DataDirect Connect ODBC Help

This Help file contains reference information for INTERSOLV DataDirect Connect ODBC for Windows 9*x* and Windows NT. The Connect ODBC components are called te ODBC drivers. To get help for a particular driver:

1 Click **Contents** on this Help window's toolbar. Then click the **Contents** tab on the Help Topics window.

2 Double-click a driver's book icon to display the topics for that driver.

3 Double-click the topic about which you need information.

Clicking a driver topic takes you to the Help file for that driver. Click **Back** on the Help window's toolbar to return to this Help file, or click **Contents** and then select a General Help topic.

Note: For important last-minute information about the drivers, see the README file in the ODBC drivers installation directory.

DataDirect Connect ODBC

About INTERSOLV DataDirect Connect ODBC

The INTERSOLV DataDirect Connect ODBC drivers are compliant with the Microsoft® Open Database Connectivity (ODBC) specification. ODBC is a specification for an application program interface (API) that enables applications to access multiple database management systems using Structured Query Language (SQL).

ODBC permits maximum interoperability: a single application can access many different database management systems. This enables an ODBC developer to develop, compile, and ship an application without targeting a specific type of data source. Users can then add the database drivers, which link the application to the database management systems of their choice.

Using Connect ODBC on Windows NT and Windows 9*x*

On both Windows 9*x* and Windows NT systems, the ODBC drivers are 32-bit drivers. All required network software supplied by your database system vendors must be 32-bit compliant. For specific requirements of each relational database driver, click **Contents** on this Help window's toolbar, then click the **Contents** tab on the Help Topics window. Click the book icon for any driver, and then click the "System Requirements" topic.

Starting the ODBC Administrator

The section "Configuring Data Sources" in each driver topic instructs you to start the ODBC Administrator. To start the ODBC Administrator, double-click the ODBC icon in the Control Panel.

Driver Names

On Windows NT and Windows 9*x*, all ODBC drivers start with IV. The file extension is .DLL. For example, the dBASE driver is IVDBF*nn*.DLL, where *nn* is the revision number of the driver. For specific driver filenames, click **Contents** on this Help window's toolbar, then click the **Contents** tab on the Help Topics window. Click the book icon for any driver, then click the "About the driver" topic.

Disk Space and Memory Requirements

Disk space requirements are 15 MB of free space on the disk driver where Windows NT or Windows 9*x* is installed.

Memory requirements vary, depending on the database driver. If you are using a flat-file database driver, you need at least 8 MB of memory on Windows 9*x* or at least 16 MB of memory on Windows NT. If your system is hosting a relational database system, additional memory may be required. Consult your relational database documentation to determine the exact memory requirements.

ODBC Functions Supported

The DataDirect Connect ODBC database drivers support the API functions described in the Microsoft ODBC specification.

All DataDirect ODBC database drivers support:

- ODBC 2.x conformance functions
- ODBC 3.0 conformance functions
- Scalar functions

Some drivers support additional functions or support specific functions differently. These differences are listed in the "ODBC Conformance Level" topic for each driver.

About Application/Driver Compatibility

ODBC version 3.0 adds new features and changes the behavior of some API functions. In general, all existing ODBC applications are compatible with the 3.0 drivers. The 3.0 drivers work with any version of an ODBC 2.x or 3.0 application.

Note: A purely 3.0 ODBC application(that is, one that calls any ODBC 3.0 function or expects 3.0 behavior) is not compatible with a 32-bit, ODBC version 2.x driver. An ODBC 2.x driver exports only ODBC 2.x functions and exhibits ODBC 2.x behavior.

In ODBC version 3.x, certain 2.x functions are deprecated (no longer supported). However, when an ODBC 2.x application uses a deprecated function, the Driver Manager substitutes a comparable ODBC 3.0 function. Click a function name for information about whether the function:

- is supported by a driver conforming to the ODBC 3.x specification
- can be used in an ODBC 3.x application
- is supported by a driver conforming to the ODBC 2.x specification
- when called by an ODBC 3.x driver, will be mapped to another function by the ODBC Driver Manager
- is supported by all drivers in the DataDirect ODBC 3.x Pack.

For more information, refer to the *ODBC Programmer's Reference, Volume 1,* especially Chapter 17, *Programming Considerations*. The section, "Backwards Compatibility and Standards Compliance" discusses which ODBC applications and drivers are incompatible.

ODBC 2.x Conformance Functions

About Application/Driver Compatibility

All database drivers are ODBC Level 1-compliant; they support all ODBC Core and Level 1 functions. A limited set of Level 2 functions is also supported. The drivers support the functions listed in the following table. Any additions to these supported functions or differences in the support of specific functions are listed in the "ODBC Conformance Level" topic for each individual driver.

Core Functions	Level 1 Functions
SQLAllocConnect	SQLColumns
SQLAllocEnv	SQLDriverConnect
SQLAllocStmt	SQLGetConnectOption
SQLBindCol	SQLGetData
SQLBindParameter*	SQLGetFunctions
SQLCancel	SQLGetInfo
SQLColAttributes	SQLGetStmtOption
SQLConnect	SQLGetTypeInfo
SQLDescribeCol	SQLParamData
SQLDisconnect	SQLPutData
SQLDrivers*	SQLSetConnectOption
SQLError	SQLSetStmtOption
SQLExecDirect	SQLSpecialColumns
SQLExecute	SQLStatistics
SQLFetch	SQLTables
SQLFreeConnect	**Level 2 Functions**
SQLFreeEnv	SQLBrowseConnect
SQLFreeStmt	SQLDataSources
SQLGetCursorName	SQLExtendedFetch (forward scrolling only)
SQLNumResultCols	SQLMoreResults
SQLPrepare	SQLNativeSql
SQLRowCount	SQLNumParams
SQLSetCursorName	SQLParamOptions
SQLTransact	SQLSetScrollOptions

ODBC 3.0 Conformance Functions

About Application/Driver Compatibility

SQLAllocConnect
SQLAllocEnv
SQLAllocHandle
SQLAllocStmt
SQLBindCol
SQLBindParameter
SQLBrowseConnect
(except for PROGRESS)
SQLBulkOperations
SQLCancel
SQLCloseCursor
SQLColAttribute
SQLColAttributes
SQLColumns
SQLConnect
SQLCopyDesc
SQLDataSources
SQLDescribeCol
SQLDisconnect
SQLDriverConnect
SQLDrivers
SQLEndTran
SQLError
SQLExecDirect
SQLExecute
SQLExtendedFetch
SQLFetch
SQLFetchScroll
(forward scrolling only)
SQLFreeConnect
SQLFreeEnv
SQLFreeHandle
SQLFreeStmt
SQLGetConnectAttr
SQLGetCursorName
SQLGetData
SQLGetDescField
SQLGetDescRec
SQLGetDiagField
SQLGetDiagRec
SQLGetEnvAttr
SQLGetFunctions
SQLGetInfo
SQLGetStmtAttr
SQLGetTypeInfo
SQLMoreResults
SQLNativeSql
SQLNumParens
SQLNumResultCols
SQLParamData
SQLPrepare
SQLPutData
SQLRowCount
SQLSetConnectAttr
SQLSetCursorName
SQLSetDescField
SQLSetDescRec
SQLSetEnvAttr
SQLSetParam
SQLSetStmtAttr
SQLSpecialColumns
SQLStatistics
SQLTables
SQLTransact

Scalar Functions

ODBC supports scalar functions. Your database system might not support all of these functions. See the documentation for your database system to find out which functions are supported. Refer to the *DataDirect ODBC Drivers Reference* for a list of all of the ODBC scalar functions.

You can use scalar functions in SQL statements using the following syntax:

```
{fn scalar-function}
```

where *scalar-function* is one of the supported functions. For example,

```
SELECT {fn UCASE(NAME)} FROM EMP
```

Error Messages

Error messages may come from

- An ODBC driver
- The database system
- The ODBC driver manager

An error reported on an ODBC driver has the following format:

```
[vendor] [ODBC_component] message
```

ODBC_component is the component in which the error occurred. For example, an error message from INTERSOLV's SQL Server driver would look like this:

```
[INTERSOLV] [ODBC SQL Server driver] Invalid precision specified.
```

If you get this type of error, check the last ODBC call your application made for possible problems or contact your ODBC application vendor.

An error that occurs in the data source includes the data source name, in the following format:

```
[vendor] [ODBC_component] [data_source] message
```

With this type of message, *ODBC_component* is the component that received the error from the data source indicated. For example, you may get the following message from an Oracle data source:

```
[INTERSOLV] [ODBC Oracle driver] [Oracle] ORA-0919:  specified length too long for CHAR column
```

If you get this type of error, you did something incorrectly with the database system. Check your database system documentation for more information or consult your database administrator. In this example, you would check your Oracle documentation.

The Driver Manager is a DLL that establishes connections with drivers, submits requests to drivers, and returns results to applications. For example, an error that occurs in the Microsoft driver manager has the following format:

```
[Microsoft] [ODBC Driver Manager] Driver does not support this function
```

If you get this type of error, consult the programming documentation for the Microsoft ODBC Software Development Kit available from Microsoft.

- Not supported by ODBC 3.x drivers
- Not used in ODBC 3.x applications
- Supported in ODBC 2.x specification
- Mapped by ODBC 3.x Driver Manager with 3.x drivers
- Not supported by all drivers in DataDirect Connect ODBC Pack 3.0

- Supported by ODBC 3.x drivers
- Can be used in ODBC 3.x applications
- Not supported in ODBC 2.x specification
- Not mapped by ODBC 3.x Driver Manager with 3.x drivers
- Supported by all drivers in DataDirect Connect ODBC Pack 3.0

- Supported by ODBC 3.x drivers
- Can be used in ODBC 3.x applications
- Supported in ODBC 2.x specification
- Not mapped by ODBC 3.x Driver Manager with 3.x drivers
- Supported by all drivers in DataDirect Connect ODBC Pack 3.0

- Supported by ODBC 3.x drivers
- Not supported in ODBC 3.x applications
- Supported in ODBC 2.x specification
- Not mapped by ODBC 3.x Driver Manager with 3.x drivers
- Supported by all drivers in DataDirect Connect ODBC Pack 3.0

- Supported by ODBC 3.x drivers
- Can be used in ODBC 3.x applications
- Not supported in ODBC 2.x specification
- Mapped by ODBC 3.x Driver Manager with 3.x drivers
- Supported by all drivers in DataDirect Connect ODBC Pack 3.0

- Not supported by ODBC 3.x drivers
- Not supported in ODBC 3.x applications
- Not supported in ODBC 2.x specification
- Mapped by ODBC 3.x Driver Manager with 3.x drivers
- Not supported by all drivers in DataDirect Connect ODBC Pack 3.0

- Supported by ODBC 3.x drivers
- Can be used in ODBC 3.x applications
- Supported in ODBC 2.x specification
- Not mapped by ODBC 3.x Driver Manager with 3.x drivers
- Not supported by all drivers in DataDirect Connect ODBC Pack 3.0

Using Microsoft Query '97 with Flat-File Data Sources

To use a flat-file database driver with Microsoft Query ’97, you must alter the data source alias outside of Microsoft Query ’97. The data source alias created within Microsoft Query sets a default database, which is the Microsoft Query ’97 working directory. Most likely, the Microsoft Query ’97 working directory does not contain your data files.

For more information, select the following topics:

Creating a Flat-File Data Source for Use with Microsoft Query ’97

Using Microsoft Query ’97 with Single-Connect Data Sources

Creating a Flat-File Data Source for Use with Microsoft Query '97

The following steps describe how to update a flat-file data source. In this example, a dBASE data file is used:

1. Open WordPad or another text editor and edit the data source file that was created in Microsoft Query ’97. The file will have a .dsn extension and will be located in the Microsoft Query ’97 working directory.

   The information will look similar to the section below:

   ```
   [ODBC]
   DRIVER={INTERSOLV 3.01 32-BIT dBASEFile (*.dbf)}
   DB=d:\msoffice\query97
   ```

2. Edit the DB entry to specify the directory that contains your data files. For example, the modified file may look similar to:

   ```
   [ODBC]
   DRIVER={INTERSOLV 3.01 32-BIT dBASEFile (*.dbf)}
   DB=c:\data\sales
   ```

3. Save the file and exit the editor.

Using Microsoft Query '97 with Single-Connect Data Sources

To use data source that are limited to a single connection per session (Btrieve and DB2 data sources), you must create the data source as a FileDSN through the ODBC Administrator.

Follow these steps to create a Btrieve or DB2 data source. A DB2 data source is used in this example:

1. Start the ODBC Administrator.
2. Click the **File DSN** tab.
3. Click **Add**.
4. Select the INTERSOLV 3.01 32-BIT DB2 data source. Click **Next**.

   You are prompted for the path name of the new file data source name.
5. Type the Microsoft Query ’97 directory followed by your new data source name. For example,
   ```
   d:\msoffice\msquery\ DB2Test.dsn
   ```

   Click **Next**.
6. Click **Finish** to validate the new File DSN entry.

   The driver’s logon box is displayed. Specify the logon options as you normally would for the DB2 database. Click OK.
7. Exit the ODBC Administrator.
8. Open WordPad or another text editor and edit the data source file you created in the ODBC Administrator. The file has a .dsn extension and will be located in the Microsoft Query ’97 working directory. In our example, the file name is DB2Test.dsn.

   The information will look similar to the following:
   ```
   [ODBC]
   DRIVER={INTERSOLV 3.01 32-BIT DB2}
   DB=test
   UID=TEST1
   ```
9. Add the following entry to specify the applicable driver option:
   ```
   WA=4
   ```

   The modified file will look similar to the following:
   ```
   [ODBC]
   DRIVER={INTERSOLV 3.01 32-BIT DB2}
   WA=4
   DB=test
   UID=TEST1
   ```
10. Exit the editor.

SQL for Flat-File Drivers

To enable your application to be portable across other databases, you can use SQL statements with the flat-file drivers (Btrieve, dBASE, Excel, Paradox, and Text). The database drivers parse SQL statements and translate them into a form that the database can understand.

This section of Help describes the SQL statements you can use with all of the flat-file drivers. Syntax differences may exist for certain drivers. See the Help topics for each driver for more information.

The following SQL statements let you read, insert, update, and delete records from a database, create new tables, and drop existing tables:

Select Statement

Create Table Statement

Drop Table Statement

Insert Statement

Update Statement

Delete Statement

The following topics describe specific elements of the SQL statements supported:

Aggregate Functions

SQL Expressions

Reserved Keywords

Select Statement

The form of the Select statement supported by the flat-file drivers is:

```
SELECT [DISTINCT] {* | column_expression, ...}
FROM table_names [table_alias] ...
[ WHERE expr1 rel_operator expr2 ]
[ GROUP BY {column_expression, ...} ]
[ HAVING expr1 rel_operator expr2 ]
[ UNION [ALL] (SELECT...) ]
[ ORDER BY {sort_expression [DESC | ASC]}, ... ]
[ FOR UPDATE [OF {column_expression, ...}] ]
```

Click any keyword to see a description of its statement clause.

Select Clause

Follow Select with a list of column expressions you want to retrieve or an asterisk (*) to retrieve all fields:

```
SELECT [DISTINCT] {* | column_expression, [[AS] column_alias] ...}
```

column_expression can be a field name (for example, LAST_NAME). More complex expressions may include mathematical operations or string manipulation (for example, SALARY * 1.05). See SQL Expressions for more information.

column_alias can be used to give the column a more descriptive name. For example, to assign the alias DEPARTMENT to the column DEP:

```
SELECT dep AS department FROM emp
```

Separate multiple column expressions with commas (for example, LAST_NAME, FIRST_NAME, HIRE_DATE).

Field names can be prefixed with the table name or alias. For example, EMP.LAST_NAME or E.LAST_NAME, where E is the alias for the table EMP.

The Distinct operator can precede the first column expression. This operator eliminates duplicate rows from the result of a query. For example:

```
SELECT DISTINCT dep FROM emp
```

Select statements can also include aggregate functions.

Aggregate Functions

Aggregate functions can also be a part of a Select clause. Aggregate functions return a single value from a set of records. An aggregate can be used with a field name (for example, AVG(SALARY)) or in combination with a more complex column expression (for example, AVG(SALARY * 1.07)). The column expression can be preceded by the Distinct operator. The Distinct operator eliminates duplicate values from an aggregate expression. For example:

```
COUNT (DISTINCT last_name)
```

In this example, only distinct last name values are counted.

The following are valid aggregates:

Aggregate	Returns
SUM	The total of the values in a numeric field expression. For example, SUM(SALARY) returns the sum of all salary field values.
AVG	The average of the values in a numeric field expression. For example, AVG(SALARY) returns the average of all salary field values.
COUNT	The number of values in any field expression. For example, COUNT(NAME) returns the number of name values. When using COUNT with a field name, COUNT returns the number of non-null field values. A special example is COUNT(*), which returns the number of records in the set, including records with null values.
MAX	The maximum value in any field expression. For example, MAX(SALARY) returns the maximum salary field value.
MIN	The minimum value in any field expression. For example, MIN(SALARY) returns the minimum salary field value.

From Clause

The From clause indicates the tables that will be used in the Select statement. The format of the From clause is:

```
FROM table_names [table_alias]
```

table_names can be one or more simple table names in the current working directory or complete pathnames.

table_alias is a name used to refer to this table in the rest of the Select statement. Database field names may be prefixed by the table alias. Given the table specification:

```
FROM emp E
```

you may refer to the LAST_NAME field as E.LAST_NAME. Table aliases must be used if the Select statement joins a table to itself. For example:

```
SELECT * FROM emp E, emp F WHERE E.mgr_id = F.emp_id
```

The equal sign (=) includes only matching rows in the results.

If you are joining more than one table, you may use LEFT OUTER JOIN, which includes nonmatching rows in the first table you name. For example:

```
SELECT * FROM T1 LEFT OUTER JOIN T2 on T1.key = T2.key
```

Where Clause

The Where clause specifies the conditions that records must meet to be retrieved. The Where clause contains conditions in the form:

```
WHERE expr1 rel_operator expr2
```

expr1 and *expr2* may be field names, constant values, or expressions.

rel_operator is the relational operator that links the two expressions. See SQL Expressions for more information.

For example, the following Select statement retrieves the names of employees that make at least $20,000:

```
SELECT last_name,first_name FROM emp WHERE salary >= 20000
```

Group By Clause

The Group By clause specifies the names of one or more fields by which the returned values should be grouped. This clause is used to return a set of aggregate values.

It has the following form:

```
GROUP BY column_expressions
```

column_expressions must match the column expression used in the Select clause. A column expression can be one or more field names of the database table, separated by a comma (,) or one or more expressions, separated by a comma (,). See SQL Expressions for more information.

The following example sums the salaries in each department:

```
SELECT dept_id, sum(salary) FROM emp GROUP BY dept_id
```

This statement returns one row for each distinct department ID. Each row contains the department ID and the sum of the salaries of the employees in the department.

Having Clause

The Having clause enables you to specify conditions for groups of records (for example, display only the departments that have salaries totaling more than $200,000). This clause is valid only if you have already defined a Group By clause.

It has the following form:

```
HAVING expr1 rel_operator expr2
```

expr1 and *expr2* can be field names, constant values, or expressions. These expressions do not have to match a column expression in the Select clause.

rel_operator is the relational operator that links the two expressions. See SQL Expressions for more information.

The following example returns only the departments whose sums of salaries are greater than $200,000:

```
SELECT dept_id, sum(salary) FROM emp
GROUP BY dept_id HAVING sum(salary) > 200000
```

Union Operator

The Union operator combines the results of two Select statements into a single result. The single result is all of the returned records from both Select statements. By default, duplicate records are not returned. To return duplicate records, use the All keyword (UNION ALL). The form is

```
SELECT statement
UNION [ALL]
SELECT statement
```

When using the Union operator, the select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order. For example,

```
SELECT last_name, salary, hire_date FROM emp
UNION
SELECT name, pay, birth_date FROM person
```

This example has the same number of column expressions, and each column expression, in order, has the same data type.

The following example is *not* valid because the data types of the column expressions are different (SALARY from EMP has a different data type than LAST_NAME from RAISES). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type:

```
SELECT last_name, salary FROM emp
UNION
SELECT salary, last_name FROM raises
```

Order By Clause

The Order By clause indicates how the records are to be sorted. The form is:

```
ORDER BY {sort_expression [DESC | ASC]}, ...
```

sort_expression can be field names, expressions, or the positional number of the column expression to use.

The default is to perform an ascending (ASC) sort.

For example, to sort by LAST_NAME you could use either of the following Select statements:

```
SELECT emp_id, last_name, first_name FROM emp
ORDER BY last_name
```

or

```
SELECT emp_id, last_name, first_name FROM emp
ORDER BY 2
```

In the second example, LAST_NAME is the second column expression following Select, so Order By 2 sorts by LAST_NAME.

For Update Of Clause

The For Update Of clause locks the records of the database table selected by the Select statement. The form is:

```
FOR UPDATE [OF column_expressions]
```

column_expressions is a list of field names in the database table that you intend to update, separated by a comma (,). Note that *column_expressions* is optional.

The following example returns all records in the employee database that have a SALARY field value of more than $20,000. When each record is fetched, it is locked. If the record is updated or deleted, the lock is held until you commit the change. Otherwise, the lock is released when you fetch the next record:

```
SELECT * FROM emp WHERE salary > 20000
           FOR UPDATE OF last_name, first_name, salary
```

SQL Expressions

Expressions are used in the Where clauses, Having clauses, and Order By clauses of SQL Select statements.

Expressions enable you to use mathematical operations as well as character string and date manipulation operators to form complex database queries.

The most common expression is a simple field name. You can combine a field name with other expression elements.

Other valid expression elements are as follows:

- Constants
- Numeric operators
- Exponential notation
- Character operators
- Date operators
- Relational operators
- Logical operators
- Functions

The order in which these elements are evaluated is described in Operator Precedence.

Constants

Constants are values that do not change. For example, in the expression PRICE * 1.05, the value 1.05 is a constant.

You must enclose character constants in pairs of single (') or double quotation marks ("). To include a single quotation mark in a character constant enclosed by single quotation marks, use two single quotation marks together (for example, 'Don''t'). Similarly, if the constant is enclosed by double quotation marks, use two double quotation marks to include one.

You must enclose date and time constants in braces ({}); for example, {01/30/89} and {12:35:10}. The form for date constants is MM/DD/YY or MM/DD/YYYY. The form for time constants is HH:MM:SS.

The logical constants are .T. and 1 for True and .F. and 0 for False. For portability, use 1 and 0.

Exponential Notation

You may include exponential notation in SQL expressions. For example:

```
SELECT col1, 3.4E+& FROM table1 WHERE calc < 3.4E-6 * col2
```

Numeric Operators

You may include the following operators in numeric expressions:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
^	Exponentiation

The following table shows examples of numeric expressions. For these examples, assume SALARY is 20000.

Example	Resulting value
`salary + 10000`	`30000`
`salary * 1.1`	`22000`
`2 ** 3`	`8`

You can precede numeric expressions with a unary plus (+) or minus (-). For example, `-(salary * 1.1)` is -22000.

Character Operators

Character expressions may include the following operators:

Operator	Meaning
+	Concatenation keeping trailing blanks.
-	Concatenation moving trailing blanks to the end.

The following chart shows examples of character expressions. In the examples, LAST_NAME is `'JONES '` and FIRST_NAME is `'ROBERT '`.

Example	Resulting value
`first_name + last_name`	`'ROBERT JONES '`
`first_name - last_name`	`'ROBERTJONES '`

Note: Some flat-file drivers return character data with trailing blanks as shown in the table. However, you cannot rely on the driver to return blanks. Therefore, if you want an expression that works with drivers that do and do not return trailing blanks, use the TRIM function before concatenating strings to make the expression portable. For example:

```
TRIM(first_name) + '' + TRIM(last_name)
```

Date Operators

You may include the following operators in date expressions:

Operator	Meaning
+	Add a number of days to a date to produce a new date.
-	The number of days between two dates, or subtract a number of days from a date to produce a new date.

The following chart shows examples of date expressions. In these examples, hire_date is {01/30/90}.

Example	Resulting value
`hire_date + 5`	`{02/04/90}`
`hire_date - {01/01/90}`	`29`
`hire_date - 10`	`{01/20/90}`

Relational Operators

The relational operators separating the two expressions may be any one of the following:

Operator	Meaning
=	Equal
<>	Not Equal
>	Greater Than
>=	Greater Than or Equal
<	Less Than
<=	Less Than or Equal
Like	Matching a pattern
Not Like	Not matching a pattern
Is Null	Equal to Null
Is Not Null	Not Equal to Null
Between	Range of values between a lower and upper bound.
In	A member of a set of specified values or a member of a subquery.
Exists	True if a subquery returned at least one record.
Any	Compares a value to each value returned by a subquery. Any must be prefaced by =, <>, >, >=, <, or <=. =Any is equivalent to In.
All	Compares a value to each value returned by a subquery. All must be prefaced by =, <>, >, >=, <, or <=.

The following list shows some examples of relational operators:

```
salary <= 40000

dept = 'D101'

hire_date > {01/30/89}

salary + commission >= 50000

last_name LIKE 'Jo%'

salary IS NULL

salary BETWEEN 10000 AND 20000

WHERE salary = ANY (SELECT salary FROM emp WHERE dept = 'D101')

WHERE salary > ALL (SELECT salary FROM emp WHERE dept = 'D101')
```

Logical Operators

Two or more conditions may be combined to form more complex criteria. When two or more conditions are present, they must be related by AND or OR. For example:

```
salary = 40000 AND exempt = 1
```

The logical NOT operator is used to reverse the meaning. For example:

```
NOT (salary = 40000 AND exempt = 1)
```

Operator Precedence

As expressions become more complex, the order in which the expressions are evaluated becomes important. The following table shows the order in which the operators are evaluated. The operators in the first line are evaluated first, then those in the second line, and so on. Operators in the same line are evaluated left to right in the expression.

Precedence	Operator
1	Unary -, Unary +
2	**
3	*, /
4	+, -
5	=, <>, <, <=, >, >=, Like, Not Like, Is Null, Is Not Null, Between, In, Exists, Any, All
6	Not
7	AND
8	OR

The following example shows the importance of precedence:

```
WHERE salary > 40000 OR
hire_date > {01/30/89} AND
dept = 'D101'
```

Because AND is evaluated first, this query retrieves employees in department D101 hired after January 30, 1989, as well as every employee making more than $40,000, no matter what department or hire date.

To force the clause to be evaluated in a different order, use parentheses to enclose the conditions to be evaluated first. For example:

```
WHERE (salary > 40000 OR hire_date > {01/30/89})
AND dept = 'D101'
```

retrieves employees in department D101 that either make more than $40,000 or were hired after January 30, 1989.

Functions

The flat-file drivers support a number of functions that you may use in expressions. In the following topics, the functions are grouped according to the type of result they return.

Functions that Return Character Strings

Functions that Return Numbers

Functions that Return Dates

Functions that Return Character Strings

The following functions return character string values:

Function	Description
CHR	Converts an ASCII code into a one-character string. `CHR(67)` returns `C`.
RTRIM	Removes trailing blanks from a string. `RTRIM('ABC  ')` returns `ABC`.
TRIM	Removes trailing blanks from a string. `TRIM('ABC  ')` returns `ABC`.
LTRIM	Removes leading blanks from a string. `LTRIM('  ABC')` returns `ABC`.
UPPER	Changes each letter of a string to uppercase. `UPPER('Allen')` returns `ALLEN`.
LOWER	Changes each letter of a string to lowercase. `LOWER('Allen')` returns `allen`.
LEFT	Returns leftmost characters of a string. `LEFT('Mattson',3)` returns `Mat`.
RIGHT	Returns rightmost characters of a string. `RIGHT('Mattson',4)` returns `tson`.
SUBSTR	Returns a substring of a string. Parameters are the string, the first character to extract, and the number of characters to extract (optional). `SUBSTR('Conrad',2,3)` returns `onr`. `SUBSTR('Conrad',2)` returns `onrad`.
SPACE	Generates a string of blanks. `SPACE(5)` returns `'  '`.
DTOC	Converts a date to a character string. An optional second parameter determines the format of the result: 0 (the default) returns MM/DD/YY 1 returns DD/MM/YY 2 returns YY/MM/DD 10 returns MM/DD/YYYY 11 returns DD/MM/YYYY 12 returns YYYY/MM/DD An optional third parameter specifies the date separator character. If not specified, a slash (/) is used. `DTOC({01/30/89})` returns `01/30/89`

`DTOC({01/30/89}, 0)` returns `01/30/89`

`DTOC({01/30/89}, 1)` returns `30/01/89`

`DTOC({01/30/89}, 2,'-')` returns `89-01-30`

DTOS — Converts a date to a character string using the format YYYYMMDD.

`DTOS({01/23/90})` returns `19900123`.

IIF — Returns one of two values. Parameters are a logical expression, the true value, and the false value. If the logical expression evaluates to True, the function returns the true value. Otherwise, it returns the false value.

`IIF(salary>20000,'BIG','SMALL')` returns `BIG` if SALARY is greater than 20000. If not, it returns `SMALL`.

STR — Converts a number to a character string. Parameters are the number, the total number of output characters (including the decimal point), and optionally the number of digits to the right of the decimal point.

`STR(12.34567,4)` returns `12`

`STR(12.34567,4,1)` returns `12.3`

`STR(12.34567,6,3)` returns `12.346`

STRVAL — Converts a value of any type to a character string.

`STRVAL('Woltman')` returns `Woltman`

`STRVAL({12/25/53})` returns `12/25/53`

`STRVAL (5 * 3)` returns `15`

`STRVAL (4 = 5)` returns 'False'

TIME — Returns the time of day as a string.

At 9:49 PM, `TIME()` returns `21:49:00`

TTOC — **Note:** This function is applicable only for those flat-file drivers that support SQL_TIMESTAMP, the Btrieve, Excel 4, Excel 5, FoxPro 3.0, and Paradox 5 drivers.

Converts a timestamp to a character string. An optional second parameter determines the format of the result:

0 or none, the default, returns MM/DD/YY HH:MM:SS AM

1 returns YYYYMMDDHHMMSS, which is a suitable format for indexing.

`TTOC({1992-04-02 03:27:41})` returns `04/02/92 03:27:41 AM`.

`TTOC({1992-04-02 03:27:41, 1})` returns `19920402032741`

USERNAME — Returns the name in the UID attribute as a character string. If UID=Allen, `USERNAME()` returns `Allen`.

For Btrieve, the logon ID specified at connect time is returned. For Paradox and Paradox 5 drivers, the user name specified during configuration is returned. For all other flat file drivers, an empty string is returned.

Functions that Return Numbers

The following functions return number values:

Function	Description
MOD	Divides two numbers and returns the remainder of the division. `MOD(10,3)` returns `1`
LEN	Returns the length of a string. `LEN('ABC')` returns `3`
MONTH	Returns the month part of a date. `MONTH({01/30/89})` returns `1`
DAY	Returns the day part of a date. `DAY({01/30/89})` returns `30`
YEAR	Returns the year part of a date. `YEAR({01/30/89})` returns `1989`
MAX	Returns the larger of two numbers. `MAX(66,89)` returns `89`
MIN	Returns the smaller of two numbers. `MIN(66,89)` returns `66`
POW	Raises a number to a power. `POW(7,2)` returns `49`
INT	Returns the integer part of a number. `INT(6.4321)` returns `6`

ROUND	Rounds a number.
	`ROUND(123.456, 0)` returns `123`
	`ROUND(123.456, 2)` returns `123.46`
	`ROUND(123.456, -2)` returns `100`

NUMVAL	Converts a character string to a number. If the character string is not a valid number, a zero is returned. `NUMVAL('123')` returns the number `123`
VAL	Converts a character string to a number. If the character string is not a valid number, a zero is returned. `VAL('123')` returns the number `123`

Functions that Return Dates

The following functions return date values:

Function	Description

DATE	Returns today's date. If today is 12/25/79, `DATE()` returns `{12/25/79}`
TODAY	Returns today's date. If today is 12/25/79, `TODAY()` returns `{12/25/79}`
DATEVAL	Converts a character string to a date. `DATEVAL('01/30/89')` returns `{01/30/89}`

CTOD	Converts a character string to a date. An optional second parameter specifies the format of the character string: 0 (the default) returns MM/DD/YY, 1 returns DD/MM/YY, and 2 returns YY/MM/DD.
	`CTOD('01/30/89')` returns `{01/30/89}`
	`CTOD('01/30/89',1)` returns `{30/01/89}`

The following examples use some of the number and date functions.

Retrieve all employees that have been with the company at least 90 days:

```
SELECT first_name, last_name FROM emp
WHERE DATE() - hire_date >= 90
```

Retrieve all employees hired in January of this year or last year:

```
SELECT first_name, last_name FROM emp
WHERE MONTH(hire_date) = 1 AND (YEAR(hire_date) = YEAR(DATE())
OR YEAR(hire_date) = YEAR(DATE()) - 1)
```

Create Table Statement

The Create Table statement is used to create database files. The form of the Create Table statement is

```
CREATE TABLE filename (col_definition[,col_definition, ...])
```

filename can be a simple filename or a full pathname. A simple filename is preferred for portability to other SQL data sources. If it is a simple filename, the file is created in the directory you specified as the database directory in the connection string. If you did not specify a database directory in the connection string, the file is created in the directory you specified as the database directory in the system information. If you did not specify a database directory in either place, the file is created in the current working directory at the time you connected to the driver.

col_definition is the column name, followed by the data type, followed by an optional column constraint definition. Values for column names are database specific. The data type specifies a column's data type.

The only column constraint definition currently supported by some flat-file drivers is "not null." Not all flat-file tables support "not null" columns. In the cases where not null is not supported, this restriction is ignored and the driver returns a warning if "not null" is specified for a column. The "not null" column constraint definition is allowed in the driver so that you can write a database-independent application (and not be concerned about the driver raising an error on a Create Table statement with a "not null" restriction).

A sample Create Table statement to create an employee database table is:

```
CREATE TABLE emp (last_name CHAR(20) NOT NULL,
           first_name CHAR(12) NOT NULL,
           salary NUMERIC (10,2) NOT NULL,
           hire_date DATE NOT NULL)
```

Drop Table Statement

The Drop Table statement is used to delete database files. The form of the Drop Table statement is

```
DROP TABLE filename
```

filename may be a simple filename (EMP) or a full pathname. A simple filename is preferred for portability to other SQL data sources. If it is a simple filename, the file is dropped from the directory you specified as the database directory in the connection string. If you did not specify a database directory in the connection string, the file is deleted from the directory you specified as the database directory in the system information. If you did not specify a database directory in either of these places, the file is dropped from the current working directory at the time you connected to the driver.

A sample Drop Table statement to delete the employee database table is:

```
DROP TABLE emp
```

Insert Statement

The SQL Insert statement is used to add new records to a database table. With it, you can specify either of the following:

- A list of values to be inserted as a new record
- A Select statement that copies data from another table to be inserted as a set of new records

The form of the Insert statement is

```
INSERT INTO table_name [(col_name, ...)]
VALUES (expr, ...)
```

table_name may be a simple filename or a full pathname. A simple filename is preferred for portability to other SQL data sources.

col_name is an optional list of column names giving the name and order of the columns whose values are specified in the Values clause. If you omit *col_name*, the value expressions (*expr*) must provide values for all columns defined in the file and must be in the same order that the columns are defined for the file.

expr is the list of expressions giving the values for the columns of the new record. Usually, the expressions are constant values for the columns. Character string values must be enclosed in single or double quotation marks, date values must be enclosed in braces {}, and logical values that are characters must be enclosed in periods (for example, .T. or .F.).

An example of an Insert statement that uses a list of expressions is:

```
INSERT INTO emp (last_name, first_name, emp_id, salary, hire_date)
VALUES ('Smith', 'John', 'E22345', 27500, {4/6/91})
```

Each Insert statement adds one record to the database table. In this case a record has been added to the employee database table, EMP. Values are specified for five columns. The remaining columns in the table are assigned a blank value, meaning Null.

select_statement is a query that returns values for each *col_name* value specified in the column name list. Using a Select statement instead of a list of value expressions lets you select a set of rows from one table and insert it into another table using a single Insert statement.

An example of an Insert statement that uses a Select statement is:

```
INSERT INTO emp1 (first_name, last_name, emp_id, dept, salary)
SELECT first_name, last_name, emp_id, dept, salary from emp
    WHERE dept = ‘D050’
```

In this type of Insert statement, the number of columns to be inserted must match the number of columns in the Select statement. The list of columns to be inserted must coresond to the columns in the Select statement just as it would to a list of value expressions in the other type of Insert statement. That is, the first column inserted corresponds to the first column selected; the second inserted to the second, etc.

The size and data type of these corresponding columns must be compatible. Each column in the Select list should have a data type that the ODBC driver accepts on a regular Insert/Update of the corresponding column in the Insert list. Values are truncated when the size of the value in the Select list column is greater than the size of the corresponding Insert list column.

The *select_statement* is evaluated before any values are inserted. This query cannot be made on the table into which values are inserted.

Update Statement

The SQL Update statement is used to change records in a database file. The form of the Update statement supported for flat-file drivers is:

```
UPDATE filename SET col_name = expr, ...
[ WHERE { conditions | CURRENT OF cursor_name } ]
```

filename may be a simple filename or a full pathname. A simple filename is preferred for portability to other SQL data sources.

col_name is the name of a column whose value is to be changed. Several columns can be changed in one statement.

expr is the new value for the column. The expression can be a constant value or a subquery. Character string values must be enclosed with single or double quotation marks, date values must be enclosed by braces {}, and logical values that are letters must be enclosed by periods (for example, .T. or .F.). Subqueries must be enclosed in parentheses.

The Where clause is any valid clause. It determines which records are to be updated.

The Where Current Of *cursor_name* clause can be used only by developers coding directly to the ODBC API. It causes the row at which *cursor_name* is positioned to be updated. This is called a "positioned update." You must first execute a Select...For Update statement with a named cursor and fetch the row to be updated.

An example of an Update statement on the employee table is:

```
UPDATE emp SET salary=32000, exempt=1
WHERE emp_id = 'E10001'
```

The Update statement changes every record that meets the conditions in the Where clause. In this case the salary and exempt status is changed for all employees having the employee ID E10001. Since employee IDs are unique in the employee table, only one record is updated.

An example using a subquery is:

```
UPDATE emp SET salary = (SELECT avg(salary) from emp)
WHERE emp_id = 'E10001'
```

In this case, the salary is changed to the average salary in the company for the employee having employee ID E10001.

Delete Statement

The SQL Delete statement is used to delete records from a database table. The form of the Delete statement supported for flat-file drivers is:

```
DELETE FROM filename
[ WHERE { conditions | CURRENT OF cursor_name } ]
```

filename may be a simple filename or a full pathname. A simple filename is preferred for portability to other SQL data sources.

The Where clause is any valid clause. It determines which records are to be deleted. If you include only the keyword Where, all records in the table are deleted but the file is left intact.

The Where Current Of *cursor_name* clause can be used only by developers coding directly to the ODBC API. It causes the row at which *cursor_name* is positioned to be deleted. This is called a "positioned delete." You must first execute a Select...For Update statement with a named cursor and fetch the row to be deleted.

An example of a Delete statement on the employee table is:

```
DELETE FROM emp WHERE emp_id = 'E10001'
```

Each Delete statement removes every record that meets the conditions in the Where clause. In this case every record having the employee ID E10001 is deleted. Since employee IDs are unique in the employee table, at most one record is deleted.

Reserved Keywords

The following words are reserved for use in SQL statements. If they are used for file or column names in a database that you use, you must enclose them in quotation marks in any SQL statement where they appear as file or column names.

ALL	FULL	NOT
AND	GROUP	NULL
BETWEEN	HAVING	ON
COMPUTE	INNER	OPTIONS
CROSS	INTO	OR
DISTINCT	LEFT	ORDER
FOR	LIKE	RIGHT
FROM	NATURAL	WHERE