

SQL Aggregate Functions

Using the [SQL aggregate functions](#), you can determine various statistics on sets of values. You can use these functions in a query and aggregate expressions in the [SQL](#) property of a [QueryDef](#) object or when creating a [Recordset](#) object based on an SQL query.

[Avg Function](#)

[Count Function](#)

[First, Last Functions](#)

[Min, Max Functions](#)

[StDev, StDevP Functions](#)

[Sum Function](#)

[Var, VarP Functions](#)

SQL Expressions

An [SQL expression](#) is a [string](#) that makes up all or part of an [SQL statement](#). For example, the [FindFirst](#) method on a [Recordset](#) object uses an SQL expression consisting of the selection criteria found in an SQL [WHERE clause](#).

The [Microsoft Jet database engine](#) uses the Microsoft® Visual Basic® for Applications (or VBA) expression service to perform simple arithmetic and function evaluation. All of the operators used in Microsoft Jet SQL expressions (except [Between](#), [In](#), and [Like](#)) are defined by the VBA expression service. In addition, the VBA expression service offers over 100 VBA functions that you can use in SQL expressions. For example, you can use these VBA functions to compose an SQL query in the Microsoft Access query Design view, and you can also use these functions in an SQL query in the [DAO OpenRecordset](#) method in Microsoft Visual C++®, Microsoft Visual Basic, and Microsoft Excel code.

SQL Reserved Words

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)

The following list includes all words reserved by the [Microsoft Jet database engine](#) for use in [SQL statements](#). The words in the list that are not in all uppercase letters are also reserved by other applications. Consequently, the individual Help topics for these words provide general descriptions that do not focus on [SQL](#) usage.

Note Words followed by an asterisk (*) are reserved but currently have no meaning in the context of a Microsoft® Jet SQL statement (for example, **Level** and **TableID**). Words that are not underlined do not have linked explanations.

A

ABSOLUTE

[ADD](#)

ADMINDB

[ALL](#)

[Alphanumeric](#) — See TEXT

[ALTER](#)

[ALTER TABLE](#)

[And](#)

[AS](#)

[ANY](#)

ARE

[AS](#)

[ASC](#)

ASSERTION

AUTHORIZATION

[AUTOINCREMENT](#) — See COUNTER

[Avg](#)

B-C

[BEGIN](#)

[Between](#)

[BINARY](#)

[BIT](#)

BIT_LENGTH

[BOOLEAN](#) — See BIT

BOTH

[BY](#)

COLLATION

[COLUMN](#)

[COMMIT](#)

[COMP, COMPRESSION](#)

CONNECT

CONNECTION

[CONSTRAINT, CONSTRAINTS](#)

[CONTAINER](#)

[BYTE](#)
[CASCADE](#)
CATALOG
[CHAR, CHARACTER](#) — See
TEXT
[CHAR_LENGTH](#)
[CHARACTER_LENGTH](#)
CHECK
CLOSE
CLUSTERED
COALESCE
COLLATE

D

[DATABASE](#)
[DATE](#) — See DATETIME
[DATETIME](#)
DAY
[DEC, DECIMAL](#)
DECLARE
[DELETE](#)
[DESC](#)

E-H

Eqv
EXCLUSIVECONNECT
EXEC, EXECUTE
[EXISTS](#)
EXTRACT
FALSE
FETCH

CONTAINS
CONVERT
[Count](#)
[COUNTER](#)

[CREATE](#)
[CURRENCY](#)
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURSOR

[DISALLOW](#)
DISCONNECT
[DISTINCT](#)
[DISTINCTROW](#)
DOMAIN
[DOUBLE](#)
[DROP](#)

[FOREIGN](#)
[FROM](#)
[FROM Clause](#)
[GENERAL](#) — See LONGBINARY
[GRANT](#)
[GROUP](#)
[GUID](#)

[FIRST](#)
[FLOAT, FLOAT8](#) — See DOUBLE
[FLOAT4](#) — See SINGLE

[HAVING](#)
HOUR

I

[IDENTITY](#)
[IEEEDOUBLE](#) — See DOUBLE
[IEEESINGLE](#) — See SINGLE
[IGNORE](#)
[IMAGE](#)

Imp

In

[IN](#)

[INDEX](#)

INDEXCREATEDB

[INNER](#)

INPUT

INSENSITIVE

[INSERT](#)

[INSERT INTO](#)

[INT, INTEGER, INTEGER4](#) — See
LONG

[INTEGER1](#) — See BYTE

[INTEGER2](#) — See SHORT

[INTERVAL](#)

[INTO](#)

Is

ISOLATION

J-M

[JOIN](#)

[KEY](#)

LANGUAGE

[LAST](#)

[LEFT](#)

Level*

Like

[LOGICAL, LOGICAL1](#) — See BIT

[LONG](#)

[LONGBINARY](#)

[LONGCHAR](#)

[LONGTEXT](#)

LOWER

MATCH

Max

[MEMO](#) — See LONGTEXT

Min

MINUTE

Mod

[MONEY](#) — See CURRENCY

MONTH

N-P

[NATIONAL](#)

[NCHAR](#)

NONCLUSTERED

Not

[NTEXT](#)

[NULL](#)

[NUMBER](#) — See DOUBLE

[NUMERIC](#) — See DECIMAL

[NVARCHAR](#)

OCTET_LENGTH

[OLEOBJECT](#) — See

LONGBINARY

[ON](#)

OPEN

[OPTION](#)

Or

[ORDER](#)

Outer*

OUTPUT

[OWNERACCESS](#)

PAD

[PARAMETERS](#)

PARTIAL

PASSWORD

[PERCENT](#)

[PIVOT](#)

POSITION

PRECISION

PREPARE

[PRIMARY](#)

PRIVILEGES

[PROC, PROCEDURE](#)

PUBLIC

Q-S

[REAL](#) — See SINGLE

[REFERENCES](#)

RESTRICT

[REVOKE](#)

[RIGHT](#)

[ROLLBACK](#)

[SCHEMA](#)

SECOND

[SELECT](#)

[SELECTSCHEMA](#)

[SELECTSECURITY](#)

SMALLDATETIME

[SMALLINT](#) — See SHORT

SMALLMONEY

[SOME](#)

SPACE

SQL

SQLCODE, SQLERROR, SQLSTATE

[StDev](#)

[StDevP](#)

[STRING](#) — See TEXT

SUBSTRING

[SET](#)
[SHORT](#)
[SINGLE](#)
SIZE

T-Z

[TABLE](#)
TableID*
[TEMPORARY](#)
[TEXT](#)
[TIME](#) — See DATETIME
[TIMESTAMP](#)
TIMEZONE_HOUR
TIMEZONE_MINUTE
[TINYINT](#)
[TO](#)
[TOP](#)
TRAILING
[TRANSACTION](#)
[TRANSFORM](#)
TRANSLATE
TRANSLATION
TRIM
TRUE
[UNION](#)
[UNIQUE](#)
[UNIQUEIDENTIFIER](#)
UNKNOWN
[UPDATE](#)
[UPDATEIDENTITY](#)

[Sum](#)
SYSNAME
SYSTEM_USER

[UPDATEOWNER](#)
[UPDATESECURITY](#)
UPPER
USAGE
[USER](#)
USING
[VALUE](#)
[VALUES](#)
Var
[VARBINARY](#) — See BINARY
[VARCHAR](#) — See TEXT
VarP
[VARYING](#)
[VIEW](#)
WHEN
WHENEVER
[WHERE](#)
[WITH](#)
[WORK](#)
Xor
YEAR
[YESNO](#) — See BIT
ZONE

SQL Data Types

The [Microsoft Jet database engine SQL](#) data types consist of 13 primary data types defined by the Microsoft® Jet database engine and several valid synonyms recognized for these data types.

The following table lists the primary data types. The synonyms are identified in [Microsoft Jet Database Engine SQL Reserved Words](#).

Data type	Storage size	Description
BINARY	1 byte per character	Any type of data may be stored in a field of this type. No translation of the data (for example, to text) is made. How the data is input in a binary field dictates how it will appear as output.
BIT	1 byte	Yes and No values and fields that contain only one of two values.
TINYINT	1 byte	An integer value between 0 and 255.
MONEY	8 bytes	A scaled integer between – 922,337,203,685,477.5808 and 922,337,203,685,477.5807.
DATETIME (See DOUBLE)	8 bytes	A date or time value between the years 100 and 9999.
UNIQUEIDENTIFIER	128 bits	A unique identification number used with remote procedure calls.
REAL	4 bytes	A single-precision floating-point value with a range of – 3.402823E38 to – 1.401298E-45 for negative values, 1.401298E-45 to 3.402823E38 for positive values, and 0.
FLOAT	8 bytes	A double-precision floating-point value with a range of – 1.79769313486232E308 to – 4.94065645841247E-324 for

		negative values, 4.94065645841247E-324 to 1.79769313486232E308 for positive values, and 0.
SMALLINT	2 bytes	A short integer between – 32,768 and 32,767. (See Notes)
INTEGER	4 bytes	A long integer between – 2,147,483,648 and 2,147,483,647. (See Notes)
DECIMAL	17 bytes	An exact numeric data type that holds values from 1028 - 1 through - 1028 - 1. You can define both precision (1 - 28) and scale (0 - defined precision). The default precision and scale are 18 and 0, respectively.
TEXT	2 bytes per character (See Notes)	Zero to a maximum of 2.14 gigabytes.
IMAGE	As required	Zero to a maximum of 2.14 gigabytes. Used for OLE objects.
CHARACTER	2 bytes per character (See Notes)	Zero to 255 characters.

Notes

Both the [seed](#) and the [increment](#) can be modified using an [ALTER TABLE statement](#). New rows inserted into the table will have values, based on the new seed and increment values, that are automatically generated for the column. If the new seed and increment can yield values that match values generated based on the preceding seed and increment, duplicates will be generated. If the column is a [primary key](#), then inserting new rows may result in errors when duplicate values are generated.

To find the last value that was used for an auto-increment column, you can use the following statement: `SELECT @@IDENTITY`. You cannot specify a table name. The value returned is from the last table, containing an auto-increment column, that was updated.

Characters in fields defined as either [TEXT](#) (also known as MEMO) or [CHAR](#) (also known as TEXT(n) with a specific length) are stored in [the Unicode representation format](#). Unicode characters uniformly require two bytes to store each character. For existing Microsoft Jet databases that contain predominately character data, this could mean that the database file would nearly double in size when converted to the Microsoft Jet 4.0 format. Yet Unicode representation of many character sets, those formerly denoted as SBCS (Single-Byte Character Sets) can easily be compressed to a single byte. For details, see [CREATE TABLE](#). If you define a CHAR column with the COMPRESSION attribute, data will automatically be compressed as it is stored and uncompressed when it is retrieved from the column.

See Also

[Equivalent ANSI SQL Data Types](#) [Microsoft Jet Database Engine SQL Reserved Words](#)

ODBC Scalar Functions

Microsoft® Jet SQL supports the use of the [ODBC](#) defined syntax for scalar functions. For example, the query:

```
SELECT DAILYCLOSE, DAILYCHANGE FROM DAILYQUOTE  
WHERE {fn ABS(DAILYCHANGE)} > 5
```

Would return all rows where the absolute value of the change in the price of a stock was greater than five.

A subset of the ODBC defined scalar functions is supported. The following table lists the functions that are supported.

For a description of the arguments and a complete explanation of the escape syntax for including functions in a SQL statement, see the ODBC documentation.

String Functions

ASCII	LENGTH	RTRIM
CHAR	LOCATE	SPACE
CONCAT	LTRIM	SUBSTRING
LCASE	RIGHT	UCASE
LEFT		

Numeric Functions

ABS	FLOOR	SIN
ATAN	LOG	SQRT
CEILING	POWER	TAN
COS	RAND	MOD
EXP	SIGN	

Time & Date Functions

CURDATE	DAYOFYEAR	MONTH
CURTIME	YEAR	WEEK
NOW	HOUR	QUARTER
DAYOFMONTH	MINUTE	MONTHNAME
DAYOFWEEK	SECOND	DAYNAME

Data Type Conversion

CONVERT String literals can be converted to the following data types: SQL_FLOAT, SQL_DOUBLE, SQL_NUMERIC, SQL_INTEGER, SQL_REAL, SQL_SMALLINT, SQL_VARCHAR and SQL_DATETIME.

See Also

[Configuring the Microsoft Jet Database Engine for ODBC Access](#)

Comparison of Microsoft Jet SQL and ANSI SQL

[Microsoft Jet database engine SQL](#) is generally [ANSI-89](#) Level 1 compliant. However, certain ANSI SQL features are not implemented in Microsoft® Jet SQL. With the release of Microsoft Jet version 4.X, the Microsoft OLE DB Provider for Jet exposes more [ANSI-92 SQL](#) syntax. Conversely, Microsoft Jet SQL includes reserved words and features not supported in ANSI SQL.

Major Differences

Microsoft Jet SQL and ANSI SQL each have different reserved words and data types. For more information, see [Microsoft Jet Database Engine SQL Reserved Words](#) and [Equivalent ANSI SQL Data Types](#). Using the Microsoft OLE DB Provider for Jet with Jet 4.X, there are additional reserved words.

Different rules apply to the [Between...And](#) construct, which has the following syntax:

expr1 [NOT] **Between** *value1* **And** *value2*

In Microsoft Jet SQL, *value1* can be greater than *value2*; in ANSI SQL, *value1* must be equal to or less than *value2*.

Microsoft Jet SQL supports both ANSI SQL wildcard characters and Microsoft Jet-specific [wildcard characters](#) to use with the [Like](#) operator. The use of the ANSI and Microsoft Jet wildcard characters is mutually exclusive. You must use one set or the other and cannot mix them. The ANSI SQL wildcards are only available when using Jet 4.X and the Microsoft OLE DB Provider for Jet. If you try to use the ANSI SQL wildcards through Microsoft Access or DAO, then they will be interpreted as literals. The opposite is true when using the Microsoft OLE DB Provider for Jet and Jet 4.X.

Matching character	Microsoft Jet SQL	ANSI SQL
Any single character	?	_ (underscore)
Zero or more characters	*	%

Microsoft Jet SQL is generally less restrictive. For example, it permits grouping

and ordering on expressions.

Microsoft Jet SQL supports more powerful expressions.

Enhanced Features of Microsoft Jet SQL

Microsoft Jet SQL provides the following enhanced features:

- The [TRANSFORM](#) statement, which provides support for [crosstab queries](#).

Additional [aggregate functions](#), such as **StDev** and **VarP**.

- The [PARAMETERS](#) declaration for defining [parameter queries](#).

ANSI SQL Features Not Supported in Microsoft Jet SQL

Microsoft Jet SQL does not support the following ANSI SQL features:

DISTINCT aggregate function references. For example, Microsoft Jet SQL does not allow SUM(DISTINCT *columnname*).

The LIMIT TO *nn* ROWS clause used to limit the number of rows returned by a query. You can use only the [WHERE clause](#) to limit the scope of a query.

See Also

[Equivalent ANSI SQL Data Types](#)

[Like Operator \(Microsoft Jet SQL\)](#)

[Microsoft Jet Database Engine SQL Data Types](#)

[Microsoft Jet Database Engine SQL Reserved Words](#)

[SQL Aggregate Functions \(SQL\)](#)

[Using Wildcard Characters in String Comparisons](#)

Equivalent ANSI SQL Data Types

The following table lists ANSI [SQL](#) data types, their equivalent [Microsoft Jet database engine](#) SQL data types, and their valid synonyms. It also lists the equivalent Microsoft® SQL Server™ data types.

ANSI SQL data type	Microsoft Jet SQL data type	Synonym	Microsoft SQL Server data type
BIT, BIT VARYING	BINARY (See Notes)	VARBINARY, BINARY VARYING, BIT VARYING	BINARY, VARBINARY
Not supported	BIT (See Notes)	BOOLEAN, LOGICAL, LOGICAL1, YESNO	BIT
Not supported	TINYINT	INTEGER1, BYTE	TINYINT
Not supported	COUNTER (See Notes)	AUTOINCREMENT (See Notes)	
Not supported	MONEY	CURRENCY	MONEY
DATE, TIME, TIMESTAMP	DATETIME	DATE, TIME (See Notes)	DATETIME
Not supported	UNIQUEIDENTIFIER	GUID	UNIQUEIDENTIFIER
DECIMAL	DECIMAL	NUMERIC, DEC	DECIMAL
REAL	REAL	SINGLE, FLOAT4, IEEE SINGLE	REAL
DOUBLE PRECISION, FLOAT	FLOAT	DOUBLE, FLOAT8, IEEE DOUBLE, NUMBER (See Notes)	FLOAT
SMALLINT	SMALLINT	SHORT, INTEGER2	SMALLINT
INTEGER	INTEGER	LONG, INT, INTEGER4	INTEGER
INTERVAL	Not supported		Not supported
Not supported	IMAGE	LONGBINARY, GENERAL, OLEOBJECT	IMAGE

Not supported	TEXT (See Notes)	LONGTEXT, LONGCHAR, MEMO, NOTE, NTEXT (See Notes)	TEXT
CHARACTER, CHAR (See Notes)	CHARACTER	TEXT(n), ALPHANUMERIC, CHARACTER, STRING, VARCHAR, CHARACTER	CHAR, VARCHAR
VARYING, NATIONAL CHARACTER, NATIONAL CHARACTER VARYING		VARYING, NCHAR, NATIONAL CHARACTER, NATIONAL CHAR, NATIONAL CHARACTER VARYING, NATIONAL CHAR VARYING (See Notes)	NCHAR, NVARCHAR

Notes

The ANSI SQL BIT data type does not correspond to the Microsoft Jet SQL BIT data type. It corresponds to the BINARY data type instead. There is no ANSI SQL equivalent for the Microsoft Jet SQL BIT data type.

TIMESTAMP is no longer supported as a synonym for DATETIME.

NUMERIC is no longer supported as a synonym for FLOAT or DOUBLE.
NUMERIC is now used as a synonym for DECIMAL.

A LONGTEXT field is always stored in the [Unicode representation format](#).

If the data type name [TEXT](#) is used without specifying the optional length, for example TEXT(25), a LONGTEXT field is created. This enables [CREATE TABLE statements](#) to be written that will yield data types consistent with Microsoft SQL Server.

A CHAR field is always stored in the [Unicode representation format](#), which is the equivalent of the ANSI SQL NATIONAL CHAR data type.

If the data type name [TEXT](#) is used and the optional length is specified, for example TEXT(25), the data type of the field is equivalent to the CHAR data type. This preserves backwards compatibility for most Microsoft Jet applications, while enabling the [TEXT data type](#) (without a length specification) to be aligned with Microsoft SQL Server.

See Also

[Microsoft Jet Database Engine SQL Data Types](#)

Using Wildcard Characters in String Comparisons

Built-in pattern matching provides a versatile tool for making string comparisons. The following table shows the wildcard characters you can use with the **Like** operator and the number of digits or strings they match.

Character(s) in <i>pattern</i>	Matches in <i>expression</i>
? or _ (underscore)	Any single character
* or %	Zero or more characters
#	Any single digit (0 — 9)
[<i>charlist</i>]	Any single character in <i>charlist</i>
[! <i>charlist</i>]	Any single character not in <i>charlist</i>

You can use a group of one or more characters (*charlist*) enclosed in brackets ([]) to match any single character in *expression*, and *charlist* can include almost any characters in the [ANSI](#) character set, including digits. You can use the special characters opening bracket ([), question mark (?), number sign (#), and [asterisk](#) (*) to match themselves directly only if enclosed in brackets. You cannot use the closing bracket (]) within a group to match itself, but you can use it outside a group as an individual character.

In addition to a simple list of characters enclosed in brackets, *charlist* can specify a range of characters by using a hyphen (-) to separate the upper and lower bounds of the range. For example, using [A-Z] in *pattern* results in a match if the corresponding character position in *expression* contains any of the uppercase letters in the range A through Z. You can include multiple ranges within the brackets without delimiting the ranges. For example, [a-zA-Z0-9] matches any alphanumeric character.

It is important to note that the ANSI SQL wildcards (%) and (_) are only available with Microsoft® Jet version 4.X and the Microsoft OLE DB Provider for Jet. They will be treated as literals if used through Microsoft Access or DAO.

Other important rules for pattern matching include the following:

An exclamation mark (!) at the beginning of *charlist* means that a match is made if any character except those in *charlist* are found in *expression*. When used outside brackets, the exclamation mark matches itself.

You can use the hyphen (-) either at the beginning (after an exclamation mark if

one is used) or at the end of *charlist* to match itself. In any other location, the hyphen identifies a range of ANSI characters.

When you specify a range of characters, the characters must appear in ascending sort order (A-Z or 0-100). [A-Z] is a valid pattern, but [Z-A] is not.

The character sequence [] is ignored; it is considered to be a [zero-length string](#) (“”).

See Also

[Like Operator](#)

[SQL Expressions](#)

CREATE TABLE Statement

Creates a new table.

Note The [Microsoft Jet database engine](#) does not support the use of CREATE TABLE, or any of the [DDL](#) statements, with non-Microsoft Jet database engine databases. Use the DAO [Create](#) methods instead.

Syntax

```
CREATE [TEMPORARY] TABLE table (field1 type [(size)] [NOT NULL]
[WITH COMPRESSION | WITH COMP] [index1] [, field2 type [(size)] [NOT
NULL] [index2] [, ...]] [, CONSTRAINT multifieldindex [, ...]])
```

The CREATE TABLE statement has these parts:

Part	Description
<i>table</i>	The name of the table to be created.
<i>field1</i> , <i>field2</i>	The name of field or fields to be created in the new table. You must create at least one field.
<i>type</i>	The data type of <i>field</i> in the new table.
<i>size</i>	The field size in characters (Text and Binary fields only).
<i>index1</i> , <i>index2</i>	A CONSTRAINT clause defining a single-field index. For more information on how to create this index, see CONSTRAINT Clause .
<i>multifieldindex</i>	A CONSTRAINT clause defining a multiple-field index. For more information on how to create this index, see CONSTRAINT Clause .

Remarks

Use the CREATE TABLE statement to define a new table and its fields and field constraints. If NOT NULL is specified for a field, then new records are required to have valid data in that field.

A CONSTRAINT clause establishes various restrictions on a field, and can be used to establish the [primary key](#). You can also use the [CREATE INDEX](#) statement to create a primary key or additional indexes on existing tables.

You can use NOT NULL on a single field or within a named CONSTRAINT clause that applies to either a single field or to a multiple-field named CONSTRAINT. However, you can apply the NOT NULL restriction only once to a field. Attempting to apply this restriction more than once results in a run-time error.

When a TEMPORARY table is created it is visible only within the session in which it was created. It is automatically deleted when the session is terminated. Temporary tables can be accessed by more than one user.

The WITH COMPRESSION attribute can be used only with the CHARACTER and MEMO (also known as TEXT) data types and their synonyms.

The WITH COMPRESSION attribute was added for CHARACTER columns because of the change to the Unicode character representation format. Unicode characters uniformly require two bytes for each character. For existing Microsoft® Jet databases that contain predominately character data, this could mean that the database file would nearly double in size when converted to the Microsoft Jet version 4.0 format. However, [Unicode representation](#) of many character sets, those formerly denoted as Single-Byte Character Sets (SBCS) can easily be compressed to a single byte. If you define a CHARACTER column with this attribute, data will automatically be compressed as it is stored and uncompressed when retrieved from the column.

MEMO columns can also be defined to store data in a compressed format. However, there is a limitation. Only instances of MEMO columns that, when compressed, will fit within 4096 bytes or less, will be compressed. All other instances of MEMO columns will remain uncompressed. This means that within a given table, for a given MEMO column, some data may be compressed and some data may not be compressed.

See Also

[ADD USER Statement](#)

[CREATE USER or GROUP Statement](#)

[ALTER USER or DATABASE Statement](#)

[CREATE VIEW Statement](#)

[ALTER TABLE Statement](#)

[DROP Statement](#)

[CONSTRAINT Clause](#)

[DROP USER or GROUP Statement](#)

[CREATE INDEX Statement](#)

[GRANT Statement](#)

[CREATE PROCEDURE Statement REVOKE Statement](#)

Example

[CREATE TABLE Statement, CONSTRAINT Clause Example](#)

CREATE INDEX Statement

Creates a new index on an existing table.

Note For non-Microsoft Jet databases, the [Microsoft Jet database engine](#) does not support the use of CREATE INDEX (except to create a [pseudo index](#) on an [ODBC linked table](#)) or any of the [data definition language \(DDL\)](#) statements. Use the [DAO Create](#) methods instead. For more information see the Remarks section.

Syntax

```
CREATE [ UNIQUE ] INDEX index  
  ON table (field [ASC|DESC][, field [ASC|DESC], ...])  
  [WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]
```

The CREATE INDEX statement has these parts:

Part	Description
<i>index</i>	The name of the index to be created.
<i>table</i>	The name of the existing table that will contain the index.
<i>field</i>	The name of the field or fields to be indexed. To create a single-field index, list the field name in parentheses following the table name. To create a multiple-field index, list the name of each field to be included in the index. To create descending indexes, use the DESC reserved word; otherwise, indexes are assumed to be ascending.

Remarks

To prohibit duplicate values in the indexed field or fields of different records, use the UNIQUE reserved word.

In the optional WITH clause you can enforce data validation rules. You can:

Prohibit [Null](#) entries in the indexed field or fields of new records by using the DISALLOW NULL option.

Prevent records with **Null** values in the indexed field or fields from being included in the index by using the IGNORE NULL option.

Designate the indexed field or fields as the [primary key](#) by using the PRIMARY reserved word. This implies that the key is unique, so you can omit the UNIQUE reserved word.

You can use CREATE INDEX to create a [pseudo index](#) on a [linked table](#) in an [ODBC data source](#), such as Microsoft® SQL Server™, that does not already have an index. You do not need permission or access to the remote server to create a pseudo index, and the remote database is unaware of and unaffected by the pseudo index. You use the same syntax for both linked and native tables. Creating a pseudo-index on a table that would ordinarily be read-only can be especially useful.

You can also use the [ALTER TABLE](#) statement to add a single- or multiple-field index to a table, and you can use the ALTER TABLE statement or the [DROP](#) statement to remove an index created with ALTER TABLE or CREATE INDEX.

Note Do not use the PRIMARY [reserved word](#) when you create a new index on a table that already has a primary key; if you do, an error occurs.

See Also

[ADD USER Statement](#)

[CREATE USER or GROUP Statement](#)

[ALTER USER or DATABASE Statement](#)

[CREATE VIEW Statement](#)

[ALTER TABLE Statement](#)

[DROP Statement](#)

[CONSTRAINT Clause](#)

[DROP USER or GROUP Statement](#)

[CREATE PROCEDURE Statement](#)

[GRANT Statement](#)

[CREATE TABLE Statement](#)

[REVOKE Statement](#)

Example

[CREATE INDEX Statement Example](#)

CREATE PROCEDURE Statement

Creates a stored [procedure](#).

Note The [Microsoft Jet database engine](#) does not support the use of CREATE PROCEDURE, or any of the [DDL](#) statements, with non-Microsoft Jet database engine databases.

Syntax

```
CREATE PROCEDURE procedure  
    [param1 datatype[, param2 datatype[, ...]] AS sqlstatement
```

The CREATE PROCEDURE statement has these parts:

Part	Description
<i>procedure</i>	A name for the procedure. It must follow standard naming conventions .
<i>param1, param2</i>	From one to 255 field names or parameters . For example: CREATE PROCEDURE Sales_By_Country [Beginning Date] DateTime, [Ending Date] DateTime; For more information on parameters, see PARAMETERS .
<i>datatype</i>	One of the primary Microsoft Jet SQL data types or their synonyms.
<i>sqlstatement</i>	An SQL statement such as SELECT, UPDATE, DELETE, INSERT, CREATE TABLE, DROP TABLE, and so on.

Remarks

An SQL procedure consists of a PROCEDURE clause that specifies the name of the procedure, an optional list of parameter definitions, and a single [SQL statement](#).

A procedure name cannot be the same as the name of an existing table.

See Also

[ADD USER Statement](#)

[ALTER USER or DATABASE Statement](#)

[ALTER TABLE Statement](#)

[CONSTRAINT Clause](#)

[CREATE INDEX Statement](#)

[CREATE TABLE Statement](#)

[CREATE USER or GROUP Statement](#)

[CREATE VIEW Statement](#)

[DROP Statement](#)

[DROP USER or GROUP Statement](#)

[GRANT Statement](#)

[REVOKE Statement](#)

Example

[CREATE PROCEDURE Statement, PROCEDURE Clause Example](#)

CREATE USER or GROUP Statement

Creates one or more new users or groups.

Syntax

Create a user:

```
CREATE USER user password pid [, user password pid, ...]
```

Create a group:

```
CREATE GROUP group pid [, group pid, ...]
```

The CREATE USER or GROUP statement has these parts:

Part	Description
<i>user</i>	The name of a user to be added to the workgroup information file.
<i>group</i>	The name of a group to be added to the workgroup information file.
<i>password</i>	The password to be associated with the specified <i>user</i> name.
<i>pid</i>	The personal id.

Remarks

A *user* and a *group* cannot have the same name.

A *password* is required for each *user* or *group* that is created.

See Also

[ADD USER Statement](#)

[CREATE TABLE Statement](#)

[ALTER USER or DATABASE Statement](#)

[CREATE VIEW Statement](#)

[ALTER TABLE Statement](#)

[DROP Statement](#)

[CONSTRAINT Clause](#)

[DROP USER or GROUP Statement](#)

[CREATE INDEX Statement](#)

[GRANT Statement](#)

[CREATE PROCEDURE Statement](#) [REVOKE Statement](#)

CREATE VIEW Statement

Creates a new [view](#).

Note The [Microsoft Jet database engine](#) does not support the use of CREATE VIEW, or any of the [DDL](#) statements, with non-Microsoft Jet database engine databases.

Syntax

```
CREATE VIEW view [(field1[, field2[, ...]])] AS selectstatement
```

The CREATE VIEW statement has these parts:

Part	Description
<i>view</i>	The name of the view to be created.
<i>field1, field2</i>	The name of field or fields for the corresponding fields specified in <i>selectstatement</i> .
<i>selectstatement</i>	A SQL SELECT statement. For more information, see SELECT Statement .

Remarks

The SELECT statement that defines the view cannot be a [SELECT INTO](#) statement.

The SELECT statement that defines the view cannot contain any parameters.

The name of the [view](#) cannot be the same as the name of an existing table.

If the query defined by the SELECT statement is updatable, then the view is also updatable. Otherwise, the view is read-only.

If any two fields in the query defined by the SELECT statement have the same name, the view definition must include a field list specifying unique names for each of the fields in the query.

See Also

[ADD USER Statement](#)

[ALTER USER or DATABASE Statement](#)

[ALTER TABLE Statement](#)

[CONSTRAINT Clause](#)

[CREATE INDEX Statement](#)

[CREATE PROCEDURE Statement](#)

[CREATE USER or GROUP Statement](#)

[CREATE TABLE Statement](#)

[DROP Statement](#)

[DROP USER or GROUP Statement](#)

[GRANT Statement](#)

[REVOKE Statement](#)

ADD USER Statement

Adds one or more existing *users* to an existing *group*.

Syntax

ADD USER *user*[, *user*, ...] TO *group*

The ADD USER statement has these parts:

Part	Description
<i>user</i>	The name of a user to be added to the workgroup information file.
<i>group</i>	The name of a group to be added to the workgroup information file.

Remarks

Once a *user* had been added to a *group*, the *user* has all the permissions that have been granted to the *group*.

See Also

ALTER USER or DATABASE Statement	CREATE USER or GROUP Statement
ALTER TABLE Statement	CREATE VIEW Statement
CONSTRAINT Clause	DROP Statement
CREATE INDEX Statement	DROP USER or GROUP Statement
CREATE PROCEDURE Statement	GRANT Statement
CREATE TABLE Statement	REVOKE Statement

DROP USER or GROUP Statement

Deletes one or more existing *users* or *groups*, or removes one or more existing *users* from an existing *group*.

Syntax

Delete one or more *users* or remove one or more *users* from a *group*:

```
DROP USER user[, user, ...] [FROM group]
```

Delete one or more *groups*:

```
DROP GROUP group[, group, ...]
```

The DROP USER or GROUP statement has these parts:

Part	Description
<i>user</i>	The name of a user to be removed from the workgroup information file.
<i>group</i>	The name of a group to be removed from the workgroup information file.

Remarks

If the FROM keyword is used in the DROP USER statement, then each of the *users* listed in the statement will be removed from the *group* specified following the FROM keyword. However, the *users* themselves will not be deleted.

The DROP GROUP statement will delete the specified *group(s)*. The *users* who are members of the *group(s)* will not be affected, but they will no longer be members of the deleted *group(s)*.

See Also

[ADD USER Statement](#)

[CREATE TABLE Statement](#)

[ALTER USER or DATABASE Statement](#)

[CREATE USER or GROUP Statement](#)

[ALTER TABLE Statement](#)

[CREATE VIEW Statement](#)

[CONSTRAINT Clause](#)

[DROP Statement](#)

[CREATE INDEX Statement](#)

[GRANT Statement](#)

[CREATE PROCEDURE Statement](#)

[REVOKE Statement](#)

ALTER TABLE Statement

Modifies the design of a table after it has been created with the [CREATE TABLE](#) statement.

Note The [Microsoft Jet database engine](#) does not support the use of ALTER TABLE, or any of the [data definition language \(DDL\)](#) statements, with non-Microsoft Jet databases. Use the [DAO Create](#) methods instead.

Syntax

```
ALTER TABLE table {ADD {COLUMN field type[(size)] [NOT NULL]
    [CONSTRAINT index] |
    ALTER COLUMN field type[(size)] |
    CONSTRAINT multifieldindex} |
    DROP {COLUMN field | CONSTRAINT indexname} }
```

The ALTER TABLE statement has these parts:

Part	Description
<i>table</i>	The name of the table to be altered.
<i>field</i>	The name of the field to be added to or deleted from <i>table</i> . Or, the name of the field to be altered in <i>table</i> .
<i>type</i>	The data type of <i>field</i> .
<i>size</i>	The field size in characters (Text and Binary fields only).
<i>index</i>	The index for <i>field</i> . For more information on how to construct this index see CONSTRAINT Clause .
<i>multifieldindex</i>	The definition of a multiple-field index to be added to <i>table</i> . For more information on how to construct this index see CONSTRAINT Clause .
<i>indexname</i>	The name of the multiple-field index to be removed.

Remarks

Using the ALTER TABLE statement you can alter an existing table in several ways. You can:

Use `ADD COLUMN` to add a new field to the table. You specify the field name, data type, and (for Text and Binary fields) an optional size. For example, the following statement adds a 25-character Text field called Notes to the Employees table:

```
ALTER TABLE Employees ADD COLUMN Notes TEXT(25)
```

You can also define an index on that field. For more information on single-field indexes see [CONSTRAINT Clause](#).

If you specify `NOT NULL` for a field then new records are required to have valid data in that field.

Use `ALTER COLUMN` to change the data type of an existing field. You specify the field name, the new data type, and an optional size for Text and Binary fields. For example, the following statement changes the data type of a field in the Employees table called ZipCode (originally defined as Integer) to a 10-character Text field:

```
ALTER TABLE Employees ALTER COLUMN ZipCode TEXT(10)
```

Use `ADD CONSTRAINT` to add a multiple-field index. For more information on multiple-field indexes see [CONSTRAINT Clause](#).

Use `DROP COLUMN` to delete a field. You specify only the name of the field.

Use `DROP CONSTRAINT` to delete a multiple-field index. You specify only the index name following the `CONSTRAINT` reserved word.

Notes

You cannot add or delete more than one field or index at a time.

You can use the [CREATE INDEX](#) statement to add a single- or multiple-field index to a table, and you can use `ALTER TABLE` or the [DROP](#) statement to delete an index created with `ALTER TABLE` or `CREATE INDEX`.

You can use `NOT NULL` on a single field or within a named `CONSTRAINT` clause that applies to either a single field or to a multiple-field named `CONSTRAINT`. However, you can apply the `NOT NULL` restriction only once to a field. Attempting to apply this restriction more than once results in a run-time error.

See Also

[ADD USER Statement](#)

[ALTER USER or DATABASE Statement](#)

[CONSTRAINT Clause](#)

[CREATE INDEX Statement](#)

[CREATE PROCEDURE Statement](#)

[CREATE TABLE Statement](#)

[CREATE USER or GROUP Statement](#)

[CREATE VIEW Statement](#)

[DROP Statement](#)

[DROP USER or GROUP Statement](#)

[GRANT Statement](#)

[REVOKE Statement](#)

Example

[ALTER TABLE Statement Example](#)

ALTER USER or DATABASE Statement

Changes the password for an existing user or for a database.

Syntax

ALTER DATABASE PASSWORD *newpassword oldpassword*

ALTER USER *user* PASSWORD *newpassword oldpassword*

The ALTER USER or DATABASE statement has these parts:

Part	Description
<i>user</i>	The name of a user to be added to the workgroup information file.
<i>newpassword</i>	The new password to be associated with the specified <i>user</i> or <i>database</i> name.
<i>oldpassword</i>	The existing password to be associated with the specified <i>user</i> or <i>group</i> name.

See Also

[ADD USER Statement](#)

[CREATE USER or GROUP Statement](#)

[ALTER TABLE Statement](#)

[CREATE VIEW Statement](#)

[CONSTRAINT Clause](#)

[DROP Statement](#)

[CREATE INDEX Statement](#)

[DROP USER or GROUP Statement](#)

[CREATE PROCEDURE Statement](#)

[GRANT Statement](#)

[CREATE TABLE Statement](#)

[REVOKE Statement](#)

CONSTRAINT Clause

A [constraint](#) is similar to an [index](#), although it can also be used to establish a [relationship](#) with another table.

You use the CONSTRAINT clause in [ALTER TABLE](#) and [CREATE TABLE](#) statements to create or delete constraints. There are two types of CONSTRAINT clauses: one for creating a constraint on a single field and one for creating a constraint on more than one field.

Note The [Microsoft Jet database engine](#) does not support the use of CONSTRAINT, or any of the [data definition language \(DDL\)](#) statements, with non-Microsoft Jet databases. Use the [DAO Create](#) methods instead.

Syntax

Single-field constraint:

```
CONSTRAINT name {PRIMARY KEY | UNIQUE | NOT NULL |  
REFERENCES foreigntable [(foreignfield1, foreignfield2)]  
[ON UPDATE CASCADE | SET NULL]  
[ON DELETE CASCADE | SET NULL]}
```

Multiple-field constraint:

```
CONSTRAINT name  
{PRIMARY KEY (primary1 [, primary2 [, ...]]) |  
UNIQUE (unique1 [, unique2 [, ...]]) |  
NOT NULL (notnull1 [, notnull2 [, ...]]) |  
FOREIGN KEY [NO INDEX] (ref1 [, ref2 [, ...]]) REFERENCES foreigntable  
[(foreignfield1 [, foreignfield2 [, ...]])]  
[ON UPDATE CASCADE | SET NULL]  
[ON DELETE CASCADE | SET NULL]}
```

The CONSTRAINT clause has these parts:

Part	Description
<i>name</i>	The name of the constraint to be created.
<i>primary1</i> ,	The name of the field or fields to be designated the primary

<i>primary2</i>	key .
<i>unique1, unique2</i>	The name of the field or fields to be designated as a unique key.
<i>notnull1, notnull2</i>	The name of the field or fields that are restricted to non- Null values.
<i>ref1, ref2</i>	The name of a foreign key field or fields that refer to fields in another table.
<i>foreigntable</i>	The name of the foreign table containing the field or fields specified by <i>foreignfield</i> .
<i>foreignfield1, foreignfield2</i>	The name of the field or fields in <i>foreigntable</i> specified by <i>ref1, ref2</i> . You can omit this clause if the referenced field is the primary key of <i>foreigntable</i> .

Remarks

You use the syntax for a single-field constraint in the field-definition clause of an ALTER TABLE or CREATE TABLE statement immediately following the specification of the field's data type.

You use the syntax for a multiple-field constraint whenever you use the [reserved word](#) CONSTRAINT outside a field-definition clause in an ALTER TABLE or CREATE TABLE statement.

Using CONSTRAINT you can designate a field as one of the following types of constraints:

You can use the UNIQUE reserved word to designate a field as a unique key. This means that no two records in the table can have the same value in this field. You can constrain any field or list of fields as unique. If a multiple-field constraint is designated as a unique key, the combined values of all fields in the index must be unique, even if two or more records have the same value in just one of the fields.

You can use the [PRIMARY KEY](#) reserved words to designate one field or set of fields in a table as a primary key. All values in the primary key must be unique and not **Null**, and there can be only one primary key for a table.

Note Do not set a PRIMARY KEY constraint on a table that already has a primary key; if you do, an error occurs.

You can use the [FOREIGN KEY](#) reserved words to designate a field as a foreign key. If the foreign table's primary key consists of more than one field, you must use a multiple-field constraint definition, listing all of the referencing fields, the name of the foreign table, and the names of the referenced fields in the foreign table in the same order that the referencing fields are listed. If the referenced field or fields are the foreign table's primary key, you do not have to specify the referenced fields. By default the database engine behaves as if the foreign table's primary key is the referenced fields.

Foreign key constraints define specific actions to be performed when a corresponding primary key value is changed:

You can specify actions to be performed on the foreign table based on a corresponding action performed on a primary key in the table on which the CONSTRAINT is defined. For example, consider the following definition for the table Customers:

```
CREATE TABLE Customers (CustId INTEGER PRIMARY KEY,  
CLstNm NCHAR VARYING (50))
```

Consider the following definition of the table Orders, which defines a foreign key relationship referencing the primary key of the Customers table:

```
CREATE TABLE Orders (OrderId INTEGER PRIMARY KEY,  
CustId INTEGER, OrderNotes NCHAR VARYING (255),  
CONSTRAINT FKOrdersCustId FOREIGN KEY (CustId)  
REFERENCES Customers ON UPDATE CASCADE ON DELETE  
CASCADE
```

Both an ON UPDATE CASCADE and an ON DELETE CASCADE clause are defined on the foreign key. The ON UPDATE CASCADE clause means that if a customer's identifier (CustId) is updated in the Customer table, the update will be cascaded through the Orders table. Each order containing a corresponding customer identifier value will be updated automatically with the new value. The ON DELETE CASCADE clause means that if a customer is deleted from the Customer table, all rows in the Orders table containing the same customer identifier value will also be deleted.

Consider the following different definition of the table Orders, using the SET NULL action instead of the CASCADE action:

```
CREATE TABLE Orders (OrderId INTEGER PRIMARY KEY,  
CustId INTEGER, OrderNotes NCHAR VARYING (255),  
CONSTRAINT FKOrdersCustId FOREIGN KEY (CustId)  
REFERENCES Customers ON UPDATE SET NULL ON DELETE  
SET NULL
```

The ON UPDATE SET NULL clause means that if a customer's identifier (CustId) is updated in the Customer table, the corresponding foreign key values in the Orders table will automatically be set to NULL. Similarly, the ON DELETE SET NULL clause means that if a customer is deleted from the Customer table, all corresponding foreign keys in the Orders table will automatically be set to NULL.

To prevent the automatic creation of indexes for foreign keys, the modifier NO INDEX can be used. This form of foreign key definition should be used only in cases where the resulting index values would be frequently duplicated. Where the values in a foreign key index are frequently duplicated, using an index can be less efficient than simply performing a table scan. Maintaining this type of index, with rows inserted and deleted from the table, degrades performance and does not provide any benefit.

See Also

[ADD USER Statement](#)

[CREATE USER or GROUP Statement](#)

[ALTER USER or DATABASE Statement](#)

[CREATE VIEW Statement](#)

[ALTER TABLE Statement](#)

[DROP Statement](#)

[CREATE INDEX Statement](#)

[DROP USER or GROUP Statement](#)

[CREATE PROCEDURE Statement](#)

[GRANT Statement](#)

[CREATE TABLE Statement](#)

[REVOKE Statement](#)

Example

[CREATE TABLE Statement, CONSTRAINT Clause Example Example](#)

DROP Statement

Deletes an existing table, procedure, or view from a database, or deletes an existing index from a table.

Note The [Microsoft Jet database engine](#) does not support the use of DROP, or any of the [DDL](#) statements, with non-Microsoft Jet databases. Use the [DAO Delete](#) method instead.

Syntax

```
DROP {TABLE table | INDEX index ON table | PROCEDURE procedure |  
VIEW view}
```

The DROP statement has these parts:

Part	Description
<i>table</i>	The name of the table to be deleted or the table from which an index is to be deleted.
<i>procedure</i>	The name of the procedure to be deleted.
<i>view</i>	The name of the view to be deleted.
<i>index</i>	The name of the index to be deleted from <i>table</i> .

Remarks

You must close the table before you can delete it or remove an index from it.

You can also use [ALTER TABLE](#) to delete an index from a table.

You can use [CREATE TABLE](#) to create a table and [CREATE INDEX](#) or ALTER TABLE to create an index. To modify a table, use ALTER TABLE.

See Also

[ADD USER Statement](#)

[CREATE USER or GROUP Statement](#)

[ALTER USER or DATABASE Statement](#)

[CREATE TABLE Statement](#)

[ALTER TABLE Statement](#)

[CREATE VIEW Statement](#)

[CONSTRAINT Clause](#)

[DROP USER or GROUP Statement](#)

[CREATE INDEX Statement](#)

[GRANT Statement](#)

[CREATE PROCEDURE Statement](#)

[REVOKE Statement](#)

Example

[DROP Statement Example](#)

GRANT Statement

Grants specific privileges to an existing user or group.

Syntax

```
GRANT {privilege[, privilege, ...]} ON {TABLE table |  
OBJECT object |  
CONTAINER container } TO {authorizationname[, authorizationname, ...]}
```

The GRANT statement has these parts:

Part	Description
<i>privilege</i>	The privilege or privileges to be granted. Privileges are specified using the following keywords: SELECT, DELETE, INSERT, UPDATE, DROP, SELECTSECURITY, UPDATESECURITY, DBPASSWORD, UPDATEIDENTITY, CREATE, SELECTSCHEMA, SCHEMA and UPDATEOWNER.
<i>tablename</i>	Any valid table name.
<i>object</i>	This can encompass any non-table object. A stored query (view or procedure) is one example.
<i>container</i>	The name of a valid container.
<i>authorizationname</i>	A user or group name.

See Also

[ADD USER Statement](#)

[CREATE TABLE Statement](#)

[ALTER USER or DATABASE Statement](#)

[CREATE USER or GROUP Statement](#)

[ALTER TABLE Statement](#)

[CREATE VIEW Statement](#)

[CONSTRAINT Clause](#)

[DROP Statement](#)

[CREATE INDEX Statement](#)

[DROP USER or GROUP Statement](#)

CREATE PROCEDURE
Statement

REVOKE Statement

REVOKE Statement

Revokes specific privileges from an existing user or group.

Syntax

```
REVOKE {privilege[, privilege, ...]} ON  
  {TABLE table |  
  OBJECT object |  
  CONTAINER container }  
FROM {authorizationname[, authorizationname, ...]}
```

The REVOKE statement has these parts:

Part	Description
<i>privilege</i>	The privilege or privileges to be revoked. Privileges are specified using the following keywords: SELECT, DELETE, INSERT, UPDATE, DROP, SELECTSECURITY, UPDATESECURITY, DBPASSWORD, UPDATEIDENTITY, CREATE, SELECTSCHEMA, SCHEMA and UPDATEOWNER.
<i>table</i>	Any valid table name.
<i>object</i>	This can encompass any non-table object. A stored query (view or procedure) is one example.
<i>container</i>	The name of a valid container.
<i>authorizationname</i>	A user or group name.

See Also

[ADD USER Statement](#)

[CREATE TABLE Statement](#)

[ALTER USER or DATABASE Statement](#)

[CREATE USER or GROUP Statement](#)

[ALTER TABLE Statement](#)

[CREATE VIEW Statement](#)

[CONSTRAINT Clause](#)

[DROP Statement](#)

[CREATE INDEX Statement](#)

[CREATE PROCEDURE](#)

[Statement](#)

[DROP USER or GROUP Statement](#)

[GRANT Statement](#)

SELECT Statement

Instructs the [Microsoft Jet database engine](#) to return information from the database as a set of records.

Syntax

```
SELECT [predicate] { * | table.* | [table.]field1 [AS alias1] [, [table.]field2 [AS alias2] [, ...]] }  
FROM tableexpression [, ...] [IN externaldatabase]  
[WHERE... ]  
[GROUP BY... ]  
[HAVING... ]  
[ORDER BY... ]  
[WITH OWNERACCESS OPTION]
```

The SELECT statement has these parts:

Part	Description
<i>predicate</i>	One of the following predicates: ALL , DISTINCT , DISTINCTROW , or TOP . You use the predicate to restrict the number of records returned. If none is specified, the default is ALL.
*	Specifies that all fields from the specified table or tables are selected.
<i>table</i>	The name of the table containing the fields from which records are selected.
<i>field1, field2</i>	The names of the fields containing the data you want to retrieve. If you include more than one field, they are retrieved in the order listed.
<i>alias1, alias2</i>	The names to use as column headers instead of the original column names in <i>table</i> .
<i>tableexpression</i>	The name of the table or tables containing the data you want to retrieve.
<i>externaldatabase</i>	The name of the database containing the tables in <i>tableexpression</i> if they are not in the current database.

Remarks

To perform this operation, the Microsoft® Jet database engine searches the specified table or tables, extracts the chosen columns, selects rows that meet the criterion, and sorts or groups the resulting rows into the order specified.

SELECT statements do not change data in the database.

SELECT is usually the first word in an [SQL statement](#). Most SQL statements are either SELECT or [SELECT...INTO](#) statements.

The minimum syntax for a SELECT statement is:

```
SELECT fields FROM table
```

You can use an [asterisk](#) (*) to select all fields in a table. The following example selects all of the fields in the Employees table:

```
SELECT * FROM Employees;
```

If a field name is included in more than one table in the FROM clause, precede it with the table name and the . (dot) operator. In the following example, the Department field is in both the Employees table and the Supervisors table. The SQL statement selects departments from the Employees table and supervisor names from the Supervisors table:

```
SELECT Employees.Department, Supervisors.SupvName  
FROM Employees INNER JOIN Supervisors  
WHERE Employees.Department = Supervisors.Department;
```

When a [Recordset](#) object is created, the Microsoft Jet database engine uses the table's field name as the [Field](#) object name in the **Recordset** object. If you want a different field name or a name is not implied by the expression used to generate the field, use the AS [reserved word](#). The following example uses the title Birth to name the returned **Field** object in the resulting **Recordset** object:

```
SELECT BirthDate  
AS Birth FROM Employees;
```

Whenever you use [aggregate functions](#) or queries that return ambiguous or duplicate **Field** object names, you must use the AS clause to provide an alternate name for the **Field** object. The following example uses the title HeadCount to name the returned **Field** object in the resulting **Recordset** object:


```
SELECT COUNT(EmployeeID)
AS HeadCount FROM Employees;
```

You can use the other clauses in a SELECT statement to further restrict and organize your returned data. For more information, see the Help topic for the clause you are using.

See Also

[ALL DISTINCT, DISTINCTROW, ORDER BY Clause \(Microsoft Jet SQL\)](#)
[TOP Predicates \(Microsoft Jet SQL\)](#)

[DELETE Statement \(Microsoft Jet SQL\)](#) [SELECT...INTO Statement \(Microsoft Jet SQL\)](#)

[FROM Clause \(Microsoft Jet SQL\)](#) [SQL Aggregate Functions \(SQL\)](#)

[GROUP BY Clause \(Microsoft Jet SQL\)](#) [UNION Operation \(Microsoft Jet SQL\)](#)

[HAVING Clause \(Microsoft Jet SQL\)](#) [UPDATE Statement \(Microsoft Jet SQL\)](#)

[IN Clause \(Microsoft Jet SQL\)](#) [WHERE Clause \(Microsoft Jet SQL\)](#)

[INSERT INTO Statement \(Microsoft Jet SQL\)](#) [WITH OWNERACCESS OPTION Declaration \(Microsoft Jet SQL\)](#)

Example

[SELECT Statement, FROM Clause Example](#)

SELECT...INTO Statement

Creates a [make-table query](#).

Syntax

```
SELECT field1[, field2[, ...]] INTO newtable [IN externaldatabase]  
FROM source
```

The SELECT...INTO statement has these parts:

Part	Description
<i>field1</i> , <i>field2</i>	The name of the fields to be copied into the new table.
<i>newtable</i>	The name of the table to be created. It must conform to standard naming conventions . If <i>newtable</i> is the same as the name of an existing table, a trappable error occurs.
<i>externaldatabase</i>	The path to an external database . For a description of the path, see the IN clause.
<i>source</i>	The name of the existing table from which records are selected. This can be single or multiple tables or a query.

Remarks

You can use make-table queries to archive records, make backup copies of your tables, or make copies to export to another database or to use as a basis for reports that display data for a particular time period. For example, you could produce a Monthly Sales by Region report by running the same make-table query each month.

Notes

You may want to define a [primary key](#) for the new table. When you create the table, the fields in the new table inherit the [data type](#) and field size of each field in the query's underlying tables, but no other field or table properties are transferred.

To add data to an existing table, use the [INSERT INTO](#) statement instead to create an [append query](#).

To find out which records will be selected before you run the make-table query, first examine the results of a [SELECT](#) statement that uses the same selection criteria.

See Also

[ALL, DISTINCT, DISTINCTROW, TOP Predicates \(Microsoft Jet SQL\)](#)

[FROM Clause \(Microsoft Jet SQL\)](#) [UNION Operation \(Microsoft Jet SQL\)](#)

[IN Clause \(Microsoft Jet SQL\)](#) [WHERE Clause \(Microsoft Jet SQL\)](#)

[INSERT INTO Statement \(Microsoft Jet SQL\)](#)

Example

[SELECT...INTO Statement Example](#)

INSERT INTO Statement

Adds a record or multiple records to a table. This is referred to as an [append query](#).

Syntax

Multiple-record append query:

```
INSERT INTO target [(field1[, field2[, ...]])] [IN externaldatabase]  
  SELECT [source.]field1[, field2[, ...]]  
  FROM tableexpression
```

Single-record append query:

```
INSERT INTO target [(field1[, field2[, ...]])]  
  VALUES (value1[, value2[, ...]])
```

The INSERT INTO statement has these parts:

Part	Description
<i>target</i>	The name of the table or query to append records to.
<i>field1</i> , <i>field2</i>	Names of the fields to append data to, if following a <i>target</i> argument, or the names of fields to obtain data from, if following a <i>source</i> argument.
<i>externaldatabase</i>	The path to an external database . For a description of the path, see the IN clause.
<i>source</i>	The name of the table or query to copy records from.
<i>tableexpression</i>	The name of the table or tables from which records are inserted. This argument can be a single table name or a compound resulting from an INNER JOIN , LEFT JOIN , or RIGHT JOIN operation or a saved query.
<i>value1</i> , <i>value2</i>	The values to insert into the specific fields of the new record. Each value is inserted into the field that corresponds to the value's position in the list: <i>value1</i> is inserted into <i>field1</i> of the new record, <i>value2</i> into <i>field2</i> , and so on. You must separate values with a comma, and enclose text fields in quotation marks (' ').

Remarks

You can use the INSERT INTO statement to add a single record to a table using the single-record append query syntax as shown above. In this case, your code specifies the name and value for each field of the record. You must specify each of the fields of the record that a value is to be assigned to and a value for that field. When you do not specify each field, the default value or **Null** is inserted for missing columns. Records are added to the end of the table.

You can also use INSERT INTO to append a set of records from another table or query by using the SELECT ... FROM clause as shown above in the multiple-record append query syntax. In this case, the SELECT clause specifies the fields to append to the specified *target* table.

The *source* or *target* table may specify a table or a query. If a query is specified, the [Microsoft Jet database engine](#) appends records to any and all tables specified by the query.

INSERT INTO is optional but when included, precedes the [SELECT](#) statement.

If your destination table contains a [primary key](#), make sure you append unique, non-**Null** values to the primary key field or fields; if you do not, the [Microsoft Jet database engine](#) will not append the records.

If you append records to a table with an [AutoNumber](#) field and you want to renumber the appended records, do not include the AutoNumber field in your query. Do include the AutoNumber field in the query if you want to retain the original values from the field.

Use the IN clause to append records to a table in another database.

To create a new table, use the [SELECT... INTO](#) statement instead to create a [make-table query](#).

To find out which records will be appended before you run the append query, first execute and view the results of a [select query](#) that uses the same selection criteria.

An append query copies records from one or more tables to another. The tables that contain the records you append are not affected by the append query.

Instead of appending existing records from another table, you can specify the value for each field in a single new record using the VALUES clause. If you omit the field list, the VALUES clause must include a value for every field in the

table; otherwise, the INSERT operation will fail. Use an additional INSERT INTO statement with a VALUES clause for each additional record you want to create.

See Also

[FROM Clause \(Microsoft Jet SQL\)](#) [SELECT Statement \(Microsoft Jet SQL\)](#)

[IN Clause \(Microsoft Jet SQL\)](#) [SELECT...INTO Statement \(Microsoft Jet SQL\)](#)

[INNER JOIN Operation \(Microsoft Jet SQL\)](#) [WHERE Clause \(Microsoft Jet SQL\)](#)

[LEFT JOIN, RIGHT JOIN Operations \(Microsoft Jet SQL\)](#)

Example

[INSERT INTO Statement Example](#)

UPDATE Statement

Creates an [update query](#) that changes values in fields in a specified table based on specified criteria.

Syntax

```
UPDATE table  
    SET newvalue  
    WHERE criteria;
```

The UPDATE statement has these parts:

Part	Description
<i>table</i>	The name of the table containing the data you want to modify.
<i>newvalue</i>	An expression that determines the value to be inserted into a particular field in the updated records.
<i>criteria</i>	An expression that determines which records will be updated. Only records that satisfy the expression are updated.

Remarks

UPDATE is especially useful when you want to change many records or when the records that you want to change are in multiple tables.

You can change several fields at the same time. The following example increases the Order Amount values by 10 percent and the Freight values by 3 percent for shippers in the United Kingdom:

```
UPDATE Orders  
SET OrderAmount = OrderAmount * 1.1,  
    Freight = Freight * 1.03  
WHERE ShipCountry = 'UK';
```

Important

UPDATE does not generate a result set. Also, after you update records using an update query, you cannot undo the operation. If you want to know which records were updated, first examine the results of a [select query](#) that uses the same criteria, and then run the update query.

Maintain backup copies of your data at all times. If you update the wrong records, you can retrieve them from your backup copies.

See Also

[SELECT Statement \(Microsoft Jet SQL\)](#) [WHERE Clause \(Microsoft Jet SQL\)](#)

Example

[UPDATE Statement Example](#)

DELETE Statement

Creates a [delete query](#) that removes records from one or more of the tables listed in the [FROM](#) clause that satisfy the [WHERE](#) clause.

Syntax

```
DELETE [table.*]  
FROM table  
WHERE criteria
```

The DELETE statement has these parts:

Part	Description
<i>table</i>	The optional name of the table from which records are deleted.
<i>table</i>	The name of the table from which records are deleted.
<i>criteria</i>	An expression that determines which records to delete.

Remarks

DELETE is especially useful when you want to delete many records.

To drop an entire table from the database, you can use the [Execute](#) method with a [DROP](#) statement. If you delete the table, however, the structure is lost. In contrast, when you use DELETE, only the data is deleted; the table structure and all of the table properties, such as field attributes and indexes, remain intact.

You can use DELETE to remove records from tables that are in a [one-to-many relationship](#) with other tables. [Cascade delete](#) operations cause the records in tables that are on the many side of the relationship to be deleted when the corresponding record in the one side of the relationship is deleted in the query. For example, in the relationship between the Customers and Orders tables, the Customers table is on the one side and the Orders table is on the many side of the relationship. Deleting a record from Customers results in the corresponding Orders records being deleted if the cascade delete option is specified.

A delete query deletes entire records, not just data in specific fields. If you want to delete values in a specific field, create an [update query](#) that changes the values to [Null](#).

Important

After you remove records using a delete query, you cannot undo the operation. If you want to know which records were deleted, first examine the results of a [select query](#) that uses the same criteria, and then run the delete query.

Maintain backup copies of your data at all times. If you delete the wrong records, you can retrieve them from your backup copies.

See Also

[DROP Statement \(Microsoft Jet SQL\)](#)

[SELECT Statement \(Microsoft Jet SQL\)](#)

[FROM Clause \(Microsoft Jet SQL\)](#)

[UPDATE Statement \(Microsoft Jet SQL\)](#)

[IN Clause \(Microsoft Jet SQL\)](#)

[WHERE Statement \(Microsoft Jet SQL\)](#)

[INNER JOIN Operation \(Microsoft Jet SQL\)](#)

Example

[DELETE Statement Example](#)

EXECUTE Statement

Used to invoke the execution of a [procedure](#).

Syntax

```
EXECUTE procedure [param1[, param2[, ...]]]
```

The EXECUTE statement has these parts:

Part	Description
<i>procedure</i>	The name of the procedure that is to be executed.
<i>param1</i> , <i>param2</i> , ...	Values for the parameters defined by the procedure.

See Also

[PARAMETERS Declaration \(Microsoft Jet SQL\)](#)

[PROCEDURE Clause \(Microsoft Jet SQL\)](#)

Example

[CREATE PROCEDURE Statement, PROCEDURE Clause Example](#)

TRANSACTION Statement

Used to initiate and conclude explicit [transactions](#).

Syntax

Initiate a new transaction.

BEGIN TRANSACTION

Conclude a transaction by committing all work performed during the transaction.

COMMIT [TRANSACTION | WORK]

Conclude a transaction by [rolling back](#) all work performed during the transaction.

ROLLBACK [TRANSACTION | WORK]

Remarks

Transactions are not started automatically. To start a transaction, you must do so explicitly using BEGIN TRANSACTION.

Transactions can be nested up to five levels deep. To start a nested transaction, use BEGIN TRANSACTION within the context of an existing transaction.

Transactions are not supported for linked tables.

TRANSFORM Statement

Creates a [crosstab query](#).

Syntax

```
TRANSFORM aggfunction  
  selectstatement  
  PIVOT pivotfield [IN (value1[, value2[, ...]])]
```

The TRANSFORM statement has these parts:

Part	Description
<i>aggfunction</i>	An SQL aggregate function that operates on the selected data.
<i>selectstatement</i>	A SELECT statement.
<i>pivotfield</i>	The field or expression you want to use to create column headings in the query's result set.
<i>value1, value2</i>	Fixed values used to create column headings.

Remarks

When you summarize data using a crosstab query, you select values from specified fields or expressions as column headings so you can view data in a more compact format than with a [select query](#).

TRANSFORM is optional but when included is the first statement in an [SQL string](#). It precedes a SELECT statement that specifies the fields used as row headings and a [GROUP BY](#) clause that specifies row grouping. Optionally, you can include other clauses, such as [WHERE](#), that specify additional selection or sorting criteria. You can also use [subqueries](#) as predicates — specifically, those in the WHERE clause — in a crosstab query.

The values returned in *pivotfield* are used as column headings in the query's result set. For example, pivoting the sales figures on the month of the sale in a crosstab query would create 12 columns. You can restrict *pivotfield* to create headings from fixed values (*value1, value2*) listed in the optional IN clause. You can also include fixed values for which no data exists to create additional

columns.

See Also

[FROM Clause \(Microsoft Jet SQL\)](#) [SELECT Statement \(Microsoft Jet SQL\)](#)

[GROUP BY Clause \(Microsoft Jet SQL\)](#) [SQL Aggregate Functions \(SQL\)](#)

[INNER JOIN Operation \(Microsoft Jet SQL\)](#) [SQL Subqueries](#)

[ORDER BY Clause \(Microsoft Jet SQL\)](#) [WHERE Clause \(Microsoft Jet SQL\)](#)

Example

[TRANSFORM Statement Example](#)

INNER JOIN Operation

Combines records from two tables whenever there are matching values in a common field.

Syntax

FROM *table1* INNER JOIN *table2* ON *table1.field1* *compopr* *table2.field2*

The INNER JOIN operation has these parts:

Part	Description
<i>table1, table2</i>	The names of the tables from which records are combined.
<i>field1, field2</i>	The names of the fields that are joined. If they are not numeric, the fields must be of the same data type and contain the same kind of data, but they do not have to have the same name.
<i>compopr</i>	Any relational comparison operator: "=", "<," ">," "<=," ">=," or "<>."

Remarks

You can use an INNER JOIN operation in any [FROM](#) clause. This is the most common type of join. Inner joins combine records from two tables whenever there are matching values in a field common to both tables.

You can use INNER JOIN with the Departments and Employees tables to select all the employees in each department. In contrast, to select all departments (even if some have no employees assigned to them) or all employees (even if some are not assigned to a department), you can use a [LEFT JOIN](#) or [RIGHT JOIN](#) operation to create an [outer join](#).

If you try to join fields containing [Memo](#) or [OLE Object](#) data, an error occurs.

You can join any two numeric fields of like types. For example, you can join on [AutoNumber](#) and [Long](#) fields because they are like types. However, you cannot join [Single](#) and [Double](#) types of fields.

The following example shows how you could join the Categories and Products tables on the CategoryID field:

```
SELECT CategoryName, ProductName
```

FROM Categories INNER JOIN Products
ON Categories.CategoryID = Products.CategoryID;

In the preceding example, CategoryID is the joined field, but it is not included in the query output because it is not included in the [SELECT](#) statement. To include the joined field, include the field name in the SELECT statement — in this case, Categories.CategoryID.

You can also link several ON clauses in a JOIN statement, using the following syntax:

```
SELECT fields  
FROM table1 INNER JOIN table2  
ON table1.field1 compopr table2.field1 AND  
ON table1.field2 compopr table2.field2) OR  
ON table1.field3 compopr table2.field3);
```

You can also nest JOIN statements using the following syntax:

```
SELECT fields  
FROM table1 INNER JOIN  
(table2 INNER JOIN [( table3  
[INNER JOIN [( tablex [INNER JOIN ...])]  
ON table3.field3 compopr tablex.fieldx])  
ON table2.field2 compopr table3.field3)  
ON table1.field1 compopr table2.field2;
```

A LEFT JOIN or a RIGHT JOIN may be nested inside an INNER JOIN, but an INNER JOIN may not be nested inside a LEFT JOIN or a RIGHT JOIN.

See Also

[FROM Clause \(Microsoft Jet SQL\)](#) [TRANSFORM Statement \(Microsoft Jet SQL\)](#)

[LEFT JOIN, RIGHT JOIN Operations \(Microsoft Jet SQL\)](#) [UNION Operation \(Microsoft Jet SQL\)](#)

[SELECT Statement \(Microsoft Jet SQL\)](#)

Example

[INNER JOIN Operation Example](#)

LEFT JOIN, RIGHT JOIN Operations

Combines source-table records when used in any [FROM](#) clause.

Syntax

```
FROM table1 [ LEFT | RIGHT ] JOIN table2  
  ON table1.field1 comopr table2.field2
```

The LEFT JOIN and RIGHT JOIN operations have these parts:

Part	Description
<i>table1, table2</i>	The names of the tables from which records are combined.
<i>field1, field2</i>	The names of the fields that are joined. The fields must be of the same data type and contain the same kind of data, but they do not need to have the same name.
<i>comopr</i>	Any relational comparison operator: "=", "<," ">," "<=," ">=," or "<>."

Remarks

Use a LEFT JOIN operation to create a [left outer join](#). Left outer joins include all of the records from the first (left) of two tables, even if there are no matching values for records in the second (right) table.

Use a RIGHT JOIN operation to create a [right outer join](#). Right outer joins include all of the records from the second (right) of two tables, even if there are no matching values for records in the first (left) table.

For example, you could use LEFT JOIN with the Departments (left) and Employees (right) tables to select all departments, including those that have no employees assigned to them. To select all employees, including those who are not assigned to a department, you would use RIGHT JOIN.

The following example shows how you could join the Categories and Products tables on the CategoryID field. The query produces a list of all categories, including those that contain no products:

```
SELECT CategoryName,
```

```
ProductName  
FROM Categories LEFT JOIN Products  
ON Categories.CategoryID = Products.CategoryID;
```

In this example, CategoryID is the joined field, but it is not included in the query results because it is not included in the [SELECT](#) statement. To include the joined field, enter the field name in the SELECT statement — in this case, Categories.CategoryID.

Notes

To create a query that includes only records in which the data in the joined fields is the same, use an [INNER JOIN](#) operation.

A LEFT JOIN or a RIGHT JOIN can be nested inside an INNER JOIN, but an INNER JOIN cannot be nested inside a LEFT JOIN or a RIGHT JOIN. See the discussion of nesting in the INNER JOIN topic to see how to nest joins within other joins.

You can link multiple ON clauses. See the discussion of clause linking in the INNER JOIN topic to see how this is done.

If you try to join fields containing [Memo](#) or [OLE Object](#) data, an error occurs.

See Also

[FROM Clause \(Microsoft Jet SQL\)](#) [UNION Operation \(Microsoft Jet SQL\)](#)
[INNER JOIN Operation \(Microsoft Jet SQL\)](#)

Example

[LEFT JOIN, RIGHT JOIN Operations Example](#)

UNION Operation

Creates a [union query](#), which combines the results of two or more independent queries or tables.

Syntax

```
[TABLE] query1 UNION [ALL] [TABLE] query2 [UNION [ALL] [TABLE] queryn [ ... ]]
```

The UNION operation has these parts:

Part	Description
<i>query1-n</i>	A SELECT statement , the name of a stored query, or the name of a stored table preceded by the TABLE keyword.

Remarks

You can merge the results of two or more queries, tables, and SELECT statements, in any combination, in a single UNION operation. The following example merges an existing table named New Accounts and a SELECT statement:

```
TABLE [New Accounts] UNION ALL  
SELECT *  
FROM Customers  
WHERE OrderAmount > 1000;
```

By default, no duplicate records are returned when you use a UNION operation; however, you can include the [ALL](#) predicate to ensure that all records are returned. This also makes the query run faster.

All queries in a UNION operation must request the same number of fields; however, the fields do not have to be of the same size or [data type](#).

Use [aliases](#) only in the first SELECT statement because they are ignored in any others. In the ORDER BY clause, refer to fields by what they are called in the first SELECT statement.

Notes

You can use a [GROUP BY](#) or [HAVING](#) clause in each *query* argument to group the returned data.

You can use an [ORDER BY](#) clause at the end of the last *query* argument to display the returned data in a specified order.

See Also

[ALL, DISTINCT, DISTINCTROW, TOP Predicates \(Microsoft Jet SQL\)](#) [ORDER BY Clause \(Microsoft Jet SQL\)](#)

[GROUP BY Clause \(Microsoft Jet SQL\)](#) [SELECT Statement \(Microsoft Jet SQL\)](#)

[HAVING Clause \(Microsoft Jet SQL\)](#) [SQL Subqueries](#)

[INNER JOIN Operation \(Microsoft Jet SQL\)](#) [WHERE Clause \(Microsoft Jet SQL\)](#)

[LEFT JOIN, RIGHT JOIN Operations \(Microsoft Jet SQL\)](#)

Example

[UNION Operation Example](#)

PARAMETERS Declaration

Declares the name and data type of each [parameter](#) in a [parameter query](#).

Syntax

PARAMETERS *name datatype* [, *name datatype* [, ...]]

The PARAMETERS declaration has these parts:

Part	Description
<i>name</i>	The name of the parameter. Assigned to the Name property of the Parameter object and used to identify this parameter in the Parameters collection. You can use <i>name</i> as a string that is displayed in a dialog box while your application runs the query. Use brackets ([]) to enclose text that contains spaces or punctuation. For example, [Low price] and [Begin report with which month?] are valid <i>name</i> arguments.
<i>datatype</i>	One of the primary Microsoft Jet SQL data types or their synonyms.

Remarks

For queries that you run regularly, you can use a PARAMETERS declaration to create a parameter query. A parameter query can help automate the process of changing query [criteria](#). With a parameter query, your code will need to provide the parameters each time the query is run.

The PARAMETERS declaration is optional but when included precedes any other statement, including [SELECT](#).

If the declaration includes more than one parameter, separate them with commas. The following example includes two parameters:

```
PARAMETERS [Low price] Currency, [Beginning date] DateTime;
```

You can use *name* but not *datatype* in a [WHERE](#) or [HAVING](#) clause. The following example expects two parameters to be provided and then applies the criteria to records in the Orders table:

```
PARAMETERS [Low price] Currency,
```

```
[Beginning date] DateTime;  
SELECT OrderID, OrderAmount  
FROM Orders  
WHERE OrderAmount > [Low price]  
AND OrderDate >= [Beginning date];
```

See Also

[EXECUTE Statement \(Microsoft Jet SQL\)](#) [SELECT Statement \(Microsoft Jet SQL\)](#)

[HAVING Clause \(Microsoft Jet SQL\)](#) [WHERE Clause \(Microsoft Jet SQL\)](#)

[Microsoft Jet Database Engine SQL Data Types](#)

Example

[PARAMETERS Declaration Example](#)

WITH OWNERACCESS OPTION Declaration

In a multiuser environment with a [secure workgroup](#), use this declaration with a query to give the user who runs the query the same [permissions](#) as the query's owner.

Syntax

```
sqlstatement WITH OWNERACCESS OPTION
```

Remarks

The WITH OWNERACCESS OPTION declaration is optional.

The following example enables the user to view salary information (even if the user does not otherwise have permission to view the Payroll table), provided that the query's owner does have that permission:

```
SELECT LastName,  
FirstName, Salary  
FROM Employees  
ORDER BY LastName  
WITH OWNERACCESS OPTION;
```

If a user is otherwise prevented from creating or adding to a table, you can use WITH OWNERACCESS OPTION to enable the user to run a [make-table](#) or [append query](#).

If you want to enforce workgroup security settings and users' permissions, do not include the WITH OWNERACCESS OPTION declaration.

This option requires you to have access to the System.mdw file associated with the database. It is useful only in secured multiuser implementations.

See Also

[SELECT Statement \(Microsoft Jet SQL\)](#)

PROCEDURE Clause

Defines a name and optional [parameters](#) for a query.

Note The PROCEDURE clause has been superseded by the PROCEDURE statement. Although the PROCEDURE clause is still supported, the PROCEDURE statement provides a superset of the capability of the PROCEDURE clause and is the recommended syntax.

Syntax

PROCEDURE *name* [*param1 datatype*[, *param2 datatype*[, ...]]

The PROCEDURE clause has these parts:

Part	Description
<i>name</i>	A name for the procedure. It must follow standard naming conventions .
<i>param1, param2</i>	One or more field names or parameters . For example: PROCEDURE Sales_By_Country [Beginning Date] DateTime, [Ending Date] DateTime; For more information on parameters, see parameters .
<i>datatype</i>	One of the primary Microsoft Jet SQL data types or their synonyms.

Remarks

An SQL procedure consists of a PROCEDURE clause (which specifies the name of the procedure), an optional list of parameter definitions, and a single [SQL statement](#). For example, the procedure Get_Part_Number might run a query that retrieves a specified part number.

Notes

If the clause includes more than one field definition (that is, *param-datatype* pairs), separate them with commas.

The PROCEDURE clause must be followed by an SQL statement (for example, a [SELECT](#) or [UPDATE](#) statement).

See Also

[DELETE Statement \(Microsoft Jet SQL\)](#) [PARAMETERS Declaration \(Microsoft Jet SQL\)](#)

[EXECUTE Statement \(Microsoft Jet SQL\)](#) [SELECT Statement \(Microsoft Jet SQL\)](#)

[Microsoft Jet Database Engine SQL Data Types](#) [UPDATE Statement \(Microsoft Jet SQL\)](#)

Example

[CREATE PROCEDURE Statement, PROCEDURE Clause Example](#)

SQL Subqueries

A subquery is a [SELECT](#) statement nested inside a [SELECT](#), [SELECT...INTO](#), [INSERT...INTO](#), [DELETE](#), or [UPDATE](#) statement or inside another subquery.

Syntax

You can use three forms of syntax to create a subquery:

comparison [ANY | ALL | SOME] (*sqlstatement*)

expression [NOT] IN (*sqlstatement*)

[NOT] EXISTS (*sqlstatement*)

A subquery has these parts:

Part	Description
<i>comparison</i>	An expression and a comparison operator that compares the expression with the results of the subquery.
<i>expression</i>	An expression for which the result set of the subquery is searched.
<i>sqlstatement</i>	A SELECT statement, following the same format and rules as any other SELECT statement. It must be enclosed in parentheses.

Remarks

You can use a subquery instead of an expression in the field list of a [SELECT](#) statement or in a [WHERE](#) or [HAVING](#) clause. In a subquery, you use a [SELECT](#) statement to provide a set of one or more specific values to evaluate in the [WHERE](#) or [HAVING](#) clause expression.

Use the [ANY](#) or [SOME](#) predicate, which are synonymous, to retrieve records in the main query that satisfy the comparison with any records retrieved in the subquery. The following example returns all products whose unit price is greater than that of any product sold at a discount of 25 percent or more:

```
SELECT * FROM Products
WHERE UnitPrice > ANY
```

```
(SELECT UnitPrice FROM OrderDetails
WHERE Discount >= .25);
```

Use the [ALL](#) predicate to retrieve only those records in the main query that satisfy the comparison with all records retrieved in the subquery. If you changed ANY to ALL in the previous example, the query would return only those products whose unit price is greater than that of all products sold at a discount of 25 percent or more. This is much more restrictive.

Use the IN predicate to retrieve only those records in the main query for which some record in the subquery contains an equal value. The following example returns all products with a discount of 25 percent or more:

```
SELECT * FROM Products
WHERE ProductID IN
(SELECT ProductID FROM OrderDetails
WHERE Discount >= .25);
```

Conversely, you can use NOT IN to retrieve only those records in the main query for which no record in the subquery contains an equal value.

Use the EXISTS predicate (with the optional NOT reserved word) in true/false comparisons to determine whether the subquery returns any records.

You can also use table name [aliases](#) in a subquery to refer to tables listed in a [FROM](#) clause outside the subquery. The following example returns the names of employees whose salaries are equal to or greater than the average salary of all employees having the same job title. The Employees table is given the alias "T1":

```
SELECT LastName,
FirstName, Title, Salary
FROM Employees AS T1
WHERE Salary >=
(SELECT Avg(Salary)
FROM Employees
WHERE T1.Title = Employees.Title) Order by Title;
```

In the preceding example, the AS [reserved word](#) is optional.

Some subqueries are allowed in [crosstab queries](#) — specifically, as predicates

(those in the WHERE clause). Subqueries as output (those in the SELECT list) are not allowed in crosstab queries.

See Also

[ALL, DISTINCT, DISTINCTROW, TOP Predicates \(Microsoft Jet SQL\)](#) [SELECT Statement \(Microsoft Jet SQL\)](#)
[DELETE Statement \(Microsoft Jet SQL\)](#) [SELECT INTO Statement \(Microsoft Jet SQL\)](#)
[HAVING Clause \(Microsoft Jet SQL\)](#) [UNION Operation \(Microsoft Jet SQL\)](#)
[INNER JOIN Operation \(Microsoft Jet SQL\)](#) [UPDATE Statement \(Microsoft Jet SQL\)](#)
[INSERT INTO Statement \(Microsoft Jet SQL\)](#) [WHERE Clause \(Microsoft Jet SQL\)](#)
[LEFT JOIN, RIGHT JOIN Operations \(Microsoft Jet SQL\)](#)

Example

[SQL Subqueries Example](#)

Customizing Windows Registry Settings for Microsoft Jet

If your application cannot work correctly with the default functionality of the [Microsoft Jet database engine](#), you may have to change the settings in the Microsoft® Windows® Registry to suit your needs. The Windows Registry can also be used to tune the operation of the [installable ISAM](#) and [ODBC driver](#).

You can customize the settings in the Windows Registry in four different ways:

[Using Regedit.exe to Overwrite the Default Settings](#)

[Creating a Portion in Your Application's Registry Tree to Manage the Settings](#)

[Using the SetOption Method from DAO](#)

[Using the Connection Properties in the Microsoft OLE DB Provider for Jet](#)

You can also edit the Windows Registry to specify the following:

Settings used for interaction with Microsoft Excel, Lotus, Paradox, and dBASE databases. See [Initializing the Paradox Database Driver](#), [Initializing the Microsoft Excel Driver](#), [Initializing the Lotus Driver](#), and [Initializing the dBASE Database Driver](#).

Settings used by Microsoft ODBC for interaction with SQL databases. See [Initializing the Microsoft ODBC Database Driver](#).

Settings that affect how the Microsoft Jet database engine reads and saves data. See [Initializing the Microsoft Jet 2.5 Database Engine Driver](#), [Initializing the Microsoft Jet 3.5 Database Engine Driver](#), and [Initializing the Microsoft Jet 4.0 Database Engine Driver](#).

Settings used to control how the Microsoft Jet database engine interacts with the Internet and Microsoft Exchange. See [Initializing the Text and HTML Data Source Driver](#) and [Initializing the Microsoft Exchange Data Source Driver](#).

Settings used to control how the Microsoft Jet database engine interacts with data imported as plain text. See [Initializing the Text and HTML Data Source Driver](#).

See Also

[Initializing the dBase Database Driver](#)

[Initializing the Lotus Driver](#)

[Initializing the Microsoft Jet 2.5 Database Engine Driver](#)

[Initializing the Microsoft Jet 3.5 Database Engine Driver](#)

[Initializing the Microsoft Jet 4.0 Database Engine Driver](#)

[Configuring the Microsoft Jet Database Engine for ODBC Access](#)

[Initializing the Paradox Database Driver](#)

[Initializing the Text and HTML Data Source Driver](#)

[Initializing the Microsoft Exchange Data Source Driver](#)

[Initializing the Microsoft Excel Driver](#)

Initializing the dBASE Database Driver

When you install the dBASE database driver, the Setup program writes a set of default values to the Microsoft® Windows® Registry in the Engines and ISAM Formats subkeys. You should not modify these settings directly; use the setup program for your application to add, remove, or change these settings. The following sections describe initialization and ISAM format settings for the dBASE database driver.

The Paradox database driver will work in one of two modes, depending upon whether the Borland Database Engine (BDE) is installed. Paradox data is only updateable with the BDE. Without the BDE, the Paradox data can be Read, Exported, or Linked to read-only.

dBASE Initialization Settings

The Jet\4.0\Engines\Xbase folder includes initialization settings for the msxbde40.dll driver, used for access to external dBASE data sources. Typical settings for the entries in this folder are shown in the following example.

```
win32=<path>\msxbde40.dll
NetworkAccess=On
PageTimeout=600
INFPATH=C:\DBASE\SYSTEM
CollatingSequence=ASCII
DataCodePage=OEM
Deleted=On
DbcsStr=On
Century=Off
Date=MDY
Mark=47
Exact=Off
```

The Microsoft Jet database engine uses the Xbase folder entries as follows.

Entry	Description
win32	The location of Msxbse35.dll. The full path is determined

	at the time of installation. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
NetworkAccess	A binary indicator for file locking preference. If NetworkAccess is set to 00, tables are opened for exclusive access, regardless of the settings of the OpenDatabase and OpenRecordset methods' <i>exclusive</i> argument. The default value is 01. Values are of type Binary for Windows 95 and Windows NT 4.0, and of type REG_BINARY for Windows NT 3.51.
PageTimeout	The length of time between when data is placed in an internal cache and when it is invalidated. The value is specified in 100 millisecond units. The default is 600 units or 60 seconds. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.
INFPATH	The full path to the .inf file directory. The Microsoft Jet database engine first looks for an .inf file in the directory containing the table. If the .inf file is not in the database directory, it looks in the INFPATH. If there is no INFPATH, it uses whatever index files (.cdx or .mdx) it finds in the database directory. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
	This entry is not written by the installation procedure.
CollatingSequence	This setting is only used if the BDE is not present on the machine. The collating sequence for all dBASE tables created or opened using the Microsoft Jet database engine. Possible values are ASCII and International. The default is ASCII. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
DataCodePage	This setting is only used if the BDE is not present on the machine. An indicator of how text pages are stored. Possible settings

are:

- OEM — OemToAnsi and AnsiToOem conversions done.
- ANSI — OemToAnsi and AnsiToOem conversions not done.

The default is OEM. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.

Deleted	A binary indicator that determines how records marked for deletion are handled by the Microsoft Jet database engine. A value of 01 corresponds to the dBASE command SET DELETED ON and indicates never to retrieve or position on a deleted record. A value of 00 corresponds to the dBASE command SET DELETED OFF and indicates to treat a deleted record like any other record. The default is 00. Values are of type Binary for Windows 95 and Windows NT 4.0, and of type REG_BINARY for Windows NT 3.51.
DdbcStr	A binary indicator that determines how the string functions will handle Far East character set data. A value of 01 corresponds to the dBASE command SET KANJISTRING ON and indicates that strings should be treated as a dbcs character stream. A value of 00 indicates that strings should be treated as a simple byte stream.
Century	A binary indicator for formatting the century component of dates in cases where date-to-string functions are used in index expressions. A value of 01 corresponds to the dBASE command SET CENTURY ON and a value of 00 corresponds to the dBASE command SET CENTURY OFF. The default is 00. Values are of type Binary for Windows 95 and Windows NT 4.0, and of type REG_BINARY for Windows NT 3.51.
Date	The date formatting style to use in cases where date-to-string functions are used in index expressions. The possible settings for this entry, which corresponds to the dBASE SET DATE command, are American, ANSI, British,

French, DMY, German, Italian, Japan, MDY, USA, and YMD. The default is MDY. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.

Mark

The decimal value of the ASCII character used to separate date parts. The default depends on the Date setting as follows:

- "/" (American, MDY)
- "." (ANSI)
- "/" (British, French, DMY)
- "." (German)
- "-" (Italian)
- "/" (Japan, YMD)
- "-" (USA)

A value of 0 specifies that the system should use the separator usually associated with the selected date format.

The default is 0. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

Exact

A binary indicator for string comparisons. A value of 01 corresponds to the dBASE command SET EXACT ON. A value of 00 corresponds to the dBASE command SET EXACT OFF. The default is 00. Values are of type Binary for Windows 95 and Windows NT 4.0, and of type REG_BINARY for Windows NT 3.51.

dBASE ISAM Formats

The Jet\4.0\ISAM Formats\dBASE III folder contains the following entries.

	Windows NT 3.51	Windows 95 and Windows NT 4.0	
--	----------------------------	--	--

Entry name	Type	Type	Value
Engine	REG_SZ	String	Xbase
ExportFilter	REG_SZ	String	dBASE III (*.dbf)
ImportFilter	REG_SZ	String	dBASE III (*.dbf)
CanLink	REG_BINARY	Binary	01
OneTablePerFile	REG_BINARY	Binary	01
IsamType	REG_DWORD	DWORD	0
IndexDialog	REG_BINARY	Binary	01
IndexFilter	REG_SZ	String	dBASE Index (*.ndx)
CreateDBOnExport	REG_BINARY	Binary	00
ResultTextImport	REG_SZ	String	Import data from the external file into the current database. Changing data in the current database will not change data in the external file.
ResultTextLink	REG_SZ	String	Create a table in the current database that is linked to the external file. Changing data in the current database will change data in the external file.
ResultTextExport	REG_SZ	String	Export data from the current database into a dBASE III file. This process will overwrite the data if exported to an existing file.
SupportsLongNames	REG_BINARY	Binary	00

The Jet\4.0\ISAM Formats\dBASE IV folder contains the following entries.

	Windows NT	Windows 95 and	

Entry name	3.51 Type	Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Xbase
ExportFilter	REG_SZ	String	dBASE IV (*.dbf)
ImportFilter	REG_SZ	String	dBASE IV (*.dbf)
CanLink	REG_BINARY	Binary	01
OneTablePerFile	REG_BINARY	Binary	01
IsamType	REG_DWORD	DWORD	0
IndexDialog	REG_BINARY	Binary	01
IndexFilter	REG_SZ	String	dBASE Index (*.ndx; *.mdx)
CreateDBOnExport	REG_BINARY	Binary	00
ResultTextImport	REG_SZ	String	Import data from the external file into the current database. Changing data in the current database will not change data in the external file.
ResultTextLink	REG_SZ	String	Create a table in the current database that is linked to the external file. Changing data in the current database will change data in the external file.
ResultTextExport	REG_SZ	String	Export data from the current database into a dBASE IV file. This process will overwrite the data if exported to an existing file.
SupportsLongNames	REG_BINARY	Binary	00

The Jet\4.0\ISAM Formats\dBASE 5.x folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Xbase
ExportFilter	REG_SZ	String	dBASE 5 (*.dbf)
ImportFilter	REG_SZ	String	dBASE 5 (*.dbf)
CanLink	REG_BINARY	Binary	01
OneTablePerFile	REG_BINARY	Binary	01
IsamType	REG_DWORD	DWORD	0
IndexDialog	REG_BINARY	Binary	01
IndexFilter	REG_SZ	String	dBASE Index (*.ndx; *.mdx)
CreateDBOnExport	REG_BINARY	Binary	00
ResultTextImport	REG_SZ	String	Import data from the external file into the current database. Changing data in the current database will not change data in the external file.
ResultTextLink	REG_SZ	String	Create a table in the current database that is linked to the external file. Changing data in the current database will change data in the external file.
ResultTextExport	REG_SZ	String	Export data from the current database into a dBASE 5 file. This process will overwrite the data if exported to an existing file.
SupportsLongNames	REG_BINARY	Binary	00

Note When you change Windows Registry settings, you must exit and then restart the database engine for the new settings to take effect.

See Also

[Customizing Windows Registry Settings for Microsoft Jet](#)

Initializing the Lotus Driver

When you install the Lotus database driver, the Setup program writes a set of default values to the Microsoft® Windows® Registry in the Engines and ISAM Formats subkeys. You should not modify these settings directly; use the setup program for your application to add, remove, or change these settings. The following sections describe initialization and ISAM Format settings for the Lotus database driver.

Lotus Initialization Settings

The Jet\4.0\Engines\Lotus folder includes initialization settings for the msltus40.dll driver, used for external access to Lotus spreadsheets. Typical settings for the entries under this heading are shown in the following example.

```
win32=<path>\msltus40.dll
TypeGuessRows=8
ImportMixedTypes=Text
AppendBlankRows=4
FirstRowHasNames=Yes
```

The Microsoft Jet database engine uses the Lotus folder entries as follows.

Entry	Description
win32	The location of msltus40.dll. The full path is determined at the time of installation. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
TypeGuessRows	The number of rows to be checked for the data type. The data type is determined based on the most frequently found data type in the selection. If there is a tie, the data type is determined in the following order: Number, Currency, Date, Text, Long Text. If data is encountered that does not match the data type guessed for the column, it is returned as a Null value. On import, if a column has mixed data types, the entire column will be converted according to the ImportMixedTypes setting.

The default number of rows to be checked is 8. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

ImportMixedTypes Can be set to MajorityType or Text. If set to MajorityType, columns of mixed data types will be cast to the predominate data type on import. If set to Text, columns of mixed data types will be cast to Text on import. The default is Text. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.

AppendBlankRows The number of blank rows to be appended to the end of a WK1 worksheet before new data is added. For example, if AppendBlankRows is set to 4, Microsoft Jet will append 4 blank rows to the end of the worksheet before appending rows that contain data. Integer values for this setting can range from 0 to 16; the default is 01 (one additional row appended). Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

FirstRowHasNames A binary value that indicates whether the first row of the table contains column names. A value of 01 indicates that, during import, column names are taken from the first row. A value of 00 indicates no column names in the first row; column names appear as F1, F2, F3, and so on. The default is 01. Values are of type Binary for Windows 95 and Windows NT 4.0, and of type REG_BINARY for Windows NT 3.51.

Lotus ISAM Formats

The Jet\4.0\ISAM Formats\Lotus WK1 folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Lotus
ExportFilter	REG_SZ	String	Lotus 1-2-3 WK1

ImportFilter	REG_SZ	String	(*.*wk1) Lotus 1-2-3 (*.wk*)
CanLink	REG_BINARY	Binary	00
OneTablePerFile	REG_BINARY	Binary	00
IsamType	REG_DWORD	DWORD	1
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	01
ResultTextImport	REG_SZ	String	Import data from the external file into the current database. Changing data in the current database will not change data in the external file.
ResultTextExport	REG_SZ	String	Export data from the current database into a Lotus 1-2-3 Version 2 file. This process will overwrite the data if exported to an existing file.
SupportsLongNames	REG_BINARY	Binary	01

The Jet\4.0\ISAM Formats\Lotus WK3 folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Lotus
ExportFilter	REG_SZ	String	Lotus 1-2-3 WK3 (*.wk3)
CanLink	REG_BINARY	Binary	00
OneTablePerFile	REG_BINARY	Binary	00
IsamType	REG_DWORD	DWORD	1

IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	01
ResultTextExport	REG_SZ	String	Export data from the current database into a Lotus 1-2-3 Version 3 file. This process will overwrite the data if exported to an existing file.
SupportsLongNames	REG_BINARY	Binary	01

The Jet\4.0\ISAM Formats\Lotus WK4 folder contains the following entries.

	Windows NT 3.51	Windows 95 and Windows NT 4.0	
Entry name	Type	Type	Value
Engine	REG_SZ	String	Lotus
CanLink	REG_BINARY	Binary	00
OneTablePerFile	REG_BINARY	Binary	00
IsamType	REG_DWORD	DWORD	1
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	01
SupportsLongNames	REG_BINARY	Binary	01

The Jet\4.0\ISAM Formats\Lotus WJ2 folder contains the following entries.

	Windows NT 3.51	Windows 95 and Windows NT 4.0	
Entry name	Type	Type	Value
Engine	REG_SZ	String	Lotus
ExportFilter	REG_SZ	String	Lotus 1-2-3 WJ2 (*.wj2)
ImportFilter	REG_SZ	String	Lotus 1-2-3/DOS

			(*.wj*)
CanLink	REG_BINARY	Binary	00
OneTablePerFile	REG_BINARY	Binary	00
IsamType	REG_DWORD	DWORD	1
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	01
ResultTextExport	REG_SZ	String	Export data from the current database into a Lotus 1-2-3 Version 2 file. This process will overwrite the data if exported to an existing file.
SupportsLongNames	REG_BINARY	Binary	01

The Jet\4.0\ISAM Formats\Lotus WJ3 folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Lotus
CanLink	REG_BINARY	Binary	00
OneTablePerFile	REG_BINARY	Binary	00
IsamType	REG_DWORD	DWORD	1
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	01
SupportsLongNames	REG_BINARY	Binary	01

Note When you change Windows Registry settings, you must exit and then restart the database engine for the new settings to take effect.

See Also

[Customizing Windows Registry Settings for Microsoft Jet](#)

Initializing the Microsoft Excel Driver

When you install the Microsoft® Excel driver, the Setup program writes a set of default values to the Microsoft Windows® Registry in the Engines and ISAM Formats subkeys. You should not modify these settings directly; use the setup program for your application to add, remove, or change these settings. The following sections describe initialization and ISAM Format settings for the Microsoft Excel database driver.

Microsoft Excel Initialization Settings

The Jet\4.0\Engines\Excel folder includes initialization settings for the msexcl40.dll driver, used for external access to Microsoft Excel worksheets. Typical settings for the entries in this folder are shown in the following example.

```
win32=<path>\ msexcl40.dll
TypeGuessRows=8
ImportMixedTypes=Text
AppendBlankRows=1
FirstRowHasNames=Yes
```

The Microsoft Jet database engine uses the Excel folder entries as follows.

Entry	Description
win32	The location of msexcl40.dll. The full path is determined at the time of installation. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
TypeGuessRows	The number of rows to be checked for the data type. The data type is determined given the maximum number of kinds of data found. If there is a tie, the data type is determined in the following order: Number, Currency, Date, Text, Boolean. If data is encountered that does not match the data type guessed for the column, it is returned as a Null value. On import, if a column has mixed data types, the entire column will be cast according to the ImportMixedTypes setting.

The default number of rows to be checked is 8. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

ImportMixedTypes Can be set to MajorityType or Text. If set to MajorityType, columns of mixed data types will be cast to the predominate data type on import. If set to Text, columns of mixed data types will be cast to Text on import. The default is Text. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.

AppendBlankRows The number of blank rows to be appended to the end of a Version 3.5 or Version 4.0 worksheet before new data is added. For example, if AppendBlankRows is set to 4, Microsoft Jet will append 4 blank rows to the end of the worksheet before appending rows that contain data. Integer values for this setting can range from 0 to 16; the default is 01 (one additional row appended). Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

FirstRowHasNames A binary value that indicates whether the first row of the table contains column names. A value of 01 indicates that, during import, column names are taken from the first row. A value of 00 indicates no column names in the first row; column names appear as F1, F2, F3, and so on. The default is 01. Values are of type Binary for Windows 95 and Windows NT 4.0, and of type REG_BINARY for Windows NT 3.51.

Microsoft Excel ISAM Formats

The Jet\4.0\ISAM Formats\Excel 3.0 folder contains the following entries.

Entry name	Windows NT Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Excel
ExportFilter	REG_SZ	String	Microsoft Excel 3 (*.xls)

CanLink	REG_BINARY	Binary	01
OneTablePerFile	REG_BINARY	Binary	00
IsamType	REG_DWORD	DWORD	1
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	01
ResultTextExport	REG_SZ	String	Export data from the current database into a Microsoft Excel 3.0 file. This process will overwrite the data if exported to an existing file.
SupportsLongNames	REG_BINARY	Binary	01

The Jet\4.0\ISAM Formats\Excel 4.0 folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Excel
ExportFilter	REG_SZ	String	Microsoft Excel 4 (*.xls)
CanLink	REG_BINARY	Binary	01
OneTablePerFile	REG_BINARY	Binary	00
IsamType	REG_DWORD	DWORD	1
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	01
ResultTextExport	REG_SZ	String	Export data from the current database into a Microsoft Excel 4.0 file. This process will overwrite the data if exported to an existing file.

The Jet\4.0\ISAM Formats\Excel 5.0 folder contains the following entries, which apply to Microsoft Excel versions 5.0 and 7.0.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Excel
ExportFilter	REG_SZ	String	Microsoft Excel 5-7 (*.xls)
ImportFilter	REG_SZ	String	Microsoft Excel (*.xls)
CanLink	REG_BINARY	Binary	01
OneTablePerFile	REG_BINARY	Binary	00
IsamType	REG_DWORD	DWORD	1
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	01
ResultTextImport	REG_SZ	String	Import data from the external file into the current database. Changing data in the current database will not change data in the external file.
ResultTextLink	REG_SZ	String	Create a table in the current database that is linked to the external file. Changing data in the current database will change data in the external file.
ResultTextExport	REG_SZ	String	Export data from the current database into a Microsoft Excel 5.0 file. This process will

overwrite the data if exported to an existing file.

SupportsLongNames REG_BINARY Binary 01

The Jet\4.0\ISAM Formats\Excel 8.0 folder contains the following entries, which apply to Microsoft Excel 97.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Excel
ExportFilter	REG_SZ	String	Microsoft Excel 97-2000 (*.xls)
CanLink	REG_BINARY	Binary	01
OneTablePerFile	REG_BINARY	Binary	00
IsamType	REG_DWORD	DWORD	1
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	01
ResultTextExport	REG_SZ	String	Export data from the current database into a Microsoft Excel 97 file. This process will overwrite the data if exported to an existing file.
SupportsLongNames	REG_BINARY	Binary	01

Note When you change Windows Registry settings, you must exit and then restart the database engine for the new settings to take effect.

See Also

[Customizing Windows Registry Settings for Microsoft Jet](#)

Initializing the Microsoft Exchange Data Source Driver

When you install the Microsoft® Exchange Data Source driver, the Setup program writes a set of default values to the Microsoft Windows® Registry in the Engines and ISAM Formats subkeys. You should not modify these settings directly; use the setup program for your application to add, remove, or change these settings. The following sections describe initialization and ISAM Format settings for the Microsoft Exchange Data Source driver.

Microsoft Exchange Data Source Initialization Settings

The Jet\4.0\Engines\Exchange folder includes initialization settings for the msexch40.dll driver, used for external access to Microsoft Outlook and Microsoft Exchange folders. The only entry in this folder is the following:

```
win32=<path>\msexch40.dll
```

The Microsoft Jet database engine uses this setting to indicate the location of msexch40.dll. The full path is determined at the time of installation. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.

The results of using the Outlook ISAM format and of using the Exchange client ISAM format are similar. The only difference is that the two different clients use different names for the same columns. The two ISAM formats have been created so that Microsoft Jet can return the column names in the particular style that the user desires.

Microsoft Outlook Client ISAM Formats

The Jet\4.0\ISAM Formats\Exchange 9.0 folder contains the following entries.

	Windows NT 3.51	Windows 95 and Windows NT 4.0	
Entry name	Type	Type	Value
Engine	REG_SZ	String	Exchange
ImportFilter	REG_SZ	String	Outlook()
CanLink	REG_BINARY	Binary	01

OneTablePerFile	REG_BINARY Binary	00
IsamType	REG_DWORD DWORD	3
IndexDialog	REG_BINARY Binary	00
CreateDBOnExport	REG_BINARY Binary	00
SupportsLongNames	REG_BINARY Binary	01

Note When you change Windows Registry settings, you must exit and then restart the database engine for the new settings to take effect.

Microsoft Exchange Client ISAM Formats

The Jet\4.0\ISAM Formats\Exchange 4.0 folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Exchange
ImportFilter	REG_SZ	String	Exchange()
CanLink	REG_BINARY Binary		01
OneTablePerFile	REG_BINARY Binary		00
IsamType	REG_DWORD DWORD		3
IndexDialog	REG_BINARY Binary		00
CreateDBOnExport	REG_BINARY Binary		00
SupportsLongNames	REG_BINARY Binary		01

Note When you change Windows Registry settings, you must exit and then restart the database engine for the new settings to take effect.

Customizing the Schema.ini File for Outlook and Exchange Data

The Schema.ini file is used by the Outlook and Exchange ISAM in much the same way that it is used by the Text ISAM. Schema.ini contains the specifics of

a data source: how the data is formatted, and the names of columns that should be accessed.

It is not necessary to modify the Schema.ini file before data can be read, imported, or exported for Outlook and Exchange. Many of the settings inside the Schema.ini file for Outlook and Exchange are specific to internal tags that MAPI requires. You should not attempt to modify those tag values.

See Also

[Customizing Windows Registry Settings for Microsoft Jet](#)

Initializing the Microsoft Jet 4.0 Database Engine Driver

When you install the Microsoft® Jet version 4.0 Engine database driver, the Setup program writes a set of default values to the Microsoft Windows® Registry in the Engines and ISAM Formats subkeys. You must use the Registry Editor to add, remove, or change these settings. The following sections describe initialization and ISAM Format settings for the Microsoft Jet Engine database driver.

Microsoft Jet Engine Initialization Settings

The Jet\4.0\Engines folder includes initialization settings for the msjet40.dll database engine, used for access to Microsoft Access databases. Typical initialization settings for the entries in this folder are shown in the following example.

SystemDB = <path>\System.mdb

CompactBYPkey = 1

PrevFormatCompactWithUNICODECompression=1

The Microsoft Jet database engine uses the following entries.

Entry	Description
SystemDB	Specifies the full path and file name of the workgroup information file. The default is the appropriate path followed by the file name System.mdb. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
CompactByPKey	Specifies that when you compact tables they are copied in primary-key order, if a primary key exists on the table. If no primary key

exists on a table, the tables are copied in base-table order.

A value of 0 indicates that tables should be compacted in base-table order; a non-zero value indicates that tables should be compacted in primary-key order, if a primary key exists. The default value is non-zero. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

Note This setting only applies to compacting databases created with the Microsoft Jet database engine version 3.0 or later; when you compact databases created with the Microsoft Jet database engine version 2.x, the data is always copied in the order of the base table.

PrevFormatCompactWithUNICODECompression Microsoft Jet 4.0 databases use the Unicode character set to store textual data. Compressing the Unicode data can significantly improve the performance of the database because of the reduced number of page read/write operations that are needed afterwards.

This key determines if databases created by the Microsoft Jet database engine version 3.x or earlier should be created with compressed Unicode or uncompressed Unicode.

Note This setting does not apply to compacting Microsoft Jet 4.0 databases. Microsoft Jet 4.0 databases will default to keep the compression settings with which they were created.

The Jet\4.0\Engines\Jet 4.0 folder includes initialization settings for the msjet40.dll database engine, used for access to Microsoft Access databases. Typical initialization settings for the entries in this folder are shown in the following example.

```
FlushTransactionTimeout=500
LockDelay=100
LockRetry=20
MaxBufferSize= 0
MaxLocksPerFile= 9500
PageTimeout=5000
Threads=3
UserCommitSync=Yes
ImplicitCommitSync=No
ExclusiveAsyncDelay=2000
SharedAsyncDelay=0
RecycleLVs=0
PagesLockedToTableLock=0
```

The Microsoft Jet database engine uses the following entries.

Entry	Description
-------	-------------

PageTimeout	The length of time between the time when data that is not read-locked is placed in an internal cache and the time when it is invalidated, expressed in milliseconds. The default is 5000 milliseconds or 5 seconds. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.
FlushTransactionTimeout	This entry disables both the ExclusiveAsyncDelay and SharedAsyncDelay registry entries. To enable those entries, a value of zero must be entered. FlushTransactionTimeout changes the Microsoft Jet database engine's method for doing asynchronous writes to a database file. Previously, the Microsoft Jet database engine would use either the ExclusiveAsyncDelay or SharedAsyncDelay to determine how long it would wait before forcing asynchronous writes. FlushTransactionTimeout changes that behavior by having a value that will start asynchronous writes only after the specified amount of time has expired and no pages have been added to the cache. The only exception to this is if the cache exceeds the MaxBufferSize, at which point the cache will start asynchronous writing regardless of whether or not the time has expired. Microsoft Jet 3.5 database engine will wait 500 milliseconds of non-activity or until the cache size is exceeded before starting asynchronous writes.
LockDelay	This setting works in conjunction with the LockRetry setting in that it causes each LockRetry to wait 100 milliseconds before issuing another lock request. The LockDelay setting was added to prevent bursting that would occur with certain networking operating systems.
MaxLocksPerFile	This setting prevents transactions in Microsoft Jet from exceeding the specified value. If the locks in a transaction attempt to exceed this value, then the transaction is split into two or more parts and partially committed. This setting was added to

prevent Netware 3.1 server crashes when the specified Netware lock limit was exceeded, and to improve performance with both Netware and NT.

LockRetry

The number of times to repeat attempts to access a locked page before returning a lock conflict message. The default is 20. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

RecycleLVs

This setting, when enabled, will cause Microsoft Jet to recycle long value (LV) pages (Memo, Long Binary [OLE object], and Binary data types). Microsoft Jet 3.0 would not recycle those types of pages until the last user closed the database. If the RecycleLVs setting is enabled, Microsoft Jet 3.5 will start to recycle most LV pages when the database is expanded (that is, when groups of pages are added).

Note By enabling this feature, users will notice a performance degradation when manipulating long value data types. Microsoft Access 97 automatically enables and disables this feature when manipulating modules, forms, and reports, thus eliminating the need to turn it on when modifying those objects. The default value is 0. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

MaxBufferSize

The size of the database engine internal cache, measured in kilobytes (K). MaxBufferSize must be an integer value greater than or equal to 512. The default is based on the following formula:

$$((\text{TotalRAM in MB} - 12 \text{ MB}) / 4) + 512 \text{ KB}$$

For example, on a system with 32 MB of RAM, the default buffer size is $((32 \text{ MB} - 12 \text{ MB}) / 4) + 512 \text{ KB}$ or 5632 KB. To set the value to the default, set the registry key to

MaxBufferSize=

Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

Threads	The number of background threads available to the Microsoft Jet database engine. The default is 3. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.
UserCommitSync	Specifies whether the system waits for a commit to finish. A value of Yes instructs the system to wait; a value of No instructs the system to perform the commit asynchronously. The default is Yes. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
ImplicitCommitSync	Specifies whether the system waits for a commit to finish. A value of No instructs the system to proceed without waiting for the commit to finish; a value of Yes instructs the system to wait for the commit to finish. The default is No. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
ExclusiveAsyncDelay	Specifies the length of time, in milliseconds, to defer an asynchronous flush of an exclusive database. The default value is 2000, or 2 seconds. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.
SharedAsyncDelay	Specifies the length of time, in milliseconds, to defer an asynchronous flush of a shared database. The default value is 0. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.
PagesLockedToTableLock	During bulk operations it is often more efficient to lock a whole table, instead of obtaining locks for each individual page of the table as you try to access it.

This setting specifies the number of pages that Microsoft Jet will allow to be locked in any particular transaction before Jet attempts to escalate to an exclusive table lock

The default value of 0 indicates that Jet will never automatically change from page locking to table locking.

Note This setting should be used carefully. If a database is needed for multi-user access, then locking a whole table could cause locking conflicts for other users. This would be especially severe if a small number was used for this setting. Even when a larger number was used, such as 25 or 50, the operation for other users might become unpredictable.

Microsoft Jet Engine Replication Settings

The Jet\4.0\Transporter key includes initialization settings for Jet Replication synchronizations via the Jet Synchronizer. These settings determine the order that specific transports will be attempted by Synchronizer synchs. The valid values are 0-100. A value of 0 implies that a transport should not be attempted. All transport types with non-zero values will be attempted in ascending key value order until one successfully executes the synchronization or until all attempts fail.

Typical initialization settings for the entries in this folder are shown in the following example.

Priority_FS=1

Priority_Internet=2

Priority_direct=3

The Microsoft Jet database engine uses the following [priority](#) entries.

Entry	Description
Priority_FS	File System (Indirect) Synchronization relies on a series of

message exchanges between replicas. The Synchronizer that manages each replica collects changes into message files (*.msg), which are then copied to a shared folder called a dropbox, used by the partner Synchronizer and accessible via the file system.

Priority_Internet [Internet \(Indirect\) Synchronization](#) relies on a series of message exchanges between replicas. Only the replica managed by the Synchronizer on the Internet Server has a dropbox that is accessible via a HTTP or FTP connection.

Priority_direct [Direct synchronization](#) is the process of exchanging data and design changes between two members of a replica set that are directly connected, either on the same computer or over a network. The Synchronizer opens both databases.

Note When you change Windows Registry settings, you must exit and then restart the database engine for the new settings to take effect.

See Also

[Customizing Windows Registry Settings for Microsoft Jet](#)

Initializing the Microsoft Jet 3.5 Database Engine Driver

When you install the Microsoft® Jet version 3.5 Engine database driver, the Setup program writes a set of default values to the Microsoft Windows® Registry in the Engines and ISAM Formats subkeys. You must use the Registry Editor to add, remove, or change these settings. The following sections describe initialization and ISAM Format settings for the Microsoft Jet Engine database driver.

Microsoft Jet Engine Initialization Settings

The Jet\4.0\Engines\Jet 3.x folder includes initialization settings for the msrd3x40.dll driver, used for access to Microsoft Access 97 worksheets. Typical initialization settings for the entries in this folder are shown in the following example.

```
win32=<path>\msrd3x40.dll
FlushTransactionTimeout=500
LockDelay=100
LockRetry=20
MaxBufferSize= 0
MaxLocksPerFile= 9500
PageTimeout=5000
Threads=3
UserCommitSync=Yes
ImplicitCommitSync=No
ExclusiveAsyncDelay=2000
SharedAsyncDelay=0
RecycleLVs=0
SortMemorySource=0
```

The Microsoft Jet database engine uses the following entries.

Entry	Description
win32	Location of the database engine driver (.dll). The path is determined at the time of installation. Values are of type String for Windows 95 and Windows NT 4.0,

PageTimeout	<p>and of type REG_SZ for Windows NT 3.51.</p> <p>The length of time between the time when data that is not read-locked is placed in an internal cache and when it is invalidated, expressed in milliseconds. The default is 5000 milliseconds or 5 seconds. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.</p>
FlushTransactionTimeout	<p>This entry disables both the ExclusiveAsyncDelay and SharedAsyncDelay registry entries. To enable those entries, a value of zero must be entered.</p> <p>FlushTransactionTimeout changes the Microsoft Jet database engine's method for doing asynchronous writes to a database file. Previously, the Microsoft Jet database engine would use either the ExclusiveAsyncDelay or SharedAsyncDelay to determine how long it would wait before forcing asynchronous writes. FlushTransactionTimeout changes that behavior by having a value that will start asynchronous writes only after the specified amount of time has expired and no pages have been added to the cache. The only exception to this is if the cache exceeds the MaxBufferSize, at which point the cache will start asynchronous writing regardless if the time has expired. Microsoft Jet 3.5 database engine will wait 500 milliseconds of non-activity or until the cache size is exceeded before starting asynchronous writes.</p>
LockDelay	<p>This setting works in conjunction with the LockRetry setting in that it causes each LockRetry to wait 100 milliseconds before issuing another lock request. The LockDelay setting was added to prevent "bursting" that would occur with certain networking operating systems.</p>
MaxLocksPerFile	<p>This setting prevents transactions in Microsoft Jet from exceeding the specified value. If the locks in a transaction attempts to exceed this value, then the transaction is split into two or more parts and partially</p>

committed. This setting was added to prevent Netware 3.1 server crashes when the specified Netware lock limit was exceeded and to improve performance with both Netware and NT.

LockRetry

The number of times to repeat attempts to access a locked page before returning a lock conflict message. The default is 20. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

RecycleLVs

This setting, when enabled, will cause Microsoft Jet to recycle long value (LV) pages (Memo, Long Binary [OLE object], and Binary data types). Microsoft Jet 3.0 would not recycle those types of pages until the last user closed the database. If the RecycleLVs setting is enabled, Microsoft Jet 3.5 will start to recycle most LV pages when the database is expanded (that is, when groups of pages are added).

Note By enabling this feature, users will notice a performance degradation when manipulating long value data types. Microsoft Access 97 automatically enables and disables this feature when manipulating modules, forms, and reports, thus eliminating the need to turn it on when modifying those objects. The default value is 0. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

MaxBufferSize

The size of the database engine internal cache, measured in kilobytes (K). MaxBufferSize must be an integer value greater than or equal to 512. The default is based on the following formula:

$$((\text{TotalRAM in MB} - 12 \text{ MB}) / 4) + 512 \text{ KB}$$

For example, on a system with 32 MB of RAM, the default buffer size is $((32 \text{ MB} - 12 \text{ MB}) / 4) + 512 \text{ KB}$ or 5632 KB. To set the value to the default, set the registry key to

MaxBufferSize=

Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

Threads	The number of background threads available to the Microsoft Jet database engine. The default is 3. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.
UserCommitSync	Specifies whether the system waits for a commit to finish. A value of Yes instructs the system to wait; a value of No instructs the system to perform the commit asynchronously. The default is Yes. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
ImplicitCommitSync	Specifies whether the system waits for a commit to finish. A value of No instructs the system to proceed without waiting for the commit to finish; a value of Yes instructs the system to wait for the commit to finish. The default is No. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
ExclusiveAsyncDelay	Specifies the length of time, in milliseconds, to defer an asynchronous flush of an exclusive database. The default value is 2000 or 2 seconds. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.
SharedAsyncDelay	Specifies the length of time, in milliseconds, to defer an asynchronous flush of a shared database. The default value is 0. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.
SortMemorySource	Specifies how Microsoft Jet obtains the memory that is used for sort keys. A value of 0 indicates that memory should be taken from the heap. A value of 1 indicates that memory should be taken from global

memory using the malloc function call.

Microsoft Jet Engine ISAM Formats

The Jet\4.0\ISAM Formats\Jet 3.x folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Jet 3.x
OneTablePerFile	REG_BINARY	Binary	00
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	00
IsamType	REG_DWORD	DWORD	0

Note When you change Windows Registry settings, you must exit and then restart the database engine for the new settings to take effect.

See Also

[Customizing Windows Registry Settings for Microsoft Jet](#)

Initializing the Microsoft Jet 2.5 Database Engine Driver

When you install the Microsoft® Jet version 2.5 Engine database driver, the Setup program writes a set of default values to the Microsoft Windows® Registry in the Engines and ISAM Formats subkeys. You must use the Registry Editor to add, remove, or change these settings. The following sections describe initialization and ISAM Format settings for the Microsoft Jet Engine database driver.

Microsoft Jet Engine Initialization Settings

The Jet\4.0\Engines\Jet 2.x folder includes initialization settings for the msrd2x40.dll driver, used for access to Microsoft Access 2.0 worksheets. Typical initialization settings for the entries in this folder are shown in the following example.

```
win32=<path>\ msrd2x40.dll
PageTimeout=5
LockedPageTimeout=5
CursorTimeout=5
LockRetry=20
CommitLockRetry=20
MaxBufferSize=512
ReadAheadPages=16
IdleFrequency=10
ForceOsFlush = 0
```

The following entries are used to configure the Microsoft Jet database engine.

Entry	Description
win32	Location of the database engine driver (.dll). The path is determined at the time of installation. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
PageTimeout	The length of time between when data that is not read-locked is placed in an internal cache and when it is invalidated, expressed in 100 millisecond units. The

default is 5 units (or 0.5 seconds). Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

LockedPageTimeout The length of time between when data that is read-locked is placed in an internal cache and when it is invalidated, expressed in 100 millisecond units. The default is 5 units (or 0.5 seconds). Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

CursorTimeout The length of time a reference to a page will remain on that page, expressed in 100 millisecond units. The default is 5 units (or 0.5 seconds). This setting applies only to databases created with version 1.x of the Microsoft Jet database engine. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

LockRetry The number of times to repeat attempts to access a locked page before returning a lock conflict message. The default is 20 times; LockRetry is related to CommitLockRetry. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

CommitLockRetry The number of times the Microsoft Jet database engine attempts to acquire a lock on data to commit changes to that data. If the Microsoft Jet database engine cannot acquire a commit lock, changes to the data will be unsuccessful.

The number of attempts the Microsoft Jet database engine makes to get a commit lock is directly related to the LockRetry value. For each attempt made to acquire a commit lock, the Microsoft Jet database engine will make as many attempts as specified by the LockRetry value to acquire a lock. For example, if CommitLockRetry is set to 20 and LockRetry is set to 20, the Microsoft Jet database engine will try to acquire a commit lock as many as 20 times; for each of those attempts, the Microsoft Jet database engine will try to acquire a lock as many as 20 times, for a total of 400 attempts.

The default value for CommitLockRetry is 20. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

- MaxBufferSize The size of the database engine internal cache, measured in kilobytes (K). MaxBufferSize must be a whole number value between 9 and 4096, inclusive. The default is 512. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.
- ReadAheadPages The number of pages to read ahead when performing sequential scans. The default is 16. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.
- ForceOSFlush Any setting other than 0 means a commit or a write will force flushing the OS cache to disk. A setting of 0 (the default setting) means no force flush occurs. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.
- IdleFrequency The amount of time, in 100 millisecond units, that Microsoft Jet will wait before releasing a read lock. The default is 10 units or one second. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

Microsoft Jet Engine ISAM Formats

The Jet\4.0\ISAM Formats\Jet 2.x folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Jet 2.x
OneTablePerFile	REG_BINARY	Binary	00
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	00
IsamType	REG_DWORD	DWORD	0

Note When you change Windows Registry settings, you must exit and then restart the database engine for the new settings to take effect.

See Also

[Customizing Windows Registry Settings for Microsoft Jet](#)

Initializing the Paradox Database Driver

When you install the Paradox database driver, the Setup program writes a set of default values to the Microsoft® Windows® Registry in the Engines and ISAM Formats subkeys. You should not modify these settings directly (unless necessary); use the setup program for your application to add, remove, or change these settings. The following sections describe initialization and ISAM Format settings for the Paradox database driver.

The Paradox database driver will work in one of two modes, depending upon whether the Borland Database Engine (BDE) is installed. Paradox data is only updateable with the BDE. Without the BDE, the Paradox data can be Read, Exported, or Linked to read-only.

Paradox Initialization Settings

The Jet\4.0\Engines\Paradox folder includes initialization settings for the Mspdox35.dll driver, used for access to external Paradox data. Typical settings for the entries in this folder are shown in the following example.

```
win32=<path>\ mspbde40.dll
PageTimeout=600
CollatingSequence=ASCII
DataCodePage=OEM
ParadoxUserName=Kimberly
ParadoxNetPath=P:\PDOXDB
ParadoxNetStyle=4.X
```

The Microsoft Jet database engine uses the Paradox folder entries as follows.

Entry	Description
win32	The location of mspbde40.dll. The full path is determined at the time of installation. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
PageTimeout	The length of time between when data is placed in an internal cache and when it is invalidated. The value is specified in 100 millisecond units. The default is 600 units

or 60 seconds. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

CollatingSequence This setting is only used if the BDE is not present on the machine.

The collating sequence for all Paradox tables created or opened using the Microsoft Jet database engine. Possible values are ASCII, International, Norwegian-Danish, and Swedish-Finnish. The CollatingSequence entry must match the collating sequence used when the Paradox table was built. The default is ASCII. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.

DataCodePage This setting is only used if the BDE is not present on the machine.

An indicator of how text pages are stored. Possible settings are:

- OEM — OemToAnsi and AnsiToOem conversions done.
- ANSI — OemToAnsi and AnsiToOem conversions not done.

The default is OEM. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.

ParadoxUserName The name to be displayed by Paradox if a table is locked by the Paradox ISAM and an interactive user accessing the data from Paradox (rather than the ISAM) attempts to place an incompatible lock. This entry is not added by the setup program. You must create this entry. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.

Note If you indicate a ParadoxUserName, you must also specify a ParadoxNetPath and a ParadoxNetStyle or you will receive an error when trying to access external Paradox data. Also, if you are accessing a Paradox database in

multi-user mode over the network, you will need to add or modify this registry entry manually.

ParadoxNetPath The full path to the directory containing the PARADOX.NET file (for Paradox 3.x) or the PDOXUSRS.NET file (for Paradox 4.x). This entry is not added by the setup program. You must create this entry. The full ParadoxNetPath (including the drive letter) must be consistent for all users sharing a particular database (directory). Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.

Note If you indicate a ParadoxNetPath, you must also specify a ParadoxUserName and a ParadoxNetStyle or you will receive an error when trying to access external Paradox data. Also, if you are accessing a Paradox database in multi-user mode over the network, you will need to add or modify this registry entry manually.

ParadoxNetStyle The network access style to use when accessing Paradox data. Possible values are:

- 3.x
- 4.x

Note Paradox 3.x users cannot set this to 4.x or the driver will use the wrong locking method. Paradox 5.0 users must use the 4.x ParadoxNetStyle setting to ensure proper locking behavior.

This entry is not added by the setup program. You must create this entry. This entry should correspond to whatever version of Paradox the users in the group are using. It must be consistent for all users sharing a particular database (directory). The default is 4.x. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.

Note If you indicate a ParadoxNetStyle, you must also

specify a ParadoxUserName and a ParadoxNetPath or you will receive an error when trying to access external Paradox data.

Paradox ISAM Formats

The Jet\4.0\ISAM Formats\Paradox 3.x folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Paradox
ExportFilter	REG_SZ	String	Paradox 3 (*.db)
ImportFilter	REG_SZ	String	Paradox (*.db)
CanLink	REG_BINARY	Binary	01
OneTablePerFile	REG_BINARY	Binary	01
IsamType	REG_DWORD	DWORD	0
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	00
ResultTextImport	REG_SZ	String	Import data from the external file into the current database. Changing data in the current database will not change data in the external file.
ResultTextLink	REG_SZ	String	Create a table in the current database that is linked to the external file. Changing data in the current database will change data in the external file.
ResultTextExport	REG_SZ	String	Export data from the current database into a Paradox 3 file. This

process will overwrite the data if exported to an existing file.

SupportsLongNames REG_BINARY Binary 00

The Jet\4.0\ISAM Formats\Paradox 4.x folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Paradox
ExportFilter	REG_SZ	String	Paradox 4 (*.db)
CanLink	REG_BINARY	Binary	01
OneTablePerFile	REG_BINARY	Binary	01
IsamType	REG_DWORD	DWORD	0
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	00
ResultTextExport	REG_SZ	String	Export data from the current database into a Paradox 4 file. This process will overwrite the data if exported to an existing file.
SupportsLongNames	REG_BINARY	Binary	00

The Jet\4.0\ISAM Formats\Paradox 5.x folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Paradox
ExportFilter	REG_SZ	String	Paradox 5 (*.db)
CanLink	REG_BINARY	Binary	01

OneTablePerFile	REG_BINARY	Binary	01
IsamType	REG_DWORD	DWORD	0
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	00
ResultTextExport	REG_SZ	String	Export data from the current database into a Paradox 5 file. This process will overwrite the data if exported to an existing file.
SupportsLongNames	REG_BINARY	Binary	00

The Jet\4.0\ISAM Formats\Paradox 7.x folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Paradox
ExportFilter	REG_SZ	String	Paradox 7 (*.db)
CanLink	REG_BINARY	Binary	01
OneTablePerFile	REG_BINARY	Binary	01
IsamType	REG_DWORD	DWORD	0
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	00
ResultTextExport	REG_SZ	String	Export data from the current database into a Paradox 7 file. This process will overwrite the data if exported to an existing file.
SupportsLongNames	REG_BINARY	Binary	00

Note When you change Windows Registry settings, you must exit and then

restart the database engine for the new settings to take effect.

See Also

[Customizing Windows Registry Settings for Microsoft Jet](#)

Initializing the Text and HTML Data Source Driver

The same database driver is used for both Text Data sources and for HTML data sources.

When you install the Text Data Source database driver, the Setup program writes a set of default values to the Microsoft® Windows® Registry in the Engines and ISAM Formats subkeys. You should not modify these settings directly; use the setup program for your application to add, remove, or change these settings. The following sections describe initialization and ISAM Format settings for the Text Data Source database driver.

Text Data Source Initialization Settings

The Jet\4.0\Engines\Text folder includes initialization settings for the Mstext35.dll driver, used for external access to text data files. Typical settings for the entries in this folder are shown in the following example.

```
win32=<path>\ mstext40.dll
MaxScanRows=25
FirstRowHasNames=True
CharacterSet= ANSI
Format=CSVDelimited
Extensions= txt, csv, tab, asc
ExportCurrencySymbols=Yes
```

The Microsoft Jet database engine uses the Text folder entries as follows.

Entry	Description
win32	The location of mstext40.dll. The full path is determined at the time of installation. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
MaxScanRows	The number of rows to be scanned when guessing the column types. If set to 0, the entire file will be searched. The default is 25. Values are of type DWORD for Windows 95 and Windows NT 4.0, and of type REG_DWORD for Windows NT 3.51.

FirstRowHasNames	A binary value that indicates whether the first row of the table contains column names. A value of 01 indicates that, during import, column names are taken from the first row. A value of 00 indicates no column names in the first row. The default is 01. Values are of type Binary for Windows 95 and Windows NT 4.0, and of type REG_BINARY for Windows NT 3.51.
CharacterSet	<p>An indicator of how text pages are stored. Possible settings are:</p> <ul style="list-style-type: none"> • ANSI — The ANSI code page of the machine. AnsiToUnicode and UnicodeToAnsi conversions done. • OEM — — The OEM code page of the machine. OemToUnicode and UnicodeToOem conversions done. • Unicode — codepage conversions not done. • <decimal number> — The code page number of a specific character set. Conversions to and from Unicode will be done. <p>The default is ANSI. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.</p>
Format	Can be any of the following: TabDelimited, CSVDelimited, Delimited (<single character>). The single-character delimiter in the Delimited format can be any single character except a double quotation mark ("). The default is CSVDelimited. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
Extensions	The extension of any files to be browsed when looking for text-based data. The default is txt, csv, tab, asc. Values are of type String for Windows 95 and Windows NT 4.0, and of type REG_SZ for Windows NT 3.51.
ExportCurrencySymbols	A binary value that indicates whether the appropriate

currency symbol is included when currency fields are exported. A value of 01 indicates that the symbol is included. A value of 00 indicates that only the numeric data is exported. The default is 01. Values are of type Binary for Windows 95 and Windows NT 4.0, and of type REG_BINARY for Windows NT 3.51.

Text Data Source ISAM Formats

The Jet\4.0\ISAM Formats\Text folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Text
ExportFilter	REG_SZ	String	Text Files (*.txt; *.csv; *.tab; *.asc)
ImportFilter	REG_SZ	String	Text Files (*.txt; *.csv; *.tab; *.asc)
CanLink	REG_BINARY	Binary	01
OneTablePerFile	REG_BINARY	Binary	01
IsamType	REG_DWORD	DWORD	2
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	00
ResultTextImport	REG_SZ	String	Import data from the external file into the current database. Changing data in the current database will not change data in the external file.
ResultTextLink	REG_SZ	String	Create a table in the current database that is linked to the external file. Changing data in the current database

ResultTextExport	REG_SZ	String	will change data in the external file. Export data from the current database into a text file. This process will overwrite the data if exported to an existing file.
SupportsLongNames	REG_BINARY	Binary	01

Note When you change Windows Registry settings, you must exit and then restart the database engine for the new settings to take effect.

HTML Import ISAM Formats

The Jet\4.0\ISAM Formats\HTML Import folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Text
ImportFilter	REG_SZ	String	HTML Files (*.ht*)
CanLink	REG_BINARY	Binary	01
OneTablePerFile	REG_BINARY	Binary	00
IsamType	REG_DWORD	DWORD	2
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	00
ResultTextImport	REG_SZ	String	Import data from the external file into the current database. Changing data in the current database will not change data in the external file.

ResultTextLink	REG_SZ	String	Create a table in the current database that is linked to the external file. Changing data in the current database will change data in the external file.
SupportsLongNames	REG_BINARY	Binary	01

Note When you change Windows Registry settings, you must exit and then restart the database engine for the new settings to take effect.

HTML Export ISAM Formats

The Jet\4.0\ISAM Formats\HTML Export folder contains the following entries.

Entry name	Windows NT 3.51 Type	Windows 95 and Windows NT 4.0 Type	Value
Engine	REG_SZ	String	Text
ExportFilter	REG_SZ	String	HTML Files (*.htm)
CanLink	REG_BINARY	Binary	00
OneTablePerFile	REG_BINARY	Binary	01
IsamType	REG_DWORD	DWORD	2
IndexDialog	REG_BINARY	Binary	00
CreateDBOnExport	REG_BINARY	Binary	00
ResultTextExport	REG_SZ	String	Export data from the current database into a text file. This process will overwrite the data if exported to an existing file.
SupportsLongNames	REG_BINARY	Binary	01

Note When you change Windows Registry settings, you must exit and then restart the database engine for the new settings to take effect.

Customizing the Schema.ini File for Text and HTML Data

To read, import, or export text and HTML data, you need to create a Schema.ini file in addition to including the Text ISAM information in the .ini file.

Schema.ini contains the specifics of a data source: how the text file is formatted, how it is read at import time, and what the default export format is for files. The following examples show the layout for a fixed-width file, Filename.txt:

```
[Filename.txt]
ColNameHeader=False
Format=FixedLength
FixedFormat= RaggedEdge
MaxScanRows=25
CharacterSet=OEM
Col1=columnname Char Width 24
Col2=columnname2 Date Width 9
Col3=columnname7 Float Width 10
Col4=columnname8 Integer Width 10
Col5=columnname9 LongChar Width 10
```

Similarly, the format for a delimited file is specified as follows:

```
[Delimit.txt]
ColNameHeader=True
Format=Delimited()
MaxScanRows=0
CharacterSet=OEM
Col1=username char width 50
Col2=dateofbirth Date width 9
```

If you are exporting data into a delimited text file, specify the format for that file as well:

```
[Export: My Special Export]
```

ColNameHeader=True
 Format=TabDelimited
 MaxScanRows=25
 CharacterSet=OEM
 DateTimeFormat=mm.dd.yy.hh.mm.ss
 CurrencySymbol=Dm
 CurrencyPosFormat=0
 CurrencyDigits=2
 CurrencyNegFormat=0
 CurrencyThousandSymbol=,
 CurrencyDecimalSymbol=.
 DecimalSymbol=,
 NumberDigits=2
 NumberLeadingZeros=0
 TextDelimiter="

The My Special Export example refers to a specific export option; you can specify any variation of export options at connect time. This last example also corresponds to a data source name (DSN) that can be optionally passed at connect time. All three format sections can be included in the same .ini file.

The Microsoft Jet database engine uses the Schema.ini entries as follows.

Entry	Description
ColNameHeader	Can be set to either True (indicating that the first record of data specifies the column names) or False .
Format	Can be set to one of the following values: TabDelimited, CSVDelimited, Delimited (<single character>), or FixedLength. The delimiter specified for the Delimited file format can be any single character except a double quotation mark (").
FixedFormat	Only used when the Format is FixedLength, this can be set to one of the following values: RaggedEdge or TrueFixedLength. RaggedEdge allows rows to be terminated with a Carriage Return character.

TrueFixedLength requires each row to be an exact number of characters, and any Carriage Return characters not at a row boundary are assumed to be embedded in a field.

If this setting is not present, the default value is RaggedEdge.

MaxScanRows	Indicates the number of rows to be scanned when guessing the column data types. If this is set to 0, the entire file is searched.
CharacterSet	Can be set to OEM, ANSI, UNICODE, or the decimal number of a valid code page, and indicates the character set of the source file.
DateTimeFormat	Can be set to a format string indicating dates and times. This entry should be specified if all date/time fields in the import/export are handled with the same format. All of the Microsoft Jet database engine formats except AM and PM are supported. In the absence of a format string, the Windows Control Panel short date picture and time options are used.
CurrencySymbol	Indicates the currency symbol to be used for currency values in the text file. Examples include the dollar sign (\$) and Dm. If this entry is absent, the default value in the Windows Control Panel is used.
CurrencyPosFormat	<p>Can be set to any of the following values:</p> <ul style="list-style-type: none">Currency symbol prefix with no separation (\$1)Currency symbol suffix with no separation (1\$)Currency symbol prefix with one character separation (\$ 1)Currency symbol suffix with one character separation (1 \$) <p>If this entry is absent, the default value in the Windows Control Panel is used.</p>

CurrencyDigits Specifies the number of digits used for the fractional part of a currency amount. If this entry is absent, the default value in the Windows Control Panel is used.

CurrencyNegFormat Can be one of the following values:

(\$1)

-\$1

\$-1

\$1-

(1\$)

-1\$

1-\$

1\$-

-1 \$

-\$ 1

1 \$-

\$ 1-

\$ -1

1- \$

(\$ 1)

(1 \$)

The dollar sign is shown for purposes of this example, but it should be replaced with the appropriate CurrencySymbol value in the actual program. If this entry is absent, the default value in the Windows Control Panel is used.

CurrencyThousandSymbol Indicates the single-character symbol to be used for separating currency values by thousands in the text file. If this entry is absent, the default value in the Windows Control Panel is used.

CurrencyDecimalSymbol	Can be set to any single character that is used to separate the whole from the fractional part of a currency amount. If this entry is absent, the default value in the Windows Control Panel is used.
DecimalSymbol	Can be set to any single character that is used to separate the integer from the fractional part of a number. If this entry is absent, the default value in the Windows Control Panel is used.
NumberDigits	Indicates the number of decimal digits in the fractional portion of a number. If this entry is absent, the default value in the Windows Control Panel is used.
NumberLeadingZeros	Specifies whether a decimal value less than 1 and greater than -1 should contain leading zeros; this value can either be False (no leading zeros) or True.
Col1, Col2, ...	Lists the columns in the text file to be read. The format of this entry should be:

Coln=columnName type [Width #]

columnName: Column names with embedded spaces should be enclosed in quotation marks.

type: Can be Bit, Byte, Short, Long, Decimal, Currency, Single, Double, DateTime, Binary, OLE, Text, or Memo.

In addition, the following ODBC Text Driver types are supported:

Char (same as Text)

Float (same as Double)

Integer (same as Short)

LongChar (same as Memo)

Date *date format*

In the case of a Memo type an additional format marker [Attribute Hyperlink] can be used to specify columns that should be active URLs in Microsoft Access.

In the case of a Decimal type the additional format markers [Scale #] Precision #] should be used.

TextDelimiter

Can be set to any single character that is used to delimit strings that contain any of the other special characters.

E.g. “abc”,”xyz,pqr”,”hij”

If this entry is not present the default delimiter is a double quote. If this entry is the string “none” then no characters will be treated as delimiters.

Note When you change Schema.ini file settings, you must exit and then restart the database engine for the new settings to take effect.

See Also

[Customizing Windows Registry Settings for Microsoft Jet](#)

Configuring the Microsoft Jet Database Engine for ODBC Access

The following sections describe Microsoft® Windows® Registry settings for the Microsoft Jet database engine for connection to an ODBC database.

Initialization Settings for Microsoft Jet-connected ODBC Databases

The \HKEY_LOCAL_MACHINE\Software\Microsoft\Jet\4.0\Engines\ODBC folder contains initialization settings for the Microsoft Jet database engine.

Note Typical settings for the entries in the Jet\4.0\Engines\ODBC folder are shown in the following example.

```
LoginTimeout=20
QueryTimeout=60
ConnectionTimeout=600
AsyncRetryInterval=500
AttachCaseSensitive=0
AttachableObjects='TABLE','VIEW','SYSTEM
TABLE','ALIAS','SYNONYM'
SnapshotOnly=0
TraceSQLMode=0
TraceODBCAPI=0
DisableAsync=1
TryJetAuth=1
PreparedInsert=0
PreparedUpdate=0
FastRequery=0
FatBlastRows=-1
FatBlastTimeout=3
ODBCISAMAttach=0
```

The Microsoft Jet database engine uses the ODBC entries as follows.

Entry	Description
-------	-------------

LoginTimeout	The number of seconds a login attempt can continue before timing out. The default is 20 (values are of type REG_DWORD).
QueryTimeout	The number of seconds a query can run (total processing time) before timing out. If DisableAsync=0 (the default), then QueryTimeout is the number of seconds to wait for a response from the server between polls for query completion. The default is 60 (values are of type REG_DWORD).
ConnectionTimeout	The number of seconds a cached connection can remain idle before timing out. The default is 600 (values are of type REG_DWORD).
AsyncRetryInterval	The number of milliseconds between polls to determine if the server is done processing a query. This entry is used for asynchronous processing only. The default is 500 (values are of type REG_DWORD).
AttachCaseSensitive	An indicator of whether to match table names exactly when linking. Values are 0 (link the first table matching the specified name, regardless of case) and 1 (link a table only if the name matches exactly). The default is 0 (values are of type REG_DWORD).
AttachableObjects	A list of server object types to which linking will be allowed. The default is: 'TABLE', 'VIEW', 'SYSTEM TABLE', 'ALIAS', 'SYNONYM' (values are of type REG_SZ).
SnapshotOnly	An indicator of whether Recordset objects are forced to be of snapshot type. Values are 0 (allow dynasets) and 1 (force snapshots only). The default is 0 (values are of type REG_DWORD).
TraceSQLMode	An indicator of whether the Microsoft Jet database engine will trace SQL statements sent to an ODBC data source in SLOUT.txt. Values are 0 (no) and 1 (yes). The default is 0 (values are of type REG_DWORD). This entry is interchangeable with SQLTraceMode.
TraceODBCAPI	An indicator of whether to trace ODBC API calls in ODBCAPI.txt. Values are 0 (no) and 1 (yes). The default is 0 (values are of type REG_DWORD).

DisableAsync	An indicator of whether to force synchronous query execution. Values are 0 (use asynchronous query execution if possible) and 1 (force synchronous query execution). The default is 1 (values are of type REG_DWORD).
TryJetAuth	An indicator of whether to try using the Microsoft Access user name and password to log in to the server before prompting. Values are 0 (no) and 1 (yes). The default is 1 (values are of type REG_DWORD).
PreparedInsert	An indicator of whether to use a prepared INSERT statement that inserts data in all columns. Values are 0 (use a custom INSERT statement that inserts only non- Null values) and 1 (use a prepared INSERT statement). The default is 0 (values are of type REG_DWORD). Using prepared INSERT statements can cause Nulls to overwrite server defaults and can cause triggers to execute on columns that were not inserted explicitly.
PreparedUpdate	An indicator of whether to use a prepared UPDATE statement that updates data in all columns. Values are 0 (use a custom UPDATE statement that sets only columns that have changed) and 1 (use a prepared UPDATE statement). The default is 0 (values are of type REG_DWORD). Using prepared UPDATE statements can cause triggers to execute on unchanged columns.
FastRequery	An indicator of whether to use a prepared SELECT statement for parameterized queries. Values are 0 (no) and 1 (yes). The default is 0 (values are of type REG_DWORD).
FatBlastRows	
FatBlastTimeout	
ODBCISAMAttach	

Note When you change Windows Registry settings, you must exit and then restart the database engine for the new settings to take effect.

See Also

[Customizing Windows Registry Settings for Microsoft Jet](#)

Avg Function

Calculates the arithmetic mean of a set of values contained in a specified field on a query.

Syntax

Avg(*expr*)

The *expr* placeholder represents a [string expression](#) identifying the field that contains the numeric data you want to average or an expression that performs a calculation using the data in that field. Operands in *expr* can include the name of a table field, a constant, or a function (which can be either intrinsic or user-defined but not one of the other [SQL aggregate](#) functions).

Remarks

The average calculated by **Avg** is the arithmetic mean (the sum of the values divided by the number of values). You could use **Avg**, for example, to calculate average freight cost.

The **Avg** function does not include any [Null](#) fields in the calculation.

You can use **Avg** in a query expression and in the [SQL](#) property of a [QueryDef](#) object or when creating a [Recordset](#) object based on an SQL query.

See Also

[Calculating Fields in SQL Functions](#)

[SQL Aggregate Functions \(SQL\)](#)

Example

[Avg Function Example](#)

Count Function

Calculates the number of records returned by a query.

Syntax

Count(*expr*)

The *expr* placeholder represents a [string expression](#) identifying the field that contains the data you want to count or an expression that performs a calculation using the data in the field. Operands in *expr* can include the name of a table field or function (which can be either intrinsic or user-defined but not other [SQL aggregate functions](#)). You can count any kind of data, including text.

Remarks

You can use **Count** to count the number of records in an underlying query. For example, you could use **Count** to count the number of orders shipped to a particular country.

Although *expr* can perform a calculation on a field, **Count** simply tallies the number of records. It does not matter what values are stored in the records.

The **Count** function does not count records that have [Null](#) fields unless *expr* is the asterisk (*) [wildcard character](#). If you use an asterisk, **Count** calculates the total number of records, including those that contain **Null** fields. **Count(*)** is considerably faster than **Count([Column Name])**. Do not enclose the asterisk in quotation marks (' '). The following example calculates the number of records in the Orders table:

```
SELECT Count(*)  
AS TotalOrders FROM Orders;
```

If *expr* identifies multiple fields, the **Count** function counts a record only if at least one of the fields is not **Null**. If all of the specified fields are **Null**, the record is not counted. Separate the field names with an ampersand (&). The following example shows how you can limit the count to records in which either ShippedDate or Freight is not **Null**:

```
SELECT  
Count('ShippedDate & Freight')
```

AS [Not Null] FROM Orders;

You can use **Count** in a query expression. You can also use this expression in the **SQL** property of a **QueryDef** object or when creating a **Recordset** object based on an SQL query.

See Also

[SELECT Statement \(Microsoft Jet SQL\)](#)

[Sum Function \(Microsoft Jet SQL\)](#)

[SQL Aggregate Functions \(SQL\)](#)

Example

[Count Function Example](#)

First, Last Functions

Return a field value from the first or last record in the result set returned by a query.

Syntax

First(*expr*)

Last(*expr*)

The *expr* placeholder represents a [string expression](#) identifying the field that contains the data you want to use or an expression that performs a calculation using the data in that field. Operands in *expr* can include the name of a table field, a constant, or a function (which can be either intrinsic or user-defined but not one of the other [SQL aggregate](#) functions).

Remarks

The **First** and **Last** functions are analogous to the [MoveFirst](#) and [MoveLast](#) methods of a [DAO Recordset](#) object. They simply return the value of a specified field in the first or last record, respectively, of the result set returned by a query. Because records are usually returned in no particular order (unless the query includes an [ORDER BY](#) clause), the records returned by these functions will be arbitrary.

See Also

[Calculating Fields in SQL Functions](#)

[SQL Aggregate Functions \(SQL\)](#)

Example

[First, Last Functions Example](#)

Min, Max Functions

Return the minimum or maximum of a set of values contained in a specified field on a query.

Syntax

Min(*expr*)

Max(*expr*)

The *expr* placeholder represents a [string expression](#) identifying the field that contains the data you want to evaluate or an expression that performs a calculation using the data in that field. Operands in *expr* can include the name of a table field, a constant, or a function (which can be either intrinsic or user-defined but not one of the other [SQL aggregate](#) functions).

Remarks

You can use **Min** and **Max** to determine the smallest and largest values in a field based on the specified aggregation, or grouping. For example, you could use these functions to return the lowest and highest freight cost. If there is no aggregation specified, then the entire table is used.

You can use **Min** and **Max** in a query expression and in the [SQL](#) property of a [QueryDef](#) object or when creating a [Recordset](#) object based on an SQL query.

See Also

[SQL Aggregate Functions \(SQL\)](#)

Example

[Min, Max Functions Example](#)

StDev, StDevP Functions

Return estimates of the [standard deviation](#) for a population or a population sample represented as a set of values contained in a specified field on a query.

Syntax

StDev(*expr*)

StDevP(*expr*)

The *expr* placeholder represents a [string expression](#) identifying the field that contains the numeric data you want to evaluate or an expression that performs a calculation using the data in that field. Operands in *expr* can include the name of a table field, a constant, or a function (which can be either intrinsic or user-defined but not one of the other [SQL aggregate](#) functions).

Remarks

The **StDevP** function evaluates a population, and the **StDev** function evaluates a population sample.

If the underlying query contains fewer than two records (or no records, for the **StDevP** function), these functions return a **Null** value (which indicates that a standard deviation cannot be calculated).

You can use the **StDev** and **StDevP** functions in a query expression. You can also use this expression in the [SQL](#) property of a [QueryDef](#) object or when creating a [Recordset](#) object based on an SQL query.

See Also

[Avg Function \(Microsoft Jet SQL\)](#) [SQL Aggregate Functions \(SQL\)](#)
[SELECT Statement \(Microsoft Jet SQL\)](#) [SUM Function \(Microsoft Jet SQL\)](#)

Example

[StDev, StDevP Functions Example](#)

Sum Function

Returns the sum of a set of values contained in a specified field on a query.

Syntax

Sum(*expr*)

The *expr* placeholder represents a [string expression](#) identifying the field that contains the numeric data you want to add or an expression that performs a calculation using the data in that field. Operands in *expr* can include the name of a table field, a constant, or a function (which can be either intrinsic or user-defined but not one of the other [SQL aggregate](#) functions).

Remarks

The **Sum** function totals the values in a field. For example, you could use the **Sum** function to determine the total cost of freight charges.

The **Sum** function ignores records that contain **Null** fields. The following example shows how you can calculate the sum of the products of UnitPrice and Quantity fields:

```
SELECT  
Sum(UnitPrice * Quantity)  
AS [Total Revenue] FROM [Order Details];
```

You can use the **Sum** function in a query expression. You can also use this expression in the [SQL](#) property of a [QueryDef](#) object or when creating a [Recordset](#) based on an SQL query.

See Also

[Count Function \(Microsoft Jet SQL\)](#)

[SQL Aggregate Functions \(SQL\)](#)

[SELECT Statement \(Microsoft Jet SQL\)](#)

Example

Sum Function Example

Var, VarP Functions

Return estimates of the [variance](#) for a population or a population sample represented as a set of values contained in a specified field on a query.

Syntax

Var(*expr*)

VarP(*expr*)

The *expr* placeholder represents a [string expression](#) identifying the field that contains the numeric data you want to evaluate or an expression that performs a calculation using the data in that field. Operands in *expr* can include the name of a table field, a constant, or a function (which can be either intrinsic or user-defined but not one of the other [SQL aggregate](#) functions).

Remarks

The **VarP** function evaluates a population, and the **Var** function evaluates a population sample.

If the underlying query contains fewer than two records, the **Var** and **VarP** functions return a **Null** value, which indicates that a variance cannot be calculated.

You can use the **Var** and **VarP** functions in a query expression or in an [SQL statement](#).

See Also

[SQL Aggregate Functions \(SQL\)](#)

Example

[VAR, VARP Functions Example](#)

WHERE Clause

Specifies which records from the tables listed in the [FROM](#) clause are affected by a [SELECT](#), [UPDATE](#), or [DELETE](#) statement.

Syntax

```
SELECT fieldlist  
FROM tableexpression  
WHERE criteria
```

A SELECT statement containing a WHERE clause has these parts:

Part	Description
<i>fieldlist</i>	The name of the field or fields to be retrieved along with any field-name aliases , selection predicates (ALL , DISTINCT , DISTINCTROW , or TOP), or other SELECT statement options.
<i>tableexpression</i>	The name of the table or tables from which data is retrieved.
<i>criteria</i>	An expression that records must satisfy to be included in the query results.

Remarks

The [Microsoft Jet database engine](#) selects the records that meet the conditions listed in the WHERE clause. If you do not specify a WHERE clause, your query returns all rows from the table. If you specify more than one table in your query and you have not included a WHERE clause or a JOIN clause, your query generates a [Cartesian product](#) of the tables.

WHERE is optional, but when included, follows FROM. For example, you can select all employees in the sales department (WHERE Dept = 'Sales') or all customers between the ages of 18 and 30 (WHERE Age Between 18 And 30).

If you do not use a JOIN clause to perform SQL join operations on multiple tables, the resulting **Recordset** object will not be updatable.

WHERE is similar to [HAVING](#). WHERE determines which records are selected. Similarly, once records are grouped with [GROUP BY](#), HAVING determines which records are displayed.

Use the WHERE clause to eliminate records you do not want grouped by a GROUP BY clause.

Use various expressions to determine which records the SQL statement returns. For example, the following SQL statement selects all employees whose salaries are more than \$21,000:

```
SELECT LastName, Salary
FROM Employees
WHERE Salary > 21000;
```

A WHERE clause can contain up to 40 expressions linked by logical operators, such as **And** and **Or**.

When you enter a field name that contains a space or punctuation, surround the name with brackets ([]). For example, a customer information table might include information about specific customers :

```
SELECT [Customer's Favorite Restarant]
```

When you specify the *criteria* argument, [date literals](#) must be in U.S. format, even if you are not using the U.S. version of the Microsoft® Jet database engine. For example, May 10, 1996, is written 10/5/96 in the United Kingdom and 5/10/96 in the United States. Be sure to enclose your date literals with the number sign (#) as shown in the following examples.

To find records dated May 10, 1996 in a United Kingdom database, you must use the following SQL statement:

```
SELECT *
FROM Orders
WHERE ShippedDate = #5/10/96#;
```

You can also use the **DateValue** function which is aware of the international settings established by Microsoft Windows®. For example, use this code for the United States:

```
SELECT *
```

```
FROM Orders
WHERE ShippedDate = DateValue('5/10/96');
```

And use this code for the United Kingdom:

```
SELECT *
FROM Orders
WHERE ShippedDate = DateValue('10/5/96');
```

Note If the column referenced in the criteria string is of type [GUID](#), the criteria expression uses a slightly different syntax:

```
WHERE ReplicaID = {GUID {12345678-90AB-CDEF-1234-567890ABCDEF}}
```

Be sure to include the nested braces and hyphens as shown.

See Also

ALL DISTINCT, DISTINCTROW, TOP Predicates (Microsoft Jet SQL)	LEFT JOIN, RIGHT JOIN Operations (Microsoft Jet SQL)
DELETE Statement (Microsoft Jet SQL)	ORDER BY Clause (Microsoft Jet SQL)
FROM Clause (Microsoft Jet SQL)	SELECT Statement (Microsoft Jet SQL)
GROUP BY Clause (Microsoft Jet SQL)	SELECT...INTO Statement (Microsoft Jet SQL)
HAVING Clause (Microsoft Jet SQL)	SQL Aggregate Functions (SQL)
IN Clause (Microsoft Jet SQL)	UPDATE Statement (Microsoft Jet SQL)
INNER JOIN Operation (Microsoft Jet SQL)	

Example

[WHERE Clause Example](#)

Between...And Operator

Determines whether the value of an expression falls within a specified range of values. You can use this operator within [SQL statements](#).

Syntax

expr [**Not**] **Between** *value1* **And** *value2*

The **Between...And** operator syntax has these parts:

Part	Description
<i>expr</i>	Expression identifying the field that contains the data you want to evaluate.
<i>value1</i> , <i>value2</i>	Expressions against which you want to evaluate <i>expr</i> .

Remarks

If the value of *expr* is between *value1* and *value2* (inclusive), the **Between...And** operator returns **True**; otherwise, it returns **False**. You can include the **Not** logical operator to evaluate the opposite condition (that is, whether *expr* lies outside the range defined by *value1* and *value2*).

You might use **Between...And** to determine whether the value of a field falls within a specified numeric range. The following example determines whether an order was shipped to a location within a range of postal codes. If the postal code is between 98101 and 98199, the **Iif** function returns “Local”. Otherwise, it returns “Nonlocal”.

```
SELECT Iif(PostalCode Between 98101 And 98199, “Local”,  
“Nonlocal”)  
FROM Publishers
```

If *expr*, *value1*, or *value2* is **Null**, **Between...And** returns a **Null** value.

Because [wildcard characters](#), such as *, are treated as literals, you cannot use them with the **Between...And** operator. For example, you cannot use 980* and 989* to find all postal codes that start with 980 to 989. Instead, you have two

alternatives for accomplishing this. You can add an expression to the query that takes the left three characters of the text field and use **Between...And** on those characters. Or you can pad the high and low values with extra characters — in this case, 98000 to 98999, or 98000 to 98999 – 9999 if using extended postal codes. (You must omit the – 0000 from the low values because otherwise 98000 is dropped if some postal codes have extended sections and others do not.)

See Also

[IN Clause \(Microsoft Jet SQL\)](#)

[SQL Expressions](#)

[WHERE Clause \(Microsoft Jet SQL\)](#)

Example

[SQL Subqueries Example](#)

In Operator

Determines whether the value of an expression is equal to any of several values in a specified list.

Syntax

expr [**Not**] **In**(*value1*, *value2*, . . .)

Remarks

The **In** operator syntax has these parts:

Part	Description
<i>expr</i>	Expression identifying the field that contains the data you want to evaluate.
<i>value1</i> , <i>value2</i>	Expression or list of expressions against which you want to evaluate <i>expr</i> .

If *expr* is found in the list of values, the **In** operator returns **True**; otherwise, it returns **False**. You can include the **Not** logical operator to evaluate the opposite condition (that is, whether *expr* is not in the list of values).

For example, you can use **In** to determine which orders are shipped to a set of specified regions:

```
SELECT *  
FROM Orders  
WHERE ShipRegion In ('Avon','Glos','Som')
```

See Also

[SQL Expressions](#)

[WHERE Clause \(Microsoft Jet SQL\)](#)

Example

[In Operator Example](#)

Like Operator

Compares a [string expression](#) to a pattern in an [SQL](#) expression.

Syntax

expression **Like** "pattern"

The **Like** operator syntax has these parts:

Part	Description
<i>expression</i>	SQL expression used in a WHERE clause .
<i>pattern</i>	String or character string literal against which <i>expression</i> is compared.

Remarks

You can use the **Like** operator to find values in a field that match the pattern you specify. For *pattern*, you can specify the complete value (for example, Like "Smith"), or you can use [wildcard characters](#) to find a range of values (for example, Like "Sm*").

In an expression, you can use the **Like** operator to compare a field value to a string expression. For example, if you enter Like "C*" in an SQL query, the query returns all field values beginning with the letter C. In a [parameter query](#), you can prompt the user for a pattern to search for.

The following example returns data that begins with the letter P followed by any letter between A and F and three digits:

Like "P[A-F]###"

The following table shows how you can use **Like** to test expressions for different patterns.

Kind of match	Pattern	Match (returns True)	No match (returns False)
Multiple characters	a*a	aa, aBa, aBBBa	aBC

	ab	abc, AABB, Xab	aZb, bac
Special character	a[*]a	a*a	aaa
Multiple characters	ab*	abcdefg, abc	cab, aab
Single character	a?a	aaa, a3a, aBa	aBBBa
Single digit	a#a	a0a, a1a, a2a	aaa, a10a
Range of characters	[a-z]	f, p, j	2, &
Outside a range	[!a-z]	9, &, %	b, a
Not a digit	[!0-9]	A, a, &, ~	0, 1, 9
Combined	a[!b-m]#	An9, az0, a99	abc, aj0

See Also

[SQL Expressions](#)

[Using Wildcard Characters in String Comparisons](#)

[WHERE Clause \(Microsoft Jet SQL\)](#)

Example

[Like Operator Example](#)

ALL

The ALL keyword is used in these contexts:

[ALL, DISTINCT, DISTINCTROW, TOP Predicates](#)

[SQL Subqueries](#)

[UNION Operation](#)

ASC/DESC

The ASC and DESC keywords are used in these contexts:

[CREATE INDEX Statement](#)

[ORDER BY Clause](#)

ALTER

The ALTER keyword is used in these contexts:

[ALTER TABLE Statement](#)

[ALTER USER or DATABASE Statement](#)

AS

The AS keyword is used in these contexts:

[SELECT Statement](#)

[CREATE VIEW Statement](#)

[CREATE PROCEDURE Statement](#)

BY

The BY keyword is used in these contexts:

[GROUP BY Clause](#)

[ORDER BY Clause](#)

CONTAINER

The CONTAINER keyword is used in these contexts:

[GRANT Statement](#)

[REVOKE Statement](#)

CREATE

The CREATE keyword is used in these contexts:

[CREATE INDEX Statement](#)

[CREATE TABLE Statement](#)

[CREATE VIEW Statement](#)

[CREATE PROCEDURE Statement](#)

[CREATE USER or GROUP Statement](#)

[GRANT Statement](#)

[REVOKE Statement](#)

IN Clause

Identifies tables in any [external database](#) to which the [Microsoft Jet database engine](#) can connect, such as a dBASE or Paradox database or an external Microsoft® Jet database.

Syntax

To identify a destination table:

```
[SELECT | INSERT] INTO destination IN  
  {path | ["path" "type"] | ["" [type; DATABASE = path]]}
```

To identify a source table:

```
FROM tableexpression IN  
  {path | ["path" "type"] | ["" [type; DATABASE = path]]}
```

A SELECT statement containing an IN clause has these parts:

Part	Description
<i>destination</i>	The name of the external table into which data is inserted.
<i>tableexpression</i>	The name of the table or tables from which data is retrieved. This argument can be a single table name, a saved query, or a compound resulting from an INNER JOIN , LEFT JOIN , or RIGHT JOIN .
<i>path</i>	The full path for the directory or file containing <i>table</i> .
<i>type</i>	The name of the database type used to create <i>table</i> if a database is not a Microsoft Jet database (for example, dBASE III, dBASE IV, Paradox 3.x, or Paradox 4.x).

Remarks

You can use IN to connect to only one [external database](#) at a time.

In some cases, the *path* argument refers to the directory containing the database files. For example, when working with dBASE, Microsoft FoxPro®, or Paradox database tables, the *path* argument specifies the directory containing .dbf or .db files. The table file name is derived from the *destination* or *tableexpression*

argument.

To specify a non-Microsoft Jet database, append a semicolon (;) to the name, and enclose it in single (' ') or double (" ") quotation marks. For example, either 'dBASE IV;' or "dBASE IV;" is acceptable.

You can also use the DATABASE reserved word to specify the external database. For example, the following lines specify the same table:

```
... FROM Table IN "" [dBASE IV;  
DATABASE=C:\DBASE\DATA\SALES;];  
... FROM Table IN "C:\DBASE\DATA\SALES" "dBASE IV;"
```

Notes

For improved performance and ease of use, use a [linked table](#) instead of IN.

You can also use the IN reserved word as a comparison operator in an [expression](#). For more information, see the [In](#) operator.

See Also

[FROM Clause \(Microsoft Jet SQL\)](#) [SELECT Statement \(Microsoft Jet SQL\)](#)
[INNER JOIN Operation \(Microsoft Jet SQL\)](#) [SELECT...INTO Statement \(Microsoft Jet SQL\)](#)
[INSERT INTO Statement \(Microsoft Jet SQL\)](#) [SQL Aggregate Functions \(SQL\)](#)
[LEFT JOIN, RIGHT JOIN Operations \(Microsoft Jet SQL\)](#)

Example

[IN Clause Example](#)

ALL, DISTINCT, DISTINCTROW, TOP Predicates

Specifies records selected with [SQL](#) queries.

Syntax

```
SELECT [ALL | DISTINCT | DISTINCTROW | [TOP n [PERCENT]]]  
FROM table
```

A [SELECT](#) statement containing these predicates has the following parts:

Part	Description
ALL	<p>Assumed if you do not include one of the predicates. The Microsoft Jet database engine selects all of the records that meet the conditions in the SQL statement. The following two examples are equivalent and return all records from the Employees table:</p> <pre>SELECT ALL * FROM Employees ORDER BY EmployeeID; SELECT * FROM Employees ORDER BY EmployeeID;</pre>
DISTINCT	<p>Omits records that contain duplicate data in the selected fields. To be included in the results of the query, the values for each field listed in the SELECT statement must be unique. For example, several employees listed in an Employees table may have the same last name. If two records contain Smith in the LastName field, the following SQL statement returns only one record that contains Smith:</p> <pre>SELECT DISTINCT LastName FROM Employees;</pre> <p>If you omit DISTINCT, this query returns both Smith records.</p>

If the SELECT clause contains more than one field, the combination of values from all fields must be unique for a given record to be included in the results.

The output of a query that uses DISTINCT is not updatable and does not reflect subsequent changes made by other users.

DISTINCTROW Omits data based on entire duplicate records, not just duplicate fields. For example, you could create a query that joins the Customers and Orders tables on the CustomerID field. The Customers table contains no duplicate CustomerID fields, but the Orders table does because each customer can have many orders. The following SQL statement shows how you can use DISTINCTROW to produce a list of companies that have at least one order but without any details about those orders:

```
SELECT DISTINCTROW CompanyName
FROM Customers INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
ORDER BY CompanyName;
```

If you omit DISTINCTROW, this query produces multiple rows for each company that has more than one order.

DISTINCTROW has an effect only when you select fields from some, but not all, of the tables used in the query. DISTINCTROW is ignored if your query includes only one table, or if you output fields from all tables.

TOP *n*
[PERCENT] Returns a certain number of records that fall at the top or the bottom of a range specified by an ORDER BY clause. Suppose you want the names of the top 25 students from the class of 1994:

```
SELECT TOP 25
FirstName, LastName
FROM Students
WHERE GraduationYear = 1994
ORDER BY GradePointAverage DESC;
```


If you do not include the ORDER BY clause, the query will return an arbitrary set of 25 records from the Students table that satisfy the WHERE clause.

The TOP predicate does not choose between equal values. In the preceding example, if the twenty-fifth and twenty-sixth highest grade point averages are the same, the query will return 26 records.

You can also use the PERCENT reserved word to return a certain percentage of records that fall at the top or the bottom of a range specified by an ORDER BY clause. Suppose that, instead of the top 25 students, you want the bottom 10 percent of the class:

```
SELECT TOP 10 PERCENT  
FirstName, LastName  
FROM Students  
WHERE GraduationYear = 1994  
ORDER BY GradePointAverage ASC;
```

The ASC predicate specifies a return of bottom values. The value that follows TOP must be an unsigned [Integer](#).

TOP does not affect whether or not the query is updatable.

table The name of the table from which records are retrieved.

See Also

[FROM Clause \(Microsoft Jet SQL\)](#) [SELECT Statement \(Microsoft Jet SQL\)](#)

Example

[ALL, DISTINCT, DISTINCTROW, TOP Predicates Example](#)

DELETE

The DELETE keyword is used in these contexts:

[DELETE Statement](#)

[CREATE PROCEDURE Statement](#)

[GRANT Statement](#)

[REVOKE Statement](#)

DROP

The DROP keyword is used in these contexts:

[ALTER TABLE Statement](#)

[DROP Statement](#)

[DROP USER or GROUP Statement](#)

[GRANT Statement](#)

[REVOKE Statement](#)

FROM

The FROM keyword is used in these contexts:

[SELECT Statement](#)

[DROP USER or GROUP Statement](#)

[REVOKE Statement](#)

FROM Clause

Specifies the tables or queries that contain the fields listed in the [SELECT](#) statement.

Syntax

```
SELECT fieldlist  
FROM tableexpression [IN externaldatabase]
```

A SELECT statement containing a FROM clause has these parts:

Part	Description
<i>fieldlist</i>	The name of the field or fields to be retrieved along with any field-name aliases , SQL aggregate functions , selection predicates (ALL , DISTINCT , DISTINCTROW , or TOP), or other SELECT statement options.
<i>tableexpression</i>	An expression that identifies one or more tables from which data is retrieved. The expression can be a single table name, a saved query name, or a compound resulting from an INNER JOIN , LEFT JOIN , or RIGHT JOIN .
<i>externaldatabase</i>	The full path of an external database containing all the tables in <i>tableexpression</i> .

Remarks

FROM is required and follows any SELECT statement.

The order of the table names in *tableexpression* is not important.

For improved performance and ease of use, it is recommended that you use a [linked table](#) instead of an IN clause to retrieve data from an external database.

The following example shows how you can retrieve data from the Employees table:

```
SELECT LastName, FirstName  
FROM Employees;
```

See Also

[ALL DISTINCT, DISTINCTROW, TOP Predicates \(Microsoft Jet SQL\)](#) [SELECT Statement \(Microsoft Jet SQL\)](#)

[IN Clause \(Microsoft Jet SQL\)](#) [SQL Aggregate Functions \(SQL\)](#)

[LEFT JOIN, RIGHT JOIN Operations \(Microsoft Jet SQL\)](#) [WHERE Clause \(Microsoft Jet SQL\)](#)

[INNER JOIN Operation \(Microsoft Jet SQL\)](#)

Example

[SELECT Statement, FROM Clause Example](#)

GROUP BY Clause

Combines records with identical values in the specified field list into a single record. A summary value is created for each record if you include an [SQL aggregate function](#), such as **Sum** or **Count**, in the [SELECT](#) statement.

Syntax

```
SELECT fieldlist
  FROM table
  WHERE criteria
  [GROUP BY groupfieldlist]
```

A SELECT statement containing a GROUP BY clause has these parts:

Part	Description
<i>fieldlist</i>	The name of the field or fields to be retrieved along with any field-name aliases , SQL aggregate functions, selection predicates (ALL , DISTINCT , DISTINCTROW , or TOP), or other SELECT statement options.
<i>table</i>	The name of the table from which records are retrieved. For more information, see the FROM clause.
<i>criteria</i>	Selection criteria. If the statement includes a WHERE clause, the Microsoft Jet database engine groups values after applying the WHERE conditions to the records.
<i>groupfieldlist</i>	The names of up to 10 fields used to group records. The order of the field names in <i>groupfieldlist</i> determines the grouping levels from the highest to the lowest level of grouping.

Remarks

GROUP BY is optional.

Summary values are omitted if there is no SQL aggregate function in the SELECT statement.

[Null](#) values in GROUP BY fields are grouped and are not omitted. However, **Null** values are not evaluated in any SQL aggregate function.

Use the WHERE clause to exclude rows you do not want grouped, and use the [HAVING](#) clause to filter records after they have been grouped.

Unless it contains [Memo](#) or [OLE Object](#) data, a field in the GROUP BY field list can refer to any field in any table listed in the FROM clause, even if the field is not included in the SELECT statement, provided the SELECT statement includes at least one SQL aggregate function. The Microsoft® Jet database engine cannot group on Memo or OLE Object fields.

All fields in the SELECT field list must either be included in the GROUP BY clause or be included as arguments to an SQL aggregate function.

See Also

[ALL DISTINCT, DISTINCTROW, TOP Predicates \(Microsoft Jet SQL\)](#)

[SELECT Statement \(Microsoft Jet SQL\)](#)

[FROM Clause \(Microsoft Jet SQL\)](#)

[SELECT...INTO Statement \(Microsoft Jet SQL\)](#)

[HAVING Clause \(Microsoft Jet SQL\)](#)

[SQL Aggregate Functions \(SQL\)](#)

[ORDER BY Clause \(Microsoft Jet SQL\)](#)

[WHERE Clause \(Microsoft Jet SQL\)](#)

Example

[GROUP BY Clause Example](#)

FIRST

The FIRST keyword is used in these contexts:

[First, Last Functions](#)

[First, Last Functions Example](#)

HAVING Clause

Specifies which grouped records are displayed in a [SELECT](#) statement with a [GROUP BY](#) clause. After [GROUP BY](#) combines records, HAVING displays any records grouped by the [GROUP BY](#) clause that satisfy the conditions of the HAVING clause.

Syntax

```
SELECT fieldlist
FROM table
WHERE selectcriteria
GROUP BY groupfieldlist
[HAVING groupcriteria]
```

A SELECT statement containing a HAVING clause has these parts:

Part	Description
<i>fieldlist</i>	The name of the field or fields to be retrieved along with any field-name aliases , SQL aggregate functions , selection predicates (ALL , DISTINCT , DISTINCTROW , or TOP), or other SELECT statement options.
<i>table</i>	The name of the table from which records are retrieved. For more information, see the FROM clause.
<i>selectcriteria</i>	Selection criteria. If the statement includes a WHERE clause, the Microsoft Jet database engine groups values after applying the WHERE conditions to the records.
<i>groupfieldlist</i>	The names of up to 10 fields used to group records. The order of the field names in <i>groupfieldlist</i> determines the grouping levels from the highest to the lowest level of grouping.
<i>groupcriteria</i>	An expression that determines which grouped records to display.

Remarks

HAVING is optional.

HAVING is similar to WHERE, which determines which records are selected. After records are grouped with GROUP BY, HAVING determines which records

are displayed:

```
SELECT CategoryID,  
Sum(UnitsInStock)  
FROM Products  
GROUP BY CategoryID  
HAVING Sum(UnitsInStock) > 100 And Like "BOS*";
```

A HAVING clause can contain up to 40 expressions linked by logical operators, such as **And** and **Or**.

See Also

ALL DISTINCT, DISTINCTROW, TOP Predicates (Microsoft Jet SQL)	SELECT Statement (Microsoft Jet SQL)
FROM Clause (Microsoft Jet SQL)	SELECT...INTO Statement (Microsoft Jet SQL)
GROUP BY Clause (Microsoft Jet SQL)	SQL Aggregate Functions (SQL)
ORDER BY Clause (Microsoft Jet SQL)	WHERE Clause (Microsoft Jet SQL)

Example

[HAVING Clause Example](#)

INSERT

The INSERT keyword is used in these contexts:

[INSERT INTO Statement](#)

[CREATE PROCEDURE Statement](#)

[GRANT Statement](#)

[REVOKE Statement](#)

IN

The IN keyword is used in these contexts:

[In Operator](#)

[IN Clause](#)

[SQL Subqueries](#)

[TRANSFORM Statement](#)

INDEX

The INDEX keyword is used in these contexts:

[CREATE INDEX Statement](#)

[DROP Statement](#)

[GRANT Statement](#)

[REVOKE Statement](#)

INTO

The INTO keyword is used in these contexts:

[INSERT INTO Statement](#)

[SELECT ... INTO Statement](#)

JOIN

The JOIN keyword is used in these contexts:

[INNER JOIN Operation](#)

[LEFT JOIN, RIGHT JOIN Operations](#)

LAST

The LAST keyword is used in these contexts:

[First, Last Functions](#)

[First, Last Functions Example](#)

ON

The ON keyword is used in these contexts:

[INNER JOIN Operation](#)

[LEFT JOIN, RIGHT JOIN Operations](#)

[CONSTRAINT Clause](#)

[GRANT Statement](#)

[REVOKE Statement](#)

ORDER BY Clause

Sorts a query's resulting records on a specified field or fields in ascending or descending order.

Syntax

```
SELECT fieldlist  
FROM table  
WHERE selectcriteria  
[ORDER BY field1 [ASC | DESC ][, field2 [ASC | DESC ]][, ...]]
```

A SELECT statement containing an ORDER BY clause has these parts:

Part	Description
<i>fieldlist</i>	The name of the field or fields to be retrieved along with any field-name aliases , SQL aggregate functions , selection predicates (ALL , DISTINCT , DISTINCTROW , or TOP), or other SELECT statement options.
<i>table</i>	The name of the table from which records are retrieved. For more information, see the FROM clause.
<i>selectcriteria</i>	Selection criteria. If the statement includes a WHERE clause, the Microsoft Jet database engine orders values after applying the WHERE conditions to the records.
<i>field1, field2</i>	The names of the fields on which to sort records.

Remarks

ORDER BY is optional. However, if you want your data displayed in sorted order, then you must use ORDER BY.

The default [sort order](#) is ascending (A to Z, 0 to 9). Both of the following examples sort employee names in last name order:

```
SELECT LastName, FirstName  
FROM Employees  
ORDER BY LastName;  
SELECT LastName, FirstName
```

```
FROM Employees
ORDER BY LastName ASC;
```

To sort in descending order (Z to A, 9 to 0), add the DESC reserved word to the end of each field you want to sort in descending order. The following example selects salaries and sorts them in descending order:

```
SELECT LastName, Salary
FROM Employees
ORDER BY Salary DESC, LastName;
```

If you specify a field containing [Memo](#) or [OLE Object](#) data in the ORDER BY clause, an error occurs. The Microsoft Jet database engine does not sort on fields of these types.

ORDER BY is usually the last item in an [SQL statement](#).

You can include additional fields in the ORDER BY clause. Records are sorted first by the first field listed after ORDER BY. Records that have equal values in that field are then sorted by the value in the second field listed, and so on.

See Also

[ALL DISTINCT, DISTINCTROW, SELECT Statement \(Microsoft Jet SQL\)](#)
[TOP Predicates \(Microsoft Jet SQL\)](#)

[FROM Clause \(Microsoft Jet SQL\)](#) [SELECT...INTO Statement \(Microsoft Jet SQL\)](#)

[GROUP BY Clause \(Microsoft Jet SQL\)](#) [SQL Aggregate Functions \(SQL\)](#)

[HAVING Clause \(Microsoft Jet SQL\)](#) [WHERE Clause \(Microsoft Jet SQL\)](#)

Example

[ORDER BY Clause Example](#)

SCHEMA

The SCHEMA keyword is used in these contexts:

[GRANT Statement](#)

[REVOKE Statement](#)

SELECT

The SELECT keyword is used in these contexts:

[SELECT Statement](#)

[SELECT ... INTO Statement](#)

[CREATE VIEW Statement](#)

[CREATE PROCEDURE Statement](#)

[GRANT Statement](#)

[REVOKE Statement](#)

SELECTSCHEMA

The SELECTSCHEMA keyword is used in these contexts:

[GRANT Statement](#)

[REVOKE Statement](#)

SELECTSECURITY

The SELECTSECURITY keyword is used in these contexts:

[GRANT Statement](#)

[REVOKE Statement](#)

TABLE

The TABLE keyword is used in these contexts:

[ALTER TABLE Statement](#)

[CREATE INDEX Statement](#)

[CREATE TABLE Statement](#)

[DROP Statement](#)

[UNION Operation](#)

[GRANT Statement](#)

[REVOKE Statement](#)

UPDATEOWNER

The UPDATEOWNER keyword is used in these contexts:

[GRANT Statement](#)

[REVOKE Statement](#)

UPDATESECURITY

The UPDATESECURITY keyword is used in these contexts:

[GRANT Statement](#)

[REVOKE Statement](#)

USER

The USER keyword is used in these contexts:

[CREATE USER or GROUP Statement](#)

[ADD USER Statement](#)

[DROP USER or GROUP Statement](#)

[ALTER USER or DATABASE Statement](#)

TO

The TO keyword is used in these contexts:

[ADD USER Statement](#)

[GRANT Statement](#)

VIEW

The VIEW keyword is used in these contexts:

[CREATE VIEW Statement](#)

[DROP Statement](#)

WITH

The WITH keyword is used in these contexts:

[CREATE INDEX Statement](#)

[WITH OWNERACCESS OPTION Declaration](#)

UPDATE

The UPDATE keyword is used in these contexts:

[UPDATE Statement](#)

[CREATE PROCEDURE Statement](#)

[GRANT Statement](#)

[REVOKE Statement](#)

UPDATEIDENTITY

The UPDATEIDENTITY keyword is used in these contexts:

[GRANT Statement](#)

[REVOKE Statement](#)

CREATE TABLE Statement, CONSTRAINT Clause Example

This example creates a new table called ThisTable with two text fields.

```
Sub CreateTableX1()
    Dim dbs As Database
    ' Modify this line to include the path to Northwind
    ' on your computer.
    Set dbs = OpenDatabase("Northwind.mdb")
    ' Create a table with two text fields.
    dbs.Execute "CREATE TABLE ThisTable " _
        & "(FirstName CHAR, LastName CHAR);"
    dbs.Close
End Sub
```

This example creates a new table called MyTable with two text fields, a Date/Time field, and a unique index made up of all three fields.

```
Sub CreateTableX2()
    Dim dbs As Database
    ' Modify this line to include the path to Northwind
    ' on your computer.
    Set dbs = OpenDatabase("Northwind.mdb")
    ' Create a table with three fields and a unique
    ' index made up of all three fields.
    dbs.Execute "CREATE TABLE MyTable " _
        & "(FirstName CHAR, LastName CHAR, " _
        & "DateOfBirth DATETIME, " _
        & "CONSTRAINT MyTableConstraint UNIQUE " _
        & "(FirstName, LastName, DateOfBirth));"
    dbs.Close
End Sub
```

This example creates a new table with two text fields and an Integer field. The SSN field is the primary key.

```
Sub CreateTableX3()
```

```
Dim dbs As Database
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")
' Create a table with three fields and a primary
' key.
dbs.Execute "CREATE TABLE NewTable " _
    & "(FirstName CHAR, LastName CHAR, " _
    & "SSN INTEGER CONSTRAINT MyFieldConstraint " _
    & "PRIMARY KEY);"
dbs.Close
End Sub
```

CREATE INDEX Statement Example

This example creates an index consisting of the fields Home Phone and Extension in the Employees table.

```
Sub CreateIndexX1()
```

```
    Dim dbs As Database
```

```
    ' Modify this line to include the path to Northwind  
    ' on your computer.
```

```
    Set dbs = OpenDatabase("Northwind.mdb")
```

```
    ' Create the NewIndex index on the Employees table.
```

```
    dbs.Execute "CREATE INDEX NewIndex ON Employees " _  
        & "(HomePhone, Extension);"
```

```
    dbs.Close
```

```
End Sub
```

This example creates an index on the Customers table using the CustomerID field. No two records can have the same data in the CustomerID field, and no [Null](#) values are allowed.

```
Sub CreateIndexX2()
```

```
    Dim dbs As Database
```

```
    ' Modify this line to include the path to Northwind  
    ' on your computer.
```

```
    Set dbs = OpenDatabase("Northwind.mdb")
```

```
    ' Create a unique index, CustID, on the  
    ' CustomerID field.
```

```
    dbs.Execute "CREATE UNIQUE INDEX CustID " _  
        & "ON Customers (CustomerID) " _  
        & "WITH DISALLOW NULL;"
```

```
    dbs.Close
```

```
End Sub
```

CREATE PROCEDURE Statement, PROCEDURE Clause Example

This example names the query CategoryList.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub ProcedureX()

Dim dbs As Database, rst As Recordset

Dim qdf As QueryDef, strSql As String

' Modify this line to include the path to Northwind
' on your computer.

Set dbs = OpenDatabase("Northwind.mdb")

strSql = "PROCEDURE CategoryList; " _
& "SELECT DISTINCTROW CategoryName, " _
& "CategoryID FROM Categories " _
& "ORDER BY CategoryName;"

' Create a named QueryDef based on the SQL
' statement.

Set qdf = dbs.CreateQueryDef("NewQry", strSql)

' Create a temporary snapshot-type Recordset.

Set rst = qdf.OpenRecordset(dbOpenSnapshot)

' Populate the Recordset.

rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.

EnumFields rst, 15

' Delete the QueryDef because this is a
' demonstration.

```
dbms.QueryDefs.Delete "NewQry"
```

```
dbms.Close
```

```
End Sub
```

ALTER TABLE Statement Example

This example adds a Salary field with the data type **Money** to the Employees table.

```
Sub AlterTableX1()  
    Dim dbs As Database  
    ' Modify this line to include the path to Northwind  
    ' on your computer.  
    Set dbs = OpenDatabase("Northwind.mdb")  
    ' Add the Salary field to the Employees table  
    ' and make it a Money data type.  
    dbs.Execute "ALTER TABLE Employees " _  
        & "ADD COLUMN Salary MONEY;"  
    dbs.Close  
End Sub
```

This example changes the Salary field from the data type **Money** to the data type **Char**.

```
Sub AlterTableX2()  
    Dim dbs As Database  
    ' Modify this line to include the path to Northwind  
    ' on your computer.  
    Set dbs = OpenDatabase("Northwind.mdb")  
    ' Add the Salary field to the Employees table  
    ' and make it a Money data type.  
    dbs.Execute "ALTER TABLE Employees " _  
        & "ALTER COLUMN Salary CHAR(20);"  
    dbs.Close  
End Sub
```

This example removes the Salary field from the Employees table.

```
Sub AlterTableX3()  
    Dim dbs As Database
```

```

' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")
' Delete the Salary field from the Employees table.
dbs.Execute "ALTER TABLE Employees " _
    & "DROP COLUMN Salary;"
dbs.Close
End Sub

```

This example adds a [foreign key](#) to the Orders table. The foreign key is based on the EmployeeID field and refers to the EmployeeID field of the Employees table. In this example, you do not have to list the EmployeeID field after the Employees table in the REFERENCES clause because EmployeeID is the [primary key](#) of the Employees table.

```

Sub AlterTableX4()
    Dim dbs As Database
    ' Modify this line to include the path to Northwind
    ' on your computer.
    Set dbs = OpenDatabase("Northwind.mdb")
    ' Add a foreign key to the Orders table.
    dbs.Execute "ALTER TABLE Orders " _
        & "ADD CONSTRAINT OrdersRelationship " _
        & "FOREIGN KEY (EmployeeID) " _
        & "REFERENCES Employees (EmployeeID);"
    dbs.Close
End Sub

```

This example removes the foreign key from the Orders table.

```

Sub AlterTableX5()
    Dim dbs As Database
    ' Modify this line to include the path to Northwind
    ' on your computer.
    Set dbs = OpenDatabase("Northwind.mdb")
    ' Remove the OrdersRelationship foreign key from

```



```
' the Orders table.  
dbs.Execute "ALTER TABLE Orders " _  
    & "DROP CONSTRAINT OrdersRelationship;"  
dbs.Close  
End Sub
```

DROP Statement Example

The following example assumes the existence of a hypothetical NewIndex index on the Employees table in the Northwind database.

This example deletes the index MyIndex from the Employees table.

```
Sub DropX1()
```

```
    Dim dbs As Database
```

```
    ' Modify this line to include the path to Northwind
```

```
    ' on your computer.
```

```
    Set dbs = OpenDatabase("Northwind.mdb")
```

```
    ' Delete NewIndex from the Employees table.
```

```
    dbs.Execute "DROP INDEX NewIndex ON Employees;"
```

```
    dbs.Close
```

```
End Sub
```

This example deletes the Employees table from the database.

```
Sub DropX2()
```

```
    Dim dbs As Database
```

```
    ' Modify this line to include the path to Northwind
```

```
    ' on your computer.
```

```
    Set dbs = OpenDatabase("Northwind.mdb")
```

```
    ' Delete the Employees table.
```

```
    dbs.Execute "DROP TABLE Employees;"
```

```
    dbs.Close
```

```
End Sub
```

SELECT Statement, FROM Clause Example

Some of the following examples assume the existence of a hypothetical Salary field in an Employees table. Note that this field does not actually exist in the Northwind database Employees table.

This example creates a dynaset-type **Recordset** based on an [SQL statement](#) that selects the LastName and FirstName fields of all records in the Employees table. It calls the EnumFields procedure, which prints the contents of a **Recordset** object to the **Debug** window.

Sub SelectX1()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")
' Select the last name and first name values of all
' records in the Employees table.
Set rst = dbs.OpenRecordset("SELECT LastName, " _
    & "FirstName FROM Employees;")
' Populate the recordset.
rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset.

EnumFields rst,12

dbs.Close
End Sub
```

This example counts the number of records that have an entry in the PostalCode field and names the returned field Tally.

Sub SelectX2()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
```

```

Set dbs = OpenDatabase("Northwind.mdb")
' Count the number of records with a PostalCode
' value and return the total in the Tally field.
Set rst = dbs.OpenRecordset("SELECT Count " _
    & "(PostalCode) AS Tally FROM Customers;")
' Populate the Recordset.
rst.MoveLast
' Call EnumFields to print the contents of
' the Recordset. Specify field width = 12.
EnumFields rst, 12
dbs.Close
End Sub

```

This example shows the number of employees and the average and maximum salaries.

```

Sub SelectX3()
    Dim dbs As Database, rst As Recordset
    ' Modify this line to include the path to Northwind
    ' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")
' Count the number of employees, calculate the
' average salary, and return the highest salary.
Set rst = dbs.OpenRecordset("SELECT Count (*) " _
    & "AS TotalEmployees, Avg(Salary) " _
    & "AS AverageSalary, Max(Salary) " _
    & "AS MaximumSalary FROM Employees;")
' Populate the Recordset.
rst.MoveLast
' Call EnumFields to print the contents of
' the Recordset. Pass the Recordset object and
' desired field width.
EnumFields rst, 17
dbs.Close

```

End Sub

The **Sub** procedure EnumFields is passed a **Recordset** object from the calling procedure. The procedure then formats and prints the fields of the **Recordset** to the **Debug** window. The intFldLen variable is the desired printed field width. Some fields may be truncated.

```
Sub EnumFields(rst As Recordset, intFldLen As Integer)
```

```
    Dim lngRecords As Long, lngFields As Long
```

```
    Dim lngRecCount As Long, lngFldCount As Long
```

```
    Dim strTitle As String, strTemp As String
```

```
    ' Set the lngRecords variable to the number of
```

```
    ' records in the Recordset.
```

```
    lngRecords = rst.RecordCount
```

```
    ' Set the lngFields variable to the number of
```

```
    ' fields in the Recordset.
```

```
    lngFields = rst.Fields.Count
```

```
    Debug.Print "There are " & lngRecords _
```

```
        & " records containing " & lngFields _
```

```
        & " fields in the recordset."
```

```
    Debug.Print
```

```
    ' Form a string to print the column heading.
```

```
    strTitle = "Record "
```

```
    For lngFldCount = 0 To lngFields - 1
```

```
        strTitle = strTitle _
```

```
            & Left(rst.Fields(lngFldCount).Name _
```

```
                & Space(intFldLen), intFldLen)
```

```
    Next lngFldCount
```

```
    ' Print the column heading.
```

```
    Debug.Print strTitle
```

```
    Debug.Print
```

```

' Loop through the Recordset; print the record
' number and field values.
rst.MoveFirst
For lngRecCount = 0 To lngRecords - 1
    Debug.Print Right(Space(6) & _
        Str(lngRecCount), 6) & " ";
    For lngFldCount = 0 To lngFields - 1
        ' Check for Null values.
        If IsNull(rst.Fields(lngFldCount)) Then
            strTemp = "<null>"
        Else
            ' Set strTemp to the field contents.
            Select Case _
                rst.Fields(lngFldCount).Type
            Case 11
                strTemp = ""
            Case dbText, dbMemo
                strTemp = _
                    rst.Fields(lngFldCount)
            Case Else
                strTemp = _
                    str(rst.Fields(lngFldCount))
            End Select
        End If
        Debug.Print Left(strTemp _
            & Space(intFldLen), intFldLen);
    Next lngFldCount
    Debug.Print
    rst.MoveNext
Next lngRecCount
End Sub

```

SELECT...INTO Statement Example

This example selects all records in the Employees table and copies them into a new table named Emp Backup.

```
Sub SelectIntoX()
```

```
    Dim dbs As Database
```

```
    Dim qdf As QueryDef
```

```
    ' Modify this line to include the path to Northwind
```

```
    ' on your computer.
```

```
    Set dbs = OpenDatabase("Northwind.mdb")
```

```
    ' Select all records in the Employees table
```

```
    ' and copy them into a new table, Emp Backup.
```

```
    dbs.Execute "SELECT Employees.* INTO " _
```

```
        & "[Emp Backup] FROM Employees;"
```

```
    ' Delete the table because this is a demonstration.
```

```
    dbs.Execute "DROP TABLE [Emp Backup];"
```

```
    dbs.Close
```

```
End Sub
```

INSERT INTO Statement Example

This example selects all records in a hypothetical New Customers table and adds them to the Customers table. When individual columns are not designated, the SELECT table column names must match exactly those in the INSERT INTO table.

Sub InsertIntoX1()

```
Dim dbs As Database
```

```
' Modify this line to include the path to Northwind  
' on your computer.
```

```
Set dbs = OpenDatabase("Northwind.mdb")
```

```
' Select all records in the New Customers table  
' and add them to the Customers table.
```

```
dbs.Execute " INSERT INTO Customers " _  
    & "SELECT * " _  
    & "FROM [New Customers];"
```

```
dbs.Close
```

```
End Sub
```

This example creates a new record in the Employees table.

Sub InsertIntoX2()

```
Dim dbs As Database
```

```
' Modify this line to include the path to Northwind  
' on your computer.
```

```
Set dbs = OpenDatabase("Northwind.mdb")
```

```
' Create a new record in the Employees table. The  
' first name is Harry, the last name is Washington,  
' and the job title is Trainee.
```

```
dbs.Execute " INSERT INTO Employees " _  
    & "(FirstName,LastName, Title) VALUES " _  
    & "('Harry', 'Washington', 'Trainee');"
```


dbz.Close
End Sub

UPDATE Statement Example

This example changes values in the ReportsTo field to 5 for all employee records that currently have ReportsTo values of 2.

Sub UpdateX()

Dim dbs As Database

Dim qdf As QueryDef

' Modify this line to include the path to Northwind

' on your computer.

Set dbs = OpenDatabase("Northwind.mdb")

' Change values in the ReportsTo field to 5 for all

' employee records that currently have ReportsTo

' values of 2.

dbs.Execute "UPDATE Employees " _

& "SET ReportsTo = 5 " _

& "WHERE ReportsTo = 2;"

dbs.Close

End Sub

DELETE Statement Example

This example deletes all records for employees whose title is Trainee. When the FROM clause includes only one table, you do not have to list the table name in the DELETE statement.

```
Sub DeleteX()
```

```
    Dim dbs As Database, rst As Recordset
```

```
    ' Modify this line to include the path to Northwind
```

```
    ' on your computer.
```

```
    Set dbs = OpenDatabase("Northwind.mdb")
```

```
    ' Delete employee records where title is Trainee.
```

```
    dbs.Execute "DELETE * FROM " _
```

```
        & "Employees WHERE Title = 'Trainee';"
```

```
    dbs.Close
```

```
End Sub
```

TRANSFORM Statement Example

This example uses the SQL TRANSFORM clause to create a [crosstab query](#) showing the number of orders taken by each employee for each calendar quarter of 1994. The SQLTRANSFORMOutput function is required for this procedure to run.

Sub TransformX1()

```
Dim dbs As Database
```

```
Dim strSQL As String
```

```
Dim qdfTRANSFORM As QueryDef
```

```
strSQL = "PARAMETERS prmYear SHORT; TRANSFORM " _  
    & "Count(OrderID) " _  
    & "SELECT FirstName & "" "" & LastName AS " _  
    & "FullName FROM Employees INNER JOIN Orders " _  
    & "ON Employees.EmployeeID = " _  
    & "Orders.EmployeeID WHERE DatePart " _  
    & "(" & ""yyyy"" , OrderDate) = [prmYear] "
```

```
strSQL = strSQL & "GROUP BY FirstName & " _  
    & "" "" & LastName " _  
    & "ORDER BY FirstName & "" "" & LastName " _  
    & "PIVOT DatePart(""q"" , OrderDate)"
```

```
' Modify this line to include the path to Northwind  
' on your computer.
```

```
Set dbs = OpenDatabase("Northwind.mdb")
```

```
Set qdfTRANSFORM = dbs.CreateQueryDef _  
    ("", strSQL)
```

```
SQLTRANSFORMOutput qdfTRANSFORM, 1994
```

```
dbs.Close
```

```
End Sub
```

This example uses the SQL TRANSFORM clause to create a slightly more complex crosstab query showing the total dollar amount of orders taken by each employee for each calendar quarter of 1994. The SQLTRANSFORMOutput function is required for this procedure to run.

```
Sub TransformX2()
```

```
    Dim dbs As Database
```

```
    Dim strSQL As String
```

```
    Dim qdfTRANSFORM As QueryDef
```

```
    strSQL = "PARAMETERS prmYear SMALLINT; TRANSFORM "
```

```
    & "Sum(Subtotal) SELECT FirstName & "" "" _  
    & "& LastName AS FullName " _  
    & "FROM Employees INNER JOIN " _  
    & "(Orders INNER JOIN [Order Subtotals] " _  
    & "ON Orders.OrderID = " _  
    & "[Order Subtotals].OrderID) " _  
    & "ON Employees.EmployeeID = " _  
    & "Orders.EmployeeID WHERE DatePart" _  
    & "(" & "yyyy", OrderDate) = [prmYear] "
```

```
    strSQL = strSQL & "GROUP BY FirstName & "" "" _  
    & "& LastName " _  
    & "ORDER BY FirstName & "" "" & LastName " _  
    & "PIVOT DatePart(""q"", OrderDate)"
```

```
' Modify this line to include the path to Northwind  
' on your computer.
```

```
Set dbs = OpenDatabase("Northwind.mdb")
```

```
Set qdfTRANSFORM = dbs.CreateQueryDef _  
    ("", strSQL)
```

```
SQLTRANSFORMOutput qdfTRANSFORM, 1994
```

```

    dbs.Close
End Sub
Function SQLTRANSFORMOutput(qdfTemp As QueryDef, _
    intYear As Integer)

    Dim rstTRANSFORM As Recordset
    Dim fldLoop As Field
    Dim booFirst As Boolean
    qdfTemp.PARAMETERS!prmYear = intYear
    Set rstTRANSFORM = qdfTemp.OpenRecordset()

    Debug.Print qdfTemp.SQL
    Debug.Print
    Debug.Print , , "Quarter"
    With rstTRANSFORM
        booFirst = True
        For Each fldLoop In .Fields
            If booFirst = True Then
                Debug.Print fldLoop.Name
                Debug.Print , ;
                booFirst = False
            Else
                Debug.Print , fldLoop.Name;
            End If
        Next fldLoop
        Debug.Print

    Do While Not .EOF
        booFirst = True
        For Each fldLoop In .Fields
            If booFirst = True Then
                Debug.Print fldLoop
                Debug.Print , ;
            End If
        Next fldLoop
    Loop
End Function

```

```
        booFirst = False
    Else
        Debug.Print , fldLoop;
    End If
Next fldLoop
Debug.Print
.MoveNext
Loop
End With

End Function
```

INNER JOIN Operation Example

This example creates two [equi-joins](#): one between the Order Details and Orders tables and another between the Orders and Employees tables. This is necessary because the Employees table does not contain sales data, and the Order Details table does not contain employee data. The query produces a list of employees and their total sales.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub InnerJoinX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")

' Create a join between the Order Details and
' Orders tables and another between the Orders and
' Employees tables. Get a list of employees and
' their total sales.
Set rst = dbs.OpenRecordset("SELECT DISTINCTROW " _
    & "Sum(UnitPrice * Quantity) AS Sales, " _
    & "(FirstName & Chr(32) & LastName) AS Name " _
    & "FROM Employees INNER JOIN(Orders " _
    & "INNER JOIN [Order Details] " _
    & "ON [Order Details].OrderID = " _
    & "Orders.OrderID ) " _
    & "ON Orders.EmployeeID = " _
    & "Employees.EmployeeID " _
    & "GROUP BY (FirstName & Chr(32) & LastName);")

' Populate the Recordset.
rst.MoveLast
```



```
' Call EnumFields to print the contents of the  
' Recordset. Pass the Recordset object and desired  
' field width.  
EnumFields rst, 20  
dbs.Close  
End Sub
```

LEFT JOIN, RIGHT JOIN Operations Example

This example assumes the existence of hypothetical Department Name and Department ID fields in an Employees table. Note that these fields do not actually exist in the Northwind database Employees table.

This example selects all departments, including those without employees.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub LeftRightJoinX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")

' Select all departments, including those
' without employees.
Set rst = dbs.OpenRecordset _
("SELECT [Department Name], " _
& "FirstName & Chr(32) & LastName AS Name " _
& "FROM Departments LEFT JOIN Employees " _
& "ON Departments.[Department ID] = " _
& "Employees.[Department ID] " _
& "ORDER BY [Department Name];")

' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.
EnumFields rst, 20
dbs.Close
End Sub
```

UNION Operation Example

This example retrieves the names and cities of all suppliers and customers in Brazil.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub UnionX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")

' Retrieve the names and cities of all suppliers
' and customers in Brazil.
Set rst = dbs.OpenRecordset("SELECT CompanyName," _
    & " City FROM Suppliers" _
    & " WHERE Country = 'Brazil' UNION" _
    & " SELECT CompanyName, City FROM Customers" _
    & " WHERE Country = 'Brazil';")

' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.
EnumFields rst, 12
dbs.Close
End Sub
```

PARAMETERS Declaration Example

This example requires the user to provide a job title and then uses that job title as the criteria for the query.

This example calls the EnumFields procedure, which you can find in the [SELECT statement](#) example.

```
Sub ParametersX()
```

```
    Dim dbs As Database, qdf As QueryDef
```

```
    Dim rst As Recordset
```

```
    Dim strSql As String, strParm As String
```

```
    Dim strMessage As String
```

```
    Dim intCommand As Integer
```

```
    ' Modify this line to include the path to Northwind
```

```
    ' on your computer.
```

```
    Set dbs = OpenDatabase("NorthWind.mdb")
```

```
    ' Define the parameters clause.
```

```
    strParm = "PARAMETERS [Employee Title] CHAR; "
```

```
    ' Define an SQL statement with the parameters
```

```
    ' clause.
```

```
    strSql = strParm & "SELECT LastName, FirstName, " _
```

```
        & "EmployeeID " _
```

```
        & "FROM Employees " _
```

```
        & "WHERE Title =[Employee Title];"
```

```
    ' Create a QueryDef object based on the
```

```
    ' SQL statement.
```

```
    Set qdf = dbs.CreateQueryDef _
```

```
        ("Find Employees", strSql)
```

```
    Do While True
```

```
        strMessage = "Find Employees by Job " _
```

```
& "title:" & Chr(13) _  
& " Choose Job Title:" & Chr(13) _  
& " 1 - Sales Manager" & Chr(13) _  
& " 2 - Sales Representative" & Chr(13) _  
& " 3 - Inside Sales Coordinator"
```

```
intCommand = Val(InputBox(strMessage))
```

```
Select Case intCommand
```

```
Case 1
```

```
qdf("Employee Title") = _  
"Sales Manager"
```

```
Case 2
```

```
qdf("Employee Title") = _  
"Sales Representative"
```

```
Case 3
```

```
qdf("Employee Title") = _  
"Inside Sales Coordinator"
```

```
Case Else
```

```
Exit Do
```

```
End Select
```

```
' Create a temporary snapshot-type Recordset.  
Set rst = qdf.OpenRecordset(dbOpenSnapshot)  
' Populate the Recordset.  
rst.MoveLast
```

```
' Call EnumFields to print the contents of the  
' Recordset. Pass the Recordset object and desired  
' field width.
```

```
EnumFields rst, 12
```

```
Loop
```

```
' Delete the QueryDef because this is a  
' demonstration.  
dbs.QueryDefs.Delete "Find Employees"
```

```
dbs.Close  
End Sub
```

SQL Subqueries Example

This example lists the name and contact of every customer who placed an order in the second quarter of 1995.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub SubQueryX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")

' List the name and contact of every customer
' who placed an order in the second quarter of
' 1995.
Set rst = dbs.OpenRecordset("SELECT ContactName," _
    & " CompanyName, ContactTitle, Phone" _
    & " FROM Customers" _
    & " WHERE CustomerID" _
    & " IN (SELECT CustomerID FROM Orders" _
    & " WHERE OrderDate Between #04/1/95#" _
    & " And #07/1/95#);")

' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.
EnumFields rst, 25
dbs.Close
End Sub
```

Using Regedit.exe to Overwrite the Default Settings

To customize the Microsoft® Windows® Registry settings, you can use Regedit.exe to overwrite the default settings that are established when the Microsoft Jet database engine is registered. This method of modification is the least flexible method because all applications that do not specify another registry location will have these new default settings.

See Also

[Creating a Portion in Your Application's Registry Tree to Manage the Settings](#)

[Using the Connection Properties in the Microsoft OLE DB Provider for Jet](#)

[Using the SetOption Method from DAO](#)

Creating a Portion in Your Application's Registry Tree to Manage the Settings

To customize the Microsoft® Windows® Registry settings, you can create a Microsoft Jet portion in your application's registry tree to manage the settings for the Microsoft Jet database engine. The easiest way to accomplish this is to export the existing Microsoft Jet key and then import it into your application's tree with the Regedit.exe Export and Import commands. You can then alter the values in your new registry tree. If you have supplied any values in the Engines subfolder, Microsoft Jet loads those settings when the application starts. Any values not entered in your client application's registry tree are loaded from shadow settings.

For your application to load the appropriate portion of the Windows Registry key you must specify the location with the DAO **INIPath** property. Your application must set the **INIPath** property before executing any other DAO code. The scope of this setting is limited to your application and cannot be changed without restarting your application.

Note Although creating a Microsoft Jet portion in your application's registry is more flexible than overwriting the Microsoft Jet default entries, it requires that you maintain the registry tree. Every time changes are required in the default settings, you will need to edit the Registry.

See Also

[Using Regedit.exe to Overwrite the Default Settings](#)

[Using the Connection Properties in the Microsoft OLE DB Provider for Jet](#)

[Using the SetOption Method from DAO](#)

Using the SetOption Method from DAO

DAO version 3.6 provides a new way to modify default settings. Microsoft® Windows® Registry settings can now be modified at run time with the **SetOption** method. To customize the Windows Registry settings, you can use the **SetOption** method from DAO. With this option, your application obtains the maximum flexibility and control. This approach allows you to create applications that are easier to maintain and that are tuned for maximum performance.

The syntax for doing this is `dbEngine.SetOption, constant, NewValueSetting`. For example, the following syntax, `dbEngine.SetOption dbMaxLocksPerFile, 20000`, would allow Microsoft Jet to track 20,000 locks at one time. The names of defined constants are the same as the registry name with db added as a prefix.

This is the recommended way to fine tune registry settings for your application. This method is the most flexible approach and provides you with the most control over how the registry is changed. With The **SetOption** method you can specify new settings for any of the following default settings:

PageTimeout key

SharedAsyncDelay key

ExclusiveAsyncDelay key

LockRetry key

UserCommitSync key

ImplicitCommitSync key

MaxBufferSize key

MaxLocksPerFile key

LockDelay key

RecycleLVs

FlushTransactionTimeout key

See Also

[Using Regedit.exe to Overwrite the Default Settings](#)

[Using the Connection Properties in the Microsoft OLE DB Provider for Jet](#)

[Creating a Portion in Your Application's Registry Tree to Manage the Settings](#)

Using the Connection Properties in the Microsoft OLE DB Provider for Jet

To customize the Microsoft® Windows® Registry settings, you can use the connection properties in the Microsoft OLE DB Provider for Jet. This is accomplished by referencing a property in the connection object and changing its value. For example, assuming that your connection object is called ADOConnection, the following would yield the same results as going through ADO:

```
ADOConnection.Properties("Jet OLEDB:Max Locks Per File") = 20000
```

The property names are different than the DAO constants and the registry settings. The property names are as follows:

Jet OLEDB:Max Locks Per File

Jet OLEDB:Implicit Commit Sync

Jet OLEDB:Flush Transaction Timeout

Jet OLEDB:Lock Delay

Jet OLEDB:Max Buffer Size

Jet OLEDB>User Commit Sync

Jet OLEDB:Lock Retry

Jet OLEDB:Exclusive Async Delay

Jet OLEDB:Shared Async Delay

Jet OLEDB:Page Timeout

Jet OLEDB:Recycle Long-Valued Pages

See Also

[Using Regedit.exe to Overwrite the Default Settings](#)

[Using the SetOption Method from DAO](#)

[Creating a Portion in Your Application's Registry Tree to Manage the Settings](#)

Calculating Fields in SQL Functions

You can use the [string expression](#) argument in an [SQL aggregate function](#) to perform a calculation on values in a field. For example, you could calculate a percentage (such as a surcharge or sales tax) by multiplying a field value by a fraction.

The following table provides examples of calculations on fields from the Orders and Order Details tables in the Northwind.mdb database.

Calculation	Example
Add a number to a field	Freight + 5
Subtract a number from a field	Freight - 5
Multiply a field by a number	UnitPrice * 2
Divide a field by a number	Freight / 2
Add one field to another	UnitsInStock + UnitsOnOrder
Subtract one field from another	ReorderLevel - UnitsInStock

The following example calculates the average discount amount of all orders in the Northwind.mdb database. It multiplies the values in the UnitPrice and Discount fields to determine the discount amount of each order and then calculates the average. You can use this expression in an [SQL statement](#) in Visual Basic code:

```
SELECT Avg(UnitPrice * Discount) AS [Average Discount] FROM [Order Details];
```

See Also

[SQL Aggregate Functions \(SQL\)](#)

[SQL Expressions](#)

Avg Function Example

This example uses the Orders table to calculate the average freight charges for orders with freight charges over \$100.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub AvgX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")
' Calculate the average freight charges for orders
' with freight charges over $100.
Set rst = dbs.OpenRecordset("SELECT Avg(Freight)" _
    & " AS [Average Freight]" _
    & " FROM Orders WHERE Freight > 100;")

' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.
EnumFields rst, 25
dbs.Close
End Sub
```

Count Function Example

This example uses the Orders table to calculate the number of orders shipped to the United Kingdom.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub CountX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")

' Calculate the number of orders shipped
' to the United Kingdom.
Set rst = dbs.OpenRecordset("SELECT" _
    & " Count (ShipCountry)" _
    & " AS [UK Orders] FROM Orders" _
    & " WHERE ShipCountry = 'UK';")

' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.
EnumFields rst, 25
dbs.Close
End Sub
```

First, Last Functions Example

This example uses the Employees table to return the values from the LastName field of the first and last records returned from the table.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

```
Sub FirstLastX1()
```

```
    Dim dbs As Database, rst As Recordset
```

```
    ' Modify this line to include the path to Northwind  
    ' on your computer.
```

```
    Set dbs = OpenDatabase("Northwind.mdb")
```

```
    ' Return the values from the LastName field of the  
    ' first and last records returned from the table.
```

```
    Set rst = dbs.OpenRecordset("SELECT " _  
        & "First(LastName) as First, " _  
        & "Last(LastName) as Last FROM Employees;")
```

```
    ' Populate the Recordset.
```

```
    rst.MoveLast
```

```
    ' Call EnumFields to print the contents of the  
    ' Recordset. Pass the Recordset object and desired  
    ' field width.
```

```
    EnumFields rst, 12
```

```
    dbs.Close
```

```
End Sub
```

The next example compares using the **First** and **Last** functions with simply using the **Min** and **Max** functions to find the earliest and latest birth dates of Employees.

```
Sub FirstLastX2()
```

```
    Dim dbs As Database, rst As Recordset
```



```

' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")

' Find the earliest and latest birth dates of
' Employees.
Set rst = dbs.OpenRecordset("SELECT " _
    & "First(BirthDate) as FirstBD, " _
    & "Last(BirthDate) as LastBD FROM Employees;")

' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.
EnumFields rst, 12

Debug.Print

' Find the earliest and latest birth dates of
' Employees.
Set rst = dbs.OpenRecordset("SELECT " _
    & "Min(BirthDate) as MinBD," _
    & "Max(BirthDate) as MaxBD FROM Employees;")

' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.
EnumFields rst, 12
dbs.Close

```

End Sub

Min, Max Functions Example

This example uses the Orders table to return the lowest and highest freight charges for orders shipped to the United Kingdom.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub MinMaxX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")

' Return the lowest and highest freight charges for
' orders shipped to the United Kingdom.
Set rst = dbs.OpenRecordset("SELECT " _
    & "Min(Freight) AS [Low Freight], " _
    & "Max(Freight)AS [High Freight] " _
    & "FROM Orders WHERE ShipCountry = 'UK';")

' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.
EnumFields rst, 12
dbs.Close
End Sub
```

StDev, StDevP Functions Example

This example uses the Orders table to estimate the [standard deviation](#) of the freight charges for orders shipped to the United Kingdom.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub StDevX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")
' Calculate the standard deviation of the freight
' charges for orders shipped to the United Kingdom.
Set rst = dbs.OpenRecordset("SELECT " _
    & "StDev(Freight) " _
    & "AS [Freight Deviation] FROM Orders " _
    & "WHERE ShipCountry = 'UK';")
' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.
EnumFields rst, 15

Debug.Print

Set rst = dbs.OpenRecordset("SELECT " _
    & "StDevP(Freight) " _
    & "AS [Freight DevP] FROM Orders " _
    & "WHERE ShipCountry = 'UK';")
' Populate the Recordset.
rst.MoveLast
```

```
' Call EnumFields to print the contents of the  
' Recordset. Pass the Recordset object and desired  
' field width.  
EnumFields rst, 15  
dbs.Close  
End Sub
```

Sum Function Example

This example uses the Orders table to calculate the total sales for orders shipped to the United Kingdom.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub SumX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")
' Calculate the total sales for orders shipped to
' the United Kingdom.
Set rst = dbs.OpenRecordset("SELECT" _
    & " Sum(UnitPrice*Quantity)" _
    & " AS [Total UK Sales] FROM Orders" _
    & " INNER JOIN [Order Details] ON" _
    & " Orders.OrderID = [Order Details].OrderID" _
    & " WHERE (ShipCountry = 'UK');")
' Populate the Recordset.
rst.MoveLast
' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.
EnumFields rst, 15

dbs.Close
End Sub
```

Var, VarP Functions Example

This example uses the Orders table to estimate the [variance](#) of freight costs for orders shipped to the United Kingdom.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub VarX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")
' Calculate the variance of freight costs for
' orders shipped to the United Kingdom.
Set rst = dbs.OpenRecordset("SELECT " _
    & "Var(Freight) " _
    & "AS [UK Freight Variance] " _
    & "FROM Orders WHERE ShipCountry = 'UK';")
' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.
EnumFields rst, 20

Debug.Print

Set rst = dbs.OpenRecordset("SELECT " _
    & "VarP(Freight) " _
    & "AS [UK Freight VarianceP] " _
    & "FROM Orders WHERE ShipCountry = 'UK';")
' Populate the Recordset.
rst.MoveLast
```

```
' Call EnumFields to print the contents of the  
' Recordset. Pass the Recordset object and desired  
' field width.  
EnumFields rst, 20  
dbs.Close  
End Sub
```


WHERE Clause Example

The following example assumes the existence of a hypothetical Salary field in an Employees table. Note that this field does not actually exist in the Northwind database Employees table.

This example selects the LastName and FirstName fields of each record in which the last name is King.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub WhereX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")
' Select records from the Employees table where the
' last name is King.
Set rst = dbs.OpenRecordset("SELECT LastName, " _
    & "FirstName FROM Employees " _
    & "WHERE LastName = 'King';")

' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset.
EnumFields rst, 12
dbs.Close
End Sub
```

In Operator Example

The following example uses the Orders table in the Northwind.mdb database to create a query that includes all orders shipped to Lancashire and Essex and the dates shipped.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub InX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")
' Select records from the Orders table that
' have a ShipRegion value of Lancashire or Essex.
Set rst = dbs.OpenRecordset("SELECT " _
    & "CustomerID, ShippedDate FROM Orders " _
    & "WHERE ShipRegion In " _
    & "('Lancashire','Essex');")

' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print the contents of
' the Recordset.
EnumFields rst, 12
dbs.Close
End Sub
```

Like Operator Example

This example returns a list of employees whose names begin with the letters A through D.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub LikeX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")
' Return a list of employees whose names begin with
' the letters A through D.
Set rst = dbs.OpenRecordset("SELECT LastName, " _
    & " FirstName FROM Employees" _
    & " WHERE LastName Like '[A-D]*';")
' Populate the Recordset.
rst.MoveLast
' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.
EnumFields rst, 15

dbs.Close
End Sub
```

IN Clause Example

The following table shows how you can use the IN clause to retrieve data from an external database. In each example, assume the hypothetical Customers table is stored in an external database.

External database	SQL statement
Microsoft® Jet database	SELECT CustomerID FROM Customers IN OtherDB.mdb WHERE CustomerID Like "A*";
dBASE III or IV. To retrieve data from a dBASE III table, substitute "dBASE III;" for "dBASE IV;".	SELECT CustomerID FROM Customer IN "C:\DBASE\DATA\SALES" "dBASE IV;" WHERE CustomerID Like "A*";
dBASE III or IV using Database syntax.	SELECT CustomerID FROM Customer IN "" [dBASE IV; Database=C:\DBASE\DATA\SALES;] WHERE CustomerID Like "A*";
Paradox 3.x or 4.x. To retrieve data from a Paradox version 3.x table, substitute "Paradox 3.x;" for "Paradox 4.x;".	SELECT CustomerID FROM Customer IN "C:\PARADOX\DATA\SALES" "Paradox 4.x;" WHERE CustomerID Like "A*";
Paradox 3.x or 4.x using Database syntax.	SELECT CustomerID FROM Customer IN "" [Paradox 4.x;Database=C:\PARADOX\DATA\SALES;] WHERE CustomerID Like "A*";
A Microsoft Excel	SELECT CustomerID, CompanyName

worksheet

```
FROM [Customers$]
IN "c:\documents\xldata.xls" "EXCEL 5.0;"
WHERE CustomerID Like "A*"
ORDER BY CustomerID;
```

A named range in a
worksheet

```
SELECT CustomerID, CompanyName
FROM CustomersRange
IN "c:\documents\xldata.xls" "EXCEL 5.0;"
WHERE CustomerID Like "A*"
ORDER BY CustomerID;
```

ALL, DISTINCT, DISTINCTROW, TOP Predicates Example

This example creates a query that joins the Customers and Orders tables on the CustomerID field. The Customers table contains no duplicate CustomerID fields, but the Orders table does because each customer can have many orders. Using DISTINCTROW produces a list of companies that have at least one order but without any details about those orders.

Sub AllDistinctX()

Dim dbs As Database, rst As Recordset

' Modify this line to include the path to Northwind
' on your computer.

Set dbs = OpenDatabase("Northwind.mdb")

' Join the Customers and Orders tables on the
' CustomerID field. Select a list of companies
' that have at least one order.

Set rst = dbs.OpenRecordset("SELECT DISTINCTROW " _
& "CompanyName FROM Customers " _
& "INNER JOIN Orders " _
& "ON Customers.CustomerID = " _
& "Orders.CustomerID " _
& "ORDER BY CompanyName;")

' Populate the Recordset.

rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.

EnumFields rst, 25

dbs.Close

End Sub

GROUP BY Clause Example

This example creates a list of unique job titles and the number of employees with each title.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub GroupByX1()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")
' For each title, count the number of employees
' with that title.
Set rst = dbs.OpenRecordset("SELECT Title, " _
    & "Count([Title]) AS Tally " _
    & "FROM Employees GROUP BY Title;")

' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.
EnumFields rst, 25
dbs.Close
End Sub
```

For each unique job title, this example calculates the number of employees in Washington who have that title.

Sub GroupByX2()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
```

```
Set dbs = OpenDatabase("Northwind.mdb")

' For each title, count the number of employees
' with that title. Only include employees in the
' Washington region.
Set rst = dbs.OpenRecordset("SELECT Title, " _
    & "Count(Title) AS Tally " _
    & "FROM Employees WHERE Region = 'WA' " _
    & "GROUP BY Title;")

' Populate the Recordset.
rst.MoveLast
' Call EnumFields to print the contents of the
' Recordset. Pass the Recordset object and desired
' field width.
EnumFields rst, 25
dbs.Close
End Sub
```


HAVING Clause Example

This example selects the job titles assigned to more than one employee in the Washington region.

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub HavingX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")
' Select the job titles assigned to more than one
' employee in the Washington region.
Set rst = dbs.OpenRecordset("SELECT Title, " _
    & "Count(Title) as Total FROM Employees " _
    & "WHERE Region = 'WA' " _
    & "GROUP BY Title HAVING Count(Title) > 1;")

' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print recordset contents.
EnumFields rst, 25
dbs.Close
End Sub
```

ORDER BY Clause Example

The SQL statement shown in the following example uses the ORDER BY clause to sort records by last name in descending order (Z-A).

This example calls the EnumFields procedure, which you can find in the SELECT statement example.

Sub OrderByX()

```
Dim dbs As Database, rst As Recordset
' Modify this line to include the path to Northwind
' on your computer.
Set dbs = OpenDatabase("Northwind.mdb")
' Select the last name and first name values from
' the Employees table, and sort them in descending
' order.
Set rst = dbs.OpenRecordset("SELECT LastName, " _
    & "FirstName FROM Employees " _
    & "ORDER BY LastName DESC;")

' Populate the Recordset.
rst.MoveLast

' Call EnumFields to print recordset contents.
EnumFields rst, 12
dbs.Close
End Sub
```