Audit Login GDR Event
  - Audit Logout Event
  - Audit Object Derived Permission Event
  - Audit Object GDR Event
  - Audit Object Permission Event
  - Audit Server Starts and Stops Event
  - Audit Statement GDR Event
  - Audit Statement Permission Event
- Cursors
  - CursorClose
  - CursorExecute
  - CursorImplicitConversion
  - CursorOpen
  - CursorPrepare
  - CursorRecompile
  - CursorUnprepare
- Objects
  - Autostats
  - Object:Created
  - Object:Deleted
- Sessions
  - ExistingConnection
- Transactions
  - DTCTransaction
  - SQLTransaction
  - TransactionLog
- User Configurable
  - UserConfigurable0
  - UserConfigurable1
  - UserConfigurable2
  - UserConfigurable3
  - UserConfigurable4
  - UserConfigurable5
  - UserConfigurable6
  - UserConfigurable7
  - UserConfigurable8
  - UserConfigurable9

## SQL Profiler Default Event Classes

When a new trace is created, it is defined with a set of default event classes. You can remove these event classes and add others when you create new traces. Unless removed explicitly, the default event classes are present each time a new trace is created.

These are the default event classes for a new trace.

| Default event class | Description |
|---|---|
| **Audit Login Event** | Collects all new connection events (for example, a client requesting a connection to a server running an instance of Microsoft® SQL Server™) since the trace was started. |
| **Audit Logout Event** | Collects all new disconnect events (for example, a client issues a disconnect command) since the trace was started. |
| **ExistingConnection** | Detects activity by all users connected to an instance of SQL Server before the trace was started. |
| **RPC:Completed** | Indicates that a remote procedure call (RPC) has completed. |
| **SQL:BatchCompleted** | Indicates that a transact-SQL batch has completed. |

### See Also

Creating and Managing Traces and Templates

## SQL Profiler Data Columns

SQL Profiler allows you to select data columns when you create a template. These data columns represent the information you would like returned when a trace is running. The data displayed in SQL Profiler can be displayed either in the order the events occur or in a group based on one or a combination of data columns.

For example, to identify the user events that are taking the longest to execute, group events by **DBUserName** and **Duration**. SQL Profiler displays the execution time for each event. This functionality is similar to the Transact-SQL GROUP BY clause. For more information, see GROUP BY.

**Note** You cannot group by the **StartTime** or **EndTime** data columns.

If SQL Profiler can connect to an instance of Microsoft® SQL Server™ on which the trace data was captured, it will try to populate the **Database ID**, **Object ID**, and **Index ID** data columns with the names of the database, object, and index respectively. Otherwise, it will display identification numbers (IDs).

The following table describes the SQL Profiler data columns, and which are selected by default.

| Data column | Column Number | Description |
|---|---|---|
| **Application Name**[1] | 10 | Name of the client application that created the connection to an instance of SQL Server. This column is populated with the values passed by the application rather than the displayed name of the program. |
| **Binary Data** | 2 | Binary value dependent on the event class captured in the trace. |
| **ClientProcessID**[1] | 9 | ID assigned by the host computer to the process where the client application is running. This data column is populated if the client process ID is provided by the client. |
| | | |

| | | |
|---|---|---|
| **Column Permissions** | 44 | Indicates whether a column permission was set. Parse the statement text to determine which permissions were applied to which columns. |
| **CPU** | 18 | Amount of CPU time (in milliseconds) used by the event. |
| **Database ID**[1] | 3 | ID of the database specified by the USE *database* statement or the default database if no USE *database* statement has been issued for a given instance. SQL Profiler displays the name of the database if the **Server Name** data column is captured in the trace and the server is available. Determine the value for a database by using the DB_ID function. |
| **DatabaseName** | 35 | Name of the database in which the user statement is running. |
| **DBUserName**[1] | 40 | SQL Server user name of the client. |
| **Duration** | 13 | Amount of time (in milliseconds) taken by the event. |
| **End Time** | 15 | Time at which the event ended. This column is not populated for event classes that refer to an event starting, such as **SQL:BatchStarting** or **SP:Starting**. |
| **Error** | 31 | Error number of a given event. Often this is the error number stored in **sysmessages**. |
| **EventClass**[1] | 27 | Type of event class captured. |
| **EventSubClass**[1] | 21 | Type of event subclass, providing further information about each event class. For example, event subclass values for the Execution Warning event class represent the type of execution warning:<br><br>1 = Query wait. The query must wait for resources (for example, memory) before it can execute.<br>2 = Query time-out. The query timed out |

|  |  |  |
|---|---|---|
|  |  | while waiting for required resources to execute. This data column is not populated for all event classes. |
| **FileName** | 36 | The logical name of the file being modified. |
| **Handle** | 33 | Integer used by ODBC, OLE DB, or DB-Library to coordinate server execution. |
| **Host Name**[1] | 8 | Name of the computer on which the client is running. This data column is populated if the host name is provided by the client. To determine the host name, use the HOST_NAME function. |
| **Index ID** | 24 | ID for the index on the object affected by the event. To determine the index ID for an object, use the **indid** column of the **sysindexes** system table. |
| **Integer Data** | 25 | Integer value dependent on the event class captured in the trace. |
| **LoginName** | 11 | Name of the login of the user (either SQL Server security login or the Microsoft Windows® login credentials in the form of DOMAIN\Username). |
| **LoginSid**[1] | 41 | Security identification number (SID) of the logged-in user. You can find this information in the **sysxlogins** table of the **master** database. Each SID is unique for each login in the server. |
| **Mode** | 32 | Integer used by various events to describe a state the event has received or is requesting. |
| **NestLevel** | 29 | Integer representing the data returned by @@NESTLEVEL. |
| **NT Domain Name**[1] | 7 | Microsoft Windows NT® 4.0 or Windows 2000 domain to which the user belongs. |
| **NT User Name**[1] | 6 | Windows NT 4.0 or Windows 2000 user name. |
|  |  |  |

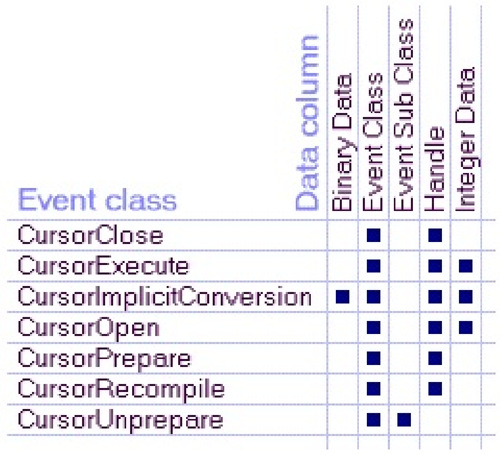
| | | |
|---|---|---|
| **Object ID** | 22 | System-assigned ID of the object. |
| **ObjectName** | 34 | Name of the object being referenced. |
| **ObjectType** | 28 | Value representing the type of the object involved in the event. This value corresponds to the **type** column in **sysobjects**. |
| **Owner Name** | 37 | Database user name of the object owner. |
| **Permissions** | 19 | Integer value representing the type of permissions checked. Values are:<br><br>1 = SELECT ALL<br>2 = UPDATE ALL<br>4 = REFERENCES ALL<br>8 = INSERT<br>16 = DELETE<br>32 = EXECUTE (procedures only)<br>4096 = SELECT ANY (at least one column)<br>8192 = UPDATE ANY<br>16384 = REFERENCES ANY |
| **Reads** | 16 | Number of logical disk reads performed by the server on behalf of the event. |
| **RoleName** | 38 | Name of an application role being enabled. |
| **Server Name**[1] | 26 | Name of the instance of SQL Server being traced. |
| **Severity** | 20 | Severity level of an exception. |
| **SPID**[1] | 12 | Server Process ID assigned by SQL Server to the process associated with the client. |
| **Start Time**[1] | 14 | Time at which the event started, when available. |
| **State** | 30 | Equivalent to an error state code. |
| **Success** | 23 | Represents whether the event was successful. Values include:<br><br>1 = Success.<br>0 = Failure<br><br>For example, a 1 means success of a permissions check and a 0 means a failure of |

| | | |
|---|---|---|
| | | that check. |
| **TargetLoginName** | 42 | For actions which target a login (for example, adding a new login), the name of the targeted login. |
| **TargetLoginSid** | 43 | For actions which target a login (for example, adding a new login), the SID of the targeted login. |
| **TargetUserName** | 39 | For actions which target a database user (for example, granting permission to a user), the name of that user. |
| **TextData** | 1 | Text value dependent on the event class captured in the trace. However, if you are tracing a parameterized query, the variables will not be displayed with data values in the TextData column. |
| **Transaction ID** | 4 | System-assigned ID of the transaction. |
| **Writes** | 17 | Number of physical disk writes performed by the server on behalf of the event. |

1 These data columns are populated by default for all events.

# Cursors Event Category

Use the **Cursors** event category to monitor cursor operations. For example, you can determine when a cursor is executed and what type of cursor is used by monitoring the **CursorOpen**, **CursorExecute**, and **CursorImplicitConversion** event classes. Tracing specific event classes can be useful to determine the actual cursor type used for an operation by an instance of Microsoft® SQL Server™, rather than the cursor type specified by the application.



| Event class / Data column | Binary Data | Event Class | Event Sub Class | Handle | Integer Data |
|---|---|---|---|---|---|
| CursorClose | | ■ | | ■ | |
| CursorExecute | | ■ | | ■ | ■ |
| CursorImplicitConversion | ■ | ■ | | ■ | ■ |
| CursorOpen | | ■ | | ■ | ■ |
| CursorPrepare | | ■ | | ■ | |
| CursorRecompile | | ■ | | ■ | |
| CursorUnprepare | | ■ | ■ | | |

## See Also

Cursors Event Classes

Cursors Data Columns

## Cursors Event Classes

The following table describes the **Cursors** event classes in the **Cursors** event category.

| Event class | Description |
|---|---|
| **CursorClose** | A cursor previously opened on a Transact-SQL statement by ODBC, OLE DB, or DB-Library is closed. |
| **CursorExecute** | A cursor previously prepared on a Transact-SQL statement by ODBC, OLE DB, or DB-Library is executed. For more information, see How to prepare and execute a statement (ODBC). |
| **CursorImplicitConversion** | A cursor on a Transact-SQL statement is converted by Microsoft® SQL Server™ from one type to another.<br><br>Triggered for ANSI and non-ANSI cursors. |
| **CursorOpen** | A cursor is opened on a Transact-SQL statement by ODBC, OLE DB, or DB-Library. |
| **CursorPrepare** | A cursor on a Transact-SQL statement is prepared for use by ODBC, OLE DB, or DB-Library. For more information, see How to prepare and execute a statement (ODBC). |
| **CursorRecompile** | A cursor opened on a Transact-SQL statement by ODBC or DB-Library has been recompiled either directly or indirectly due to a schema change.<br><br>Triggered for ANSI and non-ANSI cursors. |
| **CursorUnprepare** | A prepared cursor on a Transact-SQL statement is deleted by ODBC, OLE DB, or DB-Library. |

## See Also

Cursors Event Category

Cursors Data Columns

## Cursors Data Columns

The following table lists the data columns for each event class in the **Cursors** event category.

| Event class | Data column | Description |
|---|---|---|
| **CursorClose** | **Event Class** | Type of event recorded = 78. |
| | **Handle** | Handle of the cursor. |
| **CursorExecute** | **Event Class**<br><br>**Handle**<br><br>**Integer Data** | Type of event recorded = 74.<br><br>Handle of the cursor.<br><br>Cursor type. Values are:<br><br>1 = Keyset<br>2 = Dynamic<br>4 = Forward only<br>8 = Static<br>16 = Fast forward |
| **CursorImplicitConversion** | **Event Class** | Type of event recorded = 76. |
| | **Handle** | Handle of the cursor. |
| | **Integer Data** | Requested cursor type. Values are:<br><br>1 = Keyset<br>2 = Dynamic<br>4 = Forward only<br>8 = Static<br>16 = Fast forward |
| | **Binary Data** | Resulting cursor type. Values are:<br><br>1 = Keyset<br>2 = Dynamic<br>4 = Forward only |

| | | |
|---|---|---|
| | | 8 = Static<br>16 = Fast forward |
| **CursorOpen** | **Event Class**<br><br>**Handle**<br><br>**Integer Data** | Type of event recorded = 53.<br><br>Handle of the cursor.<br><br>Cursor type. Values are:<br><br>1 = Keyset<br>2 = Dynamic<br>4 = Forward only<br>8 = Static<br>16 = Fast forward |
| **CursorPrepare** | **Event Class** | Type of event recorded = 70. |
| | **Handle** | Handle of the prepared cursor. |
| **CursorRecompile** | **Event Class**<br><br>**Handle** | Type of event recorded = 75.<br><br>Handle of the cursor that had to be recompiled. |
| **CursorUnprepare** | **Event Class**<br><br>**Event Sub Class** | Type of event recorded = 77.<br><br>Handle of the cursor created by **CursorPrepare**. |

## See Also

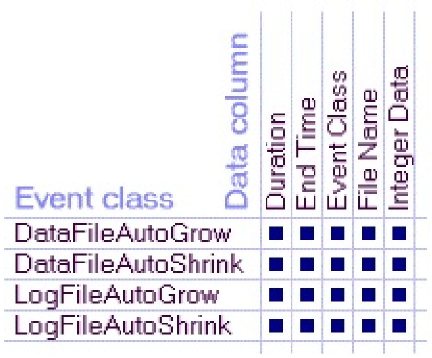
Cursors

Cursors Event Classes

Cursors Event Category

# Database Event Category

Use the **Database** event category to monitor when data or log files grow or shrink automatically.



| Event class / Data column | Duration | End Time | Event Class | File Name | Integer Data |
|---|---|---|---|---|---|
| DataFileAutoGrow | ■ | ■ | ■ | ■ | ■ |
| DataFileAutoShrink | ■ | ■ | ■ | ■ | ■ |
| LogFileAutoGrow | ■ | ■ | ■ | ■ | ■ |
| LogFileAutoShrink | ■ | ■ | ■ | ■ | ■ |

## See Also

Database Event Classes

Database Data Columns

## Database Event Classes

The following table describes the **Database** event classes in the **Database** event category.

| Event class | Description |
|---|---|
| **DataFileAutoGrow** | Indicates that the data file grew automatically. This event is not triggered if the data file is grown explicitly through ALTER DATABASE. |
| **DataFileAutoShrink** | Indicates that the data file has been shrunk. |
| **LogFileAutoGrow** | Indicates that the log file grew automatically. This event is not triggered if the log file is grown explicitly through ALTER DATABASE. |
| **LogFileAutoShrink** | Indicates that the log file has been shrunk. |

### See Also

Database Event Category

Database Data Columns

## Database Data Columns

The following lists the data columns for each event class in the **Database** event category.

| Event class | Data column | Description |
|---|---|---|
| **Data File Auto Grow** | **Event Class** | Type of event recorded = 92. |
| | **End Time** | The time the data file auto grow ended. |
| | **Duration** | The length of time (in milliseconds) necessary to extend the file. |
| | **File Name** | The logical name of the file being extended. |
| | **Integer Data** | The number of 8-kilobyte (KB) pages by which the file increased. |
| **Data File Auto Shrink** | **Event Class** | Type of event recorded = 94. |
| | **End Time** | The time the auto shrink ended. |
| | **Duration** | The time (in milliseconds) to shrink the file. |
| | **File Name** | The logical name of the file being shrunk. |
| | **Integer Data** | The number of 8 KB pages by which the file was reduced. |
| **Log File Auto Grow** | **Event Class** | Type of event recorded = 93. |
| | **End Time** | The time the log file auto grow ended. |
| | **Duration** | The time (in milliseconds) needed to extend the file. |
| | **File Name** | The logical name of the file being extended. |
| | **Integer Data** | The number of 8 KB pages by which the file increased. |

| | | |
|---|---|---|
| **Log File Auto Shrink** | **Event Class** | Type of event recorded = 95. |
| | **End Time** | The time the log file auto shrink ended. |
| | **Duration** | The time (in milliseconds) needed to shrink the file. |
| | **File Name** | The logical name of the file being shrunk. |
| | **Integer Data** | The number of 8 KB pages by which the file was reduced. |

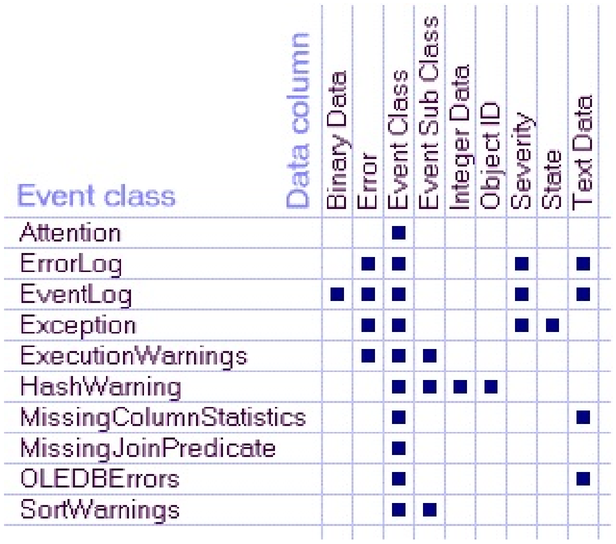
## See Also

Database Event Category

Database Event Classes

## Errors and Warnings Event Category

Use the **Errors and Warnings** event category to monitor many of the errors and warnings raised by Microsoft® SQL Server™ and components such as OLE DB. Typically, you use the following event classes to look for problems that may be encountered while running applications or executing procedures.



| Event class / Data column | Binary Data | Error | Event Class | Event Sub Class | Integer Data | Object ID | Severity | State | Text Data |
|---|---|---|---|---|---|---|---|---|---|
| Attention | | | ■ | | | | | | |
| ErrorLog | | ■ | ■ | | | | ■ | | ■ |
| EventLog | ■ | ■ | ■ | | | | ■ | | ■ |
| Exception | | ■ | ■ | | | | ■ | ■ | |
| ExecutionWarnings | | ■ | ■ | ■ | | | | | |
| HashWarning | | | ■ | ■ | ■ | ■ | | | |
| MissingColumnStatistics | | | ■ | | | | | | ■ |
| MissingJoinPredicate | | | ■ | | | | | | |
| OLEDBErrors | | | ■ | | | | | | ■ |
| SortWarnings | | | ■ | ■ | | | | | |

### See Also

Errors and Warnings Event Classes

Errors and Warnings Data Columns

## Errors and Warnings Event Classes

The following table describes the **Errors and Warnings** event classes in the **Errors and Warnings** event category.

| Event class | Description |
|---|---|
| **Attention** | Collects all attention events, such as client-interrupt requests or when a client connection is broken. |
| **ErrorLog** | Error events have been logged in the Microsoft® SQL Server™ error log. |
| **EventLog** | Events have been logged in the Microsoft Windows® application log. |
| **Exception** | Exception has occurred in SQL Server. |
| **Execution Warnings** | Any warnings that occurred during the execution of a SQL Server statement or stored procedure. |
| **Hash Warning** | Hashing operation may have encountered a problem. |
| **Missing Column Statistics** | Column statistics for the query optimizer are not available. |
| **Missing Join Predicate** | Executing query has no join predicate. This can result in a long-running query. |
| **OLEDB Errors** | OLE DB error has occurred. |
| **Sort Warnings** | Sort operations do not fit into memory. This does not include sort operations from the creation of indexes, only sort operations within a query (for example, an ORDER BY clause used in a SELECT statement). |

The **Execution Warnings** event class can be monitored to determine how long, if at all, queries had to wait for resources before proceeding. This is important for determining whether there are any contention issues in the system that can affect performance and therefore need investigating. Use the **Locks** event classes to determine the objects affected.

The **Hash Warning** event class can be used to monitor when a hash recursion or hash bail has occurred during a hashing operation. Hash recursion occurs when the build input does not fit into memory, resulting in input split into multiple partitions, which are processed separately. If any of these partitions still do not fit into memory, they are split further into sub-partitions, which then are processed separately. This process continues until each partition fits into memory or the maximum recursion level is reached (displayed in the **Integer Data** data column), thus causing hash bail.

Hash bail occurs when a hashing operation reaches its maximum recursion depth and reverts to an alternate plan to process its remaining partitioned data. Hash bail is due usually to skewed data, trace flags, or bit counting. To eliminate or reduce the chance of hash bail, verify that statistics exist on the columns being joined or grouped. For more information, see Statistical Information.

If hash bail continues to occur each time the query is executed, consider using an optimizer hint to force a different algorithm to be used by the query optimizer and then compare the performance of the query. For more information about join hints, see FROM.

By monitoring the **Missing Column Statistics** event class, you can determine whether there are statistics missing for a column used by a query. Missing statistics can cause the optimizer to choose a less-efficient query plan. For more information about creating column statistics, see Statistical Information.

The **Sort Warnings** event class can be used to monitor query performance. If a query involving a sort operation generates a **Sort Warnings** event class with an **Event Sub Class** data column value of 2, the performance of the query can be affected because multiple passes over the data are required to sort the data. Investigate the query further to determine whether the sort operation can be eliminated.

## See Also

Errors and Warnings Event Category

Errors and Warnings Data Columns

## Errors and Warnings Data Columns

These are the data columns for each event class in the **Errors and Warnings** event category.

| Event class | Data column | Description |
|---|---|---|
| **Attention** | **Event Class** | Type of event recorded = 16. |
| **ErrorLog** | **Event Class** | Type of event recorded = 22. |
| | **Error** | Error number. |
| | **Severity** | Severity of the error generated. |
| | **Text Data** | Text of the error message. |
| **EventLog** | **Event Class** | Type of event recorded = 21. |
| | **Binary Data** | Binary value dependent on the event class captured in the trace. |
| | **Error** | Error number. |
| | **Severity** | Error severity. |
| | **Text Data** | Text of the error message, if available. |
| **Exception** | **Event Class** | Type of event recorded = 33. |
| | **Error** | Error number. |
| | **State** | Server state. |
| | **Severity** | Error severity. |
| **Execution Warnings** | **Event Class** | Type of event recorded = 67. |
| | **Event Sub Class** | The type of execution warning. Can have these values:<br><br>1 = Query wait. The query must wait for resources (for example, memory) before it can execute.<br>2 = Query time-out. The query timed out while waiting for required resources to execute. |

|  |  |  |
|---|---|---|
|  | **Error** | Error number. |
| **Hash Warning** | **Event Class** | Type of event recorded = 55. |
|  | **Event Sub Class** | Type of hash operation. Can have these values:<br><br>0 = Hash recursion.<br>1 = Hash bail. |
|  | **Integer Data** | Recursion level (hash recursion only). |
|  | **Object ID** | Node ID of the root of the hash involved in the repartition. |
| **Missing Column Statistics** | **Event Class** | Type of event recorded = 79. |
|  | **Text Data** | List of the columns with missing statistics. |
| **Missing Join Predicate** | **Event Class** | Type of event recorded = 80. |
| **OLEDB Errors** | **Event Class** | Type of event recorded = 61. |
|  | **Text Data** | Error message from OLE DB. |
| **Sort Warnings** | **Event Class** | Type of event recorded = 69. |
|  | **Event Sub Class** | Type of sort warning. Can have these values:<br><br>1 = Single pass. When the sort table was written to disk, only a single additional pass over the data to be sorted was required to obtain sorted output.<br>2 = Multiple pass. When the sort table was written to disk, multiple passes over the data were required to obtain sorted output. |

## See Also

Error Messages

Monitoring the Error Logs

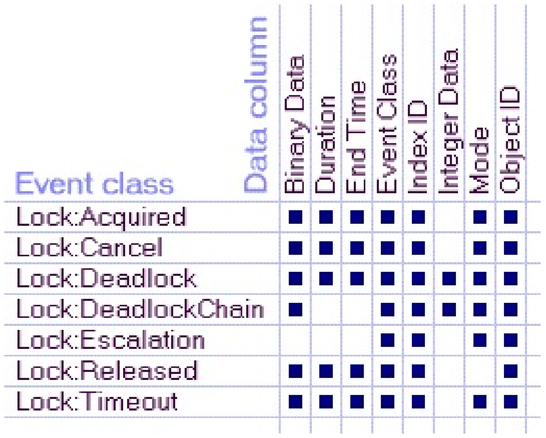
Errors and Warnings Event Classes

Errors and Warnings Event Category

# Locks Event Category

Use the **Locks** event category to monitor Microsoft® SQL Server™ lock activity. By monitoring the Locks event classes, you can investigate contention issues caused by users and applications using a database concurrently.

Because lock events are so prolific, capturing the lock event classes can incur significant overhead on the server being traced and result in very large trace files or trace tables.



| Event class / Data column | Binary Data | Duration | End Time | Event Class | Index ID | Integer Data | Mode | Object ID |
|---|---|---|---|---|---|---|---|---|
| Lock:Acquired | ■ | ■ | ■ | ■ | ■ |  | ■ | ■ |
| Lock:Cancel | ■ | ■ | ■ | ■ | ■ |  | ■ | ■ |
| Lock:Deadlock | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ |
| Lock:DeadlockChain | ■ |  |  | ■ | ■ | ■ | ■ | ■ |
| Lock:Escalation |  |  |  | ■ | ■ |  | ■ | ■ |
| Lock:Released | ■ | ■ | ■ | ■ | ■ |  |  | ■ |
| Lock:Timeout | ■ | ■ | ■ | ■ | ■ |  | ■ | ■ |

## See Also

Locks Event Classes

Locks Data Columns

## Locks Event Classes

The following table describes the **Locks** event classes in the **Locks** event category.

| Event class | Description |
|---|---|
| **Lock:Acquired** | Acquisition of a lock on a resource, such as a data page, has been achieved. For more information about resources that can be locked, see Understanding Locking in SQL Server. |
| **Lock:Cancel** | Acquisition of a lock on a resource has been canceled (for example, due to a deadlock). |
| **Lock:Deadlock** | Two concurrent transactions have deadlocked each other by trying to obtain incompatible locks on resources that the other transaction owns. For more information, see Deadlocking. |
| **Lock:Deadlock Chain** | This event is produced for each of the events leading up to the deadlock. |
| **Lock:Escalation** | A finer-grained lock has been converted to a coarser-grained lock (for example, a row lock that is converted to a page lock). |
| **Lock:Released** | A lock on a resource, such as a page, has been released. |
| **Lock:Timeout** | A request for a lock on a resource, such as a page, has timed out due to another transaction holding a blocking lock on the required resource. Time-out is determined by the @@LOCK_TIMEOUT system function and can be set with the SET LOCK_TIMEOUT statement. For more information, see Customizing the Lock Time-out. |

The **Lock:Acquired** and **Lock:Released** event classes can be used to monitor when objects are being locked, the type of locks taken, and for how long the locks were retained. Locks retained for long periods of time may cause contention and should be investigated. For example, an application can be acquiring locks on rows in a table, and then waiting for user input. Because user input can take a long time, the locks can block other users. In this instance, the application should be redesigned to make lock requests only when necessary and not require user input when locks have been acquired.

The **Lock:Deadlock**, **Lock:Deadlock Chain**, and **Lock:Timeout** event classes can be used to monitor when deadlocks and time-out conditions occur, and which objects are involved. This information is useful to determine whether deadlocks and time-outs are affecting the performance of your application significantly, and which objects are involved. The application code that modifies these objects can then be examined to determine whether changes to minimize deadlocks and time-outs can be made. For more information about reducing deadlocks, see Avoiding Deadlocks.

## See Also

Locks Event Category

Locks Data Columns

## Locks Data Columns

The following table lists the data columns for each event class in the **Locks** event category.

| Event class | Data column | Description |
|---|---|---|
| **Lock:Acquired** | **Event Class** | Type of event recorded = 24. |
| | **Mode** | Lock mode, such as intent exclusive, of the lock that was acquired. |
| | **Binary Data** | Resource type. |
| | **End Time** | End time of the event. |
| | **Duration** | Wait between the time the lock request was issued and the time the lock was acquired. |
| | **Object ID** | ID of the object on which the lock was acquired. |
| | **Index ID** | ID of the index, if the object lock was on an index. |
| **Lock:Cancel** | **Event Class** | Type of event recorded = 26. |
| | **Mode** | Mode of the lock that was canceled. |
| | **Binary Data** | Resource type. |
| | **End Time** | End time of the event. |
| | **Duration** | Wait between the time the lock requested was issued and the time the lock was canceled. |
| | **Object ID** | ID of the object on which the lock was canceled. |
| | **Index ID** | ID of the index, if the object lock was on an index. |
| **Lock:Deadlock** | **Event Class** | Type of event recorded = 25. |
| | **Mode** | Lock mode of the lock that triggered the deadlock. |
| | **Binary Data** | Resource type. |

| | | |
|---|---|---|
| | **End Time** | End time of the deadlock. |
| | **Integer Data** | Deadlock number. Numbers are assigned, beginning with zero, when the server is started and incremented for each deadlock. |
| | **Duration** | Wait between the time the lock request was issued and the time the deadlock occurred. |
| | **Object ID** | ID of the object in contention. |
| | **Index ID** | ID of the index, if the object lock was on an index. |
| **Lock:Deadlock Chain** | **Event Class** | Type of event recorded = 59. |
| | **Mode** | Lock mode of each lock in the deadlock chain. |
| | **Binary Data** | Resource type. |
| | **Integer Data** | Deadlock number. Numbers are assigned, beginning with zero, when the server is started and incremented for each deadlock. |
| | **Object ID** | ID of the object that was locked. |
| | **Index ID** | ID of the index, if the object lock was on an index. |
| **Lock:Escalation** | **Event Class** | Type of event recorded = 60. |
| | **Object ID** | ID of the object on which the lock was escalated. |
| | **Index ID** | ID of the index, if the object lock was on an index. |
| | **Mode** | Lock mode after the escalation. |
| **Lock:Released** | **Event Class** | Type of event recorded = 23. |
| | **Binary Data** | Resource type. |
| | **End Time** | End time of the event. |
| | **Duration** | Wait time between the time the lock request was issued and the time the lock was released. |

| | | |
|---|---|---|
| | **Object ID** | ID of the object on which the lock was released. |
| | **Index ID** | ID of the index, if the object lock was on an index. |
| **Lock:Timeout** | **Event Class** | Type of event recorded = 27. |
| | **Mode** | Lock mode of the requested lock that has timed out. |
| | **Binary Data** | Resource type. |
| | **End Time** | End time of the event. |
| | **Duration** | Wait time between the time the lock request was issued and the time the lock was released. |
| | **Object ID** | ID of the object on which the lock was timed out. |
| | **Index ID** | ID of the index, if the object lock was on an index. |

## See Also

Locking

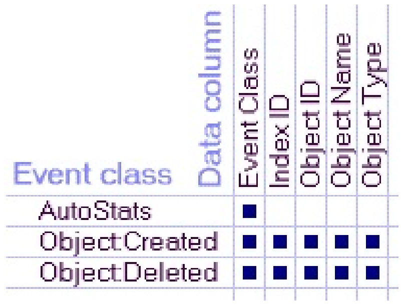
Locks Event Category

Locks Event Classes

## Objects Event Category

Use the **Objects** event classes to monitor when an object (for example, a database, table, index, view, or stored procedure) is opened, created, deleted, or used.

Because object events are so prolific, capturing the object event classes can incur significant overhead on the server being traced and result in large trace files or trace tables.



| Event class / Data column | Event Class | Index ID | Object ID | Object Name | Object Type |
|---|---|---|---|---|---|
| AutoStats | ■ | | | | |
| Object:Created | ■ | ■ | ■ | ■ | ■ |
| Object:Deleted | ■ | ■ | ■ | ■ | ■ |

### See Also

Objects Event Classes

Objects Data Columns

# Objects Event Classes

The following table describes the **Objects** event classes in the **Objects** event category.

| Event class | Description |
|---|---|
| **Auto Stats** | Indicates when the automatic creation and updating of statistics has occurred. |
| **Object:Closed** | Indicates when an open object has been closed (for example, such as at the end of a SELECT, INSERT, or DELETE statement). |
| **Object:Created** | Object has been created (for example, for CREATE INDEX, CREATE TABLE, and CREATE DATABASE statements). |
| **Object:Deleted** | Object has been deleted (for example, for DROP INDEX and DROP TABLE statements). |
| **Object:Opened** | Indicates when an object has been accessed (for example, for SELECT, INSERT, or DELETE statements). |

The **Object:Created** and **Object:Deleted** event classes can be used to determine whether many ad hoc objects are being created or deleted (for example, by ODBC applications that often create temporary stored procedures). By monitoring the **DBUserName** and **NT User Name** default data columns in addition to the Objects event classes, you can determine the name of the user who is creating, deleting, or accessing objects. This can be used to determine whether your security policies are correctly implemented, for example, to confirm that users who are not allowed to create or delete objects are not doing so.

## See Also

Objects Event Category

Objects Data Columns

## Objects Data Columns

The following table lists the data columns for each event class in the **Objects** event category.

| Event class | Data column | Description |
|---|---|---|
| **Auto Stats** | **Event Class** | Type of event recorded = 58. |
| **Object:Created** | **Event Class** | Type of event recorded = 46. |
| | **Object Type** | Type of object created. |
| | **Object Name** | Name of the object that was created. |
| | **Object ID** | ID of the object that was created. |
| | **Index ID** | Index ID, if an index was created. |
| **Object:Deleted** | **Event Class** | Type of event recorded = 47. |
| | **Object Type** | Type of the object that was deleted. |
| | **Object Name** | Name of the object that was deleted. |
| | **Object ID** | ID of the object that was deleted. |
| | **Index ID** | Index ID, if an index was deleted. |

### See Also

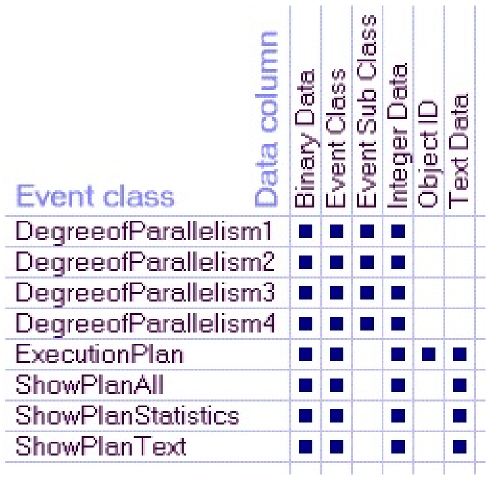
Creating and Maintaining Databases Overview

Objects Event Category

Objects Event Classes

## Performance Event Category

Use the **Performance** event category to monitor showplan event classes and event classes that are produced from the execution of SQL data manipulation language (DML) operators.



| Event class | Binary Data | Event Class | Event Sub Class | Integer Data | Object ID | Text Data |
|---|---|---|---|---|---|---|
| DegreeofParallelism1 | ■ | ■ | ■ | ■ | | |
| DegreeofParallelism2 | ■ | ■ | ■ | ■ | | |
| DegreeofParallelism3 | ■ | ■ | ■ | ■ | | |
| DegreeofParallelism4 | ■ | ■ | ■ | ■ | | |
| ExecutionPlan | ■ | ■ | | ■ | ■ | ■ |
| ShowPlanAll | ■ | ■ | | ■ | | ■ |
| ShowPlanStatistics | ■ | ■ | | ■ | | ■ |
| ShowPlanText | ■ | ■ | | ■ | | ■ |

### See Also

Performance Event Classes

Performance Data Columns

## Performance Event Classes

The following table describes the **Performance** event classes in the **Performance** event category.

| Event class | Description |
|---|---|
| **Degree Of Parallelism1**[1] | Describes the degree of parallelism assigned to the SQL statement. Occurs before a SELECT, INSERT, UPDATE, or DELETE statement is executed. If you are tracing a Microsoft® SQL Server™ version 7.0 server, this event will trace an INSERT statement. |
| **Degree of Parallelism2**[1] | Describes the degree of parallelism assigned to the SQL statement. Occurs before a SELECT, INSERT, UPDATE, or DELETE statement is executed. If you are tracing a SQL Server 7.0 server, this event will trace an UPDATE statement. |
| **Degree of Parallelism3**[1] | Describes the degree of parallelism assigned to the SQL statement. Occurs before a SELECT, INSERT, UPDATE, or DELETE statement is executed. If you are tracing a SQL Server 7.0 server, this event will trace a DELETE statement. |
| **Degree of Parallelism4**[1] | Describes the degree of parallelism assigned to the SQL statement. Occurs before a SELECT, INSERT, UPDATE, or DELETE statement is executed. If you are tracing a SQL Server 7.0 server, this event will trace a SELECT statement. |
| **Execution Plan** | Displays the plan tree of the SQL statement being executed. |
| **Show Plan All** | Displays the query plan with full compile-time details (for example, costing estimates and column lists) of the SQL statement being executed. |
| **Show Plan Statistics** | Displays the query plan with full run-time details, including actual number of rows passing through each operation, of the SQL statement which was executed. |

| | |
|---|---|
| **Show Plan Text** | Displays the query plan tree of the SQL statement being executed. |

[1] If you are tracing a SQL Server 2000 server, this event will trace SELECT, INSERT, UPDATE, or DELETE statements.

## See Also

Performance Event Category

Performance Data Columns

## Performance Data Columns

The following table lists the data columns for each event class in the **Performance** event category.

| Event class | Data column | Description |
|---|---|---|
| **Degree of Parallelism1** | **Event Class** | Type of event recorded = 28. |
| | **Event Sub Class** | If you are tracing a Microsoft® SQL Server™ 2000 server, the Event Sub Class can have these values, which reflect the type of statement:<br><br>1 = Select<br><br>2 = Insert<br><br>3 = Update<br><br>4 = Delete |
| | **Binary Data** | Supplied binary data, which is the number of CPUs used to perform the statement. |
| | **Integer Data** | Pages used in memory for the query plan. |
| **Degree of Parallelism2** | **Event Class** | Type of event recorded =28. |
| | **Event Sub Class** | If you are tracing a SQL Server 2000 server, the Event Sub Class can have these values, which reflect the type of statement:<br><br>1 = Select<br><br>2 = Insert |

| | | |
|---|---|---|
| | | 3 = Update<br><br>4 = Delete |
| | **Binary Data** | Supplied binary data, which is the number of CPUs used to perform the statement. |
| | **Integer Data** | Pages used in memory for the query plan. |
| **Degree of Parallelism3** | **Event Class** | Type of event recorded = 28. |
| | **Event Sub Class** | If you are tracing a SQL Server 2000 server, the Event Sub Class can have these values, which reflect the type of statement:<br><br>1 = Select<br><br>2 = Insert<br><br>3 = Update<br><br>4 = Delete |
| | **Binary Data** | Supplied binary data, which is the number of CPUs used to perform the statement. |
| | **Integer Data** | Pages used in memory for the query plan. |
| **Degree of Parallelism4** | **Event Class** | Type of event recorded = 28. |
| | **Event Sub Class** | If you are tracing a SQL Server 2000 server, the Event Sub Class can have these values, which reflect the type of statement:<br><br>1 = Select<br><br>2 = Insert<br><br>3 = Update |

| | | |
|---|---|---|
| | | 4 = Delete |
| | **Binary Data** | Supplied binary data, which is the number of CPUs used to perform the statement. |
| | **Integer Data** | Pages used in memory for the query plan. |
| **Execution Plan** | **Event Class** | Type of event recorded = 68. |
| | **Binary Data** | Estimated cost of the execution plan. |
| | **Object ID** | ID of the object that had its statistics updated. |
| | **Integer Data** | Estimated number of rows returned. |
| | **Text Data** | Execution plan tree. Only SQL statements are expressed. Transact-SQL constructs are not represented. |
| **Show Plan All** | **Event Class** | Type of event recorded = 97. |
| | **Binary Data** | Estimated cost of the query. |
| | **Integer Data** | Expected number of rows to be returned. |
| | **Text Data** | Showplan ALL results of the statement. |
| **Show Plan Statistics** | **Event Class** | Type of event recorded = 98. |
| | **Binary Data** | Estimated cost of the query. |
| | **Integer Data** | Expected number of rows to be returned. |
| | **Text Data** | Showplan Statistics results of the statement. |
| **Show Plan Text** | **Event Class** | Type of event recorded = 96. |
| | **Binary Data** | Estimated cost of the query. |
| | **Integer Data** | Expected number of rows to be returned. |

| | | |
|---|---|---|
| | **Text Data** | Showplan Text results of the statement. |

## See Also

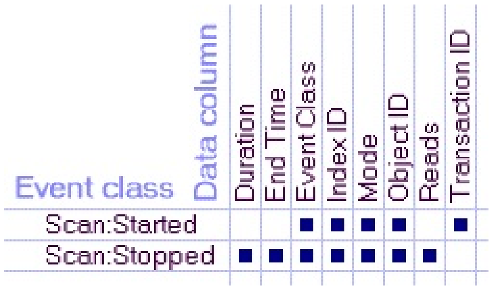
Performance Event Category

Performance Event Classes

## Scans Event Category

Use the **Scans** event category to monitor when a table or index is being scanned during the execution of a query.

Using the **Scan:Started** and **Scan:Stopped** event classes, it is possible to monitor the type of scans being performed by a query on a specific object.



| Event class \ Data column | Duration | End Time | Event Class | Index ID | Mode | Object ID | Reads | Transaction ID |
|---|---|---|---|---|---|---|---|---|
| Scan:Started | | | ■ | ■ | ■ | ■ | | ■ |
| Scan:Stopped | ■ | ■ | ■ | ■ | ■ | ■ | ■ | |

### See Also

Scans Event Classes

Scans Data Columns

## Scans Event Classes

The following table describes the **Scans** event classes in the **Scans** event category.

| Event class | Description |
|---|---|
| **Scan: Started** | Table or index scan has started. |
| **Scan: Stopped** | Table or index scan has stopped. |

### See Also

Scans Event Category

Scans Data Columns

## Scans Data Columns

The following table lists the data columns for each event class in the **Scans** event category.

| Event class | Data column | Description |
|---|---|---|
| **Scan: Started** | **Event Class** | Type of event recorded = 51. |
| | **Mode** | Scan mode. Can have these values:<br><br>1 = Normal<br>2 = First<br>4 = Back<br>8 = Unordered<br>16 = No data<br>32 = Reserved<br>64 = Exlatch<br>128 = Index supplied<br>256 = Marker |
| | **Object ID** | ID of the object that is being scanned. |
| | **Index ID** | ID of the index, if an index is being scanned. |
| | **Transaction ID** | ID of the transaction of which the scan is a part. |
| **Scan: Stopped** | **Event Class** | Type of event recorded = 52. |
| | **Mode** | Mode that was used to perform the scan. Can have these values:<br><br>1 = Normal<br>2 = First<br>4 = Back<br>8 = Unordered<br>16 = No data |

| | | |
|---|---|---|
| | | 32 = Reserved<br>64 = Exlatch<br>128 = Index supplied<br>256 = Marker |
| | **End Time** | End time of the event. |
| | **Duration** | Duration of the scan. |
| | **Reads** | Number of logical pages read. |
| | **Index ID** | ID of the index, if an index is being scanned. |
| | **Object ID** | ID of the object that is being scanned. |

By monitoring the **Index ID** default data column, you can determine the identification number of the index being used by a specific query. The **Index ID** data column contains either:

- The value 1 when the clustered index of the table is being scanned.

  -or-

- The value greater than 2 and less than 255 when a non-clustered index of the table is being scanned.

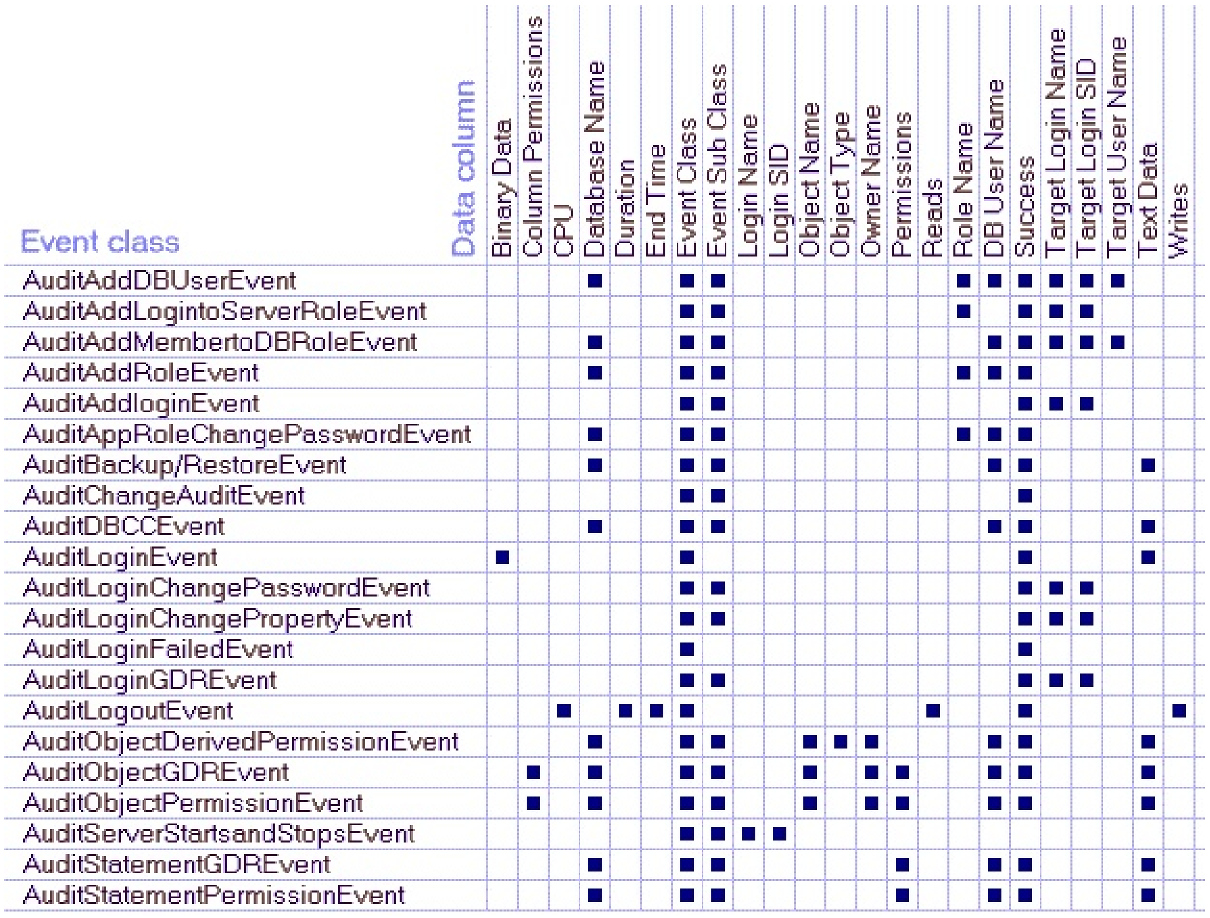
## See Also

Scans Event Category

Scans Event Classes

## Security Audit Event Category

Use the **Security Audit** event category to monitor auditing activity.



| Event class \ Data column | Binary Data | Column Permissions | CPU | Database Name | Duration | End Time | Event Class | Event Sub Class | Login Name | Login SID | Object Name | Object Type | Owner Name | Permissions | Reads | Role Name | DB User Name | Success | Target Login Name | Target Login SID | Target User Name | Text Data | Writes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AuditAddDBUserEvent | | | | ■ | | | ■ | ■ | | | | | | | | ■ | ■ | ■ | ■ | ■ | ■ | | |
| AuditAddLogintoServerRoleEvent | | | | | | | ■ | ■ | | | | | | | | ■ | | ■ | ■ | ■ | | | |
| AuditAddMembertoDBRoleEvent | | | | ■ | | | ■ | ■ | | | | | | | | | ■ | ■ | ■ | ■ | ■ | | |
| AuditAddRoleEvent | | | | ■ | | | ■ | ■ | | | | | | | | ■ | ■ | ■ | | | | | |
| AuditAddloginEvent | | | | | | | ■ | ■ | | | | | | | | | | ■ | ■ | ■ | | | |
| AuditAppRoleChangePasswordEvent | | | | ■ | | | ■ | ■ | | | | | | | | ■ | ■ | ■ | | | | | |
| AuditBackup/RestoreEvent | | | | ■ | | | ■ | ■ | | | | | | | | | ■ | ■ | | | | ■ | |
| AuditChangeAuditEvent | | | | | | | ■ | ■ | | | | | | | | | | ■ | | | | | |
| AuditDBCCEvent | | | | ■ | | | ■ | ■ | | | | | | | | | ■ | ■ | | | | ■ | |
| AuditLoginEvent | ■ | | | | | | ■ | | | | | | | | | | | ■ | | | | ■ | |
| AuditLoginChangePasswordEvent | | | | | | | ■ | ■ | | | | | | | | | | ■ | ■ | ■ | | | |
| AuditLoginChangePropertyEvent | | | | | | | ■ | ■ | | | | | | | | | | ■ | ■ | ■ | | | |
| AuditLoginFailedEvent | | | | | | | ■ | | | | | | | | | | | ■ | | | | | |
| AuditLoginGDREvent | | | | | | | ■ | ■ | | | | | | | | | | ■ | ■ | ■ | | | |
| AuditLogoutEvent | | | ■ | | ■ | ■ | ■ | | | | | | | | ■ | | | ■ | | | | | ■ |
| AuditObjectDerivedPermissionEvent | | | | ■ | | | ■ | ■ | | | ■ | ■ | ■ | | | | ■ | ■ | | | | ■ | |
| AuditObjectGDREvent | | ■ | | ■ | | | ■ | ■ | | | ■ | | ■ | ■ | | | ■ | ■ | | | | ■ | |
| AuditObjectPermissionEvent | | ■ | | ■ | | | ■ | ■ | | | ■ | | ■ | ■ | | | ■ | ■ | | | | ■ | |
| AuditServerStartsandStopsEvent | | | | | | | ■ | ■ | ■ | ■ | | | | | | | | | | | | | |
| AuditStatementGDREvent | | | | ■ | | | ■ | ■ | | | | | | ■ | | | ■ | ■ | | | | ■ | |
| AuditStatementPermissionEvent | | | | ■ | | | ■ | ■ | | | | | | ■ | | | ■ | ■ | | | | ■ | |

### See Also

Security Audit Event Classes

Security Audit Data Columns

## Security Audit Event Classes

The following table describes the **Security Audit** event classes in the **Security Audit** event category.

| Event class | Description |
|---|---|
| **Audit Add DB User Event** | Records the addition and removal of database users (Microsoft Windows NT® 4.0, Microsoft Windows® 2000, or Microsoft SQL Server™). |
| **Audit Add Login to Server Role Event** | Records the addition or removal of logins to and from a fixed server role for **sp_addsrvrolemember** and **sp_dropsrvrolemember**. |
| **Audit Add Member to DB Role Event** | Records the addition and removal of members to and from a database role (fixed or user-defined) for **sp_addrolemember**, **sp_droprolemember**, and **sp_changegroup**. |
| **Audit Add Role Event** | Records add or drop actions on database roles for **sp_addrole** and **sp_droprole**. |
| **Audit Addlogin Event** | Records add and drop actions on SQL Server logins for **sp_addlogin** and **sp_droplogin**. |
| **Audit App Role Change Password Event** | Records changes to the password of an application. |
| **Audit Backup/Restore Event** | Records BACKUP and RESTORE events. |
| **Audit Change Audit Event** | Records AUDIT modifications. |
| **Audit DBCC Event** | Records DBCC commands that have been issued. |
| **Audit Login Event** | Collects all new connection events since the trace was started (for example, a client requesting a connection to a server running an instance of SQL Server). |
| | |

| | |
|---|---|
| **Audit Login Change Password Event** | Records SQL Server login password changes. Passwords are not recorded.<br><br>If you are a member of the **sysadmin** or **securityadmin** fixed server role and you reset your own password by using **sp_password** with all three arguments specified ('*old_password*', '*new_password*', '*login*'), the audit record will reflect that you are changing someone else's password. |
| **Audit Login Change Property Event** | Records modifications on login property, except passwords for **sp_defaultdb** and **sp_defaultlanguage**. |
| **Audit Login Failed Event** | Indicates that a login attempt to an instance of SQL Server from a client has failed. |
| **Audit Login GDR Event** | Records grant, revoke, and deny actions on Windows NT 4.0 or Windows 2000 account login rights for **sp_grantlogin**, **sp_revokelogin**, and **sp_denylogin**. |
| **Audit Logout Event** | Collects all new disconnect events since the trace was started, such as when a client issues a disconnect command. |
| **Audit Object Derived Permission Event** | Records when a CREATE, ALTER, or DROP command is issued for the specified object. |
| **Audit Object GDR Event** | Records permissions events for GRANT, DENY, REVOKE objects. |
| **Audit Object Permission Event** | Records the successful or unsuccessful use of object permissions. |
| **Audit Server Starts and Stops Event** | Records shut down, start, and pause activities for services. |
| **Audit Statement GDR Event** | Records permission events for GRANT, DENY, REVOKE statements. |
| **Audit Statement Permission Event** | Records the use of statement permissions. |

## See Also

Security Audit Event Category

Security Audit Data Columns

## Security Audit Data Columns

The following table lists the data columns for each event class in the **Security Audit** event category.

| Event class | Data column | Description |
|---|---|---|
| **Audit Add DB User Event** | **Event Class** | Type of event recorded = 109. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| | **Event Sub Class** | Class of event within the event. Values are:<br><br>1 = sp_adduser<br>2 = sp_dropuser<br>3 = grantdbaccess<br>4 = revokedbaccess |
| | **Database Name** | Name of the database to which the user is being added. |
| | **DBUserName** | The issuer's user name in the database. |
| | **Target Login SID** | SID of the targeted Microsoft® Windows® login. |
| | **Target Login Name** | Name of the targeted Windows login. |
| | **Target User Name** | Name of the database user being added to the database. |
| | **Role Name** | Name of a role to which the new database user is being added. |
| **Audit Add Login to** | **Event Class** | Type of event recorded = 108. |

|  |  |  |
|---|---|---|
| **Server Role Event** |  |  |
|  | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
|  | **Event Sub Class** | Class of event within the event. Values are:<br><br>1 = Add<br>2 = Drop |
|  | **Target Login SID** | Security identification number (SID) of the targeted Windows login. |
|  | **Target Login Name** | Name of the targeted Windows login. |
|  | **Role Name** | Name of the role to which the login is being added. |
| **Audit Add Member to DB Role Event** | **Event Class** | Type of event recorded = 110. |
|  | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
|  | **Event Sub Class** | Class of event within the event. Values are:<br><br>1 = Add<br>2 = Drop |
|  | **Database Name** | Name of the database in which the command is being run. |
|  | **DBUserName** | The issuer's user name in the database. |
|  | **Target Login SID** | The SID of the targeted login. |

| | | |
|---|---|---|
| | **Target Login Name** | The name of the login that is having role membership modified. |
| | **Target User Name** | Name of the user that is having role membership modified. |
| **Audit Add Role Event** | **Event Class** | Type of event recorded = 111. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| | **Event Sub Class** | Class of event within the event. Values are:<br><br>1 = Add<br>2 = Drop |
| | **Database Name** | Name of the database in which the command is being run. |
| | **DBUserName** | The issuer's user name in the database. |
| | **Role Name** | Name of the role being created in the database. |
| **Audit Addlogin Event** | **Event Class** | Type of event being recorded = 104. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| | **Event Sub Class** | Class of event within the event. Values are:<br><br>1 = Add<br>2 = Drop |
| | **Target Login** | Security identification number |

| | | |
|---|---|---|
| | **SID** | (SID) assigned to the login being added. |
| | **Target Login Name** | Name of the login being added. |
| **Audit App Role Change Password Event** | **Event Class** | Type of event recorded = 112. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| | **Event Sub Class** | Class of event within the event. Value is:<br><br>Always = 1 |
| | **Database Name** | Name of the database in which the command is being run. |
| | **DBUserName** | The issuer's user name in the database. |
| | **Role Name** | Database application role name whose password is being changed. |
| **Audit Backup/Restore Event** | **Event Class** | Type of event recorded = 115. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| | **Event Sub Class** | Class of event within the event. Values are:<br><br>1 = Backup<br>2 = Restore |
| | **Database Name** | Name of the database in which the |

| | | |
|---|---|---|
| | | command is being run. |
| | **DBUserName** | The issuer's user name in the database. |
| | **Text Data** | The SQL text of the backup/restore statement. |
| **Audit Change Audit Event** | **Event Class** | Type of event recorded = 117. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| | **Event Sub Class** | Class of event within the event. Values are:<br><br>1 = New audit started<br>2 = Audit stopped |
| **Audit DBCC Event** | **Event Class** | Type of event recorded = 116. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| | **Event Sub Class** | Class of event within the event. Value is:<br><br>Always = 1 |
| | **Database Name** | Name of the database in which the command is being run. |
| | **DBUserName** | The issuer's user name in the database. |
| | **Text Data** | The SQL text of the DBCC command. |
| **Audit Login Event** | **Event Class** | Type of event being recorded = 14. |
| | **Text Data** | A delimited list of all set options. |
| | | |

| | | |
|---|---|---|
| | **Binary Data** | Session level settings, including ANSI nulls, ANSI padding, cursor close on commit, null concatenation, and quoted identifiers. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| **Audit Login Change Password Event** | **Event Class** | Type of event recorded = 107. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| | **Event Sub Class** | Class of event within the event. Values are:<br><br>1 = User changed his or her own password.<br>2 = User changed the password of another user. |
| | **Target Login SID** | Security identification number (SID) of the targeted Windows login. |
| | **Target Login Name** | Name of the targeted Windows login. |
| **Audit Login Change Property Event** | **Event Class** | Type of event being recorded = 106. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |

| | | |
|---|---|---|
| | **Event Sub Class** | Class of event within the event. Values are:<br><br>1 = Default database<br>2 = Default language |
| | **Target Login SID** | Security identification number (SID) of the targeted Windows login. |
| | **Target Login Name** | Name of the targeted Windows login. |
| **Audit Login Failed Event** | **Event Class** | Type of event being recorded = 20 |
| | **Success** | The success or failure of the audit indicator. Value will always be:<br><br>0 = Failure |
| **Audit Login GDR Event** | **Event Class** | Type of event being recorded = 105. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| | **Event Sub Class** | Class of event within the event. Values are:<br><br>1 = Grant<br>2 = Revoke<br>3 = Deny |
| | **Target Login SID** | Security identification number (SID) of the targeted Windows login. |
| | **Target Login Name** | Name of the targeted Windows login. |
| **Audit Logout Event** | **Event Class** | Type of event being recorded = 15. |
| | | |

| | | |
|---|---|---|
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| | **End Time** | The end time of the log out. |
| | **Duration** | The approximate amount of time since the user logged in. |
| | **Reads** | The amount of logical read I/Os issued by this user during the connection. |
| | **Writes** | The amount of logical write I/Os issued by this user during the connection. |
| | **CPU** | The amount of CPU used by this user during the connection. |
| **Audit Object Derived Permission Event** | **Event Class** | Type of event being recorded = 118. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| | **Event Sub Class** | Class of event within the event. Values are:<br><br>1 = Create object<br>2 = Alter object<br>3 = Drop object |
| | **Database Name** | The name of the database in which the object is being created, altered, or dropped. |
| | **DBUserName** | The issuer's user name in the database. |
| | **Object Type** | Type of object being created, |

| | | |
|---|---|---|
| | | altered, or dropped. Values are:<br><br>1 = Index<br>2 = Database<br>3 = User object<br>4 = CHECK constraint<br>5 = Default or DEFAULT constraint<br>6 = FOREIGN KEY constraint<br>7 = PRIMARY KEY constraint<br>8 = Stored procedure<br>9 = User-defined function (UDF)<br>10 = Rule<br>11 = Replication filter stored procedure<br>12 = System table<br>13 = Trigger<br>14 = Inline function<br>15 = Table valued UDF<br>16 = UNIQUE constraint<br>17 = User table<br>18 = View<br>19 = Extended stored procedure<br>20 = Ad-hoc query<br>21 = Prepared query<br>22 = Statistics |
| | **Object Name** | The name of the object that is being created, altered, or dropped. |
| | **Owner Name** | The database username of the object owner of the object being created, altered, or dropped. |
| | **Text Data** | The SQL text of the statement. |
| **Audit Object GDR Event** | **Event Class** | Type of event being recorded = 103. |
| | **Success** | The success or failure of the audit indicator. Values are: |

| | | |
|---|---|---|
| | | 0 = Failure<br>1 = Success |
| | **Event Sub Class** | Class of event within the event. Values are:<br><br>1 = Grant<br>2 = Revoke<br>3 = Deny |
| | **Database Name** | Name of the database that the GRANT/DENY/REVOKE of the object permission is run in. |
| | **DBUserName** | The issuer's user name in the database. |
| | **Owner Name** | Name of the user who owns the object against which the GRANT/DENY/REVOKE statement is being run. |
| | **Object Name** | Name of the object to which the permissions are being applied. |
| | **Permissions** | Type of statement issued. Values are:<br><br>1 = SELECT ALL<br>2 = UPDATE ALL<br>4 = REFERENCES ALL<br>8 = INSERT<br>16 = DELETE<br>32 = EXECUTE (procedures only) |
| | **Column Permissions** | Indicates whether a column permission was set. Values are:<br><br>0 = No<br>1 = Yes |
| | **Text Data** | The SQL text of the GRANT/REVOKE/DENY |

| | | |
|---|---|---|
| | | statement. |
| **Audit Object Permission Event** | **Event Class** | Type of event recorded = 114. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| | **Event Sub Class** | Class of event within the event. Value is:<br><br>Always = 1 |
| | **Database Name** | Name of the database in which the command is being run. |
| | **DBUserName** | The issuer's user name in the database. |
| | **Owner Name** | Owner name of the object for which the permissions are being checked. |
| | **Object Name** | Name of the object whose permissions are being checked. |
| | **Permissions** | Type of statement issued. Values are:<br><br>1 = SELECT ALL<br>2 = UPDATE ALL<br>4 = REFERENCES ALL<br>8 = INSERT<br>16 = DELETE<br>32 = EXECUTE (procedures only) |
| | **Column Permissions** | Indicates whether a column permission was used. Parse the statement text to determine which permissions were applied to which columns. |
| | | |

| | | |
|---|---|---|
| | **Text Data** | Text value dependent on the event class captured. |
| **Audit Server Starts and Stops Event** | **Event Class** | Type of event recorded = 118. |
| | **Event Sub Class** | Class of event within the event. Values are:<br><br>1 = Instance Shutdown<br>2 = Instance Started<br>3 = Instance Pause<br>4 = Instance Continued |
| | **Login SID** | Security identification number (SID) of the login running the GRANT/DENY/REVOKE statement for the Windows login. |
| | **Login Name** | Name of the login running GRANT/DENY/REVOKE statement for the Windows login. |
| **Audit Statement GDR Event** | **Event Class** | Type of event being recorded = 102. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| | **Event Sub Class** | Class of event within the event. Values are:<br><br>1 = GRANT<br>2 = REVOKE<br>3 = DENY |
| | **Database Name** | Name of the database to which the GRANT/DENY/REVOKE statement permission is being applied. |
| | **DBUserName** | The issuer's user name in the |

| | | |
|---|---|---|
| | | database. |
| | **Permissions** | Type of statement issued. Values are:<br><br>1 = CREATE DATABASE (**master** database only)<br>2 = CREATE TABLE<br>4 = CREATE PROCEDURE<br>8 = CREATE VIEW<br>16 = CREATE RULE<br>32 = CREATE DEFAULT<br>64 = BACKUP DATABASE<br>128 = BACKUP LOG<br>512 = CREATE FUNCTION |
| | **Text Data** | The SQL text of the GRANT/DENY/REVOKE statement. |
| **Audit Statement Permission Event** | **Event Class** | Type of event recorded = 113. |
| | **Success** | The success or failure of the audit indicator. Values are:<br><br>0 = Failure<br>1 = Success |
| | **Event Sub Class** | Class of event within the event. Value is:<br><br>Always = 1 |
| | **Database Name** | Name of the database in which the command is being run. |
| | **DBUserName** | The issuer's user name in the database. |
| | **Permissions** | Type of statement issued. Values are:<br><br>1 = CREATE DATABASE (**master** database only) |

|  |  |  |
|---|---|---|
|  |  | 2 = CREATE TABLE<br>4 = CREATE PROCEDURE<br>8 = CREATE VIEW<br>16 = CREATE RULE<br>32 = CREATE DEFAULT<br>64 = BACKUP DATABASE<br>128 = BACKUP LOG<br>512 = CREATE FUNCTION |
|  | **Text Data** | Text value dependent on the event class captured. |

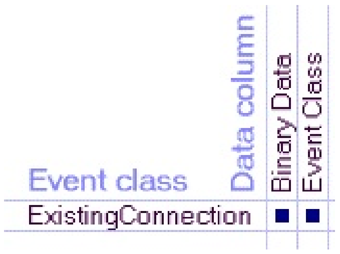
## See Also

Security Audit Event Category

Security Audit Event Classes

## Sessions Event Category

Use the **Sessions** event category to monitor Microsoft® SQL Server™ user connections.



| Event class \ Data column | Binary Data | Event Class |
|---|---|---|
| ExistingConnection | ■ | ■ |

### See Also

Sessions Event Classes

Sessions Data Columns

## Sessions Event Classes

The following table describes the **Sessions** event classes in the **Sessions** event category.

| Event class | Description |
|---|---|
| **ExistingConnection** | Detects activity by all users connected to Microsoft® SQL Server™ before the trace was started. |

Using the **ExistingConnection** event classes, it is possible to monitor the length of time each user connection was connected to an instance of SQL Server, and the amount of SQL Server processor time the queries submitted on the connection took to execute. This information can be useful for determining:

- The amount of time and the volume of activity used by each SQL Server user. This can be useful for tracking database activity and charging each user for the time and SQL Server CPU time (**CPU** data column) used.

- The security of the system, by checking the users connecting to and using an instance of SQL Server.

### See Also

Sessions Event Category

Sessions Data Columns

## Sessions Data Columns

The following table lists the data columns for each event class in the **Sessions** event category.

| Event class | Data column | Description |
|---|---|---|
| **Existing Connection** | **Event Class** | Type of event recorded = 17. |
| | **Binary Data** | Session level settings, including ANSI nulls, ANSI padding, cursor close on commit, null concatenation, and quoted identifiers. For more information, see SET. |

In addition to the data columns that are specific to the **Existing Connection** event class, by monitoring the **DBUserName** and **NT User Name** default data columns, you can map the name of the user to each connection.

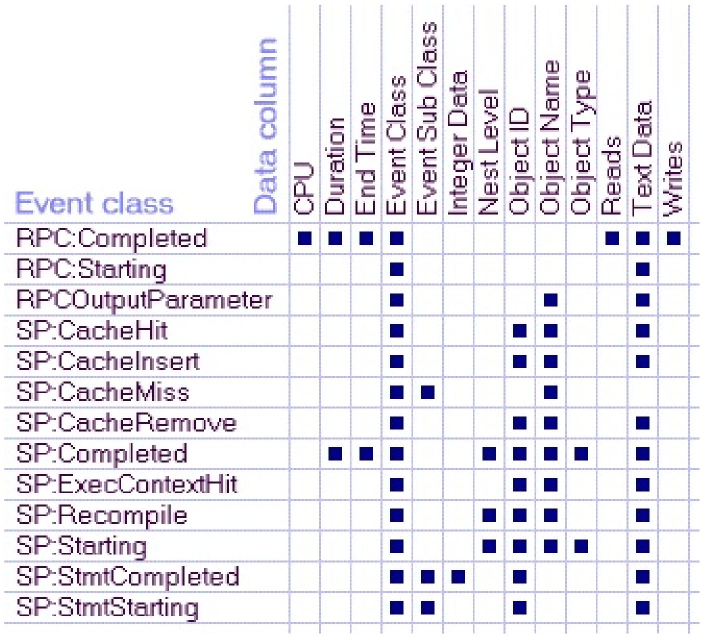
### See Also

Sessions Event Category

Sessions Event Classes

## Stored Procedures Event Category

Use the **Stored Procedures** event category to monitor the execution of stored procedures.



| Event class \ Data column | CPU | Duration | End Time | Event Class | Event Sub Class | Integer Data | Nest Level | Object ID | Object Name | Object Type | Reads | Text Data | Writes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RPC:Completed | ■ | ■ | ■ | ■ | | | | | | | ■ | ■ | ■ |
| RPC:Starting | | | | ■ | | | | | | | | ■ | |
| RPCOutputParameter | | | | ■ | | | | | ■ | | | ■ | |
| SP:CacheHit | | | | ■ | | | | ■ | ■ | | | ■ | |
| SP:CacheInsert | | | | ■ | | | | ■ | ■ | | | ■ | |
| SP:CacheMiss | | | | ■ | ■ | | | | ■ | | | | |
| SP:CacheRemove | | | | ■ | | | | ■ | ■ | | | ■ | |
| SP:Completed | | ■ | ■ | ■ | | | ■ | ■ | ■ | ■ | | ■ | |
| SP:ExecContextHit | | | | ■ | | | | ■ | ■ | | | ■ | |
| SP:Recompile | | | | ■ | | | ■ | ■ | ■ | | | ■ | |
| SP:Starting | | | | ■ | | | ■ | ■ | ■ | ■ | | ■ | |
| SP:StmtCompleted | | | | ■ | ■ | ■ | | ■ | | | | ■ | |
| SP:StmtStarting | | | | ■ | ■ | | | ■ | | | | ■ | |

### See Also

Stored Procedures Event Classes

Stored Procedures Data Columns

## Stored Procedures Event Classes

The following table describes the **Stored Procedures** event classes in the **Stored Procedures** event category.

| Event class | Description |
|---|---|
| **RPC Output Parameter** | Displays information about output parameters of a previously executed remote procedure call (RPC). |
| **RPC:Completed** | Occurs when an RPC has been completed. |
| **RPC:Starting** | Occurs when an RPC has started. |
| **SP:CacheHit** | Procedure is found in the cache. |
| **SP:CacheInsert** | Item is inserted into the procedure cache. |
| **SP:CacheMiss** | Stored procedure is not found in the procedure cache. |
| **SP:CacheRemove** | Item has been removed from the procedure cache. |
| **SP:Completed** | Stored procedure has completed. |
| **SP: ExecContextHit** | Execution version of a stored procedure has been found in the cache. |
| **SP:Recompile** | Stored procedure has been recompiled. |
| **SP:Starting** | Stored procedure has started. |
| **SP: StmtCompleted** | Statement within a stored procedure has completed. |
| **SP:StmtStarting** | Statement within a stored procedure has started. |

By monitoring the **SP:CacheHit** and **SP:CacheMiss** event classes, you can determine how often executed stored procedures are found in the cache. For example, if the **SP:CacheMiss** event class occurs frequently, it can indicate that more memory should be made available to Microsoft® SQL Server™, thereby increasing the size of the procedure cache. By monitoring the Object ID of the **SP:CacheHit** event class, you can determine which stored procedures reside in the cache.

The **SP:CacheInsert**, **SP:CacheRemove**, and **SP:Recompile** event classes can be used to determine which stored procedures are brought into cache (first executed), removed from the cache (aged out of the cache), and recompiled. For more information about recompiling stored procedures, see Recompiling a Stored Procedure. This information is useful to determine how stored procedures are being used by applications.

A stored procedure has a compiled version with shared data and an execution context version with session-specific data. When a stored procedure is looked up in the cache, execution contexts are looked for first. If none are found, the cache is searched for compiled plans. Use the **SP:ExecContextHit** event class to monitor execution contexts. If the **SP:ExecContextHit** event class is not generated for a stored procedure, then the stored procedure has no execution time cachable queries.

The execution of a stored procedure can be monitored by the **SP:Starting**, **SP:StmtStarting**, **SP:StmtCompleted**, and **SP:Completed** event classes and all the TSQL event classes.

**Note** **SP:StmtStarting** is provided for backward compatibility only. You should now use **SQL:StmtStarting** to trace this event. If you do choose to trace **SP:StmtStarting**, SQL Profiler will trace **SQL:StmtStarting**, as the two events are mapped together.

## See Also

Stored Procedures Event Category

Stored Procedures Data Columns

## Stored Procedures Data Columns

The following table lists the data columns for each event class in the **Stored Procedure** event category.

| Event class | Data column | Description |
|---|---|---|
| **RPC Output Parameter** | **Event Class** | Type of event recorded = 100. |
| | **Object Name** | Name of the output parameter from the RPC event (for example, handle). |
| | **Text Data** | Value of the parameter named in object name that was returned by the remote procedure call (RPC). |
| **RPC:Completed** | **Event Class** | Type of event recorded = 10. |
| | **End Time** | End time of the RPC. |
| | **Duration** | Duration of the RPC. |
| | **CPU** | Amount of CPU used by the RPC. |
| | **Reads** | Number of page reads issued by the RPC. |
| | **Writes** | Number of page writes issued by the RPC. |
| | **Text Data** | Text of the RPC. |
| **RPC:Starting** | **Event Class** | Type of event recorded = 11. |
| | **Text Data** | Text of the RPC. |
| **SP:CacheHit** | **Event Data** | Type of event recorded = 38. |
| | **Object ID** | Object ID of the stored procedure found in the cache. |
| | **Object Name** | Name of the stored procedure found in the cache. |
| | **Text Data** | Text of the SQL statement that was found in the cache. |
| **SP:CacheInsert** | **Event Class** | Type of event recorded = 35. |

| | | |
|---|---|---|
| | **Object ID** | Object ID of the stored procedure. |
| | **Object Name** | Name of the stored procedure found in the cache. |
| | **Text Data** | Text of the SQL statement that is being cached. |
| **SP:CacheMiss** | **Event Class** | Type of event recorded = 34. |
| | **Event Sub Class** | Nesting level of the stored procedure. |
| | **Object Name** | The name of the stored procedure found in the cache. |
| **SP:CacheRemove** | **Event Class** | Type of event recorded = 36. |
| | **Object ID** | Object ID of the stored procedure. |
| | **Object Name** | Name of the stored procedure found in the cache. |
| | **Text Data** | Text of the SQL statement being removed from the cache. |
| **SP:Completed** | **Event Class** | Type of event recorded = 43. |
| | **Nest Level** | Nesting level of the stored procedure. |
| | **End Time** | End time of the event. |
| | **Duration** | Length of time the stored procedure ran. |
| | **Object ID** | Object ID of the stored procedure. |
| | **Object Name** | Name of the stored procedure found in the cache. |
| | **Object Type** | Type of stored procedure that was called. |
| | **Text Data** | Text of the stored procedure call. |
| **SP:ExecContextHit** | **Event Class** | Type of event recorded = 39. |
| | **Object ID** | Object ID of the stored procedure. |
| | **Object Name** | The name of the stored procedure found in the cache. |
| | **Text Data** | The text of the stored procedure call found in the cache. |
| | | |

|  |  |  |
|---|---|---|
| **SP:Recompile** | **Event Class** | Type of event recorded = 37. |
|  | **Nest Level** | Nesting level of the stored procedure. |
|  | **Object ID** | The object ID of the stored procedure. |
|  | **Object Name** | The name of the stored procedure found in the cache. |
|  | **Text Data** | The text of the stored procedure call that triggered the recompile. |
| **SP:Starting** | **Event Class** | Type of event recorded = 42. |
|  | **Nest Level** | Nesting level of the stored procedure. |
|  | **Object ID** | The object ID of the stored procedure. |
|  | **Object Name** | The name of the stored procedure found in the cache. |
|  | **Object Type** | The type of stored procedure being started. |
|  | **Text Data** | The text of the stored procedure call. |
| **SP:StmtCompleted** | **Event Class** | Type of event recorded = 45. |
|  | **Event Sub Class** | Nesting level of the stored procedure. |
|  | **Integer Data** | Actual rows returned by the statement. |
|  | **Object ID** | System-assigned ID of the stored procedure. |
|  | **Text Data** | Text of the statement in the stored procedure. |
| **SP:StmtStarting** | **Event Class** | Type of event recorded = 44. |
|  | **Event Sub Class** | Nesting level of the stored procedure. |
|  | **Object ID** | System-assigned ID of the stored procedure. |
|  |  |  |

| | | |
|---|---|---|
| | **Text Data** | Text of the statement in the stored procedure. |

## See Also

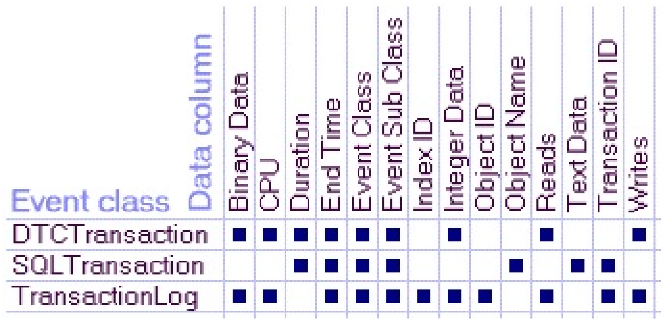
Stored Procedures

Stored Procedures Event Category

Stored Procedures Event Classes

## Transactions Event Category

The **Transactions** event classes can be used to monitor the status of transactions.



| Event class / Data column | Binary Data | CPU | Duration | End Time | Event Class | Event Sub Class | Index ID | Integer Data | Object ID | Object Name | Reads | Text Data | Transaction ID | Writes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DTCTransaction | ■ | ■ | ■ | ■ | ■ | ■ |  | ■ |  |  | ■ |  |  | ■ |
| SQLTransaction |  |  | ■ | ■ | ■ | ■ |  |  |  | ■ |  | ■ | ■ |  |
| TransactionLog | ■ | ■ |  | ■ | ■ | ■ | ■ | ■ | ■ |  | ■ |  | ■ | ■ |

### See Also

Transactions Event Classes

Transactions Data Columns

# Transactions Event Classes

The following table describes the **Transactions** event classes in the **Transactions** event category.

| Event class | Description |
|---|---|
| **DTCTransaction** | Tracks Microsoft® Distributed Transaction Coordinator (MS DTC) coordinated transactions between two or more databases. |
| **SQLTransaction** | Tracks Transact-SQL BEGIN, COMMIT, SAVE, and ROLLBACK TRANSACTION statements. |
| **TransactionLog** | Tracks when transactions are written to the transaction log. |

Use the **DTCTransaction** event class to monitor the state of MS DTC transactions as they occur. This can be useful when testing an application that uses distributed transactions.

Monitor the **SQLTransaction** event class when testing your application stored procedures or triggers to determine, for example, when transactions are committed or rolled back.

Use the **TransactionLog** event class when you want to monitor activity in the Microsoft SQL Server™ transaction log, for example, to test your application and determine the types of logging activity.

## See Also

Transactions Event Category

Transactions Data Columns

## Transactions Data Columns

The following table lists the data columns for each event class in the **Transactions** event category.

| Event class | Data column | Description |
|---|---|---|
| **DTCTransaction** | **Event Class** | Type of event recorded = 19. |
| | **Event Sub Class** | Microsoft® Distributed Transaction Coordinator (MS DTC) state. For more information, see the MS DTC documentation. Possible values include:<br><br>0 = GET_DTC_ADDRESS_SUB_CLASS<br>1 = PROPAGATE_XACT_SUB_CLASS<br>2 = DOWORK_SUB_CLASS<br>3 = CLOSE_CONN_SUB_CLASS<br>4 = DTC_VIRGIN_SUB_CLASS<br>5 = DTC_IDLE_SUB_CLASS<br>6 = DTC_BEG_DIST_SUB_CLASS<br>7 = DTC_ENLISTING_SUB_CLASS<br>8 = DTC_INT_ACTIVE_SUB_CLASS<br>9 = DTC_INT_COMMIT_SUB_CLASS<br>10 = DTC_INT_ABORT_SUB_CLASS<br>11 = DTC_INT_ASYNC_ABORT_SUB_CLASS<br>12 = DTC_ACTIVE_SUB_CLASS<br>13 = DTC_INIT_PREPARE_SUB_CLASS<br>14 = DTC_PREPARING_SUB_CLASS<br>15 = DTC_PREPARED_SUB_CLASS<br>16 = DTC_ABORTING_SUB_CLASS<br>17 = DTC_COMMITTING_SUB_CLASS<br>18 = DTC_DO_ASYNC_ABORT_SUB_CLASS<br>19 = DTC_DISASTER_SUB_CLASS |

| | | |
|---|---|---|
| | | 20 = DTC_DRAIN_ABORT_SUB_CLASS<br>21 = DTC_ASYNC_ABORT_SUB_CLASS<br>22 = DTC_TM_RECOVERY_SUB_CLASS |
| | **End Time** | The end time of the event. |
| | **Duration** | The length of the DTC transaction. |
| | **Reads** | The number of page reads generated locally by the DTC transaction. |
| | **Writes** | The number of page writes generated locally by the DTC transaction. |
| | **CPU** | The amount of CPU used by the DTC transaction. |
| | **Integer Data** | Transaction isolation level. Possible values are:<br><br>256 = Read uncommitted<br>4096 = Read Committed<br>65536 = Repeatable read<br>1048576 = Serializable<br>4294967295 = Unspecified |
| | **Binary Data** | Globally unique ID (GUID), in hexadecimal form, of the transaction, if available. For possible values of the Binary Data, see Table 2 below. |
| **SQLTransaction** | **Event Class** | Type of event recorded = 50. |
| | **Event Sub Class** | Type of SQL transaction event. Possible values include:<br><br>0 = Begin Transaction<br>1 = Commit Transaction<br>2 = Rollback Transaction<br>3 = A Savepoint was issued |
| | **End Time** | The end time of the event. This option is only for a COMMIT or ROLLBACK. |
| | **Duration** | How long the transaction ran for. This option is only for a COMMIT or |

| | | |
|---|---|---|
| | | ROLLBACK. |
| | **Transaction ID** | The internal ID number of the transaction. |
| | **Text Data** | The savepoint or rollback name, if provided. |
| | **Object Name** | The transaction name, if provided. |
| **TransactionLog** | **Event Class** | Type of event recorded = 54. |
| | **Event Sub Class** | Type of transaction log event, such as BEGINXACT(null). |
| | **Integer Data** | The length of the log record. |
| | **Binary Data** | The Replication log_pubid is the publication ID that is currently being worked on. If you are using replication and look in the table for MSPublications there is a column of publication_id. This is the value represented in Binary Data. You can use this ID to find the publication and any articles associated with it. |
| | **End Time** | The end time of the event. |
| | **Reads** | The number of read I/Os issued to perform the log entry. |
| | **Writes** | The number of I/Os issued to perform the log entry. |
| | **CPU** | The amount of CPU used to write the transaction entry. |
| | **Transaction ID** | The internal ID number of the transaction. |
| | **Object ID** | The ID of the object that has logged modifications. |
| | **Index ID** | The ID of the index that has logged modifications. |

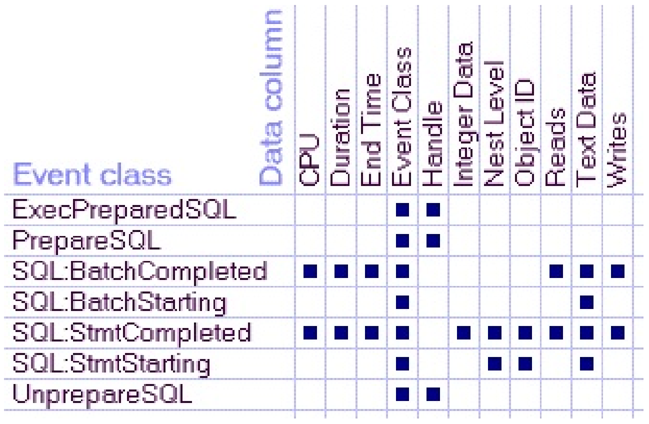
## See Also

Transactions Event Category

Transactions Event Classes

## TSQL Event Category

The **TSQL** event classes can be used to monitor the execution and completion of a batch, and a Transact-SQL statement.



| Event class \ Data column | CPU | Duration | End Time | Event Class | Handle | Integer Data | Nest Level | Object ID | Reads | Text Data | Writes |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ExecPreparedSQL | | | | ■ | ■ | | | | | | |
| PrepareSQL | | | | ■ | ■ | | | | | | |
| SQL:BatchCompleted | ■ | ■ | ■ | ■ | | | | | ■ | ■ | ■ |
| SQL:BatchStarting | | | | ■ | | | | | | ■ | |
| SQL:StmtCompleted | ■ | ■ | ■ | ■ | | ■ | ■ | ■ | ■ | ■ | ■ |
| SQL:StmtStarting | | | | ■ | | | ■ | ■ | | ■ | |
| UnprepareSQL | | | | ■ | ■ | | | | | | |

### See Also

TSQL Event Classes

TSQL Data Columns

## TSQL Event Classes

The following table describes the **TSQL** event classes in the **TSQL** event category.

| Event class | Description |
|---|---|
| **Exec Prepared SQL** | Indicates when a prepared SQL statement or statements have been executed by ODBC, OLEDB, or DB-Library. |
| **Prepare SQL** | Indicates when an SQL statement or statements have been prepared for use by ODBC, OLEDB, or DB-Library. |
| **SQL:BatchCompleted** | Transact-SQL batch has completed. |
| **SQL:BatchStarting** | Transact-SQL batch has started. |
| **SQL:StmtCompleted** | Transact-SQL statement has completed. |
| **SQL:StmtStarting** | Transact-SQL statement has started. |
| **Unprepare SQL** | Indicates when a prepared SQL statement or statements have been unprepared by ODBC, OLEDB, or DB-Library. |

By monitoring the **TSQL** event classes and monitoring the events using single stepping, you can monitor your application queries. The **SQL:BatchStarting** event class will show the Transact-SQL submitted in a batch, while the **SQL:StmtStarting** event class shows the individual statement within a batch. By replaying the **SQL:BatchCompleted** event class, any results returned by the batch are displayed and can be checked to ensure they match the expected results.

Monitoring the **Start Time**, **End Time**, and **Duration** default data columns reveals when the events start and complete, and how long each remote procedure call (RPC), batch, or statement takes to complete. By grouping events based on the **Duration** default data column, you can easily determine the longest running queries. Monitoring the **NT User Name** and **DBUserName** default data columns can also identify users who submit these queries.

## See Also

TSQL Event Category

TSQL Data Columns

## TSQL Data Columns

The following table lists the data columns for each event class in the **TSQL** event category.

| Event class | Data column | Description |
|---|---|---|
| **Exec Prepared SQL** | **Event Class** | Type of event recorded = 72. |
| | **Handle** | Handle of the prepared TSQL statement. |
| **Prepare SQL** | **Event Class** | Type of event recorded = 71. |
| | **Handle** | Handle of the prepared TSQL statement. |
| **SQL:BatchCompleted** | **Event Class** | Type of event recorded = 12. |
| | **Duration** | The duration of the event. |
| | **End Time** | The end time of the event. |
| | **Reads** | The number of page read I/Os caused by the batch. |
| | **Writes** | The number of page write I/Os caused by the batch. |
| | **CPU** | The CPU used during the batch. |
| | **Text Data** | The text of the batch. |
| **SQL:BatchStarting** | **Event Class** | Type of event recorded = 13. |
| | **Text Data** | The text of the batch. |
| **SQL:StmtCompleted** | **Event Class** | Type of event recorded = 41. |
| | **Duration** | The duration of the event. |
| | **End Time** | The end time of the event. |
| | **Reads** | The number of page reads issued by the SQL statement. |
| | **Writes** | The number of page writes issued by the SQL statement. |
| | **CPU** | The CPU used by the SQL statement. |
| | | |

| | | |
|---|---|---|
| | **Integer Data** | The number of rows returned by the SQL statement. |
| | **Object ID** | The object ID of the parent stored procedure, if the SQL statement was run within a stored procedure. |
| | **Nest Level** | The nest level of the stored procedure, if the SQL statement was run within a stored procedure. |
| | **Text Data** | The text of the statement that is about to be executed. |
| **SQL:StmtStarting** | **Event Class** | Type of event recorded = 40. |
| | **Object ID** | The object ID of the parent stored procedure, if the SQL statement was run within a stored procedure. |
| | **Nest Level** | The next level of the stored procedure, if the SQL statement was run within a stored procedure. |
| | **Text Data** | The text of the statement that is about to be executed. |
| **Unprepare SQL** | **Event Class** | Type of event recorded = 73. |
| | **Handle** | The handle of the prepared TSQL statement. |

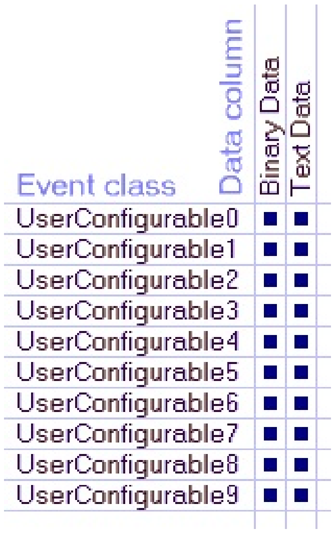
## See Also

TSQL Event Category

TSQL Event Classes

# User Configurable Event Category

Use the **User Configurable** event category to monitor user-defined events. Create user-defined events to monitor events that cannot be monitored by the system-supplied events in other event categories. For example, a user-defined event can be created to monitor the progress of the application you are testing. As the application runs, it can generate events at predefined points, allowing you to determine the current execution point in your application.



| Event class / Data column | Binary Data | Text Data |
|---|---|---|
| UserConfigurable0 | ■ | ■ |
| UserConfigurable1 | ■ | ■ |
| UserConfigurable2 | ■ | ■ |
| UserConfigurable3 | ■ | ■ |
| UserConfigurable4 | ■ | ■ |
| UserConfigurable5 | ■ | ■ |
| UserConfigurable6 | ■ | ■ |
| UserConfigurable7 | ■ | ■ |
| UserConfigurable8 | ■ | ■ |
| UserConfigurable9 | ■ | ■ |

## See Also

User Configurable Event Classes

User Configurable Data Columns

## User Configurable Event Classes

As user-defined events are generated by your application using the **sp_trace_generateevent** stored procedure, the *event_class* parameter you specify determines which of the following 10 event classes to monitor.

The following table describes the **User Configurable** event classes in **User Configurable** event category.

| Event class | Description |
|---|---|
| **UserConfigurable (0-9)** | Event data defined by the user. |

### See Also

User Configurable Event Category

User Configurable Data Columns

## User Configurable Data Columns

The following table lists the data columns for each event class in the **User Configurable** event category.

| Event class | Data column | Description |
|---|---|---|
| **UserConfigurable (0-9)** | **Text Data** | Text value dependent on the event class captured in the trace. |
| | **Binary Data** | Binary value dependent on the event class captured in the trace. |

Not all data columns will be produced for the User Configurable event classes. When you create a trace, you can select the following columns:

- Application Name
- Binary Data
- Database ID
- Host Name
- Login Name
- Login SID
- NT Domain Name
- NT User Name
- Server Name

- Text Data

### To create a user-defined event class

⊞ Transact-SQL

## See Also

User Configurable Event Category

User Configurable Event Classes

## Administering SQL Server

## Creating and Managing Traces and Templates

In Microsoft® SQL Server™, you can use SQL Profiler to create one or more templates that define the criteria for each event you want to monitor. You can save the template to a file with the .tdf extension. A template is not executed. After you define the template, you run a trace that records the data for each event you selected.

For example, you can create a template, specifying which events, data columns, and filters to use. Then you can save the template and launch a trace with the current template settings. The trace data captured is based upon the options specified in the template. You can specify where the trace results can be saved (for example, in a trace file (.trc extension file) or a trace table).

### Default Templates

Before creating a trace using SQL Profiler, you can specify a default trace template. To select a default trace template, go to the **Tools** menu, and then select **Options**.

You can also specify:

- To start the default trace immediately after making a connection.
- The number of lines of trace data buffered for display. If window auto-scrolling is disabled and the specified limit is reached, the trace pauses until you scroll down to the last row. At this point, 10 percent of the top rows are deleted and the trace continues.
- A font for the displayed trace data.
- A font size for the displayed trace data.

When creating a trace, you can specify the following:

- A trace name.

- Which instance of SQL Server to trace.

- Options for saving trace data. For example, you can choose to capture trace data to the server file. If you choose this option, you must capture trace data to the server being traced and set a maximum file size for the server file. If the maximum file size is reached, you can enable the file rollover option, which creates new files to store the trace data. You can also save a trace to a file, table, or a combination of these options. If you will be tracing a large amount of data, you should save the data to the server file. This guarantees that all events produced will be saved in the file. There is a limit of 1 gigabyte (GB) for maximum file size.

- A trace stop time.

- Events to trace. For more information about the event classes available, see Monitoring with SQL Profiler Event Categories.

- Data columns to capture. For more information about the data columns available, see Monitoring with SQL Profiler Event Categories.

- Filters that specify the criteria for determining which events to capture.

## Using System Stored Procedures

SQL Profiler uses system stored procedures to create traces and send the trace output to the appropriate destination. These system stored procedures can be used from within your own applications to create traces manually, instead of using SQL Profiler. This allows you to write custom applications specific to the needs of your enterprise. For example, when using system stored procedures to create traces, you can:

- Configure traces.

- Forward trace events from one or more servers to a file.

The following table compares the SQL Server 2000 system stored procedures to the SQL Server version 7.0 stored procedures.

| 7.0 extended stored procedure | 2000 stored procedures |
|---|---|
| xp_trace_geteventclassrequired | fn_trace_geteventinfo |
| xp_trace_getqueuecreateinfo | fn_trace_geteventinfo |
| xp_trace_getqueueproperties | fn_trace_geteventinfo |
| xp_trace_getqueuecreateinfo | fn_trace_getinfo |
| xp_trace_getqueuedestination | fn_trace_getinfo |
| xp_trace_getqueueproperties | fn_trace_getinfo |
| xp_trace_addnewqueue | sp_trace_create |
| xp_trace_setqueuecreateinfo | sp_trace_create |
| xp_trace_setqueuedestination | sp_trace_create |
| xp_trace_generate_event | sp_trace_generateevent |
| xp_trace_addnewqueue | sp_trace_setevent |
| xp_trace_eventclassrequired | sp_trace_setevent |
| xp_trace_seteventclassrequired | sp_trace_setevent |
| xp_trace_destroyqueue | sp_trace_setstatus |
| xp_trace_pausequeue | sp_trace_setstatus |
| xp_trace_restartqueue | sp_trace_setstatus |
| xp_trace_startconsumer | sp_trace_setstatus |
| xp_trace_getappfilter | fn_trace_getfilterinfo |
| xp_trace_getconnectionidfilter | fn_trace_getfilterinfo |
| xp_trace_getcpufilter | fn_trace_getfilterinfo |
| xp_trace_getdbIdfilter | fn_trace_getfilterinfo |
| xp_trace_getdurationfilter | fn_trace_getfilterinfo |
| xp_trace_geteventfilter | fn_trace_getfilterinfo |
| xp_trace_gethostfilter | fn_trace_getfilterinfo |
| xp_trace_gethpIdfilter | fn_trace_getfilterinfo |
| xp_trace_getIndIdfilter | fn_trace_getfilterinfo |
| xp_trace_getntdmfilter | fn_trace_getfilterinfo |
| xp_trace_getntnmfilter | fn_trace_getfilterinfo |

| | |
|---|---|
| xp_trace_getobjidfilter | fn_trace_getfilterinfo |
| xp_trace_getreadfilter | fn_trace_getfilterinfo |
| xp_trace_getserverfilter | fn_trace_getfilterinfo |
| xp_trace_getseverityfilter | fn_trace_getfilterinfo |
| xp_trace_getspIdfilter | fn_trace_getfilterinfo |
| xp_trace_getsysobjectsfilter | fn_trace_getfilterinfo |
| xp_trace_gettextfilter | fn_trace_getfilterinfo |
| xp_trace_getuserfilter | fn_trace_getfilterinfo |
| xp_trace_getwritefilter | fn_trace_getfilterinfo |
| xp_trace_setappfilter | sp_trace_setfilter |
| xp_trace_setconnectionidfilter | sp_trace_setfilter |
| xp_trace_setcpufilter | sp_trace_setfilter |
| xp_trace_setdbIdfilter | sp_trace_setfilter |
| xp_trace_setdurationfilter | sp_trace_setfilter |
| xp_trace_seteventfilter | sp_trace_setfilter |
| xp_trace_sethostfilter | sp_trace_setfilter |
| xp_trace_sethpIdfilter | sp_trace_setfilter |
| xp_trace_setIndIdfilter | sp_trace_setfilter |
| xp_trace_setntdmfilter | sp_trace_setfilter |
| xp_trace_setntnmfilter | sp_trace_setfilter |
| xp_trace_setobjidfilter | sp_trace_setfilter |
| xp_trace_setreadfilter | sp_trace_setfilter |
| xp_trace_setserverfilter | sp_trace_setfilter |
| xp_trace_setseverityfilter | sp_trace_setfilter |
| xp_trace_setspIdfilter | sp_trace_setfilter |
| xp_trace_setsysobjectsfilter | sp_trace_setfilter |
| xp_trace_settextfilter | sp_trace_setfilter |
| xp_trace_setuserfilter | sp_trace_setfilter |
| xp_trace_setwritefilter | sp_trace_setfilter |

System stored procedures expose the underlying architecture used to create traces. The architecture components are:

Producer

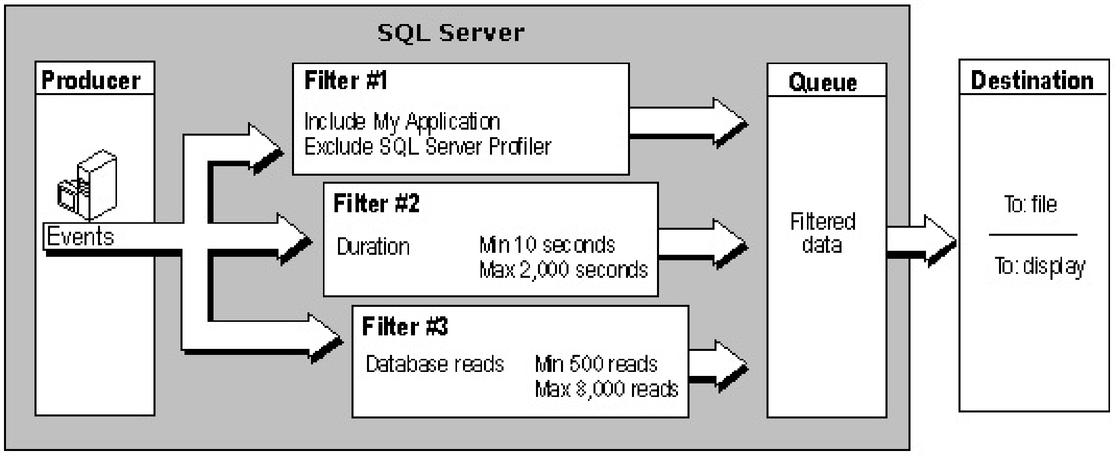
Generates the events to be monitored. The SQL Server lock manager, which generates lock events, is an example of a producer. For more information, see Locks Event Category.

Filter

Restricts the data monitored by the trace. For more information, see Limiting Traces.

Destination

Houses event data extracted from the trace in files.



To define your own trace using stored procedures:

1. Specify the events to capture using sp_trace_setevent.

2. Specify any event filters. For more information, see How to set a trace filter (Transact-SQL).

3. Specify the destination for the captured event data using sp_trace_create.

## To set trace definition defaults

## Limiting Traces

If a filter is not set, all events of the selected event classes are returned in the trace output. Filters limit the events collected in the trace. For example, limiting the Microsoft® Windows NT® 4.0 or Windows® 2000 user names in the trace to specific users reduces the output data to only those users in which you are interested.

Trace event criteria are parameters used to restrict (filter) the event data captured within the trace. For example, you can monitor the activity of a specific application or exclude an application from monitoring (the default trace event criteria excludes SQL Profiler from monitoring itself). For example, when monitoring queries to determine the batches that take the longest time to execute, you can set the trace event criteria to monitor (trace) only those batches that take longer than 30 seconds to execute (a CPU minimum value of 30,000 milliseconds). You can specify filters by right-clicking on the appropriate trace event criteria value and entering the information.

**To filter events in a trace template**

## Maximum File and Data Size

Using the **Maximum file size** and the **Maximum rows** options, you can specify the maximum size of the file or table holding the trace information and control the amount of space and resources used by a trace.

If a trace defined to save data to a file is started using SQL Profiler and the file already exists, you can append to or overwrite the file. If you choose to append to the file, and the trace file already meets or exceed the specified maximum file size, you are notified and given the opportunity to either increase the maximum file size or specify a new file. The same is true for trace tables.

### Maximum File Size

A trace with a maximum file size specified stops saving trace information to the file after the maximum file size has been reached. The maximum file size option must be used when you are saving trace data to a file. The default maximum file size is 5 megabyte (MB). The file rollover option is enabled by default when you are saving trace data to a file.

There is a limit of 1 gigabyte (GB) for the maximum file size option.

**How to set a maximum file size for a trace file**

## Datetime Filter

You can set a trace to filter out events that do not occur during a specified time. For example, if you want to view an event that started between 2:00 A.M. and 2:30 A.M., you can set a trace to automatically filter out all events starting before 2:00 A.M. and after 2:30 A.M. **Start Time** and **End Time** filters capture events that occur only between the specified times. The trace itself is active during the time and does not autostart or stop at the **Start Time** and **End Time**. These times affect only the filter.

### How to set a Start Time filter for a trace

## System SPID

You can define a trace that records only Microsoft® SQL Server™ processes while filtering out any unnecessary system events. Filtering out system server process IDs (SPIDs) saves system resources and time when you run a trace on a busy server.

**How to filter system IDs in a trace**

## Saving Traces and Templates

Saving a trace involves saving the captured event data to a specified place. Saving a template involves saving the definition of the trace, such as specified data columns, events, and filters.

### Saving Event Data

Save the captured event data to a file or a Microsoft® SQL Server™ table when you need to analyze or replay the captured data at a later time (for example, for trend forecasting or troubleshooting and debugging application problems). You can:

- Use a trace file or trace table to create a workload that is used as input for the Index Tuning Wizard.

- Use a trace file to capture events and send the trace file to the support provider for analysis.

- Use the query processing tools in SQL Server to access the data or to view the data in SQL Profiler. However, only members of the **sysadmin** fixed server role or the table creator can access the trace table directly.

  **IMPORTANT** Capturing trace data to a table is slower than capturing to a file. An alternative is to capture a trace to a file, open the trace file, and then save the trace as a trace table.

When using a trace file, SQL Profiler saves captured event data (not trace definitions) to a SQLProfiler (*.trc) file. The extension is added to the end of the file automatically when the trace file is saved, regardless of any other specified extension. For example, if you specify a trace file called Trace.dat, the file created is called Trace.dat.trc.

**Note** If SQL Profiler is running on Microsoft Windows® 2000 or Microsoft Windows NT® 4.0, you cannot open trace or script files on a Windows 98 shared directory.

## Saving Templates

The template definition of a trace includes the event classes, data columns, event criteria (filters), and all other properties (except the captured event data) used to create a trace. Templates created using SQL Profiler are saved in a file on the computer running SQL Profiler.

If you frequently monitor SQL Server, save templates in order to analyze performance. The templates capture the same event data each time and use the same trace definition to monitor the same events without having to define the event classes and data columns every time you create a trace. Additionally, a template can be given to another user to monitor specific SQL Server events. For example, a support provider can supply a customer with a template. The template is used by the customer to capture the required event data, which is then sent to the support provider for analysis.

### To save a trace to a file

## Modifying Templates

You can modify templates saved in the file on the local computer running SQL Profiler and templates derived from files. You may need to derive a template from a trace file if you cannot remember the original template that was used to create the trace, or if you want to run the same trace at a later date. Template properties, such as event classes and data columns, are modified in the same way in which the properties were originally set. Event classes and data columns can be added or removed, and filters can be changed. After the template is modified, saving it with the same name overwrites the original template. For more information, see Creating and Managing Traces and Templates.

When working with existing traces, you can view the properties, but you cannot modify them.

**WARNING** Saving a trace file with the same name overwrites the original trace file, causing any of the originally captured events or data columns that were removed or filtered to be lost.

### To derive a template from a trace file or trace table

## Starting, Pausing, and Stopping Traces

After you have created a template using SQL Profiler, you can start, pause, or stop capturing data using the new trace.

When you start a trace and the server is the defined source, Microsoft® SQL Server™ creates a queue that provides a temporary holding place for captured server events.

Each trace can have multiple producers. A producer collects events in a specific event category and sends the data to the queue. Events are read off the queue in the order in which they were placed. This method is called first-in/first-out (FIFO).

When using SQL Profiler, starting a trace opens a new trace window (if one is not already open), and data is immediately captured. When using SQL Server system stored procedures, you start a trace either manually or automatically every time an instance of SQL Server starts. A soon as the trace is started, data is captured. When a trace has been started, you can modify the name of the trace only.

Pausing a trace prevents further event data from being captured until the trace is restarted. Restarting a trace resumes trace operations. Any previously captured data is not lost. When the trace is restarted, data capturing is resumed from that point onward. When a trace is paused, you can change the name, events, columns, and filters. However, you cannot change the destination(s) to which you are sending the trace or the server connection.

Stopping a trace stops data from being captured. After a trace is stopped, it cannot be restarted without losing any previously captured data, unless the data has been captured to a trace file or trace table. All trace properties that were previously selected are preserved when a trace is stopped. When a trace is stopped, you can change the name, events, columns, and filters.

### To run a trace after it has been paused or stopped

## Viewing and Analyzing Traces

Use SQL Profiler to view captured event data in a trace. SQL Profiler displays data based on defined trace properties. One way to analyze Microsoft® SQL Server™ data is to copy the data to another program, such as SQL Query Analyzer or the Index Tuning Wizard. The Index Tuning Wizard can use a trace file that contains SQL batch and remote procedure call (RPC) events (and **Text** data columns). By specifying a server and database name when using the wizard, the captured data can be analyzed against a different server and database. For more information, see Index Tuning Wizard.

When a trace is opened using SQL Profiler, it is not necessary for the trace file to have the .trc file extension if the file was created by either SQL Profiler or the Profiler stored procedures.

**Note**  SQL Profiler can also read SQL Trace .log files and generic SQL script files. When opening a SQL Trace .log file that does not have a .log file extension, for example trace.txt, specify SQLTrace_Log as the file format.

The SQL Profiler display can be configured with customized font, font size, preview lines, and client buffer size to assist in trace analysis.

### Analyzing Data to Troubleshoot

Using SQL Profiler, you can troubleshoot data, such as queries that perform poorly or have exceptionally high numbers of logical reads, can be found by grouping traces or trace files by the **Duration**, **CPU**, **Reads**, or **Writes** data columns.

Additional information can be found by saving traces to tables and using Transact-SQL to query the event data. For example, to determine which **SQL:BatchCompleted** events had excessive wait time, execute:

```
SELECT  TextData, Duration, CPU
FROM    trace_table_name
WHERE   EventClass = 12 -- SQL:BatchCompleted events
AND     CPU < (.4 * Duration)
```

## Displaying Object Names When Viewing Traces

If you capture the **Server Name** and **Database ID** data columns in your trace, SQL Profiler displays the object name instead of the object ID (for example, **Orders** instead of the number 165575628). Similarly, if you capture the **Server Name**, **Database ID**, and **Object ID**, SQL Profiler displays the index name instead of the index ID.

If you choose to group by the **Object ID** data column, group by the **Server Name** and **Database ID** data columns first, and then **Object ID**. Similarly, if you choose to group by the **Index ID** data column, group by the **Server Name**, **Database ID**, and **Object ID** data columns first, and then **Index ID**. You need to group in this way because object and index IDs are not unique between servers and databases (and objects for index IDs).

## Finding Specific Events Within a Trace

Here are the basic steps for finding and grouping events in a trace:

1. Create your trace.
    - When defining the trace, capture the **Event Class**, **ClientProcessID**, and **Start Time** data columns in addition to any other data columns you want to capture.

    - Group the captured data by the **Event Class** data column, and capture the trace to a file or table.
2. Find the target events.
    - Open the trace file or table, and expand the node of the desired event class, for example, **Deadlock Chain**. (The file can be opened for viewing while the trace is writing to it unless the trace is located on a computer running Microsoft Windows® Windows 98. Use the **Refresh** command in the **View** menu to display the new rows.)

    - Search through the trace until you find the events for which you are looking (you can use the **Find** option on the **Edit** menu of SQL Profiler to help you find values in the trace). Note the values in the **ClientProcessID** and **Start Time** data columns of the desired events.

3. Display the events in context.

    - Display the trace data column properties, and group by **ClientProcessID** instead of **Event Class**.

    - Expand the nodes of each client process ID you want to view. Search through the trace manually, or use the **Find** option until you find the previously noted **Start Time** values of the target events. The events are displayed in chronological order with the other events that belong to each selected client process ID. For example, the **Deadlock** and **Deadlock Chain** events, captured within the trace, will be immediately after the **SQL:BatchStarting** events within the expanded client process ID.

The same technique can be used to find events grouped by **Server Name**, **Database ID**, and **Object ID**. Once you have found the events for which you are looking, group by **ClientProcessID**, **Application Name**, or another event class to view related activity in chronological order.

### To view a saved trace

## Replaying Traces

When you create or edit a trace, you can save the trace to replay it later. SQL Profiler features a multithreaded playback engine that can simulate user connections and SQL Server Authentication, allowing the user to reproduce the activity captured in the trace. Therefore, replay is useful when troubleshooting an application or process problem. When you have identified the problem and implemented corrections, run the trace that found the potential problem against the corrected application or process, then replay the original trace to compare results.

Trace replay supports debugging using break points and run-to-cursor, which especially improves the analysis of long scripts. For more information, see Single-Stepping Traces.

### Replay Requirements

In addition to any other event classes you want to monitor, the following event classes must be captured in a trace to allow the trace to be replayed:

- **Connect**
- **CursorExecute** (only required when replaying server-side cursors)
- **CursorOpen** (only required when replaying server-side cursors)
- **CursorPrepare** (only required when replaying server-side cursors)
- **Disconnect**
- **Exec Prepared SQL** (only required when replaying server-side prepared SQL statements)

- **ExistingConnection**
- **Prepare SQL** (only required when replaying server-side prepared SQL statements)
- **RPC:Starting**
- **SQL:BatchStarting**

In addition to any other data columns you want to capture, the following data columns must be captured in a trace to allow the trace to be replayed:

- **Application Name**
- **Binary Data**
- **ClientProcessID** or **SPID**
- **Database ID**
- **Event Class**
- **Event Sub Class**
- **Host Name**
- **Integer Data**
- **Server Name**
- **SQL User Name**

- **Start Time**

- **Text**

  **Note** Use the sample trace template **SQLProfilerTSQL_Replay** for traces capturing data for replay.

In order to replay a trace against a computer running Microsoft® SQL Server™ (the target), other than the computer originally traced (the source):

- All logins and users contained in the trace must already be created on the target and in the same database as the source.

- All logins and users in the target must have the same permissions they had in the source.

- All login passwords must be the same as the user executing the replay.

Replaying events associated with missing or incorrect logins will result in replay errors, but the replay operation will continue.

In order to replay a trace against an instance of SQL Server (the target), other than the computer originally traced (the source), either:

- Database IDs on the target must be the same as those on the source. This can be accomplished by creating from the source a backup of the master database, and any user databases referenced in the trace, and restoring them on the target.

- The default database for each login contained in the trace must be set (on the target) to the respective target database of the login. For example, the trace to be replayed contains activity for the login, Fred, in the database Fred_Db on the source. Therefore, on the target, the default database for the login, Fred, must be set to the database that matches Fred_Db (even if the database name is different). To set the default database of the login, use **sp_defaultdb** system stored procedure.

## Replay Options

Before replaying a captured trace, you can specify:

- **Server**

  The server is the name of the instance of SQL Server against which you want to replay the trace. The server must adhere to the replay requirements previously mentioned.

- **Output file name**

  The output file contains the result of replaying the trace for later viewing. If **Progress** is selected, then the output file can be also replayed at a later time. By default, SQL Profiler displays only the results of replaying the trace to the screen.

- **Replay Options**

  - **Replay events in the order they were traced. This option enables debugging.**

    Specify to replay events in the order they were traced. This allows you to use debugging methods such as stepping through each trace.

  - **Replay events using multiple threads. This option optimizes performance and disables debugging.**

    Specify to replay events using multiple threads. This optimizes performance, but debugging is disabled.

  - **Display replay results**

    Specify to display the results of the replay. This is the default option. If the trace you are replaying is very large, you may want to disable this to save disk space.

**Note** For best replay performance, it is recommended that you select to replay events using multiple threads, and do not select to display the replay results.

## Replay Considerations

SQL Profiler cannot replay traces:

- Captured from connections that connected to an instance of SQL Server using Windows Authentication Mode. For information about Windows Authentication Mode, see Authentication Modes.

- Containing replication and other transaction log activity.

- Containing operations that involve globally unique identifiers (GUID). For information about GUIDs, see Autonumbering and Identifier Columns.

- Containing operations on **text**, **ntext**, and **image** columns involving the **bcp** utility, BULK INSERT, READTEXT, WRITETEXT, and UPDATETEXT statements, and full-text operations.

- Containing session binding: **sp_getbindtoken** and **sp_bindsession** system stored procedures.

Additionally, SQL Profiler cannot replay SQL Trace .log files that contain SQL Server 6.5 server-side cursor statements (**sp_cursor**).

Unexpected results or replay errors can occur when replaying a trace containing the Sessions event classes (**Connect**, **Disconnect**, and **Existing Connection**) if the **Binary Data** data column is not also captured in the trace. The **Binary Data** data column, for the Session event classes, contains information required to set ANSI nulls, ANSI padding, cursor close on commit, concat null yields null, and quoted identifier session settings. For more information, see SET.

When replaying a trace containing concurrent connections, SQL Profiler creates a thread for each connection. Therefore, system performance of the computer replaying the trace can be affected if the trace contains many concurrent connections. To reduce the effect on system performance, filter the trace by specifying a value(s) for the **Application Name**, **SQL User Name**, or another data column captured in the trace, to focus the trace on only those events you need to monitor.

**Note** If you do not use the provided replay template (**SQLProfilerTSQL-Replay**), you may encounter difficulties capturing the current database context. For more information, see Troubleshooting SQL Profiler.

### To replay a trace table

## Single-Stepping Traces

Rather than replay all events in a trace to completion, SQL Profiler allows you to replay a trace in the following ways:

- A single event at a time

  By replaying a trace a single event at a time, you can examine the effects of each event after it has occurred. When trace replay is continued using single stepping, the next event is replayed, and then the trace is paused again.

- To a breakpoint

  By specifying one or more breakpoints in the trace, all events to the event marked with the breakpoint are replayed, as specified by the replay options without any user intervention, and then trace replay is paused. Trace replay can continue one event at a time, to the next breakpoint (if one exists), to a cursor, or to the end of the trace. Replaying a trace to a breakpoint is useful if you want to replay a trace without examining each event up to the breakpoint. For example, you have debugged your code and determined that all events up to a breakpoint execute as expected and do not need to be examined further.

- To a cursor

  By replaying a trace to a cursor (a highlighted event in the trace), all events to the highlighted event are replayed without any user intervention. However, if a breakpoint is marked in the trace between the cursor and the point in the trace where execution will next begin from, replay will stop at the breakpoint rather than continue to the cursor. Remove all breakpoints in the trace to replay the trace to the cursor. Similar to a breakpoint, replaying a trace to a cursor is useful if you want to replay a trace without examining each event up to the cursor.

Single stepping is useful for debugging the events captured in a trace. For example, you can create a trace monitoring the execution of all batches submitted. By replaying the events in the trace one at a time (single stepping), you can determine the effects of each batch as they occur, allowing you to debug your code. This is much more effective than placing large amounts of debug code between batches. Debug code generally creates more output that needs to be separated from the actual results generated, and that must be correctly removed when debugging is complete.

### To replay a single event at a time

## Deleting Traces

Deleting a trace permanently removes it. Delete only traces you no longer need. You must stop the trace before deleting it.

You can also choose to pause or stop the trace instead of deleting it. For more information, see Starting, Pausing, and Stopping Traces.

### To delete a trace

## Administering SQL Server

## SQL Profiler Performance Considerations

Here are some hints and tips that can help you use SQL Profiler more effectively.

### Running Too Many Traces

If an instance of Microsoft® SQL Server™ is running too slowly, SQL Profiler may have too many traces or a complex trace may be running. Stop any running traces to see whether performance improves. If stopping traces improves performance, then examine your traces carefully to make sure they are not tracing more information than necessary. Make sure you are not running too many complex traces simultaneously.

### Managing Large Trace Files

Large trace files can use significant amounts of disk space and can be slow and expensive to send across networks. Reduce the size of a saved trace file by removing unwanted event types and/or data columns and applying filters to limit the trace to a specific trace event criteria, such as **ClientProcessID**, **SPID**, or a set of values for **Application Name**. Save the trace file with the same name or a new name.

**WARNING** Saving a trace file with the same name overwrites the original file, causing any of the originally captured events or data columns that were removed or filtered to be lost.

# Administering SQL Server

## Monitoring with System Monitor

If you are running the Microsoft® Windows® 2000 operating system, use System Monitor (Performance Monitor in Microsoft Windows NT® 4.0) to measure the performance of Microsoft SQL Server™. You can view SQL Server objects and performance counters as well as the behavior of other objects, such as processors, memory, cache, threads, and processes. Each of these objects has an associated set of counters that measure device usage, queue lengths, delays, and other indicators of throughput and internal congestion.

System Monitor makes it possible to obtain up-to-the-second SQL Server activity and performance statistics. With this graphical tool, you can:

- View data simultaneously from any number of computers.
- View and change charts to reflect current activity, and show counter values that are updated at a user-defined frequency.
- Export data from charts, logs, alert logs, and reports to spreadsheet or database applications for further manipulation and printing.
- Add system alerts that list an event in the alert log and can notify you by reverting to the **Alert** view or issuing a network alert.
- Run a predefined application the first time or every time a counter value goes over or under a user-defined value.
- Create log files that contain data about various objects from different computers.
- Append to one file selected sections from other existing log files to form a long-term archive.

- View current-activity reports, or create reports from existing log files.
- Save individual chart, alert, log, or report settings, or the entire workspace setup for reuse when needed.

  **Note** You can use either the System Monitor or Performance Monitor to do these tasks.

For information about Windows NT 4.0 and Windows 2000 objects and counters, see the Windows NT 4.0 and Windows 2000 documentation.

# Monitoring Disk Activity

Microsoft® SQL Server™ uses Microsoft Windows NT® 4.0 or Windows® 2000 I/O calls to perform disk reads and writes. SQL Server manages when and how disk I/O is performed, but the Windows operating system performs the underlying I/O operations. The I/O subsystem includes the system bus, disk controller cards, disks, tape drives, CD-ROM drive, and many other I/O devices. Disk I/O is frequently the cause of bottlenecks in a system.

## Monitoring Disk I/O and Detecting Excess Paging

Two of the counters that can be monitored to determine disk activity include:

- **PhysicalDisk: % Disk Time**
- **PhysicalDisk: Avg. Disk Queue Length**

In System Monitor (Performance Monitor in Windows NT 4.0), the **PhysicalDisk: % Disk Time** counter monitors the percentage of time that the disk is busy with read/write activity. If the **PhysicalDisk: % Disk Time** counter is high (more than 90 percent), check the **Physical Disk: Current Disk Queue Length** counter to see how many system requests are waiting for disk access. The number of waiting I/O requests should be sustained at no more than 1.5 to 2 times the number of spindles making up the physical disk. Most disks have one spindle, although redundant array of inexpensive disks (RAID) devices usually have more. A hardware RAID device appears as one physical disk in System Monitor; RAID devices created through software appear as multiple instances.

Use the values of the **Current Disk Queue Length** and **% Disk Time** counters to detect bottlenecks within the disk subsystem. If **Current Disk Queue Length** and **% Disk Time** counter values are consistently high, consider:

- Using a faster disk drive.
- Moving some files to an additional disk or server.

- Adding additional disks to a RAID array, if one is being used.

If you are using a RAID device, the **% Disk Time** counter can indicate a value greater than 100 percent. If it does, use the **PhysicalDisk: Avg. Disk Queue Length** counter to determine how many system requests, on average, are waiting for disk access.

Applications and systems that are I/O-bound may keep the disk constantly active.

Monitor the **Memory: Page Faults/sec** counter to make sure that the disk activity is not caused by paging. In Windows NT 4.0 or Windows 2000, paging is caused by:

- Processes configured to use too much memory.
- File system activity.

If you have more than one logical partition on the same hard disk, use the **Logical Disk** counters instead of the **Physical Disk** counters. Looking at the logical disk counters will help you determine which files are heavily accessed. After you have found the disks with high levels of read/write activity, look at the read-specific and write-specific counters (for example, **Logical Disk: Disk Write Bytes/sec**) for the type of disk activity that is causing the load on each logical volume.

## Isolating Disk Activity Created by SQL Server

To determine the amount of I/O generated by SQL Server components, examine the following performance areas:

- Writing pages to disk
- Reading pages from disk

The number of page reads and writes that SQL Server performs can be monitored using the **SQL Server: Buffer Manager Page Reads/sec** and **Page Writes/sec** counters. If these values start to approach the capacity of the hardware I/O subsystem, try to reduce the values by tuning your application or database to reduce I/O operations (such as index coverage, better indexes, or normalization), increasing the I/O capacity of the hardware, or by adding memory.

## Monitoring CPU Usage

Monitor an instance of Microsoft® SQL Server™ periodically to determine if CPU usage rates are within normal ranges. A continually high CPU usage rate may indicate the need for a CPU upgrade or the addition of multiple processors. Alternately, a high CPU usage rate may indicate a poorly tuned or designed application. Optimizing the application can lower CPU utilization.

A good way to determine this is to use the **Processor:% Processor Time** counter in System Monitor (Performance Monitor in Microsoft Windows NT® 4.0). This counter monitors the amount of time the CPU spends processing a nonidle thread. A consistent state of 80 to 90 percent may indicate the need for a CPU upgrade or the addition of more processors. For multiprocessor systems, a separate instance of this counter should be monitored for each processor. This value represents the sum of processor time on a specific processor. To determine the average for all processors, use the **System: %Total Processor Time** counter instead.

Optionally, you can also monitor:

- **Processor: % Privileged Time**

  This counter corresponds to the percentage of time the processor is spending executing Windows NT 4.0 or Microsoft Windows® 2000 kernel commands such as processing SQL Server I/O requests. If this counter is consistently high when the **Physical Disk** counters is high, consider a faster or more efficient disk subsystem.

  **Note** Different disk controllers and drivers use different amounts of kernel processing time. Efficient controllers and drivers use less privileged time, leaving more processing time available for user applications, increasing overall throughput.

- **Processor: %User Time**

  This counter corresponds to the percentage of time the processor is spending executing user processes such as SQL Server.

- **System: Processor Queue Length**

  This counter corresponds to the number of threads waiting for processor time. A processor bottleneck develops when threads of a process require more processor cycles than are available. If more than a few processes are trying to utilize the processor's time, you might need to install a faster processor or an additional processor if you are using a multiprocessor system.

When you examine processor usage, consider the type of work the instance of SQL Server is performing. If SQL Server is performing a lot of calculations, such as queries involving aggregates or memory-bound queries that require no disk I/O, 100 percent of the processor's time can be used. If this causes the performance of other applications to suffer, try changing the workload (for example, by dedicating the computer to running the instance of SQL Server).

Values around 100 percent, where many client requests are executing, may indicate that processes are queuing up, waiting for processor time, and causing a bottleneck. Resolve the problem by adding more powerful processors.

## Monitoring Memory Usage

Monitor an instance of Microsoft® SQL Server™ periodically to confirm that memory usage is within typical ranges and that no processes, including SQL Server, are lacking or consuming too much memory.

To monitor for a low-memory condition, start with the following object counters:

- **Memory: Available Bytes**
- **Memory: Pages/sec**

The **Available Bytes** counter indicates how many bytes of memory are currently available for use by processes. The **Pages/sec** counter indicates the number of pages that either were retrieved from disk due to hard page faults or written to disk to free space in the working set due to page faults.

Low values for the **Available Bytes** counter can indicate that there is an overall shortage of memory on the computer or that an application is not releasing memory. A high rate for the **Pages/sec** counter could indicate excessive paging. Monitor the **Memory: Page Faults/sec** counter to make sure that the disk activity is not caused by paging.

A low rate of paging (and hence page faults) is typical, even if the computer has plenty of available memory. The Microsoft Windows NT® Virtual Memory Manager (VMM) steals pages from SQL Server and other processes as it trims the working-set sizes of those processes, causing page faults. To determine whether SQL Server rather than another process is causing excessive paging, monitor the **Process: Page Faults/sec** counter for the SQL Server process instance.

For more information about resolving excessive paging, see the Windows NT 4.0 or Microsoft Windows® 2000 documentation.

### Isolating Memory Used by SQL Server

By default, SQL Server changes its memory requirements dynamically, based on available system resources. If SQL Server needs more memory, it queries the operating system to determine whether free physical memory is available and uses the available memory. If SQL Server does not need the memory currently allocated to it, it releases the memory to the operating system. However, the option to dynamically use memory can be overridden using the **min server memory**, **max server memory**, and **set working set size** server configuration options. For more information, see Server Memory Options.

To monitor the amount of memory being used by SQL Server, examine the following performance counters:

- **Process: Working Set**

- **SQL Server: Buffer Manager: Buffer Cache Hit Ratio**

- **SQL Server: Buffer Manager: Total Pages**

- **SQL Server: Memory Manager: Total Server Memory (KB)**

The **Working Set** counter shows the amount of memory used by a process. If this number is consistently below the amount of memory SQL Server is configured to use (set by the **min server memory** and **max server memory** server options), SQL Server is configured for more memory than it needs. Otherwise, fix the size of the working set using the **set working set size server** option. For more information, see set working set size Option.

The **Buffer Cache Hit Ratio** counter is application specific; however, a rate of 90 percent or higher is desirable. Add more memory until the value is consistently greater than 90 percent, indicating that more than 90 percent of all requests for data were satisfied from the data cache.

If the **Total Server Memory (KB)** counter is consistently high compared to the amount of physical memory in the computer, it may indicate that more memory is required.

## Creating a SQL Server Database Alert

Using System Monitor (Performance Monitor in Microsoft® Windows NT® 4.0), you can create an alert to be raised when a threshold value for a System Monitor counter has been reached. In response to the alert, System Monitor can launch an application, such as a custom application written to handle the alert condition. For example, you could create an alert that is raised when the number of deadlocks exceeds a specific value.

**Note** Performance condition alerts are only available for the first 99 databases. Any databases created after the first 99 databases will not be included in the **sysperfinfo** system table, and using the **sp_add_alert** procedure will return an error.

Alerts also can be defined using SQL Server Enterprise Manager and SQL Server Agent. For more information, see Defining Alerts.

### To set up a SQL Server database alert using System Monitor

## Administering SQL Server

## System Monitor Scenarios

When monitoring Microsoft® SQL Server™ and the operating system to investigate performance-related issues, there are three main areas on which to concentrate your initial efforts:

- Disk activity.
- Processor utilization.
- Memory usage.

It can be useful to monitor Microsoft Windows NT® 4.0 or Microsoft Windows® 2000 and SQL Server counters at the same time to determine any correlation between the performance of SQL Server and Windows NT 4.0 or Windows 2000. For example, monitoring the Windows NT 4.0 or Windows 2000 disk I/O counters and the SQL Server Buffer Manager counters at the same time can show how the whole system is behaving.

Monitoring a computer using System Monitor (Performance Monitor in Windows NT 4.0) can slightly impact the performance of the computer. Therefore, either log the System Monitor data to another disk (or computer) so that it reduces the effect on the computer being monitored, or run System Monitor remotely. Monitor only the counters in which you are interested. Monitoring too many counters adds overhead to the monitoring process and will impact the computer being monitored, possibly affecting the results.

## Administering SQL Server

## Running System Monitor

System Monitor (Performance Monitor in Microsoft® Windows NT® 4.0) collects information from Microsoft SQL Server™ using remote procedure calls (RPC). Any user who has Microsoft Windows® 2000 permissions to run System Monitor can use it to monitor SQL Server.

**Note** When using either System Monitor or Performance Monitor, you cannot connect to an instance of SQL Server running on Microsoft Windows 98.

As with all performance monitoring tools, expect some performance overhead when monitoring SQL Server with System Monitor. The actual overhead in any specific instance will depend on the hardware platform, the number of counters, and the selected update interval. However, the integration of System Monitor with SQL Server is designed to minimize the impact.

**To start System Monitor**

## Administering SQL Server

## Creating Charts, Alerts, Logs, and Reports

System Monitor (Performance Monitor in Microsoft® Windows NT® 4.0) allows you to create charts, alerts, logs, and reports to monitor an instance of Microsoft SQL Server™.

### Charts

Charts can monitor the current performance of selected objects and counters (for example, the CPU usage or disk I/O). You can add to a chart various combinations of System Monitor objects and counters, as well as Windows NT 4.0 or Microsoft Windows® 2000 objects and counters.

Each chart represents a subset of information you want to monitor. For example, one chart can track memory usage statistics and a second chart can track disk I/O statistics.

Using a chart can be useful for:

- Investigating why a computer or application is slow or inefficient.
- Continually monitoring systems to find intermittent performance problems.
- Discovering why you need to increase capacity.
- Displaying a trend as a line chart.
- Displaying a comparison as a histogram chart.

Charts are useful for short-term, real-time monitoring of a local or remote computer (for example, when you want to monitor an event as it occurs).

### Alerts

Using alerts, System Monitor can track specific events and notify you of these events as requested. An alert log can monitor the current performance of selected counters and instances for objects in SQL Server. When a counter exceeds a given value, the log records the date and time of the event. An event can also generate a network alert. You can have a specified program run the first time or every time an event occurs. For example, an alert can send a network message to all system administrators that the instance of SQL Server is getting low on disk space.

## Logs

Logs allow you to record information on the current activity of selected objects and computers for later viewing and analysis. You can collect data from multiple systems into a single log file. For example, you can create various logs to accumulate information on the performance of selected objects on various computers for future analysis. You can save these selections under a file name and reuse them when you want to create another log of similar information for comparison.

Log files provide a wealth of information for troubleshooting or planning. Whereas charts, alerts, and reports on current activity provide instant feedback, log files enable you to track counters over a long period of time, thereby allowing you to examine information more thoroughly and to document system performance.

## Reports

Reports allow you to display constantly changing counter and instance values for selected objects. Values appear in columns for each instance. You can adjust report intervals, print snapshots, and export data. Use reports when you need to display the raw numbers.

For more information about charts, alerts, logs, and reports, or about Windows NT 4.0 or Windows 2000 objects and counters, see the Windows 4.0 or Windows 2000 documentation.

Administering SQL Server

## Using SQL Server Objects

Microsoft® SQL Server™ provides objects and counters that can be used by System Monitor (Performance Monitor in Microsoft Windows NT® 4.0) to monitor activity in computers running an instance of SQL Server. An object is any Windows NT 4.0, Microsoft Windows® 2000 or SQL Server resource, such as a SQL Server lock or Windows NT 4.0 or Windows 2000 process. Each object contains one or more counters that determine various aspects of the objects to monitor. For example, the **SQL Server Locks** object contains counters called **Number of Deadlocks/sec** or **Lock Timeouts/sec**.

Some objects have several instances if multiple resources of a given type exist on the computer. For example, the **Processor** object type will have multiple instances if a system has multiple processors. The **Databases** object type has one instance for each database on SQL Server. Some object types (for example, the **Memory Manager** object) have only one instance. If an object type has multiple instances, you can add counters to track statistics for each instance, or in many cases, all instances at once.

**Note** Performance condition alerts are only available for the first 99 databases. Any databases created after the first 99 databases will not be included in the **sysperfinfo** system table, and using the **sp_add_alert** procedure will return an error.

By adding or removing counters to the chart and saving the chart settings, you can specify the SQL Server objects and counters monitored when System Monitor is started.

| SQL Server object | Counter |
|---|---|
| SQL Server: Buffer Manager | **Buffer Cache Hit Ratio** |
| SQL Server: General Statistics | **User Connections** |
| SQL Server: Memory Manager | **Total Server Memory (KB)** |
| SQL Server: SQL Statistics | **SQL Compilations/sec** |
| SQL Server: Buffer Manager | **Page Reads/sec** |
| SQL Server: Buffer Manager | **Page Writes/sec** |

You can configure System Monitor to display statistics from any SQL Server counter. In addition, you can set a threshold value for any SQL Server counter and then generate an alert when a counter exceeds a threshold. For more information about setting an alert, see Creating a SQL Server Database Alert.

**Note** SQL Server statistics are displayed only when an instance of SQL Server is running. If you stop and restart an instance of SQL Server, the display of statistics is interrupted and then resumed automatically.

These are the SQL Server objects.

| SQL Server object | Description |
|---|---|
| SQL Server: Access Methods | Searches through and measures allocation of SQL Server database objects (for example, the number of index searches or number of pages that are allocated to indexes and data). |
| SQL Server: Backup Device | Provides information about backup devices used by backup and restore operations, such as the throughput of the backup device. |
| SQL Server: Buffer Manager | Provides information about the memory buffers used by SQL Server, such as free memory and **buffer cache hit ratio**. |
| SQL Server: Cache Manager | Provides information about the SQL Server cache used to store objects such as stored procedures, triggers, and query plans. |
| SQL Server: Databases | Provides information about a SQL Server database, such as the amount of free log space available or the number of active transactions in the database. There can be multiple instances of this object. |
| SQL Server: General Statistics | Provides information about general server-wide activity, such as the number of users who are connected to an instance of SQL Server. |
| SQL Server: Latches | Provides information about the latches on |

| | |
|---|---|
| | internal resources, such as database pages, that are used by SQL Server. |
| SQL Server: Locks | Provides information about the individual lock requests made by SQL Server, such as lock time-outs and deadlocks. There can be multiple instances of this object. |
| SQL Server: Memory Manager | Provides information about SQL Server memory usage, such as the total number of lock structures currently allocated. |
| SQL Server: Replication Agents | Provides information about the SQL Server replication agents currently running. |
| SQL Server: Replication Dist. | Measures the number of commands and transactions read from the distribution database and delivered to the Subscriber databases by the Distribution Agent. |
| SQL Server: Replication Logreader | Measures the number of commands and transactions read from the published databases and delivered to the distribution database by the Log Reader Agent. |
| SQL Server: Replication Merge | Provides information about SQL Server merge replication, such as errors generated or the number of replicated rows that are merged from the Subscriber to the Publisher. |
| SQL Server: Replication Snapshot | Provides information about SQL Server snapshot replication, such as the number of rows that are bulk copied from the publishing database. |
| SQL Server: SQL Statistics | Provides information about aspects of SQL queries, such as the number of batches of Transact-SQL statements received by SQL Server. |
| SQL Server: User Settable Object | Performs custom monitoring. Each counter can be a custom stored procedure or any Transact-SQL statement that |

| | |
|---|---|
| | returns a value to be monitored. |

## SQL Server: Access Methods Object

The **Access Methods** object in Microsoft® SQL Server™ provides counters to monitor how the logical pages within the database are accessed. Physical access to the database pages on disk is monitored using the **Buffer Manager** counters. Monitoring the methods used to access database pages can help you to determine whether query performance can be improved by adding or modifying indexes or by rewriting queries. The **Access Methods** counters can also be used to monitor the amount of data, indexes, and free space within the database, thereby indicating data volume and fragmentation (excessive fragmentation can impair performance).

These are the SQL Server **Access Methods** counters.

| SQL Server Access Methods counters | Description |
|---|---|
| **Extent Deallocations/sec** | Number of extents deallocated per second from database objects used for storing index or data records. |
| **Extents Allocated/sec** | Number of extents allocated per second to database objects used for storing index or data records. |
| **Forwarded Records/sec** | Number of records per second fetched through forwarded record pointers. |
| **FreeSpace Page Fetches/sec** | Number of pages returned per second by free space scans used to satisfy requests to insert record fragments. |
| **FreeSpace Scans/sec** | Number of scans per second that were initiated to search for free space in which to insert a new record fragment. |
| **Full Scans/sec** | Number of unrestricted full scans per second. These can be either base-table or full-index scans. |
| **Index Searches/sec** | Number of index searches per second. |

| | |
|---|---|
| | These are used to start range scans and single index record fetches and to reposition an index. |
| **Mixed Page Allocations/sec** | Number of pages allocated per second from mixed extents. These are used for storing the first eight pages that are allocated to an index or table. |
| **Page Deallocations/sec** | Number of pages deallocated per second from database objects used for storing index or data records. |
| **Page Splits/sec** | Number of page splits per second that occur as the result of overflowing index pages. |
| **Pages Allocated/sec** | Number of pages allocated per second to database objects used for storing index or data records. |
| **Probe Scans/sec** | Number of probe scans per second. These are used to find rows in an index or base table directly. |
| **Range Scans/sec** | Number of qualified range scans through indexes per second. |
| **Scan Point Revalidations/sec** | Number of times per second that the scan point had to be revalidated to continue the scan. |
| **Skipped Ghosted Records/sec** | Number of ghosted records per second skipped during scans. |
| **Table Lock Escalations/sec** | Number of times locks on a table were escalated. |
| **Workfiles Created/sec** | Number of work files created per second. |
| **Worktables Created/sec** | Number of work tables created per second. |
| **Worktables From Cache Ratio** | Percentage of work tables created where the initial pages were immediately available in the work table cache. |

## See Also

## SQL Server: Backup Device Object

The **Backup Device** object provides counters to monitor Microsoft® SQL Server™ backup devices used for backup and restore operations. Monitor backup devices when you want to determine the throughput or the progress and performance of your backup and restore operations on a per device basis. To monitor the throughput of the entire database backup or restore operation, use the **Backup/Restore Throughput/sec** counter of the SQL Server **Databases** object. For more information, see SQL Server: Databases Object.

This is the SQL Server **Backup Device** counter.

| SQL Server Backup Device counters | Description |
|---|---|
| **Device Throughput Bytes/sec** | Throughput of read and write operations (in bytes per second) for a backup device used when backing up or restoring databases. This counter exists only while the backup or restore operation is executing. |

### See Also

Backup Devices

## SQL Server: Buffer Manager Object

The **Buffer Manager** object provides counters to monitor how Microsoft® SQL Server™ uses:

- Memory to store data pages, internal data structures, and the procedure cache.

- Counters to monitor the physical I/O as SQL Server reads database pages from and writes database pages to disk.

Monitoring the memory and the counters used by SQL Server helps you determine:

- If bottlenecks exist due to a lack of available physical memory for storing frequently accessed data in cache, in which case SQL Server must retrieve the data from disk.

- If query performance can be improved by adding more memory or by making more memory available to the data cache or SQL Server internal structures.

- How often SQL Server needs to read data from disk. Compared to other operations, such as memory access, physical I/O consumes a lot of time. Minimizing physical I/O can improve query performance.

You can also monitor Microsoft Windows® 2000 Address Windowing Extensions (AWE) activity in SQL Server with the AWE counters. For example, you can make sure that SQL Server has enough memory allocated for AWE to run properly. For more information, see Using AWE Memory on Windows 2000 or awe enabled Option.

These are the SQL Server **Buffer Manager** counters.

| **SQL Server Buffer Manager** | |
|---|---|

| counters | Description |
|---|---|
| **AWE Lookup Maps/sec** | Number of times that a database page was requested by the server, found in the buffer pool, and mapped. When it is mapped, it is made a part of the server's virtual address space. |
| **AWE Stolen Maps/sec** | Number of times that a buffer was taken from the free list and mapped. |
| **AWE Unmap Call/Sec** | Number of calls to unmap buffers. When a buffer is unmapped, it is excluded from the virtual server address space. One or more buffers may be unmapped on each call. |
| **AWE Unmap Pages/Sec** | Number of SQL Server buffers that are unmapped. |
| **AWE Write Maps/Sec** | Number of times that it is necessary to map in a dirty buffer so it can be written to disk. |
| **Buffer Cache Hit Ratio** | Percentage of pages found in the buffer cache without having to read from disk. The ratio is the total number of cache hits divided by the total number of cache lookups since an instance of SQL Server was started. After a long period of time, the ratio moves very little. Because reading from the cache is much less expensive than reading from disk, you want this ratio to be high. Generally, you can increase the buffer cache hit ratio by increasing the amount of memory available to SQL Server. |
| **Checkpoint pages/sec** | Number of pages flushed to disk per second by a checkpoint or other operation that require all dirty pages to be flushed. |
| **Database pages** | Number of pages in the buffer pool with database content. |

| | |
|---|---|
| | |
| **Free list stall/sec** | Number of requests that had to wait for a free page. |
| **Free pages** | Total number of pages on all free lists. |
| **Lazy Writes/sec** | Number of buffers written per second by the buffer manager's lazy writer. The lazy writer is a system process that flushes out batches of dirty, aged buffers (buffers that contain changes that must be written back to disk before the buffer can be reused for a different page) and make them available to user processes. The lazy writer eliminates the need to perform frequent checkpoints in order to create available buffers. |
| **Page life expectancy** | Number of seconds a page will stay in the buffer pool without references. |
| **Page lookups/sec** | Number of requests to find a page in the buffer pool. |
| **Page Reads/sec** | Number of physical database page reads that are issued per second. This statistic displays the total number of physical page reads across all databases. Because physical I/O is expensive, you may be able to minimize the cost, either by using a larger data cache, intelligent indexes, and more efficient queries, or by changing the database design. |
| **Page Writes/sec** | Number of physical database page writes issued. |
| **Procedure cache pages** | Number of pages used to store compiled queries. |
| **Readahead Pages/sec** | Number of pages read in anticipation of use. |
| **Reserved Pages** | Number of buffer pool reserved pages. |
| **Stolen Pages** | Number of pages used for miscellaneous |

| | |
|---|---|
| | server purposes (including procedure cache). |
| **Target Pages** | Ideal number of pages in the buffer pool. |
| **Total Pages** | Number of pages in the buffer pool (includes database, free, and stolen pages). |

## See Also

Pages and Extents

Server Memory Options

SQL Server: Cache Manager Object

## SQL Server: Buffer Partition Object

The **Buffer Partition** object provides counters to monitor how Microsoft® SQL Server™ uses free pages.

| SQL Server Buffer Partition counters | Description |
|---|---|
| **Free list empty/sec** | Number of times a free page was requested and none was available. |
| **Free list requests/sec** | Number of times a free page was requested. |
| **Free pages** | Total number of pages on all free lists. |

## SQL Server: Cache Manager Object

The **Cache Manager** object provides counters to monitor how Microsoft® SQL Server™ uses memory to store objects such as stored procedures, ad hoc and prepared Transact-SQL statements, and triggers. Multiple instances of the **Cache Manager** object can be monitored at the same time, with each instance representing a different type of plan to monitor.

| Cache Manager instance | Description |
|---|---|
| **Ad hoc SQL Plans** | Query plans produced from an ad hoc Transact-SQL query, including auto-parameterized queries. SQL Server caches the plans for ad hoc SQL statements for later reuse if the identical Transact-SQL statement is later executed. |
| **Misc. Normalized Trees** | Normalized trees for views, rules, computed columns, and check constraints. |
| **Prepared SQL Plans** | Query plans that correspond to Transact-SQL statements prepared using **sp_prepare**, **sp_cursorprepare**, or auto-parameterization. User-parameterized queries (even if not explicitly prepared) are also monitored as Prepared SQL Plans. |
| **Procedure Plans** | Query plans generated by creating a stored procedure. |
| **Replication Procedure Plans** | Query plans of a replication system stored procedure. |
| **Trigger Plans** | Query plans generated by creating a trigger. |

These are the SQL Server **Cache Manager** counters.

| SQL Server Cache Manager counters | Description |
|---|---|
| **Cache Hit Ratio** | Ratio between cache hits and lookups. |
| | |

| | |
|---|---|
| **Cache Object Counts** | Number of cache objects in the cache. |
| **Cache Pages** | Number of 8-kilobyte (KB) pages used by cache objects. |
| **Cache Use Counts/sec** | Times each type of cache object has been used. |

For more information about caching query plans, see Execution Plan Caching and Reuse.

## See Also

Server Memory Options

SQL Server: Buffer Manager Object

## SQL Server: Databases Object

The **Databases** object in Microsoft® SQL Server™ provides counters to monitor bulk copy operations, backup and restore throughput, and transaction log activities. Monitor transactions and the transaction log to determine how much user activity is occurring in the database and how full the transaction log is becoming. The amount of user activity can determine the performance of the database and affect log size, locking, and replication. Monitoring low-level log activity to gauge user activity and resource usage can help you to identify performance bottlenecks.

Multiple instances of the **Databases** object, each representing a single database, can be monitored at the same time.

These are the SQL Server **Databases** counters.

| SQL Server Databases counters | Description |
|---|---|
| **Active Transactions** | Number of active transactions for the database. |
| **Backup/Restore Throughput/sec** | Read/write throughput for backup and restore operations of a database per second. For example, you can measure how the performance of the database backup operation changes when more backup devices are used in parallel or when faster devices are used. Throughput of a database backup or restore operation allows you to determine the progress and performance of your backup and restore operations. |
| **Bulk Copy Rows/sec** | Number of rows bulk copied per second. |
| **Bulk Copy Throughput/sec** | Amount of data bulk copied (in kilobytes) per second. |

| | |
|---|---|
| **Data File(s) Size (KB)** | Cumulative size (in kilobytes) of all the data files in the database including any automatic growth. Monitoring this counter is useful, for example, for determining the correct size of **tempdb**. |
| **DBCC Logical Scan Bytes/sec** | Number of logical read scan bytes per second for database consistency checker (DBCC) statements. |
| **Log Bytes Flushed/sec** | Total number of log bytes flushed. |
| **Log Cache Hit Ratio** | Percentage of log cache reads satisfied from the log cache. |
| **Log Cache Reads/sec** | Reads performed per second through the log manager cache. |
| **Log File(s) Size (KB)** | Cumulative size (in kilobytes) of all the transaction log files in the database. |
| **Log File(s) Used Size (KB)** | The cumulative used size of all the log files in the database. |
| **Log Flush Wait Time** | Total wait time (in milliseconds) to flush the log. |
| **Log Flush Waits/sec** | Number of commits per second waiting for the log flush. |
| **Log Flushes/sec** | Number of log flushes per second. |
| **Log Growths** | Total number of times the transaction log for the database has been expanded. |
| **Log Shrinks** | Total number of times the transaction log for the database has been shrunk. |
| **Log Truncations** | Total number of times the transaction log for the database has been truncated. |
| **Percent Log Used** | Percentage of space in the log that is in use. |
| **Repl. Pending Xacts** | Number of transactions in the transaction log of the publication database marked for replication, but not yet delivered to the distribution database. |
| | |

| | |
|---|---|
| **Repl. Trans. Rate** | Number of transactions per second read out of the transaction log of the publication database and delivered to the distribution database. |
| **Shrink Data Movement Bytes/sec** | Amount of data being moved per second by autoshrink operations, or DBCC SHRINKDATABASE or DBCC SHRINKFILE statements. |
| **Transactions/sec** | Number of transactions started for the database per second. |

## See Also

Transaction Logs

Transactions

## SQL Server: General Statistics Object

The **General Statistics** object in Microsoft® SQL Server™ provides counters to monitor general server-wide activity, such as the number of current connections and the number of users connecting and disconnecting per second from computers running an instance of SQL Server. This can be useful when you are working on large online transaction processing (OLTP) type systems where there are many clients connecting and disconnecting from an instance of SQL Server.

These are the SQL Server **General Statistics** counters.

| SQL Server General Statistics counters | Description |
|---|---|
| **Logins/sec** | Total number of logins started per second. |
| **Logouts/sec** | Total number of logout operations started per second. |
| **User Connections** | Number of user connections. Because each user connection consumes some memory, configuring overly high numbers of user connections could affect throughput. Set user connections to the maximum expected number of concurrent users. |

## SQL Server: Latches Object

The **Latches** object in Microsoft® SQL Server™ provides counters to monitor internal SQL Server resource locks called latches. Monitoring the latches to determine user activity and resource usage can help you to identify performance bottlenecks.

These are the SQL Server **Latches** counters.

| SQL Server Latches counters | Description |
|---|---|
| **Average Latch Wait Time (ms)** | Average latch wait time (in milliseconds) for latch requests that had to wait. |
| **Latch Waits/sec** | Number of latch requests that could not be granted immediately. |
| **Total Latch Wait Time (ms)** | Total latch wait time (in milliseconds) for latch requests in the last second. |

### See Also

Latching

## SQL Server: Locks Object

The **Locks** object in Microsoft® SQL Server™ provides information about SQL Server locks on individual resource types. Locks are held on SQL Server resources, such as rows read or modified during a transaction, to prevent concurrent use of resources by multiple transactions. For example, if an exclusive (X) lock is held on a row within a table by a transaction, no other transaction can modify that row until the lock is released. Minimizing locks increases concurrency, which can improve performance. Multiple instances of the **Locks** object can be monitored at the same time, with each instance representing a lock on a resource type.

SQL Server can lock these resources.

| Item | Description |
|---|---|
| Database | Database. |
| Extent | Contiguous group of eight data pages or index pages. |
| Key | Row lock within an index. |
| Page | 8-kilobyte (KB) data page or index page. |
| RID | Row ID. Used to lock a single row within a table. |
| Table | Entire table, including all data and indexes. |

These are the SQL Server **Locks** counters.

| SQL Server Locks counters | Description |
|---|---|
| **Average Wait Time (ms)** | Average amount of wait time (in milliseconds) for each lock request that resulted in a wait. |
| **Lock Requests/sec** | Number of new locks and lock conversions per second requested from the lock manager. |
| **Lock Timeouts/sec** | Number of lock requests per second that timed out, including internal requests for NOWAIT locks. |
| | |

| | |
|---|---|
| **Lock Wait Time (ms)** | Total wait time (in milliseconds) for locks in the last second. |
| **Lock Waits/sec** | Number of lock requests per second that required the caller to wait. |
| **Number of Deadlocks/sec** | Number of lock requests per second that resulted in a deadlock. |

## See Also

Understanding Locking in SQL Server

## SQL Server: Memory Manager Object

The **Memory Manager** object in Microsoft® SQL Server™ provides counters to monitor overall server memory usage. Monitoring overall server memory usage to gauge user activity and resource usage can help you to identify performance bottlenecks. Monitoring the memory used by an instance of SQL Server can help determine:

- If bottlenecks exist due to a lack of available physical memory for storing frequently accessed data in cache. If so, SQL Server must retrieve the data from disk.
- If query performance can be improved by adding more memory or by making more memory available to the data cache or SQL Server internal structures.

These are the SQL Server **Memory Manager** counters.

| SQL Server Memory Manager counters | Description |
|---|---|
| **Connection Memory (KB)** | Total amount of dynamic memory the server is using for maintaining connections. |
| **Granted Workspace Memory (KB)** | Total amount of memory currently granted to executing processes such as hash, sort, bulk copy, and index creation operations. |
| **Lock Blocks** | Current number of lock blocks in use on the server (refreshed periodically). A lock block represents an individual locked resource, such as a table, page, or row. |
| **Lock Blocks Allocated** | Current number of allocated lock blocks. At server startup, the number of |

| | |
|---|---|
| | allocated lock blocks plus the number of allocated lock owner blocks depends on the SQL Server **Locks** configuration option. If more lock blocks are needed, the value increases. |
| **Lock Memory (KB)** | Total amount of dynamic memory the server is using for locks. |
| **Lock Owner Blocks** | Number of lock owner blocks currently in use on the server (refreshed periodically). A lock owner block represents the ownership of a lock on an object by an individual thread. Therefore, if three threads each have a shared (S) lock on a page, there will be three lock owner blocks. |
| **Lock Owner Blocks Allocated** | Current number of allocated lock owner blocks. At server startup, the number of allocated lock owner blocks and the number of allocated lock blocks depend on the SQL Server **Locks** configuration option. If more lock owner blocks are needed, the value increases dynamically. |
| **Maximum Workspace Memory (KB)** | Maximum amount of memory available for executing processes such as hash, sort, bulk copy, and index creation operations. |
| **Memory Grants Outstanding** | Total number of processes per second that have successfully acquired a workspace memory grant. |
| **Memory Grants Pending** | Total number of processes per second waiting for a workspace memory grant. |
| **Optimizer Memory (KB)** | Total amount of dynamic memory the server is using for query optimization. |
| **SQL Cache Memory (KB)** | Total amount of dynamic memory the server is using for the dynamic SQL cache. |

| | |
|---|---|
| **Target Server Memory (KB)** | Total amount of dynamic memory the server can consume. |
| **Total Server Memory (KB)** | Total amount of dynamic memory (in kilobytes) that the server is using currently. |

## See Also

Understanding Locking in SQL Server

## SQL Server: Replication Agents Object

The **Replication Agents** object in Microsoft® SQL Server™ provides counters to monitor the SQL Server replication agents that are running currently. Monitoring the number of running Distribution and Merge Agents is useful to determine the number of Subscribers to which published databases are replicating. Multiple instances of the **Replication Agents** object can be monitored at the same time, with each instance representing a single replication agent (for example, Log Reader; Snapshot; Distribution; and Merge).

This is the SQL Server **Replication Agents** counter.

| SQL Server Replication Agents counters | Description |
|---|---|
| **Running** | Number of instances of a given replication agent running currently. |

## SQL Server: Replication Distribution Object

The **Replication Dist.** object in Microsoft® SQL Server™ provides counters to monitor the number of commands and transactions read from the distribution database and delivered to the subscriber databases by the SQL Server Distribution Agent.

These are the SQL Server **Replication Dist.** counters.

| SQL Server Replication Dist. counters | Description |
|---|---|
| **Dist:Delivered Cmds/sec** | Number of distribution commands delivered per second to the Subscriber. |
| **Dist:Delivered Trans/sec** | Number of distribution transactions delivered per second to the Subscriber. |
| **Dist:Delivery Latency** | Distribution latency (in milliseconds). The time it takes for transactions to be delivered to the Distributor and applied at the Subscriber. |

### See Also

Replication Distribution Agent Utility

## SQL Server: Replication Logreader Object

The **Replication Logreader** object in Microsoft® SQL Server™ provides counters to monitor the Log Reader Agent.

These are the SQL Server **Replication Logreader** counters.

| SQL Server Replication Logreader counters | Description |
|---|---|
| **Logreader:Delivered Cmds/sec** | Number of Log Reader Agent commands delivered per second to the Distributor. |
| **Logreader:Delivered Trans/sec** | Number of Log Reader Agent transactions delivered per second to the Distributor. |
| **Logreader:Delivery Latency** | Current amount of time, in milliseconds, elapsed from when transactions are applied at the Publisher to when they are delivered to the Distributor. |

### See Also

Replication Log Reader Agent Utility

## SQL Server: Replication Merge Object

The **Replication Merge** object in Microsoft® SQL Server™ provides counters to monitor each SQL Server merge execution that moves data changes up from a merge replication Subscriber to the Publisher, and down from the Publisher to the Subscriber.

These are the SQL Server **Replication Merge** counters.

| SQL Server Replication Merge counters | Description |
|---|---|
| **Conflicts/sec** | Number of conflicts per second that occurred in the Publisher/Subscriber upload and download. This value should always be zero. A nonzero value may require notifying the losing side, overriding the conflict, and so on. |
| **Downloaded Changes/sec** | Number of rows per second merged (inserted, updated, and deleted) from the Publisher to the Subscriber. |
| **Uploaded Changes/sec** | Number of rows per second merged (inserted, updated, and deleted) from the Subscriber to the Publisher. |

# SQL Server: Replication Snapshot Object

The **Replication Snapshot** object in Microsoft® SQL Server™ provides counters to monitor SQL Server snapshot replication.

These are the SQL Server **Replication Snapshot** counters.

| SQL Server Replication Snapshot counters | Description |
|---|---|
| **Snapshot:Delivered Cmds/sec** | Number of commands delivered per second to the Distributor. |
| **Snapshot:Delivered Trans/sec** | Number of transactions delivered per second to the Distributor. |

## See Also

Replication Snapshot Agent Utility

## SQL Server: SQL Statistics Object

The **SQL Statistics** object in Microsoft® SQL Server™ provides counters to monitor compilation and the type of requests sent to an instance of SQL Server. Monitoring the number of query compilations and recompilations and the number of batches received by an instance of SQL Server gives you an indication of how quickly SQL Server is processing user queries and how effectively the query optimizer is processing the queries.

Compilation is a significant part of a query's turnaround time. The objective of the cache is to reduce compilation by storing compiled queries for later reuse, thus eliminating the need to recompile queries when later executed. However, each unique query must be compiled at least once. Query recompilations can be caused by the following factors:

- Schema changes, including base schema changes such as adding columns or indexes to a table, or statistics schema changes such as inserting or deleting a significant number of rows from a table.

- Environment (SET statement) changes. Changes in session settings such as ANSI_PADDING or ANSI_NULLS can cause a query to be recompiled.

These are the **SQL Statistics** counters.

| **SQL Server SQL Statistics counters** | **Description** |
|---|---|
| **Auto-Param Attempts/sec** | Number of auto-parameterization attempts per second. Total should be the sum of the failed, safe, and unsafe auto-parameterizations. Auto-parameterization occurs when an instance of SQL Server attempts to reuse a cached plan for a previously executed query that is similar to, but not the same as, the current query. |

| | |
|---|---|
| | For more information, see Auto-parameterization. |
| **Batch Requests/sec** | Number of Transact-SQL command batches received per second. This statistic is affected by all constraints (such as I/O, number of users, cache size, complexity of requests, and so on). High batch requests mean good throughput. For more information, see Batch Processing. |
| **Failed Auto-Params/sec** | Number of failed auto-parameterization attempts per second. This should be small. |
| **Safe Auto-Params/sec** | Number of safe auto-parameterization attempts per second. |
| **SQL Compilations/sec** | Number of SQL compilations per second. Indicates the number of times the compile code path is entered. Includes compiles due to recompiles. After SQL Server user activity is stable, this value reaches a steady state. |
| **SQL Re-Compilations/sec** | Number of SQL recompiles per second. Counts the number of times recompiles are triggered. In general, you want the recompiles to be low. |
| **Unsafe Auto-Params/sec** | Number of unsafe auto-parameterization attempts per second. The table has characteristics that prevent the cached plan from being shared. These are designated as **unsafe**. |

## See Also

SQL Server: Cache Manager Object

## SQL Server: User Settable Object

The **User Settable** object in Microsoft® SQL Server™ allows you to create custom counter instances. Use custom counter instances to monitor aspects of the server not monitored by existing counters, such as components unique to your SQL Server database (for example, the number of customer orders logged or the product inventory).

The SQL Server **User Settable** object contains 10 instances of the query counter: **User counter 1** through **User counter 10**. These counters map to the SQL Server stored procedures **sp_user_counter1** through **sp_user_counter10**. As these stored procedures are executed by user applications, the values set by the stored procedures are displayed in System Monitor (Performance Monitor in Microsoft Windows NT® 4.0). A counter can monitor any single integer value (for example, a stored procedure that counts how many orders for a particular product have occurred in one day).

**Note** The user counter stored procedures are not polled automatically by System Monitor. They must be explicitly executed by a user application for the counter values to be updated. Use a trigger to automatically update the value of the counter. For example, to create a counter that monitors the number of rows in a table, create an INSERT and DELETE trigger on the table that executes:

SELECT COUNT(*) FROM *table*

Whenever the trigger is fired because of an INSERT or DELETE operation occurring on the table, the System Monitor counter is automatically updated.

This is the SQL Server **User Settable** counter.

| SQL Server User Settable counters | Description |
|---|---|
| **Query** | Defined by the user. |

To make use of the user counter stored procedures, execute them from your own application with a single integer parameter representing the new value for the counter. For example, to set **User counter 1** to the value 10, execute this

Transact-SQL statement:

```
EXECUTE sp_user_counter1 10
```

The user counter stored procedures can be called from anywhere other stored procedures can be called, such as your own stored procedures. For example, you can create the following stored procedure to count the number of connections and attempted connections made since an instance of SQL Server was started:

```
DROP PROC My_Proc
GO
CREATE PROC My_Proc
AS
   EXECUTE sp_user_counter1 @@CONNECTIONS
GO
```

The @@CONNECTIONS function returns the number of connections or attempted connections since an instance of SQL Server was started. This value is passed to the **sp_user_counter1** stored procedure as the parameter.

**IMPORTANT** Make the queries defined in the user counter stored procedures as simple as possible. Memory-intensive queries that perform substantial sort or hash operations or queries that perform large amounts of I/O are expensive to execute and can impact performance.

Administering SQL Server

## Monitoring with SQL Server Enterprise Manager

Use SQL Server Enterprise Manager to view the following information about current Microsoft® SQL Server™ activity:

- Current user connections and locks.
- Process number, status, locks, and commands that active users are running.
- Objects that are locked, and the kinds of locks that are present.

If you are a system administrator, you can view additional information about a selected process, send a message to a user who is connected currently to an instance of SQL Server, or terminate a selected process.

Use the current activity window in SQL Server Enterprise Manager to perform ad hoc monitoring of an instance of SQL Server. This allows you to determine, at a glance, the volume and general types of activity on the system, for example:

- Current blocked and blocking transactions.
- Currently connected users on an instance of SQL Server and the last statement executed.
- Locks that are in effect.

SQL Server activity can be monitored using the **sp_who** and **sp_lock** system stored procedures.

Here are icons and descriptions of the icons in the current activity window.

| Icon | Description |
|---|---|
| 🛢 | **Current Activity** gives process and lock information at a designated time. This information is a snapshot taken every time you open or refresh **Current Activity**. The |

| | |
|---|---|
| | time of the snapshot is displayed in the left pane. **Current Activity** provides information about the processes (connections) running, the locks a certain connection is holding or trying to acquire, and the current and waiting locks on databases and tables. |
| 🔍 | **Process Info** provides information about the current connections and activity in a system. A connection can be in three states:  running, sleeping, or background. The database context is also displayed. There are some server processes, which are started before the **master** database is brought online, that have no database context. |
| 🌍 | Running process that is waiting for a lock or user input. |
| 🌑 | Sleeping process that is waiting for a lock or user input. |
| ○ | Background process that wakes up periodically to execute work. SPID 2 (Lock Monitor), 3 (Lazy Writer) and 6 are background processes. |
| ❗👤 | Process (SPID) that is blocking one or more connections. |
| 🟥👤 | Process (SPID) that is blocked by another connection. |
| 🌐 | Process that is not blocking or being blocked. |
| 🔒 | Process that is not blocking or being blocked. |
| ▤ | Table lock. If an index is involved, the index name is listed in the index column. The resource locator of the locked part is displayed in the resource column. |
| 🛢 | Database lock. |

Here are descriptions of the process information in the Current Activity window.

| Item | Description |
|---|---|
| **Process ID** | SQL Server Process ID. |
| **Context ID** | Execution context ID used to uniquely identify the subthreads operating on behalf of a single process. |
| **User** | ID of the user who executed the command. |
| **Database** | Database currently being used by the process. |

| | |
|---|---|
| **Status** | Status of the process (for example, running, sleeping, runnable, and background). |
| **Open Transactions** | Number of open transactions for the process. |
| **Command** | Command currently being executed. |
| **Application** | Name of the application program being used by the process. |
| **Wait Time** | Current wait time in milliseconds. When the process is not waiting, the wait time is zero. |
| **Wait Type** | Indicates the name of the last or current wait type. |
| **Wait Resources** | Textual representation of a lock resource. |
| **CPU** | Cumulative CPU time for the process. The entry is updated only for processes performed on behalf of Transact-SQL statements executed when SET STATISTICS TIME ON has been activated in the same session. The CPU column is updated when a query has been executed with SET STATISTICS TIME ON. When zero is returned, SET STATISTICS TIME is OFF. |
| **Physical IO** | Cumulative disk reads and writes for the process. |
| **Memory Usage** | Number of pages in the procedure cache that are currently allocated to this process. A negative number indicates that the process is freeing memory allocated by another process. |
| **Login Time** | Time at which a client process logged into the server. For system processes, the time at which SQL Server startup occurred is displayed. |
| **Last Batch** | Last time a client process executed a remote stored procedure call or an EXECUTE statement. For system processes, the time at which SQL Server startup occurred is displayed. |
| **Host** | Name of the workstation. |
| **Network Library** | Column in which the client's network library is stored. Every client process comes in on a network connection. Network connections have a network library associated with them that allows them to make |

| | |
|---|---|
| | the connection. For more information, see Client and Server Net-Libraries. |
| **Network Address** | Assigned unique identifier for the network interface card on each user's workstation. When the user logs in, this identifier is inserted in the **Network Address** column. |
| **Blocked By** | Process ID (SPID) of a blocking process. |
| **Blocking** | Process ID (SPID) of processes that are blocked. |

Here are descriptions of the lock information in the Current Activity window.

| Item | Type | Description |
|---|---|---|
| **spid** | spid | Server process ID of the current user process. |
| **ecid** | ecid | Execution context ID. Represents the ID of a given thread associated with a specific spid. |
| **Lock type** | RID | Row identifier. Used to lock a single row individually within a table. |
| | KEY | Key; a row lock within an index. Used to protect key ranges in serializable transactions. |
| | PAG | Data or index page. |
| | EXT | Contiguous group of eight data pages or index pages. |
| | TAB | Entire table, including all data and indexes. |
| | DB | Database. |
| **Lock mode** | Shared (S) | Used for operations that do not change or update data (read-only operations), such as a SELECT statement. |
| | Update (U) | Used on resources that can be updated. Prevents a common form of deadlock that occurs when multiple sessions are reading, locking, and then potentially updating resources later. |
| | Exclusive (X) | Used for data modification operations, such as UPDATE, INSERT, or DELETE. Ensures |

| | | |
|---|---|---|
| | | that multiple updates cannot be made to the same resource at the same time. |
| | Intent | Used to establish a lock hierarchy. |
| | Schema | Used when an operation dependent on the schema of a table is executing. There are two types of schema locks: schema stability (Sch-S) and schema modification (Sch-M). |
| | Bulk update (BU) | Used when bulk copying data into a table and the **TABLOCK** hint is specified. |
| | RangeS_S | Shared range, shared resource lock; serializable range scan. |
| | RangeS_U | Shared range, update resource lock; serializable update scan. |
| | RangeI_N | Insert range, null resource lock. Used to test ranges before inserting a new key into an index. |
| | RangeX_X | Exclusive range, exclusive resource lock. Used when updating a key in a range. |
| **Status** | GRANT | Lock was obtained. |
| | WAIT | Lock is blocked by another process. |
| | CNVT | Lock is being converted to another lock. A lock being converted to another lock is held in one mode but is waiting to acquire a stronger lock mode (for example, update to exclusive). When diagnosing blocking issues, a CNVT can be considered similar to a WAIT. |
| **Owner** | Owner | The lock owner: xact (transaction), sess (session), or curs (cursor). |
| **Index** | Index | The index associated with the resource. If the index is clustered, you see the table name instead. |
| **Resource** | RID | Row identifier of the locked row within the table. The row is identified by a fileid:page:rid combination, where rid is the |

| | | |
|---|---|---|
| | | row identifier on the page. |
| | KEY | Hexadecimal number used internally by SQL Server. |
| | PAG | Page number. The page is identified by a fileid:page combination, where fileid is the **fileid** in the **sysfiles** table, and page is the logical page number within that file. |
| | EXT | First page number in the extent being locked. The page is identified by a fileid:page combination. |
| | TAB | No information is provided because the **ObjId** column already contains the object ID of the table. |
| | DB | No information is provided because the **dbid** column already contains the database ID of the database. |

## To view current server activity

⊞ Transact-SQL

Administering SQL Server

## Monitoring the Error Logs

Microsoft® SQL Server™ logs events (although only certain system events and user-defined events) to the SQL Server error log and the Microsoft Windows® application log. Use the information in the error log to troubleshoot problems related to SQL Server.

The Windows application logs provide an overall picture of events that occur on the Windows NT® 4.0 and Windows 2000 systems, as well as events in SQL Server and SQL Server Agent. Use Event Viewer to view the Windows application log and to filter the information. For example, you can filter events, such as information, warning, error, success audit, and failure audit.

Both logs automatically timestamp all recorded events.

### Comparing Error and Application Log Output

You can use both the SQL Server error log and the Windows application log to identify the cause of problems. For example, while monitoring the SQL Server error log, you may detect a certain set of messages for which you do not know the cause. By comparing the dates and times for events between these logs, you can narrow the list of probable causes.

## Administering SQL Server

## Viewing the SQL Server Error Log

View the Microsoft® SQL Server™ error log to ensure that processes have completed successfully (for example, backup and restore operations, batch commands, or other scripts and processes). This can be helpful to detect any current or potential problem areas, including automatic recovery messages (particularly if an instance of SQL Server has been stopped and restarted), kernel messages, and so on.

View the SQL Server error log by using SQL Server Enterprise Manager or any text editor. By default, the error log is located at Program Files\Microsoft SQL Server\Mssql\Log\Errorlog.

A new error log is created each time an instance of SQL Server is started, although the **sp_cycle_errorlog** system stored procedure can be used to cycle the error log files without having to restart the instance of SQL Server. Typically, SQL Server retains backups of the previous six logs and gives the most recent log backup the extension .1, the second most recent the extension .2, and so on. The current error log has no extension.

**To view the SQL Server error log**

Administering SQL Server

## Viewing the Windows Application Log

When Microsoft® SQL Server™ is configured to use the Microsoft Windows® application log, each SQL Server session writes new events to that log. Unlike the SQL Server error log, a new application log is not created each time you start an instance of SQL Server.

View and manage the Windows application log by using Event Viewer in Microsoft Windows NT® 4.0 or Windows 2000.

There are three logs that can be viewed with Event Viewer.

| Windows log type | Description |
|---|---|
| System log | Records events logged by the Windows NT 4.0 or Windows 2000 system components. For example, the failure of a driver or other system component to load during startup is recorded in the system log. |
| Security log | Records security events, such as failed login attempts. This helps track changes to the security system and identify possible breaches to security. For example, attempts to log on to the system may be recorded in the security log, depending on the audit settings in the User Manager.<br><br>Only members of the **sysadmin** fixed server role can view the security log. |
| Application log | Records events that are logged by applications. For example, a database application might record a file error in the application log. |

For more information about using Event Viewer, managing the application log, and understanding the information it presents, see the Windows NT 4.0 or Windows 2000 documentation.

### To view the Windows application log

Administering SQL Server

## Monitoring with Transact-SQL Statements

Microsoft® SQL Server™ provides several Transact-SQL statements and system stored procedures that allow ad hoc monitoring of an instance of SQL Server. Use these statements when you want to gather, at a glance, information about server performance and activity. For example:

- Current locks.
- Current user activity.
- Last command batch submitted by a user.
- Data space used by a table or database.
- Space used by a transaction log.
- Oldest active transaction (including replicated transactions) in the database.
- Performance information relating to I/O, memory, and network throughput.
- Procedure cache usage.
- General statistics about SQL Server activity and usage, such as the amount of time the CPU has been performing SQL Server operations or the amount of time SQL Server has spent performing I/O operations.

Most of this information can also be monitored using SQL Server Enterprise Manager, SQL-DMO, or System Monitor (Performance Monitor in Microsoft Windows NT® 4.0).

## To view the current locks

⊞ Transact-SQL

Administering SQL Server

## Monitoring with SNMP

Simple Network Management Protocol (SNMP) is an application protocol that offers network management services. Using SNMP, you can monitor an instance of Microsoft® SQL Server™ across different platforms (for example, Microsoft Windows NT® 4.0, Microsoft Windows® 98, and UNIX).

With SQL Server and the Microsoft SQL Server Management Information Base (MSSQL-MIB), you can use SNMP applications to:

- Monitor the status of SQL Server installations. SNMP can only be used to monitor the default instances of SQL Server.
- Monitor performance information.
- Access databases.
- View server and database configuration parameters.

Administering SQL Server

## SNMP Terminology

Simple Network Management Protocol (SNMP) terms are defined in the following table.

| Term | Description |
|---|---|
| SNMP | An application that monitors the status and performance of Microsoft® SQL Server™ installations, explores defined databases, and views server and database configuration parameters. |
| SNMP agent | SQL Server SNMP extension agent (Sqlsnmp.dll). Server software that extends the functionality of the SNMP service. The SNMP agent processes requests for data and data objects that reside on the local server. |

Administering SQL Server

## Enabling SNMP Support on SQL Server

Microsoft® SQL Server™ support of SNMP is enabled automatically if Microsoft Windows NT® 4.0 or Microsoft Windows® 2000 support of SNMP is installed on the computer when you run SQL Server Setup. If SNMP is not installed on the computer when you run the Setup program, SQL Server support of SNMP is not enabled.

Administering SQL Server

## Enabling SQL Server Support of SNMP on Windows 98

You can monitor remote connections to computers running Microsoft® Windows® 98 if your network uses Simple Network Management Protocol (SNMP).

The database controlled by an SNMP agent is known as SNMP Management Information Base (MIB). The values contained in an SNMP MIB can be shared with the SNMP MIB of another application.

Microsoft SQL Server™ Management Information Base (MSSQL-MIB), stored in the Mssql.mib file, and the SQL Server SNMP extension agent (Sqlsnmp.dll) are copied to the system by SQL Server Setup and are enabled if SNMP is running at the time of installation. SNMP can be activated or deactivated at any time by selecting the **Enable SNMP** check box in the **SQL Server Network Utility** dialog box.

For more information about SNMP, see the SNMP application documentation.

### To enable SQL Server support of SNMP on Windows 98

Administering SQL Server

# Enabling SQL Server MIB

The database controlled by a Simple Network Management Protocol (SNMP) agent is known as SNMP Management Information Base (MIB). The values contained in an SNMP MIB can be shared with the SNMP MIB of another application.

Microsoft® SQL Server™ Management Information Base (MSSQL-MIB), stored in the Mssql.mib file, and the SQL Server SNMP extension agent (Sqlsnmp.dll) are copied to the system by SQL Server Setup and are enabled if SNMP is running at the time of installation.

For more information about SNMP, see the SNMP application documentation.

## Copying the MSSQL-MIB to an SNMP Workstation

For SNMP applications to monitor the status of a SQL Server installation, a copy of MSSQL-MIB, stored in the Mssql.mib file, must be placed on the monitoring workstation and loaded into the SNMP application. MSSQL-MIB enables the SNMP application to access and monitor the SQL Server SNMP extension agent on an instance of SQL Server.

The Mssql.mib file is a text file that contains the definitions of objects available to SNMP workstations. The file consists of read-only variables for monitoring general performance counters, the status of SQL Server installation and databases, and limited discovery of configuration options and database files. Mssql.mib does not define any writable objects.

The following table describes the SNMP tables. These tables are SNMP tables, not SQL Server tables.

| SNMP table | Description |
|---|---|
| MssqlSrvTable | Contains a description of the SQL Server installation. Has a single row for each installation of SQL Server version 6.5 or earlier or multiple rows for each instance of SQL Server version 7.0 or SQL Server 2000 running on the server. |

| | |
|---|---|
| MssqlSrvInfoTable | Contains general information about the active SQL Server process, including performance counters. |
| MssqlSrvConfigParamTable | Lists SQL Server configuration parameters. |
| MssqlSrvDeviceTable | Contains an entry for each SQL Server database file defined on the system. |
| MssqlDbTable | Lists defined SQL Server databases. Contains a single row for each database. |
| MssqlDbOptionTable | Lists database options set for each SQL Server database. |

For more information, see the SNMP application documentation.

### To copy the SQL Server MSSQL-MIB to an SNMP workstation

Administering SQL Server

## Using the Web Assistant Wizard

You can use the Web Assistant Wizard to generate standard HTML files from Microsoft® SQL Server™ data. The Web Assistant Wizard generates HTML files by using Transact-SQL queries, stored procedures, and extended stored procedures. You can use the wizard to generate an HTML file on a one time basis or as a regularly scheduled SQL Server task. You also can update an HTML file using a trigger.

With the Web Assistant Wizard, you can:

- Schedule a task to update a Web page automatically. For example, you can update a price list when a new item is added or a price is changed, thereby maintaining a dynamic inventory and price list for customers and sales staff.
- Publish and distribute management reports, including the latest sales statistics, resource allocations, or other SQL Server data.
- Publish server reports with information about who is accessing the server currently, and about which locks are being held by which users.
- Publish information outside of SQL Server using extended stored procedures.
- Publish server jump lists using a table of favorite Web sites.
- Use the **sp_makewebtask** stored procedure to generate an HTML file. This system stored procedure can be called by a Transact-SQL program. You can also call system stored procedures to run or drop the task.

The Web Assistant Wizard runs from SQL Server Enterprise Manager.

### See Also

sp_dropwebtask

sp_makewebtask

sp_runwebtask

Administering SQL Server

# Configuring the Web Assistant Wizard

Before running the Web Assistant Wizard, you must:

- Set appropriate permissions.
- Choose the database to publish.
- Create queries.

## Setting Permissions

To run the Web Assistant Wizard, you must have:

- CREATE PROCEDURE permissions in the selected database.
- SELECT permissions on chosen columns.
- Permissions to create files in the account in an instance of Microsoft® SQL Server™.

## Choosing the Database to Publish

The Web Assistant Wizard works with databases created by SQL Server. Select the database to publish in the console tree of SQL Server Enterprise Manager. If the server does not appear in this list, run the Register Server Wizard.

## Creating Queries

You can run queries by:

- Using tables and columns you specify.
- Creating result sets from a stored procedure.

- Selecting data using Transact-SQL statements.

The Web Assistant Wizard requires that each job be named, and a default name is supplied. For jobs that will run at a later time or for jobs that run on a continuous basis, choose a name that will help you remember the focus of this query.

Administering SQL Server

## Receiving Query Results with the Web Assistant Wizard

To receive query results, use your own HTML template file. A template file is any HTML file with the marker <%insert_data_here%> to indicate where the query results should be inserted. If you use an alternate character set, you must insert the necessary meta tag information into the Web page manually, specifying the character chosen.