API Reference

§

Contents

µ

§

§

Introduction

The ODBC (Open Database Connectivity) interface is a C programming language interface for database connectivity. The *ODBC API Reference* contains syntax and semantic information for all ODBC functions.

The *ODBC API Reference* is organized into the following sections:

- Chapter 1, “ODBC Function Summary,” lists all ODBC functions. The information about is organized by the type of work each function performs.
- Chapter 2, "ODBC API Reference," lists ODBC functions in alphabetic order.

The appendixes contain additional reference information, including a list of error codes, a summary of ODBC functions, a list of valid data types, and a description of the SQL grammar that applications can submit with ODBC function calls.

If you are developing an application that uses the ODBC interface, refer to the *ODBC Application Programmer's Guide* for additional information about how to use ODBC functions.

If you are developing a driver that supports ODBC function calls, refer to the *ODBC Driver Developer's Guide* for information about how to implement ODBC functions.

Audience

The ODBC software development kit (SDK) is available for use with the C language in a Windows environment. Use of the ODBC interface spans four areas of knowledge: SQL statements, ODBC function calls, C programming, and Windows programming. This manual assumes the following expertise:

- Working knowledge of the C programming language.
- General database knowledge and a familiarity with SQL.

Document Conventions

This manual uses the following typographic conventions.

Format	Used for
WIN.INI	Names of applications, programs, and other files.
RETCODE SQLFetch (hDBC)	Sample command lines and program code.
argument	Information that the application must provide or word emphasis.
SQLTransact	Syntax that must be typed exactly as shown, including function names.
[]	Optional items or, if in bold text, brackets that must be included in the text string.
|	Separates two mutually exclusive choices in a syntax line.
{}	Delimits mutually exclusive choices in a syntax line.
...	Arguments that can be repeated several times.

Where to Find Additional Information

XE "SQL:references for additional information"§For more information about SQL, the following standards are available:

- Database Language - SQL with Integrity Enhancement, ANSI, 1989 ANSI X3.135-1989.

§

- X/Open and SQL Access Group SQL CAE draft specification (1991).
- Database Language SQL: ANSI X3H2 and ISO/IEC JTC1,SC21,WG3 (draft international standard).

In addition to standards and vendor-specific SQL guides, there are many books that describe SQL, including:

- Date, C. J.: *A Guide to the SQL Standard* (Addison-Wesley, 1989).
- Emerson, Sandra L., Darnovsky, Marcy, and Bowman, Judith S.: *The Practical SQL Handbook* (Addison-Wesley, 1989).
- Groff, James R. and Weinberg, Paul N.: *Using SQL* (Osborne McGraw-Hill, 1990).
- Gruber, Martin: *Understanding SQL* (Sybex, 1990).
- Hursch, Jack L. and Carolyn J.: *SQL, The Structured Query Language* (TAB Books, 1988).
- Pascal, Fabian: *SQL and Relational Basics* (M & T Books, 1990).
- Trimble, J. Harvey, Jr. and Chappell, David: *A Visual Introduction to SQL* (Wiley, 1989).
- Van der Lans, Rick F.: *Introduction to SQL* (Addison-Wesley, 1988).
- Vang, Soren: *SQL and Relational Databases* (Microtrend Books, 1990).
- Viescas, John: *Quick Reference Guide to SQL* (Microsoft Corp., 1989).

1 ODBC Function Summary

XE "Conformance"§XE "ODBC :summary of functions"§XE "Connections:summary of ODBC functions"§XE "Requests:summary of ODBC functions"§The following table lists ODBC functions, grouped by type of task, and includes the conformance designation and a brief description of the purpose of each function. For more information about the syntax and semantics for each function, refer to Chapter 2, “ODBC API Reference.”

To obtain conformance information, an application can call the **SQLGetInfo** function. To obtain information about support for a specific function, an application can call **SQLGetFunctions**.

Task	Function Name	Conformance	Purpose
Connecting to a Data Source	**SQLAllocEnv**	Core	Obtains an environment handle. One environment handle is used for one or more connections.
	SQLAllocConnect	Core	Obtains a connection handle.
	SQLConnect	Core	Connects to a specific driver by data source name, user ID, and password.
	SQLDriverConnect	ODBC Level 1	Connects to a specific driver by connection string or request that the Driver Manager and driver display connection dialogs for the user.
Obtaining Information about a Driver and Data Source	**SQLDataSources**	ODBC Level 2	Returns a list of available data sources.

	SQLGetInfo	ODBC Level 1	Returns information about a specific driver and data source.
	SQLGetFunctions	ODBC Level 1	Returns supported driver functions.
	SQLGetTypeInfo	ODBC Level 1	Returns information about supported data types .
	SQLTables	ODBC Level 1	Returns a list of tables.
	SQLTablePrivileges	ODBC Level 2	Returns a list of tables and matching privileges.
	SQLColumns	ODBC Level 1	Returns a list of column names.
	SQLColumnPrivileges	ODBC Level 2	Returns a list of columns and matching privileges.

Task	Function Name	Conformance	Purpose
Obtaining Information about a Driver and Data Source (continued)	**SQLSpecialColumns**	ODBC Level 1	Returns information about two sets of columns: the optimal set of columns for unique row identification or the set of columns that are automatically updated when any value in the row is updated.

	SQLStatistics	ODBC Level 1	Returns statistics about a table and its indexes.
	SQLPrimaryKeys	ODBC Level 2	Returns a list of column names that are used in a primary key.
	SQLForeignKeys	ODBC Level 2	Returns a list of column names that are used in a foreign key.
Setting and Retrieving Driver Options	**SQLSetConnectOption**	ODBC Level 1	Sets a connection option.
	SQLGetConnectOption	ODBC Level 1	Returns the value of a connection option.
	SQLSetStmtOption	ODBC Level 1	Sets a statement option.
	SQLGetStmtOption	ODBC Level 1	Returns the value of a statement option.
Preparing SQL Requests	**SQLAllocStmt**	Core	Allocates a statement handle.
	SQLPrepare	Core	Prepares an SQL statement for later execution.
	SQLSetParam	Core	Assigns storage for a parameter in an SQL statement.
	SQLParamOptions	ODBC Level	Specifies the use of

		2	multiple values for parameters.
	SQLGetCursorName	Core	Returns the cursor name associated with an hstmt.
	SQLSetCursorName	Core	Specifies a cursor name.
	SQLSetScrollOptions	ODBC Level 2	Sets options that control cursor behavior.
Submitting Requests	**SQLExecute**	Core	Executes a prepared statement.
	SQLExecDirect	Core	Executes a statement.

Task	Function Name	Conformance	Purpose
Submitting Requests (continued)	**SQLNativeSql**	ODBC Level 2	Returns the actual text of an SQL statement as translated by the driver.
	SQLDescribeParam	ODBC Level 2	Returns the description for a specific parameter in a prepared statement.
	SQLNumParams	ODBC Level 2	Returns the number of parameters in a statement.
Sending Large Data Values	**SQLParamData**	ODBC Level 1	Initialize processing of a long data value.
	SQLPutData	ODBC Level 1	Store a long data value in a column.
Retrieving Results and Information about Results	**SQLRowCount**	Core	Returns the number of rows affected by an insert, update, or delete request.
	SQLNumResultCols	Core	Returns the number of columns in the result set.
	SQLDescribeCol	Core	Describes a column in the result set.
	SQLColAttributes	ODBC Level 1	Describes additional attributes of a column in the result set.
	SQLBindCol	Core	Assigns storage for a result

		column and specifies the data type.
SQLFetch	Core	Returns a result row.
SQLExtendedFetch	ODBC Level 2	Returns an array of results.
SQLGetData	ODBC Level 1	Returns one column of one row of a result set. (Useful for large data types.)
SQLSetPos	ODBC Level 2	Positions a cursor within a fetched block of data.
SQLMoreResults	ODBC Level 2	Determines whether there are more result sets available and, if so, initializes processing for the next result set.
SQLError	Core	Returns additional error or status information.

Task	Function Name	Conformance	Purpose
Terminating a Statement	**SQLFreeStmt**	Core	Ends statement processing and closes the associated cursor or drops the statement handle.
	SQLCancel	Core	Cancels an SQL statement.
	SQLTransact	Core	Commits or rolls back a transaction.
Terminating a Connection	**SQLDisconnect**	Core	Closes the connection.
	SQLFreeConnect	Core	Releases the connection handle.
	SQLFreeEnv	Core	Releases the environment handle.

XE "Results:summary of ODBC functions"§XE "Terminating statements, summary of ODBC functions"§

2 ODBC API Reference

XE "ODBC:functions"§The following pages describe each ODBC function call in alphabetic order. Each function is defined as a C function. Descriptions include the following:

- Purpose
- Syntax
- Arguments
- Return values
- Comments about usage and implementation
- References to related functions

Each description indicates whether a function is a core or extended function. Error handling is described in the **SQLError** function description. The text associated with SQLSTATE values is included to provide a description of the condition, but is not intended to prescribe specific text.

Arguments

All function arguments use a naming convention of the following form:

[[<prefix>]...]<tag>[<qualifier>][<suffix>]

Optional elements are enclosed in square brackets ([]). The following prefixes are used:

Prefix	Description
c	Count of
h	Handle of
i	Index of
p	Pointer to
rg	Range (array) of

The following tags are used:

Tag	Description
b	Byte
col	Column (of a result set)
dbc	Database connection
env	Environment
f	Flag (enumerated type)
par	Parameter (of an SQL statement)
row	Row (of a result set)
stmt	Statement
sz	Character string (array of characters, terminated by zero)
v	void (an unspecified type)

Prefixes and tags combine to correspond roughly to the C types listed below. Flags (f) and byte counts (cb) do not distinguish between SWORD, UWORD, SDWORD, and UDWORD.

Combined	Prefix	Tag	C Type(s)	Description
cb	c	b	SWORD,SDWORD,UDWORD	Count of bytes
crow	c	row	SDWORD,UDWORD,UWORD	Count of rows
f		f	BOOL,SWORD,UWORD, UDWORD	Flag
hdbc	h	dbc	HDBC	Connection handle
henv	h	env	HENV	Environment handle
hstmt	h	stmt	HSTMT	Statement handle
hwnd	h	wnd	HWND	Pointer to window
ib	i	b	SWORD	Index of byte
icol	i	col	UWORD	Column index
ipar	i	par	UWORD	Parameter index
irow	i	row	SDWORD,UWORD	Index to row
pcb	pc	b	SWORD FAR *,SDWORD FAR *, UDWORD FAR *	Pointer to byte count
pccol	pc	col	SWORD FAR *	Pointer to column count

pcpar	pc	par	SWORD FAR *	Pointer to parameter count
pcrow	pc	row	SDWORD FAR *, UDWORD FAR *	Pointer to row count
phdbc	ph	dbc	HDBC FAR *	Pointer to connection handle
phenv	ph	env	HENV FAR *	Pointer to environment handle
phstmt	ph	stmt	HSTMT FAR *	Pointer to statement handle
pib	pi	b	SWORD FAR *	Pointer to byte index
pirow	pi	row	UDWORD FAR *	Pointer to row index
rgb	rg	b	PTR	Range (array) of bytes
sz	--	sz	UCHAR FAR *	String, zero terminated

Qualifiers are used to distinguish specific variables of the same type. Qualifiers consist of the concatenation of one or more capitalized English words or abbreviations.

ODBC defines one value for suffix, "Max," which denotes that the variable represents the largest value of its type for a given situation.

Examples

The following two examples illustrate the naming convention. The following argument:

cbErrorMsgMax

contains the largest possible byte count for an error message; in this case, the argument corresponds to the size in bytes of the argument szErrorMsg, a character string buffer. In the following example:

pcbErrorMsg

is a pointer to the count of bytes actually returned in the argument szErrorMsg, not including the zero termination character.

SQLAllocConnect

Core function **SQLAllocConnect** allocates memory for a connection handle within the environment identified by henv.

Syntax RETCODE SQLAllocConnect(henv,phdbc)

XE "SQLAllocConnect"§The **SQLAllocConnect** function accepts the following arguments.

Type	Argument	Use	Description
HENV	henv	Input	Environment handle.
HDBC FAR *	phdbc	Output	Pointer to storage for the connection handle.

Returns SQL_SUCCESS, SQL_INVALID_HANDLE, or SQL_ERROR.

If the driver returns SQL_ERROR for **SQLAllocConnect**, the driver sets the hdbc referenced by phdbc to SQL_NULL_HDBC. To obtain additional information, the application can call **SQLError** with hdbc and hstmt set to SQL_NULL_HDBC and SQL_NULL_HSTMT, respectively.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
80	000	General ODBC error
80	001	Memory allocation error
80	009	Invalid argument value

Comments The application passes the address of a variable to **SQLAllocConnect**. The driver allocates memory and stores the value of the associated handle into the variable. The application passes the hdbc value in all subsequent calls that require an hdbc .

The Driver Manager processes the **SQLAllocConnect** function and calls the driver's **SQLAllocConnect** function when the application calls **SQLConnect** or **SQLDriverConnect**. (For more information, refer to the description of the **SQLConnect** function.)

If the hdbc referenced by phdbc references a connection handle that is already allocated, the driver overwrites the hdbc without regard to its previous contents.

If the application calls **SQLAllocConnect** with a null phdbc, the driver returns SQLSTATE value 80 009 (invalid argument value).

The following table lists related ODBC functions.

For information about	See
Activating a connection	**SQLConnect**
Freeing a connection handle (the opposite of **SQLAllocConnect**)	**SQLFreeConnect**

SQLAllocEnv

Core function **SQLAllocEnv** allocates memory for an environment handle and initializes the ODBC call-level interface for use by an application. An application must call **SQLAllocEnv** prior to calling any other ODBC function.

Syntax RETCODE SQLAllocEnv(phenv)

XE "SQLAllocEnv"§The **SQLAllocEnv** function accepts the following argument.

Type	Argument	Use	Description
HENV FAR *	phenv	Output	Pointer to storage for the environment handle.

Returns SQL_SUCCESS or SQL_ERROR.

If the driver returns SQL_ERROR for **SQLAllocEnv**, the driver sets the henv referenced by phenv to SQL_NULL_HENV. In this case, the application can assume that the error was a memory allocation error.

If the application subsequently calls **SQLError** with a null henv, **SQLError** returns SQL_INVALID_HANDLE.

A driver cannot return SQLSTATE values directly after the call to **SQLAllocEnv**. There are two levels of **SQLAllocEnv** functions–one within the Driver Manager and one within each driver. The Driver Manager does not call the driver-level function until the application calls **SQLConnect**. If an error occurs in the driver-level **SQLAllocEnv** function, the Driver Manager returns SQLSTATE value DM 004 (the driver's **SQLAllocEnv** failed), followed by one of the following errors, after the call to **SQLConnect**:

- SQLSTATE value 80 000 (general ODBC error)
- A driver-specific SQLSTATE value, ranging from 80 000 to 80 9ZZ. For example, SQLSTATE value 80 001 (memory allocation error) indicates that the driver's henv is null after the Driver Manager's call to **SQLAllocEnv**

For additional information about the flow of function calls between the Driver Manager and a driver, refer to the **SQLConnect** function description.

Comments An environment handle references global information such as valid connection handles and active connection handles. To request an environment handle, an application passes the address of a variable to **SQLAllocEnv**. The driver allocates memory and stores the value of the associated handle into the variable. The application passes the henv value in all subsequent calls that require an henv .

The Driver Manager processes the **SQLAllocEnv** function and calls the driver's **SQLAllocEnv** function when the application calls **SQLConnect** or **SQLDriverConnect**. (For more information, refer to the description of the **SQLConnect** function.

There should never be more than one henv allocated at one time. The application should not call **SQLAllocEnv** when there is a current valid henv.

If the application calls **SQLAllocEnv** with a pointer to a valid henv, the driver overwrites the handle without regard to the previous contents of henv.

The following table lists related ODBC functions.

For information about	See
Allocating a connection handle	**SQLAllocConnect**
Establishing a connection with a database	**SQLConnect**
Freeing an environment handle	**SQLFreeEnv**

SQLAllocStmt

Core function **SQLAllocStmt** allocates memory for a statement handle and associates the statement handle with the connection specified by hdbc.

XE "SQLAllocStmt"§An application must call **SQLAllocStmt** prior to submitting SQL statements.

Syntax

```
RETCODE SQLAllocStmt(hdbc,phstmt)
```

The **SQLAllocStmt** function accepts the following arguments.

Type	Argument	Use	Description
HDBC	hdbc	Input	Connection handle.
HSTMT FAR *	phstmt	Output	Pointer to storage for the statement handle.

Returns

SQL_SUCCESS, SQL_INVALID_HANDLE, or SQL_ERROR.

If the driver returns SQL_ERROR for **SQLAllocStmt**, the driver sets the hstmt referenced by phstmt to SQL_NULL_HSTMT. The application can then obtain additional information by calling **SQLError** with the henv, hdbc, and SQL_NULL_HSTMT.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	003	Connection not open
80	000	General ODBC error
80	001	Memory allocation error
80	009	Invalid argument value

DM	001	Driver does not support this function

Comments

Prior to calling **SQLAllocStmt**, an application must successfully establish a connection.

The application passes the address of a variable to **SQLAllocStmt**. The driver allocates memory and stores the value of the associated handle into the variable. The application passes the hstmt itself in all subsequent calls that require an hstmt .

If the hstmt referenced by phstmt points to a statement handle that has been previously allocated, the driver overwrites the handle without regard to its previous contents.

The driver allocates memory for an hstmt, initializes the hstmt, and returns. The information the driver stores in the hstmt is very specific to the driver, and can include network information, error messages, and specifics for the data source. The driver should include room for descriptors, a cursor name, number of result columns or rows affected, SQLSTATE values, and status information for SQL statement processing.

If the application calls **SQLAllocStmt** with a null phstmt, a subsequent call to **SQLError** returns SQLSTATE value 80 009 (invalid argument value).

The following table lists related ODBC functions.

For information about	See
Submitting SQL statements	**SQLPrepare**, **SQLExecute**, **SQLExecDirect**
Freeing the statement handle (the opposite of **SQLAllocStmt**)	**SQLFreeStmt**

SQLBindCol

Core function **SQLBindCol** assigns storage and conversions for a column in a result set, including:

- A storage buffer that will receive the contents of a column of data
- The length of the storage buffer
- A storage location that will receive the actual length of the column of data returned by the fetch operation
- Data conversion

SQLBindCol can also remove binding of columns that were previously bound, so that the application can call **SQLGetData** to retrieve data in the columns or to simply omit columns of the result set from the set of bound columns.

XE "SQLBindCol"§Each time **SQLFetch** is called, the driver places the next row of the result set into the variables specified in previous calls to **SQLBindCol** for each column.

To retrieve a column in portions (such as data in LONG VARCHAR types), call **SQLGetData**.

Syntax

```
RETCODE SQLBindCol(hstmt,icol,fCType,rgbValue,cbValueMax,pcbValue)
```

The **SQLBindCol** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
UWORD	icol	Input	Column number of result data, ordered sequentially left to right, starting at 1.
SWORD	fCType	Input	Data type for result data. fCType must contain one of the following values: SQL_C_CHAR converts a noncharacter data type to a character string as defined by the format specified for numeric literals in Appendix C, "SQL Grammar." SQL_NO_CONVERT stores data in the buffer in its corresponding C data type.

			SQL_C_BINARY stores data in the internal form used by the database. The driver does not perform any conversion on the data. If the result data is defined as numeric data, the application can specify a numeric data type (see "Comments," below).
PTR	rgbValue	Input	Pointer to storage for the data, or a null pointer (to remove binding from a previously-bound column).
SDWORD	cbValueMax	Input	Contains the maximum number of bytes to store in rgbValue. This value must be greater than zero.
SDWORD FAR *	pcbValue	Input	After each **SQLFetch** operation, this argument returns either the actual number of bytes transferred to rgbValue or SQL_NULL_DATA. (Refer to “Comments” for additional information.)

Returns

SQL_SUCCESS, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists SQLSTATE values.

SQLSTATE Class	Subclass	Description
80	000	General ODBC error
80	001	Memory allocation error
80	002	Invalid column number
80	003	fCType argument out of range
80	009	Invalid argument value

80	C00	Driver not capable
DM	001	Driver does not support this function

Comments

The ODBC interface provides two ways to retrieve data:

- **SQLBindCol** links a buffer to a query result for a column that contains typical character and numeric data
- **SQLGetData** (an extended function) retrieves a column that contains a large data value

The driver coordinates interaction between **SQLBindCol**, **SQLFetch**, query processing, and **SQLGetData** (if the driver supports **SQLGetData**).

If an application calls **SQLBindCol**, the application must submit the call prior to a fetch operation. To rebind, the application can call **SQLBindCol** after fetching zero or more rows of data.

It is the application's responsibility to pass a valid pointer and to allocate enough storage for the form of the data specified by fCType. For variable-length data, the application must allocate the maximum length for the column, or the data may be truncated. **SQLBindCol** can convert data to a different data type. The result and success of the conversion is determined by the rules for assignment specified in Appendix D, "Data Type Definitions."

The application allocates storage buffers and uses **SQLBindCol** to pass pointers to the buffers. For each bound column, the driver fills the buffer at fetch time and stores the actual number of bytes in the result column in pcbValue. If truncation occurs, the driver returns SQL_SUCCESS_WITH_INFO at fetch time.

An application can request that binding be removed from a column either before or after the first row of the result set has been fetched. To remove binding, the application passes a null pointer for rgbvalue. If the column was previously bound, the driver removes binding associated with the column. and does not attempt to store the bytes of corresponding result columns. If the column was not previously bound, the driver ignores the request.

If the data value for a column is null, the driver sets pcbValue to SQL_NULL_DATA.

The following table lists related ODBC functions.

For information about	See
Determining the number of columns in a result set	**SQLNumResultCols**
Retrieving information about a result column	**SQLDescribeCol**
Retrieving row data	**SQLFetch**
Retrieving column data in a long data type	**SQLGetData** (extension)

SQLCancel

Core function **SQLCancel** requests cancellation of processing for an SQL statement and returns control to the application. If an application submits statements asychronously, it can call **SQLCancel**. Otherwise, the application cannot access **SQLCancel** from the Windows environment while a request is in process. If an application calls **SQLCancel** for a function called synchronously, **SQLCancel** has the same effect as **SQLFreeStmt** with the SQL_CLOSE option.

Syntax

```
RETCODE SQLCancel(hstmt)
```

XE "SQLCancel"§The **SQLCancel** function accepts the following argument.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle obtained from a call to **SQLAllocStmt**.

Returns SQL_SUCCESS, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
70	100	Server was unable to process the cancel request
80	000	General ODBC error
80	001	Memory allocation error
DM	001	Driver does not support this function

Comments If the cancel request is successful, the driver returns SQL_SUCCESS. This message does *not* indicate that the SQL operation was actually canceled; it indicates that the cancel request was processed.

Once the canceled ODBC function and associated SQL statement actually finish processing, the driver returns SQL_ERROR and the SQLSTATE value 80 008 (operation canceled) if the application calls the original function that returned SQL_STILL_EXECUTING.

SQLCancel preserves the statement handle and any error information buffered for **SQLError**.

The following table lists related ODBC functions.

For information about	See
Executing a statement	**SQLExecute**, **SQLExecDirect**
Closing a cursor and discarding results	**SQLFreeStmt**
Freeing a statement handle and discarding results	**SQLFreeStmt**

SQLColAttributes

Extension Level 1

SQLColAttributes returns descriptive information—beyond that provided by **SQLDescribeCol**—for a column in the result set.

Syntax

```
RETCODE  SQLColAttributes(hstmt,icol,pfAttributes,szTypeName,
    cbTypeNameMax,pcbTypeName)
```

XE "SQLColAttributes"§The **SQLColAttributes** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
UWORD	icol	Input	Column number.
UWORD FAR *	pfAttributes	Output	Attribute mask for the column. Refer to "Comments," below, for a description of options.
UCHAR FAR *	szTypeName	Output	Type name (as returned by **SQLGetTypeInfo**).
SWORD	cbTypeNameMax	Input	Size of buffer for szTypeName.
SWORD FAR *	pcbTypeName	Output	Number of characters in type name.

Returns

SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use

08	S01	Communication link failure
24	000	Invalid cursor state
80	000	General ODBC error
80	001	Memory allocation error
80	002	Invalid column number
80	008	Operation canceled
80	009	Invalid argument value
80	010	Function sequence error
DM	001	Driver does not support this function

Comments

The following table lists valid options for **SQLColAttributes**.

Attribute Bit Mask Value	Output
SQL_COL_UNSIGNED	SQL_UNSIGNED is valid only for numeric data types: 0 if signed, 1 if unsigned
SQL_COL_MONEY	1 if money data type; otherwise, 0.
SQL_COL_UPDATABLE	SQL_READONLY, SQL_WRITE, or SQL_READWRITE_UNKNOWN
SQL_COL_AUTO_INCREMENT	Autoincrement is possible only for numeric columns: 1 if autoincrement 0 if no autoincrement An application can insert values into this column, but cannot update values in the column.
SQL_COL_CASE_SENSITIVE	1 if the column is treated as case sensitive for collations; otherwise, 0.
SQL_COL_SEARCHABLE	1 if the column can be used in a **WHERE** clause with all comparison operators; 0 if only the LIKE predicate can be used. Columns of type SQL_LONG_VARCHAR and SQL_LONG_VARBINARY usually return 0.

The application can use the following syntax to test attributes:

```
(<attributes> & <mask>) == <output>
```

For example:

```
if ((attributes & SQL_UPDATABLE)==SQL_WRITE)
if mask == output
```

```
if (fAttributes & SQL_COL_UNSIGNED)
```

The following table lists related ODBC functions.

For information about	See
Retrieving the number of columns in a result set	**SQLNumResultCols**
Describing a column of a result set	**SQLDescribeCol**
Fetching a row of a result set	**SQLFetch**

SQLColumnPrivileges

Extension Level 2

SQLColumnPrivileges returns a list of columns and associated privileges for one or more tables. The driver returns the information as a result set on the specified hstmt.

Syntax

```
RETCODE  SQLColumnPrivileges(hstmt,szTableQualifier,cbTableQualifier,szTableOwner,
    cbTableOwner,szTableName,cbTableName,szColumnName,cbColumnName)
```

XE "SQLColumnPrivileges"§The **SQLColumnPrivileges** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle to retrieve results.
UCHAR FAR *	szTableQualifier	Input	Table qualifier.
SWORD	cbTableQualifier	Input	Length of szTableQualifier.
UCHAR FAR *	szTableOwner	Input	String search pattern for owner names.
SWORD	cbTableOwner	Input	Length of szTableOwner.
UCHAR FAR *	szTableName	Input	String search pattern for table name.
SWORD	cbTableName	Input	Length of szTableName.
UCHAR FAR *	szColumnName	Input	String search pattern for column names.
SWORD	cbColumnName	Input	Length of szColumnName.

Returns

SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use
08	S01	Communication link failure
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	C00	Driver not capable
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

The szTableOwner and szTableName arguments constrain the search for table names. Each of these arguments supports a character string search pattern. The search pattern can contain the metacharacters underscore (_) and percent (%), as follows:

- The underscore character represents any single character.
- The percent character represents any sequence of zero or more characters.
- All other characters represent themselves.

For example, an szTableName equal to %A% returns all tables with names that contain the character 'A'. An szTableName equal to B__ ('B' followed by two underscores) returns all tables with names that are three characters long and start with the character 'B'.

If the pattern string contains no underscores or percent characters, then the driver compares characters from left to right. If the pattern is of different length than the string it is being compared to, then the driver equates the two strings by treating the shorter of the two strings as though it has trailing blank characters.

The driver returns the results as a standard result set. The following table lists result columns. For each privilege, a one indicates that the privilege is granted for the column. A zero indicates that the privilege is not granted. If REFERENCES_PRIVILEGE is null, the privilege is not supported.

Column Name	Contents
TABLE_QUALIFIER	Varchar(32)
TABLE_OWNER	Varchar(32)
TABLE_NAME	Varchar(32) not null
COLUMN_NAME	Varchar(32) not null
GRANTOR	Varchar(32) not null
GRANTEE	Varchar(32) not null
SELECT_PRIVILEGE	Smallint not null

SELECT_GRANTABLE	Smallint not null
INSERT_PRIVILEGE	Smallint not null
INSERT_GRANTABLE	Smallint not null
UPDATE_PRIVILEGE	Smallint not null
UPDATE_GRANTABLE	Smallint not null
DELETE_PRIVILEGE	Smallint not null
DELETE_GRANTABLE	Smallint not null
REFERENCES_PRIVILEGE	Smallint
REFERENCES_GRANTABLE	Smallint

The following table lists a related ODBC function.

For information about	See
Listing privileges for a specified table or tables	**SQLTablePrivileges**

SQLColumns

Extension Level 1

SQLColumns returns the list of column names in specified tables. The driver returns this information as a result set on the specified hstmt.

Syntax

```
RETCODE  SQLColumns(hstmt,szTableQualifier,cbTableQualifier,szTableOwner,
    cbTableOwner,szTableName,cbTableName,szColumnName,cbColumnName)
```

XE "SQLColumns"§The **SQLColumns** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle for retrieving results.
UCHAR FAR *	szTableQualifier	Input	Qualifier name.
SWORD	cbTableQualifier	Input	Length of szTableQualifier.
UCHAR FAR *	szTableOwner	Input	String search pattern for owner name.
SWORD	cbTableOwner	Input	Length of szTableOwner.
UCHAR FAR *	szTableName	Input	String search pattern for table name.
SWORD	cbTableName	Input	Length of szTableName.
UCHAR FAR *	szColumnName	Input	String search pattern for column name.
SWORD	cbColumnName	Input	Length of szColumnName.

Returns

SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_INVALID_HANDLE, or

SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use
08	S01	Communication link failure
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	C00	Driver not capable
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

This function is used typically before statement execution, to retrieve information about columns from the data dictionary.

SQLColumns does not describe computed columns in a fetch, nor does it return the number of columns referenced by a **SELECT** statement.

The szTableName, szTableOwner, and szColumnName arguments constrain the search for table names. Each of these arguments supports a character string search pattern. The search pattern can contain the metacharacters underscore (_) and percent (%), as follows:

- The underscore character represents any single character.
- The percent character represents any sequence of zero or more characters.
- All other characters represent themselves.

For example, an szTableName equal to %A% returns all tables with names that contain the character 'A'. An szTableName equal to B__ ('B' followed by two underscores) returns all tables with names that are three characters long and start with the character 'B'.

If the pattern string contains no underscores or percent characters, then the driver compares characters from left to right. If the pattern is of different length than the string it is being compared to, then the driver equates the two strings by treating the shorter of the two strings as though it has trailing blank characters.

SQLColumns returns results as a standard result set. The following table lists result columns. Additional columns beyond column twelve (REMARKS) can be defined by the driver.

The length column in the result set contains the transfer size of the data; that is, the length in bytes of data transferred on an SQLGetData or SQLFetch operation if SQL_NO_CONVERT is specified. For numeric data, this size may be different than the size of the data stored on the server.

Result Column Name	Data Type
TABLE_QUALIFIER	Varchar(32)
TABLE_OWNER	Varchar(32)
TABLE_NAME	Varchar(32) not null
COLUMN_NAME	Varchar(32) not null
DATA_TYPE	Smallint not null; ODBC type number of the column
TYPE_NAME	Varchar(32) not null (same as returned

	by **SQLGetTypeInfo**)
DATA_PRECISION	Int
LENGTH	Int
NUMERIC_SCALE	Smallint
NUMERIC_RADIX	Smallint
NULLABLE	Smallint not null
REMARKS	Varchar(254)

The following table lists related ODBC functions.

For information about	See
Listing tables for the current connection	**SQLTables**
Listing indexes and statistics for a specified table	**SQLStatistics**

SQLConnect

Core function **SQLConnect** loads a driver and establishes a connection to a data source. The connection handle references storage of all information about the connection, including status, transaction state, and error information.

Syntax RETCODE SQLConnect(hdbc,szDSN,cbDSN,szUID,cbUID,szAuthStr,cbAuthStr)

XE "SQLConnect"§The **SQLConnect** function accepts the following arguments.

Type	Argument	Use	Description
HDBC	hdbc	Input	Connection handle.
UCHAR FAR *	szDSN	Input	Buffer containing the data source name.
SWORD	cbDSN	Input	Length of the data source name.
UCHAR FAR *	szUID	Input	Buffer containing the user identifier.
SWORD	cbUID	Input	Length of the user identifier.
UCHAR FAR *	szAuthStr	Input	Buffer containing the authentication string (typically the password).
SWORD	cbAuthStr	Input	Length of the authentication string.

Returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	001	Unable to connect to database server

08	002	Connection in use
08	S01	Communication link failure
28	000	Invalid user authorization specification
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	T00	Timeout expired
DM	001	Driver does not support this function
DM	002	Data source not found and no default driver specified
DM	003	Driver specified by data source could not be loaded
DM	004	Driver's **SQLAllocEnv** failed
DM	005	Driver's **SQLAllocConnect** failed
DM	006	Driver's **SQLSetConnectOption** failed

Comments

The Driver Manager does not load a driver or call the **SQLAllocEnv** or **SQLAllocConnect** functions until the application calls **SQLConnect** or **SQLDriverConnect** and specifies a data source name. Until that point, the Driver Manager works with internal handles and manages connection information. Once the application calls **SQLConnect** or **SQLDriverConnect**, the Driver Manager calls the driver's **SQLAllocEnv** and **SQLAllocConnect** routines and then calls either **SQLConnect** or **SQLDriverConnect**, respectively. The driver then allocates the application-accessible handles and initializes itself.

The following diagram shows the underlying command flow between the Driver Manager and the driver.

An ODBC application can establish more than one connection. The driver keeps track of which connection is active and switches connections when necessary.

The contents of szDSN affect how the Driver Manager and a driver work together to establish a connection to a data source. The Driver Manager uses the following guidelines:

- If szDSN contains a data source name, the Driver Manager locates the corresponding data source specification in the ODBC.INI file and loads the associated driver DLL. The Driver Manager passes each **SQLConnect** argument to the driver.

  If the corresponding data source specification does not exist, the Driver Manager locates the default data source specification and loads the associated driver DLL. The Driver Manager passes the application-specified data source name to the default driver.

  If szDSN contains a data source name but no corresponding data source specification exists and no default data source specification exists, the Driver Manager returns an error.

- If szDSN is null, the Driver Manager attempts to locate the default data source specification in the ODBC.INI file. If the default specification exists, the Driver Manager loads the associated driver DLL and uses "default" as the data source name.

  If no default specification exists, the Driver Manager returns an error.

A driver, upon being loaded by the Driver Manager, can locate the data source specification in the ODBC.INI file and use driver-specific information from the specification.

The following table lists related ODBC functions.

For information about	See
Initializing a connection handle	**SQLAllocConnect**
Allocating a statement handle	**SQLAllocStmt**
Closing a connection (the opposite of **SQLConnect**)	**SQLDisconnect**
Establishing a connection using a connection string	**SQLDriverConnect** (extension)

SQLDataSources

Extension Level 2

XE "SQLDataSources"§**SQLDataSources** enumerates data source names.

Syntax

```
RETCODE  SQLDataSources(henv,fDirection,szDSN,cbDSNMax,
          pcbDSN,szDescription,cbDescriptionMax,pcbDescription)
```

The **SQLDataSources** function accepts the following arguments:

Type	Argument	Use	Description
HENV	henv	Input	Environment handle.
UWORD	fDirection	Input	Determines whether the driver fetches the next data source name in the list or whether the search starts from the beginning of the list. fDirection is one of the following two values: SQL_FETCH_FIRST = fetch first source SQL_FETCH_NEXT = fetch next source
UCHAR FAR *	szDSN	Output	Data source name.
SWORD	cbDSNMax	Input	Length of szDSN buffer; this should be at least 32 bytes in length.
SWORD FAR *	pcbDSN	Output	Number of characters in szDSN. Set to actual length of data when driver stores data in szDSN.
UCHAR FAR *	szDescription	Output	Description of the data source name.
SWORD	cbDescriptionMax	Input	Length of szDescription buffer; this should be at least 255 bytes in length.

SWORD FAR *	pcbDescription	Output	Number of characters placed in szDescription. Set to actual length of data when driver stores data in szDescription.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA_FOUND, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
01	004	Data truncated
80	000	General ODBC error
80	001	Memory allocation error
80	009	Invalid argument value
DM	006	Driver's **SQLSetConnectOption** failed.

Comments

SQLDataSources is implemented solely by the Driver Manager. The function reads and enumerates data source names from the [ODBC Data Sources] section of the ODBC.INI file.

An application can call **SQLDataSources** multiple times to retrieve all data source names. When there are no more data source names, the Driver Manager returns SQL_NO_DATA_FOUND.

If the data source name or description is too large for its respective buffer, **SQLDataSources** returns SQL_SUCCESS_WITH_INFO.

The driver determines how data source names are mapped to actual data sources. The list of data source names and their corresponding drivers are maintained in the ODBC.INI file.

The following table lists related ODBC functions.

For information about	See
Establishing a connection with a data source	**SQLConnect**
Using a connection string or a dialog to connect with a data source	**SQLDriverConnect**

SQLDescribeCol

Core function **SQLDescribeCol** returns the result descriptor—column name, type, and length—for one column in the result set.

Syntax

```
RETCODE SQLDescribeCol(hstmt,icol,szColName,cbColNameMax,
    pcbColName,pfSqlType,pcbColDef,pibScale,pfNullable)
```

XE "SQLDescribeCol"§The **SQLDescribeCol** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
UWORD	icol	Input	Column number (in left-to-right order within the result set, starting with 1).
UCHAR FAR *	szColName	Output	Descriptor name for the column of data.
SWORD	cbColNameMax	Input	Length of szColName buffer.
SWORD FAR *	pcbColName	Output	Number of bytes available in szColName.
SWORD FAR *	pfSqlType	Output	A valid SQL data type number that describes the data type of the column. Refer to Appendix D, "Data Type Definitions," for a list of valid data types.
UDWORD FAR *	pcbColDef	Output	Maximum length or precision of the column as defined in the data source. (See Appendix D for more information.)

SWORD FAR *	pibScale	Output	Scale (total number of digits to the right of the decimal point) for the data type of the column as defined in the data source, or zero if not valid for the data type.
SWORD FAR *	pfNullable	Output	Indicates whether the column allows NULL values. One of the following values: SQL_NO_NULLS: the column does not allow NULL values SQL_NULLABLE: the column allows NULL values SQL_NULLABLE_UNKNOWN: the driver cannot determine if the column allows NULL values.

Scale information, stored in the pibScale argument, is defined only for SQL_DECIMAL, SQL_NUMERIC, and SQL_TIMESTAMP data types. For example, a column of type DECIMAL(5,3) has pcbColDef equal to 5 and pibScale equal to 3.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
01	004	Data truncated
80	000	General ODBC error
80	001	Memory allocation error
80	002	Invalid column number
80	008	Operation canceled
80	009	Invalid argument value
80	010	ODBC function sequence error
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

An application calls **SQLDescribeCol** typically after a call to **SQLPrepare** and before or after the associated call to **SQLExecute**. An application can also call **SQLDescribeCol** after a call to **SQLExecDirect**.

SQLDescribeCol retrieves the column name, type, and length generated by a **SELECT** statement. If the column is an expression, szColName is either an empty string or a driver-defined name.

§

Note ODBC supports SQL_NULLABLE_UNKNOWN as an extension, even though X/Open and the SQL Access Group do not specify the option for **SQLDescribeCol**.

§

The following table lists related ODBC functions.

For information about	See
Preparing a statement for execution	**SQLPrepare**
Returning the number of columns in a result set	**SQLNumResultCols**
Defining storage for a column in a result set	**SQLBindCol**
Fetching a row of a result set	**SQLFetch**

SQLDescribeParam

Extension Level 2

XE "SQLDescribeParam"§**SQLDescribeParam** returns the description of a parameter marker associated with a prepared SQL statement.

Syntax

```
RETCODE  SQLDescribeParam(hstmt,ipar,szColName,cbColNameMax,pcbColName,
    pfSqlType,pcbColDef,pibScale,pfNullable)
```

The **SQLDescribeParam** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
UWORD	ipar	Input	Parameter marker number (starting with 1, in the order the parameters were listed in **SQLPrepare**).
UCHAR FAR *	szColName	Output	Column name, if available.
SWORD	cbColNameMax	Input	Length of szColName buffer.
SWORD FAR *	pcbColName	Output	Number of bytes placed in szColName. Upon return, the driver stores the actual length of the column name.
SWORD FAR *	pfSqlType	Output	Type of the parameter.
UDWORD FAR *	pcbColDef	Output	Maximum length of the parameter.
SWORD FAR *	pibScale	Output	Scale (total number of digits to the right of the decimal point) for the data type of the column as defined in the data source, or zero if not valid

			for the data type.
SWORD FAR *	pfNullable	Output	Indicates whether the column allows NULL values. One of the following: SQL_NO_NULLS: the column does not allow NULL values (this is the default value) SQL_NULLABLE: the column allows NULL values SQL_NULLABLE_UNKNOWN: the driver cannot determine if a column allows NULL values.

Scale information, stored in the pibScale argument, is defined only for SQL_DECIMAL, SQL_NUMERIC, and SQL_TIMESTAMP data types. For example, a column of type DECIMAL(5,3) has pcbColDef equal to 5 and pibScale equal to 3.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
01	004	Data truncated
80	000	General ODBC error
80	001	Memory allocation error
80	005	Parameter number out of range
80	008	Operation canceled
80	009	Invalid argument value
80	010	Function sequence error
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

Parameter markers are numbered from left to right in the order they appear in the SQL statement.

If the column name does not fit in szColName, **SQLDescribeParam** stores the allowed number of bytes and returns SQL_SUCCESS_WITH_INFO. A subsequent call to **SQLError** returns SQLSTATE value 01 004 (data truncated).

The following table lists related ODBC functions.

For information about	See
Executing a prepared statement	**SQLExecute**

Specifying storage for a parameter in an SQL statement	**SQLSetParam**
Preparing a statement for execution	**SQLPrepare**

SQLDisconnect

Core function **SQLDisconnect** closes the connection associated with a specific connection handle.

Syntax RETCODE SQLDisconnect(hdbc)

XE "SQLDisconnect"§The **SQLDisconnect** function accepts the following argument.

Type	Argument	Use	Description
HDBC	hdbc	Input	Connection handle.

Returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
01	002	Disconnection error: Transaction rolled back.
08	003	Connection not open
80	000	General ODBC error
80	001	Memory allocation error
DM	001	Driver does not support this function

Comments If the application calls **SQLDisconnect** while there are open cursors or uncommitted transactions associated with the connection handle, the driver requests a rollback operation and returns SQL_SUCCESS_WITH_INFO and SQLSTATE value 01 002 (Disconnection error: Transaction rolled back).

The following table lists related ODBC functions.

For information about	See
Allocating a connection handle	**SQLAllocConnect**
Connecting to a database (the opposite of **SQLDisconnect**)	**SQLConnect**
Releasing a connection handle	**SQLFreeConnect**

SQLDriverConnect

Extension Level 2

SQLDriverConnect is an alternative to **SQLConnect**; it supports data sources that require connection information other than the three arguments used in **SQLConnect**.

XE "SQLDriverConnect"§**SQLDriverConnect** provides the following connection options:

- Establish a connection using a connection string that contains the data source name, one or more user IDs, one or more passwords, and other information required by the data source.
- Establish a connection using a partial connection string or no additional information; in this case, the Driver Manager and the driver can each prompt the user for connection information.

Once a connection is established, **SQLDriverConnect** returns the completed connection string. The application can use this string for subsequent connection requests to the same driver.

Syntax

```
RETCODE  SQLDriverConnect(hdbc,hwnd,szConnStrIn,cbConnStrIn,szConnStrOut,
    cbConnStrOutMax,pcbConnStrOut,fDriverCompletion)
```

The **SQLDriverConnect** function accepts the following arguments:

Type	Argument	Use	Description
HDBC	hdbc	Input	Connection handle.
HWND	hwnd	Input	Allocated window handle. The application can pass the handle of the parent window, if applicable, or NULL if either the window handle is not applicable or if **SQLDriverConnect** will not present any dialogs.
UCHAR FAR *	szConnStrIn	Input	A full connection string (see format below), a partial connection string, or a Null string.
SWORD	cbConnStrIn	Input	Length of szConnStrIn.

UCHAR FAR *	szConnStrOut	Output	A buffer. Upon successful connection to the target data source, this buffer contains the completed connection string. Applications should allocate at least 255 bytes for this buffer.
SWORD	cbConnStrOutMax	Input	Contains the length of szConnStrOut The application should set this length before calling **SQLDriverConnect**. Upon return, the driver sets pcbConnStrOut to the number of bytes in the complete connection string.
SWORD FAR *	pcbConnStrOut	Output	Number of bytes in the complete connection string.

Type	Argument	Use	Description
UWORD	fDriverCompletion	Input	SQL_DRIVER_PROMPT, SQL_DRIVER_COMPLETE, or SQL_DRIVER_NOPROMPT. (See "Comments," below, for additional information.

A connection string has the following format:

DSN=*data-source-name***;UID[***n***]=***userID***;PWD[***n***]=***password***;DBQ=***database-qualifier***;**
[UID*n***=***userID***;PWD***n***=***password***...]***[attribute***=***value]*

The following table describes the arguments in a connection string.

Argument	Description
data-source-name	Name of a data source as returned by **SQLDataSources** or the data sources dialog of **SQLDriverConnect**.

UID[n]	The n indicates the number of the userID, starting with 1. If there is only one userID and password, the 1 can be omitted.
userID	A userID.
PWD[n]	The n indicates the number of the password, starting with 1.
password	The password corresponding to the nth user ID, or NULL if there is no password for the userID.
database-qualifier	The driver-dependent qualifier, such as the database name, if the data source supports a database qualifier.
attribute	Driver-defined connection attribute (optional).
value	Driver-defined attribute value (optional).

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA_FOUND, SQL_STILL_EXECUTING, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	001	Unable to connect to database server
08	002	Connection in use
08	S01	Communication link failure
28	000	Invalid user authorization specification
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	T00	Timeout expired
DM	001	Driver does not support this function
DM	002	Data source not found and no default driver specified
DM	003	Driver specified by data source could not be loaded
DM	004	Driver's **SQLAllocEnv** failed
DM	005	Driver's **SQLAllocConnect** failed
DM	006	Driver's **SQLSetConnectOption** failed

Comments The contents of *fDriverCompletion* and *szConnStrIn* affect how the Driver Manager and a driver work together to establish a connection to a data source. The Driver Manager uses the following guidelines:

- If the value of the argument *fDriverCompletion* is SQL_DRIVER_PROMPT, the Driver Manager initiates a data sources dialog, obtains a data source name through this dialog, and inserted the name into *szConnStrIn* .
- If the value of the argument *fDriverCompletion* is SQL_DRIVER_COMPLETE, and if *szConnStrIn* is null or if there is no data source name following DSN= in the connection string, the Driver Manager displays a data sources dialog. The user can supply a data source name; the Driver Manager inserts this name into its copy of *szConnStrIn*.
- If the value of the argument *fDriverCompletion* is SQL_DRIVER_NOPROMPT, the Driver Manager returns SQL_ERROR if *szConnStrIn* is null or if there is no data source name following DSN= in the connection string.
- Once the Driver Manager has a data source name, from either the initial connection string or the data sources dialog, it attempts to locate a corresponding data source specification in the ODBC.INI file:
  - If the Driver Manager finds a corresponding data source specification, it loads the associated driver DLL and passes the arguments supplied to **SQLDriverConnect** to the driver. If the data source name is "default," the Driver Manager loads the driver DLL listed in the default data source specification and passes "default" to the driver as the data source name.
  - If the Driver Manager does not find a corresponding data source specification, it loads the driver DLL listed in the default data source specification of the ODBC.INI file and passes it the arguments supplied to **SQLDriverConnect** to the driver.
  - If the Driver Manager finds neither a corresponding data source specification nor the default data source specification, it returns SQL_ERROR.

A driver, upon being loaded by the Driver Manager, does the following:

- Locates the data source specification in the ODBC.INI file that corresponds to the data source name in *szConnStrIn*.
- Locates the default data source specification in the ODBC.INI file if no corresponding data source specification exists.
- Uses the information in the ODBC.INI file, if available, to add driver-specific information to the connection string.
- Depending on the value of *fDriverCompletion*, the driver prompts the user for missing information, including user ID, password, and other attributes:

- If *fDriverCompletion* equals SQL_DRIVER_PROMPT, the driver always initiates a dialog with the user. The driver uses information from *szConnStrIn* for defaults.
- If *fDriverCompletion* equals SQL_DRIVER_COMPLETE, the driver initiates a dialog only if there is not enough information in *szConnStrIn* to connect to the data source.
- If *fDriverCompletion* equals SQL_DRIVER_NOPROMPT, the driver attempts to connect to the data source. The driver does not display a dialog. If *szConnStrIn* is not sufficient, **SQLDriverConnect** returns SQL_ERROR.

§

Note The SQL_LOGIN_TIMEOUT connection option, set using **SQLSetConnectOption**, defines the number of seconds to wait for a login request to complete before returning to the application. If the user is prompted to complete the connection string, the waiting period for the login requests begins *after* the user has dismissed the final dialog.

§

The driver opens the connection in SQL_READ_WRITE access mode by default. To set the access mode to SQL_READ_ONLY, the application can call **SQLSetStmtOption** or **SQLSetConnectOption** with the SQL_ACCESS_MODE option prior to calling **SQLDriverConnect**.

Upon successful connection to the data source, the driver stores the complete connection string in the buffer referenced by *szConnStrOut* and sets *pcbConnStrOut* to the length of the connection string.

If the user cancels a dialog presented by the Driver Manager or the driver, **SQLDriverConnect** returns SQL_NO_DATA_FOUND.

The following table lists related ODBC functions.

For information about	See
Allocating a connection handle	**SQLAllocConnect**
Closing a connection to a data source	**SQLDisconnect**
Freeing the connection handle	**SQLFreeConnect**

SQLError

Core function **SQLError** returns error or status information.

Syntax SQLError(henv,hdbc,hstmt,szSqlState,pfNativeError,szErrorMsg,cbErrorMsgMax, pcbErrorMsg)

XE "SQLError"§The **SQLError** function accepts the following arguments.

Type	Argument	Use	Description
HENV	henv	Input	Environment handle or SQL_NULL_HENV.
HDBC	hdbc	Input	Connection handle or SQL_NULL_HDBC.
HSTMT	hstmt	Input	Statement handle or SQL_NULL_HSTMT.
UCHAR FAR *	szSqlState	Output	Error class and subclass as null terminated string. (SQLSTATE). See Appendix A, “ODBC Error Codes,” for a list of classes and subclasses.
UDWORD FAR *	pfNativeError	Output	Native error code (specific to the data source).
UCHAR FAR *	szErrorMsg	Output	Text of the message. The maximum length of szErrorMsg is SQL_MAX_MESSAGE_LENGTH-1.
SWORD	cbErrorMsgMax	Input	Length of the szErrorMsg buffer.
SWORD FAR *	pcbErrorMsg	Output	Length of the available message text.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA_FOUND, SQL_INVALID_HANDLE, or SQL_ERROR.

SQLError does not post error values for itself. If **SQLError** is unable to retrieve any error information, it typically returns SQL_NO_DATA_FOUND. If **SQLError** cannot access error values for any reason that would normally return SQL_ERROR, **SQLError** returns SQL_ERROR but does not post any error values. If the buffer for the error message is too short, **SQLError** returns SQL_SUCCESS_WITH_INFO but, again, does not return a SQLSTATE value for **SQLError**.

To determine that a truncation occurred in the error message, an application can compare cbErrorMsgMax to the actual length of the message text written to pcbErrorMsg.

Comments

The driver stores error information in the henv, hdbc, and hstmt structures and returns this information to the application when the application calls **SQLError**. Each ODBC function can post zero or more errors.

An application calls **SQLError** typically when a previous call to an ODBC function returns SQL_ERROR or SQL_SUCCESS_WITH_INFO. The application can, however, call **SQLError** after any ODBC function call. If an application does not call **SQLError** after a function call, the driver clears prior information when it processes the next function call for the associated handle.

SQLError retrieves an error from the data structure associated with the rightmost nonnull handle argument. An application requests error information as follows:

- To retrieve errors associated with an environment, the application passes the corresponding henv and includes SQL_NULL_HDBC and SQL_NULL_HSTMT in hdbc and hstmt, respectively. The driver returns the error status of the ODBC function most recently called with the same henv.
- To retrieve errors associated with a connection, the application passes the corresponding hdbc plus an hstmt equal to SQL_NULL_HSTMT. In such a case, the driver ignores the henv argument. The driver returns the error status of the ODBC function most recently called with the hdbc.
- To retrieve errors associated with a statement, an application passes the corresponding hstmt. If the call to **SQLError** contains a valid hstmt, the driver ignores the hdbc and henv arguments. The driver returns the error status of the ODBC function most recently called with the hstmt.
- To retrieve multiple errors for a function call, an application calls **SQLError** multiple times. For each error, the driver returns SQL_SUCCESS and removes that error from the list of available errors.

When there is no additional information for the rightmost nonnull handle, **SQLError** returns SQL_NO_DATA_FOUND. In this case, szSqlState equals 00 000, pfNativeError is undefined, pcbErrorMsg equals zero, and szErrorMsg contains a single null termination byte (unless cbErrorMsgMax equals zero).

For more information about error codes, refer to Appendix A, "ODBC Error Codes."

SQLExecDirect

Core function **SQLExecDirect** prepares and executes a preparable statement (listed in Appendix C, "SQL Grammar"). **SQLExecDirect** is the fastest way to submit an SQL string for one-time execution.

Syntax

```
RETCODE SQLExecDirect(hstmt,szSqlStr,cbSqlStr)
```

XE "SQLExecDirect:description"§The **SQLExecDirect** function uses the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
UCHAR FAR *	szSqlStr	Input	SQL statement to be executed.
SDWORD	cbSqlStr	Input	Length of the SQL statement.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
07	000	Dynamic SQL error
07	001	Wrong number of parameters
08	002	Connection in use
08	S01	Communication link failure
21	000	Cardinality violation

21	S01	Insert value list does not match column list
21	S02	Degree of derived table does not match column list
22	001	String data right truncation
22	003	Numeric value out of range
22	005	Error in assignment
22	012	Division by zero
23	000	Integrity constraint violation
37	000	Syntax error or access violation
40	000	Serialization failure
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	C00	Driver not capable
80	T00	Timeout expired

DM	001	Driver does not support this function

SQLSTATE Class	Subclass	Description
SG	000	Invalid table name
SG	S00	Invalid table name; table not found
SG	S01	Table already exists
SH	000	Invalid view name
SH	S00	Invalid view name; view not found
SH	S01	View already exists
SI	000	Invalid index name
SI	S00	Invalid index name; index not found
SI	S01	Index already exists
SJ	000	Invalid column name
SJ	S00	Invalid column name; column not found
SJ	S01	Column already exists

Comments

The application calls **SQLExecDirect** to send an SQL string to the driver. The driver then submits the SQL string to the data source. For a list of valid SQL statements, refer to Appendix C, "SQL Grammar."

If the SQL statement is a **SELECT** statement, and if the application called **SQLSetCursorName** to associate a cursor with hstmt, then the driver uses the specified cursor. Otherwise, the driver generates a cursor name.

If the data source requires explicit transaction initiation, and a transaction has not already been initiated, the driver issues a begin transaction prior to submitting the statement.

If **SQLExecDirect** encounters a statement that contains a large data value parameter, specified in a call to **SQLSetParam**, the driver returns SQL_NEED_DATA immediately and does not process data.

If an **INSERT** or **UPDATE** statement contains a character literal that is longer than the defined length of the destination column, SQLExecDirect returns the SQLSTATE value 22 001 (string data right truncation).

Some data sources return informational messages after processing SQL commands. For example, Microsoft SQL Server returns a message when the database context changes. When such information is returned to the driver, **SQLExecDirect** returns SQL_SUCCESS_WITH_INFO . The application can call **SQLError** to retrieve the message.

The following table lists related ODBC functions.

For information about	See
Preparing a statement for later execution	**SQLPrepare**
Specifying a cursor name	**SQLSetCursorName**
Assigning storage for a column in the result set	**SQLBindCol**
Assigning storage for a parameter	**SQLSetParam**
Specifying asynchronous execution	**SQLSetStmtOption** (extension)
Executing a prepared statement	**SQLExecute**
Retrieving data	**SQLFetch**, **SQLGetData** (extension), **SQLExtendedFetch** (extension)
Sending large data values	**SQLPutData** (extension) **SQLParamData** (extension)

SQLExecute

Core function **SQLExecute** executes a prepared statement, using the current values of the parameter marker variables if any parameter markers exist in the statement.

Syntax

```
RETCODE SQLExecute(hstmt)
```

XE "SQLExecute"§The **SQLExecute** statement accepts the following argument.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING,SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
07	000	Dynamic SQL error
07	001	Wrong number of parameters
08	002	Connection in use
08	S01	Communication link failure
21	000	Cardinality violation
21	S01	Insert value list does not match column list
21	S02	Degree of derived table does not match column list
22	001	String data right truncation

SQLSTATE Class	Subclass	Description
22	003	Numeric value out of range
22	005	Error in assignment
22	012	Division by zero
23	000	Integrity constraint violation
37	000	Syntax error or access violation
40	000	Serialization failure
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	C00	Driver not capable
80	T00	Timeout expired
DM	001	Driver does not support this function
SG	000	Invalid table name

SG	S00	Invalid table name; table not found
SG	S01	Table already exists
SH	000	Invalid view name
SH	S00	Invalid view name; view not found
SH	S01	View already exists
SI	000	Invalid index name
SI	S00	Invalid index name; index not found
SI	S01	Index already exists
SJ	000	Invalid column name
SJ	S00	Invalid column name; column not found
SJ	S01	Column already exists

Comments

To prepare and execute a statement with parameters, an application performs the following steps:

1. Call **SQLPrepare** to prepare the statement.
2. Call **SQLSetParam** to associate storage locations with parameter markers.
3. Set parameter values.
4. Call **SQLExecute**.

The driver sends the SQL string to the data source, including any parameter values.

Once the application processes or discards the results from a call to **SQLExecute**, the application can call **SQLExecute** again with new parameter values.

To execute a **SELECT** statement more than once, the application can call **SQLFreeStmt** with the SQL_CLOSE parameter before reissuing the **SELECT** statement. If the application attempts to resubmit a **SELECT** statement without first calling **SQLFreeStmt**, the driver returns SQLSTATE 08 002 (connection in use).

If the data source requires explicit transaction initiation, the driver requests a begin transaction operation before it sends the SQL string.

An application cannot use **SQLExecute** to prepare or submit a **COMMIT** or **ROLLBACK** statement. To commit or roll back a transaction, the application must call **SQLTransact**.

The driver returns control after the SQL operation is finished and result or error information is available.

If **SQLExecute** encounters a statement that contains a parameter with a large data value, as specified in a call to **SQLSetParam**, the driver returns SQL_NEED_DATA immediately and does not process data.

If an **INSERT** or **UPDATE** statement contains a character literal that is longer than the defined length of the destination column, **SQLExecute** returns the SQLSTATE value 22 001 (string data right truncation).

Some data sources return informational messages after processing SQL commands. For example, Microsoft SQL Server returns a message when the database context changes. When such information is returned to the driver, **SQLExecute** returns SQL_SUCCESS_WITH_INFO . The application can call **SQLError** to retrieve the message.

The following table lists related ODBC functions.

For information about	See
Preparing a statement prior to execution	**SQLPrepare**
Executing a statement without preparing it first	**SQLExecDirect**
Defining storage for a column in a result set	**SQLBindCol**
Defining storage for a parameter in an SQL statement	**SQLSetParam**
Associating a cursor with a statement	**SQLSetCursorName**

Canceling statement processing	**SQLFreeStmt**, **SQLCancel**
Sending large data values	**SQLPutData** (extension) **SQLParamData** (extension)

SQLExtendedFetch

Extension Level 2

SQLExtendedFetch fetches blocks of data in the following ways:

- Returns a block of data (one rowset), in the form of an array, for each bound column
- Returns a block of data according to the setting of a rowset pointer—forwards, backwards, or by index value

SQLExtendedFetch works in conjunction with **SQLSetScrollOptions**.

XE "SQLExtendedFetch"§To fetch one row of data at a time in a forward direction, an application should call **SQLFetch**.

Syntax

```
RETCODE  SQLExtendedFetch(hstmt,fFetchType,irow,pcrow,rgfRowStatus)
```

The **SQLExtendedFetch** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
UWORD	fFetchType	Input	Type of fetch, as listed in the following table.
SDWORD	irow	Input	Number of the row to fetch.
UDWORD FAR *	pcrow	Output	Number of rows actually fetched. If an error occurs during processing, pcrow points to the row that precedes the row with the error.
UWORD FAR *	rgfRowStatus	Output	An array of status values. The number of elements must equal crowRowSet (as defined in a call to **SQLSetScrollOptions**). The driver returns a status value for each row fetched. rgfRowStatus values can equal SQL_ROW_SUCCESS, SQL_ROW_DELETED, or SQL_ROW_UPDATED. If the number

of rows fetched is less than the number of elements in the status array, the driver sets remaining status elements to SQL_ROW_NOROW.

Valid values for fFetchType depend on the fetch capabilities defined for the driver. The following table lists all valid values.

fFetchType	Description
SQL_FETCH_NEXT	Fetch the next rowset.
SQL_FETCH_FIRST	Fetch the first rowset.
SQL_FETCH_LAST	Fetch the last rowset.
SQL_FETCH_PREV	Fetch the previous rowset.
SQL_FETCH_ABSOLUTE	Fetch the rowset starting with row irow.
SQL_FETCH_RELATIVE	Fetch the rowset starting with irows from the first row in the current rowset.
SQL_FETCH_RESUME	Fetch the remainder of a rowset that received an error and was partially retrieved.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_NO_DATA_FOUND, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
01	004	Data truncated
07	002	Number of bound columns doesn't match
08	S01	Communication link failure

22	003	Numeric value out of range
22	012	Division by zero
24	000	Invalid cursor state
40	000	Serialization failure
80	000	General ODBC error
80	001	Memory allocation error
80	006	Invalid conversion specified
80	008	Operation canceled
80	009	Invalid argument value
80	010	ODBC function sequence error
80	C00	Driver not capable
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

The block of data contains the number of rows indicated by the crowRowSet argument in **SQLSetScrollOptions.**

The buffer specified in **SQLBindCol** must be large enough to hold the number of rows specified by crowRowSet in **SQLScrollOptions**.

Immediately after the fetch, the driver leaves the cursor positioned on the entire rowset.

The status array rgfRowStatus allows an application to detect holes in the rowset (SQL_ROW_DELETED) or rows that have changed since they were last fetched (SQL_ROW_UPDATED). It may not be possible, however, for a driver to detect changes. If not, the driver sets the status value to SQL_ROW_SUCCESS. An application can call **SQLGetInfo** to determine if the driver can detect changes in rows obtained through **SQLExtendedFetch.**

If an error occurs during the fetch of a rowset, **SQLExtendedFetch** returns SQL_ERROR and an associated error value. The driver sets pcrow to the number of rows actually fetched. The next row contains an error. To finish processing the rowset, the application can resolve the error and call **SQLExtendedFetch** with SQL_FETCH_RESUME.

The update of a key value is interpreted as the deletion of the underlying row and the addition of a new row.

The following table lists related ODBC functions.

For information about	See
Specifying the number of rows to fetch	**SQLSetScrollOptions**
Setting cursor position within a fetched array	**SQLSetPos**
Executing a prepared statement	**SQLExecute**
Returning the number of columns in a result set	**SQLNumResultCols**
Returning information about a column in a result set	**SQLDescribeCol**
Assigning a storage location to a column in a result set	**SQLBindCol**

SQLFetch

Core function **SQLFetch** fetches a row of data from a result set. The driver returns data for all columns that were bound to storage locations with **SQLBindCol**.

Syntax

RETCODE SQLFetch(hstmt)

XE "SQLFetch"§The **SQLFetch** function accepts the following argument.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.

Returns

SQL_SUCCESS, SQL_INVALID_HANDLE, SQL_STILL_EXECUTING, SQL_ERROR, SQL_NO_DATA_FOUND, or SQL_SUCCESS_WITH_INFO.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
01	004	Data truncated
07	002	Number of bound columns doesn't match
08	S01	Communication link failure
22	003	Numeric value out of range
22	012	Division by zero
24	000	Invalid cursor state
40	000	Serialization failure
80	000	General ODBC error

80	001	Memory allocation error
80	008	Operation canceled
80	010	ODBC function sequence error
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

SQLFetch advances the cursor to the next row. If the application called **SQLBindCol** to bind columns, **SQLFetch** stores data into the locations specified by the calls to **SQLBindCol**.

The driver manages cursors during the fetch operation and places each value of a bound column into the associated variable. The driver follows these guidelines when performing a fetch operation:

- **SQLFetch** accesses column data in left-to-right order.
- After each fetch, pcbValue (specified in **SQLBindCol**) contains the actual number of bytes in the result column.
- If rgbValue is not large enough to hold the entire result, the driver stores part of the value and returns SQL_SUCCESS_WITH_INFO. A subsequent call to **SQLError** indicates that a truncation occurred. The application can compare pcbValue to cbValueMax (specified in **SQLBindCol**) to determine which column or columns were truncated. If pcbValue is greater than or equal to pcbValueMax, then truncation occurred.
- If the data value for the column is null, the driver stores SQL_NULL_DATA in pcbValue and does not modify rgbValue.

SQLFetch is valid only after a call that returns a result set.

When **SQLBindCol** specifies a conversion, **SQLFetch** can return the following results:

- If the conversion is unsuccessful because it would truncate the whole part of a number or would exceed the range of the destination data type, **SQLFetch** returns SQL_ERROR and SQLSTATE 22 003 (numeric value out of range).
- If the conversion truncates a character string or the fractional part of a number,

SQLFetch returns SQL_SUCCESS_WITH_INFO and SQLSTATE value 01 004 (data truncated).

An application can call **SQLGetData** to retrieve data that is not bound to a storage location. If the application does not call **SQLBindCol** to bind one or more columns, **SQLFetch** doesn't return any data, it just moves the cursor to the next row.

When finished with the result set, the driver returns SQL_NO_DATA_FOUND.

For information about conversions allowed by **SQLBindCol** and **SQLGetData**, refer to
Appendix D, "Data Type Definitions."

The following table lists related ODBC functions.

For information about	See
Preparing an SQL statement prior to execution	**SQLPrepare**
Executing an SQL statement	**SQLExecute**, **SQLExecDirect**
Assigning storage for result columns	**SQLBindCol**
Listing information about the result set	**SQLNumResultCols**, **SQLDescribeCol**
Returning a column with a large data value	**SQLGetData** (extension)
Handling blocks of array rows	**SQLExtendedFetch** (Extensions)
Supporting scrollable cursors	**SQLSetScrollOptions** (Extension)
Discarding result sets	**SQLCancel**, **SQLFreeStmt**

SQLForeignKeys

Extension Level 2

SQLForeignKeys returns a list of column names that comprise foreign keys, if foreign keys exist for a specified table. The driver returns the list as a result set on the specified hstmt.

Syntax

```
RETCODE  SQLForeignKeys(hstmt,szPkTableQualifier,cbPkTableQualifier,szPkTableOwner,
    cbPkTableOwner,szPkTableName,cbPkTableName,szFkTableQualifier,cbFkTableQualifier,
    szFkTableOwner,cbFkTableOwner,szFkTableName,cbFkTableName)
```

XE "SQLForeignKeys"§The **SQLForeignKeys** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle to retrieve results.
UCHAR FAR *	szPkTableQualifier	Input	Primary key table qualifier.
SWORD	cbPkTableQualifier	Input	Length of szPkTableQualifier.
UCHAR FAR *	szPkTableOwner	Input	Primary key owner name.
SWORD	cbPkTableOwner	Input	Length of szPkTableOwner.
UCHAR FAR *	szPkTableName	Input	Primary key table name.
SWORD	cbPkTableName	Input	Length of szPkTableName.
UCHAR FAR *	szFkTableQualifier	Input	Foreign key table qualifier.
SWORD	cbFkTableQualifier	Input	Length of szFkTableQualifier.

UCHAR FAR *	szFkTableOwner	Input	Foreign key owner name.
SWORD	cbFkTableOwner	Input	Length of szFkTableOwner.
UCHAR FAR *	szFkTableName	Input	Match foreign key table name.
SWORD	cbFkTableName	Input	Length of szFkTableName.

Returns

SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use
08	S01	Communication link failure
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	C00	Driver not capable
80	T00	Timeout expired

DM	001	Driver does not support this function

Comments

If the szPkTableName argument is not null, **SQLForeignKeys** returns the table names and column names that comprise foreign keys associated with the primary key table.

If the szFkTableName argument is not null, **SQLForeignKeys** returns the table names that contain the primary key associated with this foreign key.

If both table names are null, **SQLForeignKeys** returns SQLSTATE value 80 009 (invalid argument value).

SQLForeignKeys returns results as a standard result set. The following table lists result columns.

Column Name	Data Type
PKTABLE_QUALIFIER	Varchar(32)
PKTABLE_OWNER	Varchar(32)
PKTABLE_NAME	Varchar(32) not null
PKCOLUMN_NAME	Varchar(32) not null
FKTABLE_QUALIFIER	Varchar(32)
FKTABLE_OWNER	Varchar(32)
FKTABLE_NAME	Varchar(32) not null
FKCOLUMN_NAME	Varchar(32) not null
KEY_SEQ	Smallint not null; sequence number of column in multipart key
UPDATE_DELETE_RULE	Smallint not null; one of

	SQL_CASCADE, SQL_RESTRICT, or SQL_SET_NULL

The following table lists related ODBC functions.

For information about	See
Retrieving names of columns that comprise the primary key of a table	**SQLPrimaryKeys**

SQLFreeConnect

Core function **SQLFreeConnect** releases a connection handle and frees all memory associated with the handle.

Syntax

```
RETCODE SQLFreeConnect(hdbc)
```

XE "SQLFreeConnect"§The **SQLFreeConnect** function accepts the following argument.

Type	Argument	Use	Description
HDBC	hdbc	Input	Connection handle.

Returns

SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use
08	S01	Communication link failure
80	000	General ODBC error

Comments

Prior to calling **SQLFreeConnect,** an application must call **SQLDisconnect** for the hdbc and **SQLFreeStmt** for all hstmts opened on the hdbc . Otherwise, **SQLFreeConnect** returns SQL_ERROR and hdbc remains valid.

The following table lists related ODBC functions.

For information about	See
Allocating a statement handle (the opposite of **SQLFreeConnect**)	**SQLAllocConnect**

Establishing a connection to a data source	**SQLConnect, SQLDriverConnect**
Freeing a connection to a data source	**SQLDisconnect**
Freeing the environment handle	**SQLFreeEnv**

SQLFreeEnv

Core function **SQLFreeEnv** frees the environment handle and releases all memory associated with the environment handle.

Syntax

RETCODE SQLFreeEnv(henv)

XE "SQLFreeEnv"§The **SQLFreeEnv** function accepts the following argument.

Type	Argument	Use	Description
HENV	henv	Input	Environment handle.

Returns

SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use
80	000	General ODBC error
80	010	ODBC function sequence error

Comments

Prior to calling **SQLFreeEnv**, an application must call **SQLFreeConnect** for all hdbcs opened under the henv. Otherwise, **SQLFreeEnv** returns SQL_ERROR and the henv and hdbcs remain valid.

The following table lists a related ODBC function.

For information about	See
Allocating an environment handle (the opposite of **SQLFreeEnv**)	**SQLAllocEnv**

C Header Files

SQLFreeStmt

Core function **SQLFreeStmt** stops processing associated with a specific hstmt, closes any open cursors associated with the hstmt, discards pending results, and, optionally, frees all resources associated with the statement handle.

Syntax RETCODE SQLFreeStmt(hstmt,fOption)

XE "SQLFreeStmt"§The **SQLFreeStmt** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle
UWORD	fOption	Input	One of the following options: SQL_ CLOSE: Close the cursor associated with hstmt (if one was defined) and discard all pending results. The application can reopen this cursor later by executing a **SELECT** statement again later with the same or different parameter values. If no cursor is open, this option has no effect. SQL_DROP: Release the hstmt, free all resources associated with it, close the cursor (if one is open), and discard all pending rows. This option terminates all access to the hstmt. SQL_UNBIND: Release all column buffers bound by **SQLBindCol** for the given hstmt. SQL_RESET_PARAMS: Release all parameter buffers set by **SQLSetParam** for the given hstmt.

Returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description

80	000	General ODBC error
80	001	Memory allocation error
80	009	Invalid argument value
DM	001	Driver does not support this function

Comments

An application can call **SQLFreeStmt** to terminate processing of a **SELECT** statement with or without canceling the statement handle.

The SQL_DROP option frees all resources that were allocated by the **SQLAllocStmt** function.

If **SQLFreeStmt** is called for a statement that is executing asynchronously, **SQLFreeStmt** has, in addition to its own functionality, the same effect as **SQLCancel**.

The following table lists related ODBC functions.

For information about	See
Allocating a statement handle (the opposite of **SQLFreeStmt**)	**SQLAllocStmt**
Associating a name and statement with a cursor	**SQLSetCursorName**
Canceling statement processing asynchronously and retaining the statement handle	**SQLCancel**

SQLGetConnectOption

Extension Level 1

SQLGetConnectOption returns the current setting of a connection option.

Syntax

```
RETCODE  SQLGetConnectOption(hdbc,fOption,pvParam)
```

XE "SQLGetConnectOption"§The **SQLGetConnectOption** function accepts the following arguments:

Type	Argument	Use	Description
HDBC	hdbc	Input	Database connection handle.
UWORD	fOption	Input	Connection option to retrieve.
PTR	pvParam	Output	Value associated with fOption.

Returns

SQL_SUCCESS, SQL_NO_DATA_FOUND, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	003	Connection not open
80	000	General ODBC error
80	001	Memory allocation error
80	009	Invalid argument value
DM	001	Driver does not support this function

Comments

Depending on the option, an application does not need to establish a connection prior to calling **SQLGetConnectOption**.

For a list of options, refer to **SQLSetConnectOption**.

While an application can set statement options using **SQLSetConnectOption**, an application cannot use **SQLGetConnectOption** to retrieve statement option values; it must call **SQLGetStmtOption** to retrieve the setting of statement options.

The following table lists related ODBC functions.

For information about	See
Retrieving options associated with an hstmt	**SQLGetStmtOption**
Setting connection or statement options associated with an hdbc	**SQLSetConnectOption, SQLSetStmtOption**

SQLGetCursorName

Core function **SQLGetCursorName** returns the cursor name associated with a specified hstmt.

Syntax RETCODE SQLGetCursorName(hstmt,szCursor,cbCursorMax,pcbCursor)

XE "SQLGetCursorName"§The **SQLGetCursorName** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle**.**
UCHAR FAR *	szCursor	Output	Cursor name.
SWORD	cbCursorMax	Input	Length of szCursor.
SWORD FAR *	pcbCursor	Output	Number of bytes placed in szCursor.

Returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE, or SQL_SUCCESS_WITH_INFO.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
01	004	Data truncated
80	000	General ODBC error
80	001	Memory allocation error
80	009	Invalid argument value

80	015	No cursor name available
DM	001	Driver does not support this function

Comments The only ODBC operation that accepts a cursor name is a positioned update or delete (for example, "**UPDATE** *table-name* ...**WHERE CURRENT OF** *cursor-name*"). If the application does not call **SQLSetCursorName** to define a cursor name for a positioned update or delete, the driver generates a name that begins with the letters SQL_CUR.

SQLGetCursorName returns the name of a cursor regardless of whether the name was created explicitly or implicitly.

If truncation occurs, **SQLGetCursorName** returns SQL_SUCCESS_WITH_INFO. **SQLError** returns the truncation SQLSTATE value (01 004) and pcbCursor contains the length of the full cursor name.

The following table lists related ODBC functions.

For information about	See
Associating a name and statement with a cursor	**SQLSetCursorName**
Preparing a statement for execution	**SQLPrepare**
Executing a statement	**SQLExecute**, **SQLExecDirect**
Establishing a scrollable cursor	**SQLSetScrollOptions** (extension)

SQLGetData

Extension Level 1

SQLGetData returns result data for a single column in the current row. Unlike **SQLBindCol**, this function retrieves data one column at a time, and can be used to retrieve large data values (for example, SQL_LONG or SQL_VARCHAR types) in pieces.

Syntax

```
RETCODE  SQLGetData(hstmt,icol,fCType,rgbValue,cbValueMax,pcbValue)
```

XE "SQLGetData"§The **SQLGetData** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
UWORD	icol	Input	Column number. (If the application has called **SQLGetData** for this result set, icol must be greater than or equal to the column number in the preceding call to **SQLGetData**.)
SWORD	fCType	Input	Data type to convert data into: SQL_C_CHAR, SQL_NO_CONVERT, or SQL_C_BINARY (native database format without conversion).
PTR	rgbValue	Output	Output data buffer.
SDWORD	cbValueMax	Input	Length of rgbValue. cbValueMax determines the amount of data that can be received in a single call to **SQLGetData**
SDWORD FAR *	pcbValue	Output	Total number of bytes, or SQL_NO_TOTAL if the length cannot be determined in advance (see "Comments," below). If pcbValue is greater than cbValueMax or SQL_NO_TOTAL, there is more data to fetch.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA_FOUND, SQL_INVALID_HANDLE, SQL_STILL_EXECUTING, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
01	004	Data truncated
08	003	Connection not open
08	S01	Communication link failure
22	005	Error in assignment
24	000	Invalid cursor state
80	000	General ODBC error
80	001	Memory allocation error
80	002	Invalid column number
80	006	Invalid conversion specified
80	008	Operation canceled

SQLSTATE Class	Subclass	Description
80	009	Invalid argument value
80	010	Function sequence error
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

The application must call **SQLFetch** or **SQLExtendedFetch** before it calls **SQLGetData**. The driver follows these guidelines when performing a fetch operation:

- **SQLFetch** accesses column data in left-to-right order.
- After each fetch, pcbValue (specified in **SQLBindCol**) contains the actual number of bytes in the result column.
- If the allocated buffer is not large enough to hold the entire result, the driver stores part of the value and returns SQL_SUCCESS_WITH_INFO. A subsequent call to **SQLError** indicates that a truncation occurred. The application can compare pcbValue to cbValueMax (specified in **SQLBindCol**) to determine which column or columns were truncated. If pcbValue is greater than or equal to pcbValueMax, then truncation occurred.
- If the data value for the column is null, the driver stores SQL_NULL_DATA in pcbValue.

SQLFetch positions the cursor at the next row and retrieves all bound columns. **SQLExtendedFetch** retrieves all bound columns for each row in the rowset and leaves the cursor positioned on the entire rowset.

If the application called **SQLFetch**, the application can then call **SQLGetData** to retrieve data for specific, unbound columns. If the application called **SQLExtendedFetch**, and retrieved more than one row, the application must call **SQLSetPos** to position the cursor to a specific row prior to calling **SQLGetData**.

The application cannot call **SQLGetData** to retrieve a column that resides at or before the last bound column, if bound columns exist for the result set.

If more than one call to **SQLGetData** is required to receive data for a single column, the driver sets pcbValue to the total number of bytes in the result and returns

SQL_SUCCESS_WITH_INFO. A subsequent call to **SQLError** returns SQLSTATE value
01 004 (data truncated). The application can then use the same column number for subsequent calls until **SQLGetData** returns SQL_SUCCESS. The application can ignore excess data by proceeding to the next result column.

If the total number of bytes in the result set cannot be determined in advance, the driver sets pcbValue to SQL_NO_TOTAL and returns SQL_SUCCESS_WITH_INFO. A subsequent call to **SQLError** returns SQLSTATE 01 004 (data truncated). The application can use the same column number for subsequent calls until **SQLGetData** returns SQL_SUCCESS. To ignore data, the application can process the next result column.

SQLGetData can convert data to a different data type. The result and success of the conversion is determined by the rules for assignment specified in Appendix D, "Data Type Definitions."

The following table lists related ODBC functions.

For information about	See
Executing a statement	**SQLExecDirect, SQLExecute**
Binding a column to a storage location	**SQLBindCol**
Advancing the row pointer and fetching bound data associated with the row	**SQLFetch, SQLExtendedFetch** (extension)

SQLGetFunctions

Extension Level 1

SQLGetFunctions returns information about whether a driver supports a specific ODBC function.

Syntax

RETCODE SQLGetFunctions(hdbc,fFunction,pfExists)

XE "SQLGetFunctions"§The **SQLGetFunctions** function accepts the following arguments:

Type	Argument	Use	Description
HDBC	hdbc	Input	Database connection handle.
UWORD	fFunction	Input	ODBC function of interest. Set fFunction to the function of interest by supplying the function number as defined in the SQL.H or SQLEXT.H file.
UDWORD FAR *	pfExists	Output	TRUE if the specified function is supported by the driver associated with the specified hdbc; otherwise, FALSE.

Returns

SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

The possible SQLSTATE values are listed in the following table.

SQLSTATE Class	Subclass	Description
80	000	General ODBC error
80	001	Memory allocation error
80	009	Invalid argument value

Comments

The following table lists #define names for fFunction .These constants are listed in the SQL.H and SQLEXT.H files as #defined statements. They are guaranteed to be sequential. For a list of numeric values associated with these names, refer to Appendix F, " C Header Files."

Core Function

SQL_DLL_SQLALLOCCONNECT

SQL_DLL_SQLALLOCENV

SQL_DLL_SQLALLOCSTMT

SQL_DLL_SQLBINDCOL

SQL_DLL_SQLCANCEL

SQL_DLL_SQLCONNECT

SQL_DLL_SQLDESCRIBECOL

SQL_DLL_SQLDISCONNECT

SQL_DLL_SQLERROR

SQL_DLL_SQLEXECDIRECT

SQL_DLL_SQLEXECUTE

SQL_DLL_SQLFETCH

SQL_DLL_SQLFREECONNECT

SQL_DLL_SQLFREEENV

SQL_DLL_SQLFREESTMT

SQL_DLL_SQLGETCURSORNAME

SQL_DLL_SQLNUMRESULTCOLS

SQL_DLL_SQLPREPARE

SQL_DLL_SQLROWCOUNT

SQL_DLL_SQLSETCURSORNAME

SQL_DLL_SQLSETPARAM

SQL_DLL_SQLTRANSACT

The following table lists #define names for ODBC extended functions.

Extended Function
SQL_DLL_SQLCOLATTRIBUTES
SQL_DLL_SQLCOLUMNPRIVILEGES
SQL_DLL_SQLCOLUMNS
SQL_DLL_SQLDATASOURCES
SQL_DLL_SQLDESCRIBEPARAM
SQL_DLL_SQLDRIVERCONNECT
SQL_DLL_SQLEXTENDEDFETCH
SQL_DLL_SQLFOREIGNKEYS
SQL_DLL_SQLGETCONNECTOPTION
SQL_DLL_SQLGETDATA
SQL_DLL_SQLGETFUNCTIONS
SQL_DLL_SQLGETINFO
SQL_DLL_SQLGETSTMTOPTION
SQL_DLL_SQLGETTYPEINFO

SQL_DLL_SQLMORERESULTS

SQL_DLL_SQLNATIVESQL

SQL_DLL_SQLNUMPARAMS

SQL_DLL_SQLPARAMDATA

SQL_DLL_SQLPARAMOPTIONS

SQL_DLL_SQLPRIMARYKEYS

SQL_DLL_SQLPUTDATA

SQL_DLL_SQLSETCONNECTOPTIO
N

SQL_DLL_SQLSETPOS

SQL_DLL_SQLSETSCROLLOPTION
S

SQL_DLL_SQLSETSTMTOPTION

SQL_DLL_SQLSPECIALCOLUMNS

SQL_DLL_SQLSTATISTICS

SQL_DLL_SQLTABLEPRIVILEGES

SQL_DLL_SQLTABLES

The following table lists related ODBC functions.

For information about	See
Describing the driver and data source	**SQLGetInfo** (extension)
Retrieving values of options that govern specific aspects of driver operation	**SQLGetStmtOption, SQLGetConnectOption** (extensions)

SQLGetInfo

Extension Level 1

XE "SQLGetInfo"§SQLGetInfo returns general information about the driver and data source associated with an hdbc.

Syntax

RETCODE SQLGetInfo(hdbc,fInfoType,rgbInfoValue,cbInfoValueMax,pcbInfoValue)

The **SQLGetInfo** function accepts the following arguments.

Type	Argument	Use	Description
HDBC	hdbc	Input	Connection handle.
UWORD	fInfoType	Input	Type of information. fInfoType must be a value representing the type of interest (see "Comments," below).
PTR	rgbInfoValue	Output	Pointer to a buffer to contain the information. The application should reserve at least 256 bytes for this buffer.
SWORD	cbInfoValueMax	Input	Length of rgbInfoValue.
SWORD FAR *	pcbInfoValue	Output	Number of bytes available to be

written to rgbInfoValue.

Returns SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_ERROR or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
01	004	Data truncated
08	002	Connection in use
08	003	Connection not open
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	C00	Driver not capable
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments The values for fInfoType correspond to values in the following table. Type numbers from 1-999 are reserved by ODBC; type numbers from 1,000 to 64,000 can be reserved for use by specific drivers. The values in the following table are #defined in the SQLEXT.H file, listed in Appendix F, "C Header Files."

Where rgbInfoValue returns a bit mask, the application must store rgbInfoValue as an unsigned integer of the specified size. For example, the value returned for fInfoType equal to SQL_STRING_FUNCTIONS is a bit mask expressed as a 32-bit unsigned integer. The value returned for fInfoType equal to SQL_SCROLL_OPTIONS is a bit mask expressed as a 16-bit unsigned integer.

fInfoType	Returns
SQL_ACCESSIBLE_TABLES	Y if the user is guaranteed **SELECT** privileges to all tables returned by **SQLTables**, N if there may be tables returned that the user cannot access.
SQL_ACTIVE_STATEMENTS	The maximum number of active hstmts that the driver can support for an hdbc. This value can reflect a limitation imposed by either the driver or the data source. If there is no specified limit, this value is set to zero (0).
SQL_COLLATION_SEQ	Either "ISO8859-1" or "EBCDIC".
SQL_COMMIT_CURSOR_BEHAVIOR	How a COMMIT operation affects cursors in the data source. One of the following values: 0 = Close and delete cursors; the application must prepare and execute the next statement. 1 = Close cursors and position them at the first row; the application can execute or fetch without preparing the statement again. 2 = Preserve cursors in the same position as before the COMMIT operation; the application can execute or fetch without preparing the statement again.
SQL_CONVERT_BIGINT	The conversions supported by the data source for a BIGINT data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.

SQL_CONVERT_BINARY	The conversions supported by the data source for a BINARY data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_CONVERT_BIT	The conversions supported by the data source for a BIT data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.

fInfoType	Returns
SQL_CONVERT_CHAR	The conversions supported by the data source for a CHAR data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_CONVERT_DATE	The conversions supported by the data source for a DATE data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_CONVERT_DECIMAL	The conversions supported by the data source for a DECIMAL data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_CONVERT_DOUBL	The conversions supported by the data source for a

E	DOUBLE data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_CONVERT_FLOAT	The conversions supported by the data source for a FLOAT data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_CONVERT_FUNCTIONS	The canonical conversion functions supported by the driver and associated data source. The driver returns a 32-bit unsigned integer. The value can contain the boolean of one or more of the following values: SQL_FN_CVT_CONVERT.
SQL_CONVERT_INTEGER	The conversions supported by the data source for an INTEGER data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_CONVERT_LONGVARCHAR	The conversions supported by the data source for a LONGVARCHAR data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_CONVERT_NUMERIC	The conversions supported by the data source for a NUMERIC data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.

fInfoType	Returns
SQL_CONVERT_REAL	The conversions supported by the data source for a REAL data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_CONVERT_SMALLINT	The conversions supported by the data source for a SMALLINT data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_CONVERT_TIME	The conversions supported by the data source for a TIME data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_CONVERT_TIMESTAMP	The conversions supported by the data source for a TIMESTAMP data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_CONVERT_TINYINT	The conversions supported by the data source for a TINYINT data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_CONVERT_VARBINARY	The conversions supported by the data source for a VARBINARY data type. Refer to the conversion options table, below, for a list of conversion mask

	#defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_CONVERT_VARCHAR	The conversions supported by the data source for a VARCHAR data type. Refer to the conversion options table, below, for a list of conversion mask #defines. If zero, the data source does not support any conversions for the data type, including a conversion to the same type.
SQL_DATABASE_NAME	Current database in use.
SQL_DATA_SOURCE_NAME	Data source name used during **SQLConnect**.
SQL_DBMS_NAME	Name of the data source accessed by the driver.
SQL_DBMS_VER	Version of the data source accessed by the driver. At a minimum, the version is of the form ##.####, where the first two digits are the major version and the last four digits are the minor version.

fInfoType	Returns
SQL_DEFAULT_TXN_ISOLATION	One of the following values: 0 = Changes are immediately perceived by all transactions (Dirty read, nonrepeatable read and phantoms are possible) 1 = Row read by transaction 1 can be altered and committed by transaction 2 (nonrepeatable read and phantoms are possible) 2 = A transaction can add or remove rows matching the search condition or a pending transaction (phantoms are possible) 3 = Data affected by pending transaction is not

	available to other transactions 4 = Equivalent of Oracle's Read Consistency isolation
SQL_DRIVER_HENV	A pointer to the driver-level environment handle. To obtain this information, the application must set rbgInfoValue to point to the Driver Manager handle (henv).
SQL_DRIVER_HDBC	A pointer to the driver-level connection handle. To obtain this information, the application must set rbgInfoValue to point to the Driver Manager handle (hdbc).
SQL_DRIVER_HSTMT	A pointer to the driver-level statement handle. To obtain this information, the application must set rbgInfoValue to point to the Driver Manager handle (hstmt).
SQL_DRIVER_NAME	File name of the client driver.
SQL_DRIVER_VER	Version of the client driver and, optionally a description of the driver. At a minimum, the version is of the form ##.####, where the first two digits are the major version and the last four digits are the minor version.
SQL_EXPRESSIONS_IN_ ORDERBY	Y if the data source supports expressions in the ORDER BY list, N if it does not.
SQL_FETCH_DIRECTION	Options supported for fetch direction. Returns a 16-bit unsigned integer. The value can contain the boolean of one or more of the following values: SQL_FD_FETCH_NEXT SQL_FD_FETCH_FIRST SQL_FD_FETCH_LAST SQL_FD_FETCH_PREV SQL_FD_FETCH_ABSOLUTE SQL_FD_FETCH_RELATIVE

SQL_FD_FETCH_RESUME

fInfoType	Returns
SQL_IDENTIFIER_CASE	One of the following values: 1 = Names must be upper case 2 = Names must be lower case 3 = Names are case-sensitive and can be mixed 4 = Names are not case-sensitive
SQL_IDENTIFIER_QUOTE_CHAR	Type of quote character associated with the identifier that is used by the data source to specify a table name, or blank if the data source does not support quote characters for table specification. One of the following values: ' = Single quote character " = Double quote character " " = Blank A driver can define additional values.
SQL_MAX_COLUMN_NAME_LEN	Maximum length of a column name in the data source.
SQL_MAX_OWNER_NAME_LEN	Maximum length of an owner name in the data source. A length of zero indicates that the data source does not support owner name as a qualifier of a table name.
SQL_MAX_QUALIFIER_NAME_LEN	Maximum length of a qualifier name in the data source.
SQL_MAX_TABLE_NAME_LEN	Maximum length of a table name in the data source.

SQL_MULTIPLE_ACTIVE_TXN	Y if active transactions on multiple connections are allowed, N if only one connection at a time can have an active transaction.
SQL_MULT_RESULT_SETS	Y if the database supports multiple result sets, N if it does not.

fInfoType	Returns
SQL_NUMERIC_FUNCTIONS	The canonical numeric functions supported by the driver and associated data source. The driver returns a 32-bit unsigned integer that can contain the boolean of one or more of the following values: SQL_FN_NUM_ABS SQL_FN_NUM_ACOS SQL_FN_NUM_ASIN SQL_FN_NUM_ATAN SQL_FN_NUM_ATAN2 SQL_FN_NUM_CEILING SQL_FN_NUM_COS SQL_FN_NUM_COT SQL_FN_NUM_EXP SQL_FN_NUM_FLOOR SQL_FN_NUM_LOG SQL_FN_NUM_MOD SQL_FN_NUM_RAND SQL_FN_NUM_PI SQL_FN_NUM_SIGN SQL_FN_NUM_SIN SQL_FN_NUM_SQRT SQL_FN_NUM_TAN
SQL_ODBC_CONFORMANCE	Level of ODBC Conformance: 0 = None 1 = Level 1 supported 2 = Level 2 supported (For a list of functions and conformance levels,

	refer to Chapter 1, "ODBC Function Summary.)
SQL_ODBC_VER	Version of ODBC to which the driver conforms.
SQL_OUTER_JOINS	Y if the data source supports outer joins and the driver supports the canonical outer join escape syntax, N otherwise.
SQL_OWNER_TERM	Data source vendor's name for an owner; for example, 'owner', 'Authorization ID', or 'Schema'.
SQL_ROLLBACK_CURSOR_ BEHAVIOR	How a ROLLBACK operation affects cursors in the data source. One of the following values: 0 = Close and delete cursors; the application must prepare and execute the next statement. 1 = Close cursors and position them at the first row; the application can execute or fetch without preparing the statement again. 2 = Preserve cursors in the same position as before the ROLLBACK operation; the application can execute or fetch without preparing the statement again.

fInfoType	Returns
SQL_ROW_UPDATES	Y if the driver can detect changes in rows between multiple fetches of the same rows; otherwise, N.
SQL_SAG_CLI_ CONFORMANCE	Compliance to SAG specification: 0 = Not SAG-compliant; one or more core functions are not supported 1 = SAG-compliant
SQL_SAVEPOINT_SUPPORT	Y if the underlying data source supports named savepoints, N if it does not.

SQL_SCROLL_OPTIONS	Options supported for scrollable cursors. Returns a 16-bit unsigned integer. Each bit represents a scroll option. The value can contain the boolean of one or more of the following values: SQL_SO_FORWARD_ONLY SQL_SO_KEYSET_DRIVEN SQL_SO_DYNAMIC SQL_SO_MIXED
SQL_SCROLL_ CONCURRENCY	Options supported for concurrency control of scrollable cursors. Returns a 16-bit unsigned integer. Each bit represents a concurrency control option. The value can contain the boolean of one or more of the following values: SQL_SCCO_READ_ONLY SQL_SCCO_LOCK SQL_SCCO_OPT_TIMESTAMP SQL_SCCO_OPT_VALUES
SQL_SERVER_NAME	Actual data source name; useful when a data source name is used during **SQLConnect**.
SQL_STRING_FUNCTION S	The canonical string functions supported by the driver and associated data source. The driver returns a 32-bit unsigned integer that can contain the boolean of one or more of the following values: SQL_FN_STR_ASCII SQL_FN_STR_CHAR SQL_FN_STR_CONCAT SQL_FN_STR_INSERT SQL_FN_STR_LEFT SQL_FN_STR_LTRIM SQL_FN_STR_LENGTH SQL_FN_STR_LOCATE SQL_FN_STR_LCASE SQL_FN_STR_REPEAT SQL_FN_STR_REPLACE SQL_FN_STR_RIGHT SQL_FN_STR_RTRIM SQL_FN_STR_SUBSTRING SQL_FN_STR_UCASE

fInfoType	Returns
SQL_SYNTAX_ COMPATIBILITY	Support for SQL statement sets: 0 = Minimum grammar supported 1 = Core grammar supported
SQL_SYSTEM_FUNCTIONS	The canonical system functions supported by the driver and associated data source. The driver returns a 32-bit unsigned integer that can contain the boolean of one or more of the following values: SQL_FN_SYS_USERNAME SQL_FN_SYS_DBNAME SQL_FN_SYS_IFNULL
SQL_TABLE_TERM	Data source vendor's name for a table; for example, 'table' or 'file'.
SQL_TIMEDATE_ FUNCTIONS	The canonical date and time functions supported by the driver and its associated data source. The driver returns a 32-bit unsigned integer that can contain the boolean of one or more of the following values: SQL_FN_TD_NOW SQL_FN_TD_CURDATE SQL_FN_TD_DAYOFMONTH SQL_FN_TD_DAYOFWEEK SQL_FN_TD_DAYOFYEAR SQL_FN_TD_MONTH SQL_FN_TD_QUARTER SQL_FN_TD_WEEK SQL_FN_TD_YEAR SQL_FN_TD_CURTIME SQL_FN_TD_HOUR SQL_FN_TD_MINUTE SQL_FN_TD_SECOND
SQL_TXN_CAPABLE	Transaction support: Y = The underlying data source supports

	transactions N = The data source does not support transactions
SQL_USER_NAME	Name used in a particular database, which can be different than login name.

The following table lists ODBC data type conversion #defines stored in the SQLEXT.H file.

ODBC Data Type
SQL_CVT_BIGINT
SQL_CVT_BINARY
SQL_CVT_BIT
SQL_CVT_CHAR
SQL_CVT_DATE
SQL_CVT_DECIMAL
SQL_CVT_DOUBLE
SQL_CVT_FLOAT
SQL_CVT_INTEGER
SQL_CVT_LONGVARCHAR
SQL_CVT_NUMERIC
SQL_CVT_REAL
SQL_CVT_SMALLINT
SQL_CVT_TIME

SQL_CVT_TIMESTAMP

SQL_CVT_TINYINT

SQL_CVT_VARBINARY

SQL_CVT_VARCHAR

The following table lists related ODBC functions.

For information about	See
Listing functions supported by the data source	**SQLGetFunctions**
Listing information about data types supported by the data source	**SQLGetTypeInfo**
Listing current settings of options that govern aspects of driver operation	**SQLGetStmtOption, SQLGetConnectOption** (extensions)

SQLGetStmtOption

Extension Level 1

SQLGetStmtOption returns the current setting of a statement option.

Syntax

RETCODE SQLGetStmtOption(hstmt,fOption,pvParam)

XE "SQLGetStmtOption"§The **SQLGetStmtOption** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
UWORD	fOption	Input	Option to retrieve.
PTR	pvParam	Output	Value of option.

Returns

SQL_SUCCESS, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
80	000	General ODBC function
80	001	Memory allocation error
80	009	Invalid argument value
DM	001	Driver does not support this function

Comments

For a list of options, refer to **SQLSetStmtOption**.

The following table lists related ODBC functions.

For information about	See

Retrieving connection options associated with an hdbc	**SQLGetConnectOption**
Setting statement options associated with an hstmt or an hdbc	**SQLSetStmtOption**, **SQLSetConnectOption**

SQLGetTypeInfo

Extension Level 1

XE "SQLGetTypeInfo"§SQLGetTypeInfo returns information about data types supported by the data source that is associated with hstmt. The driver returns the information in the form of an SQL result set.

Syntax

```
RETCODE  SQLGetTypeInfo(hstmt,fSqlType)
```

The **SQLGetTypeInfo** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle for the result set.
SWORD	fSqlType	Input	Data type number, or SQL_ALL_TYPES to describe all types.

Returns

SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use
08	S01	Communication link failure
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value

80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

SQLGetTypeInfo returns the following columns of information for each requested data type.

Column	Type	Contents
TYPE_NAME	Varchar(32)not null	Data source-dependent data type name; for example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINARY", or "CHAR() FOR BIT DATA". The embedded parentheses allow applications to indicate where to insert the value in CREATE_PARAMS into TYPE_NAME. If there are no embedded parentheses, CREATE_PARAMS follows TYPE_NAME.
DATA_TYPE	Smallint, not null	ODBC SQL data type number from the list of all data types (listed in Appendix D, "Data Type Definitions."
LENGTH	Integer, not null	Maximum length allowed for this data type. For more information, refer to Appendix D, "Data Type Definitions."

Column	Type	Contents
LITERAL_PREFIX	Varchar(32)	Character or characters used to prefix a literal; for example, an apostrophe (') for character types or 0x for binary.
LITERAL_SUFFIX	Varchar(32)	Character or characters used to terminate a literal; for example, an apostrophe (') for

		character types, nothing for binary.
CREATE_PARAMS	Varchar(32)	Parameters for this data type, in the form of a string. For example, CREATE_PARAMS for DECIMAL would be "precision, scale"; CREATE_PARAMS for FLOAT, would equal NULL, and CREATE_PARAMS for VARCHAR would equal "max length". The driver supplies the text in CREATE_PARAMS in the language used in the local country (for example, France).
NULLABLE	Smallint not null	Equals 1 if columns of this type can contain NULL values; otherwise, equals 0.
CASE_SENSITIVE	Smallint not null	Equals 1 if columns of this type are treated as case sensitive for collations; otherwise, equals 0.
SEARCHABLE	Smallint not null	1 if the column can be used in a **WHERE** clause with all comparison operators; 0 if only the LIKE predicate can be used. Columns of type SQL_LONG_VARCHAR and SQL_LONG_VARBINARY usually return 0.
UNSIGNED_ ATTRIBUTE	Smallint	Equals 1 if a numeric type is signed, 0 if not.
MONEY	Smallint	Equals 1 if a money data type, 0 if not.
AUTO_INCREMENT	Smallint	Equals 1 if autoincrement; an application can insert values into a column, but cannot update values in the column. Equals 0 if not autoincrement. This attribute is valid only for numeric columns.
LOCAL_TYPE_ NAME	Varchar(32) not null	Localized version of the data source-dependent name. For example, DECIMAL would be DECIMALE in French.

Attribute information can apply to data types or to specific columns in a result set. **SQLGetTypeInfo** returns information about attributes associated with data types; **SQLColAttributes** returns information about attributes associated with columns in a result set.

The following table lists a related ODBC function.

For information about	See
Returning general information about the driver and the data source.	**SQLGetInfo**

SQLMoreResults

Extension Level 2

SQLMoreResults determines whether there are more result sets available from the hstmt and if so, initializes processing for the next result set.

Syntax

RETCODE SQLMoreResults(hstmt)

XE "SQLMoreResults"§The **SQLMoreResults** function accepts the following argument:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.

Returns

SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_INVALID_HANDLE, SQL_ERROR, or SQL_NO_DATA_FOUND.

If more results are available, **SQLMoreResults** returns SQL_SUCCESS. If all results are processed, **SQLMoreResults** returns SQL_NO_DATA_FOUND. If the result of the next statement in a batch failed or an error occurred in the function, **SQLMoreResults** returns SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
24	000	Invalid cursor state
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	010	Function sequence error
80	T00	Timeout expired

DM	001	Driver does not support this function

Comments

Refer to Appendix B, "ODBC State Transition Table," for additional information about the valid sequencing of result-processing functions.

The following table lists related ODBC functions.

For information about	See
Fetching a result row	**SQLFetch**
Fetching a column from a result row	**SQLGetData**

SQLNativeSql

Extension Level 2

SQLNativeSql returns the exact SQL string as translated by the driver. This function is useful if an application uses escape sequences to transmit database-specific text.

Syntax

```
RETCODE  SQLNativeSql(hdbc,szSqlStrIn,cbSqlStrIn,szSqlStr,cbSqlStrMax,pcbSqlStr)
```

XE "SQLNativeSql"§The **SQLNativeSql** function accepts the following arguments:

Type	Argument	Use	Description
HDBC	hdbc	Input	Connection handle.
UCHAR	szSqlStrIn	Input	SQL text string to be translated
SDWORD	cbSqlStrIn	Input	Length of szSqlStrIn text string
UCHAR FAR *	szSqlStr	Output	Translated SQL string.
SDWORD	cbSqlStrMax	Input	Length of szSqlStr buffer.
SDWORD FAR *	pcbSqlStr	Output	Number of bytes available to be written to szSqlStr.

Returns

SQL_SUCCESS, SQL_ERROR, SQL_INVALID_HANDLE, or SQL_SUCCESS_WITH_INFO.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
01	004	Data truncated

08	003	Connection not open
80	000	General ODBC error
80	001	Memory allocation error
80	009	Invalid argument value
DM	001	Driver does not support this function

Comments

If the SQL string does not fit in the output buffer, the driver stores the maximum number of bytes in szSqlStr, sets pcbSqlStr to the number of bytes in the complete SQL string minus one, and returns SQL_SUCCESS_WITH_INFO.

SQLNumParams

Extension Level 2	**SQLNumParams** returns the number of parameters in a SQL statement.
Syntax	RETCODE SQLNumParams(hstmt,pcpar)

XE "SQLNumParams"§The **SQLNumParams** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
SWORD FAR *	pcpar	Output	Number of parameters in the statement.

Returns

SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	010	Function sequence error
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

For a prepared statement, an application can call **SQLNumParams** after it calls **SQLPrepare** and before or after it calls **SQLExecute**. For a direct statement, an application can call **SQLNumParams** after it calls **SQLExecDirect**.

If the statement associated with hstmt does not contain parameters, **SQLNumParams** sets pcpar to zero.

The following table lists related ODBC functions.

For information about	See
Allocating storage for a parameter	**SQLSetParam**

SQLNumResultCols

Core function **SQLNumResultCols** returns the number of columns in a result set.

Syntax RETCODE SQLNumResultCols(hstmt,pccol)

XE "SQLNumResultCols"§The **SQLNumResultCols** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
SWORD FAR *	pccol	Output	Number of columns in the result set.

Returns SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	010	Function sequence error
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

For a prepared statement, an application can call **SQLNumResultCols** after it calls **SQLPrepare** and before or after it calls **SQLExecute**. For a direct statement, an application can call **SQLNumResultCols** after it calls **SQLExecDirect**.

If the statement associated with hstmt does not return columns, **SQLNumResultCols** sets pccol to zero.

The following table lists related ODBC functions.

For information about	See
Returning information (such as size and type) about a result column	**SQLDescribeCol**
Allocating storage for a column in the result set	**SQLBindCol**
Retrieving rows in the result set	**SQLFetch**
Retrieving a column with a large data value	**SQLGetData** (extension)
Retrieving rows using a scrollable cursor	**SQLSetScrollOptions, SQLExtendedFetch** (extension)

SQLParamData

Extension Level 1

SQLParamData is used in conjunction with **SQLPutData** to send large data values to a server. **SQLParamData** performs the following operations:

- Searches for the next large data value parameter in an SQL statement.
- Returns the value referenced by rgbValue (in a call to **SQLSetParam**) for the parameter.

Syntax

```
RETCODE SQLParamData(hstmt,rgbValue)
```

XE "SQLParamData"§The **SQLParamData** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
PTR	rgbValue	Output	The pointer to the address passed for rgbValue in **SQLSetParam**.

Returns

SQL_SUCCESS, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use
08	S01	Communication link failure
24	000	Invalid cursor state
80	000	General ODBC error
80	001	Memory allocation error

80	008	Operation canceled
80	009	Invalid argument value
80	LD0	No long data values pending
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

Once **SQLParamData** starts processing SQL data, it can return any SQLSTATE value that **SQLExecute** or **SQLExecDirect** can return.

SQLParamData returns SQL_SUCCESS to indicate that processing is complete for all parameters in the current row.

The following table lists related ODBC functions.

For information about	See
Executing a statement	**SQLExecute**, **SQLExecDirect**
Sending large data values to a server	**SQLPutData** (extension)
Assigning storage for parameters	**SQLSetParam**

SQLParamOptions

Extension Level 2

SQLParamOptions allows an application to specify multiple values for the set of parameters assigned by **SQLSetParam**. The ability to specify multiple values for a set of parameters is useful for bulk inserts and other work that requires the data source to process the same SQL statement multiple times with various parameter values. An application can, for example, specify three sets of values for the set of parameters associated with an **INSERT** statement, and then execute the **INSERT** statement once to perform the three insert operations.

Syntax

```
RETCODE  SQLParamOptions(hstmt,crow,pirow)
```

The **SQLParamOptions** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
UDWORD	crow	Input	If greater than 1, the rgbValue argument in **SQLSetParam** points to an array of parameter values and pcbValue points to an array of lengths.
UDWORD FAR *	pirow	Output	At execute time, points to the current set of parameters.

Returns

SQL_SUCCESS, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
80	000	General ODBC error
80	001	Memory allocation error

80	009	Invalid argument value
DM	001	Driver does not support this function

Comments

As a statement executes, the driver sets pirow to the index of the current set of parameter values. If an error occurs during execution of a set of parameter values, execution halts at that set of parameter values and **SQLExecute**, **SQLExecDirect**, or **SQLParamData** return SQL_ERROR. The contents of pirow can be used as follows:

- When **SQLParamData** returns for more information, the application can access the value in pirow to determine which set of parameters is being executed.
- When **SQLExecute** or **SQLExecDirect** returns an error, the application can access the value in pirow to find out which set of parameters failed.
- When **SQLExecute**, **SQLExecDirect**, **SQLParamData**, or **SQLPutData** succeed, the value in pirow is set to crow—the total number of sets of parameters processed.

The following table lists a related ODBC function.

For information about	See
Assigning storage for a single parameter marker value in an SQL statement	**SQLSetParam**

SQLPrepare

Core function **SQLPrepare** prepares an SQL string for execution. The data source retains the access plan (if supported by the data source) until the application frees the *hstmt* with a call to **SQLFreeStmt**.

Syntax RETCODE SQLPrepare(hstmt,szSqlStr,cbSqlStr)

XE "SQLPrepare"§The **SQLPrepare** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
UCHAR FAR *	szSqlStr	Input	SQL text string.
SDWORD	cbSqlStr	Input	Length of *szSqlStr*.

Returns SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
01	004	Data truncated
07	000	Dynamic SQL error
07	001	Wrong number of parameters
08	002	Connection in use
08	S01	Communication link failure

22	003	Numeric value out of range
22	005	Error in assignment
22	012	Division by zero
37	000	Syntax error or access violation
40	000	Serialization failure
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	T00	Timeout expired
DM	001	Driver does not support this function
SG	000	Invalid table name
SG	S00	Invalid table name; table not found
SG	S01	Table already exists

SQLSTATE Class	Subclass	Description
SH	000	Invalid view name
SH	S00	Invalid view name; view not found
SH	S01	View already exists
SI	000	Invalid index name
SI	S00	Invalid index name; index not found
SI	S01	Index already exists
SJ	000	Invalid column name
SJ	S00	Invalid column name; column not found
SJ	S01	Column already exists

Comments

SQLPrepare sends an SQL statement to the data source and associates the results with hstmt.

Once a statement is prepared, the application uses hstmt to refer to the statement in later function calls. Once the application prepares a statement, it can request information about the format of the result set.

The application can include one or more parameter markers in the SQL statement. To include a parameter marker, the application embeds a question mark (?) into the SQL string at the appropriate position.

For the driver, hstmt is similar to a statement identifier in embedded SQL code. If the data source supports statement identifiers, the driver can send a statement identifier and parameter values to the data source.

Not all drivers can return syntax errors or access violations when the application calls **SQLPrepare**. A driver may handle syntax errors and access violations, only syntax errors, or neither syntax errors nor access violations. Therefore, an application must be able to handle these conditions when calling subsequent related functions such as **SQLNumResultCols**, **SQLDescribeCol**, **SQLColAttributes**, and **SQLExecute**.

The following table lists related ODBC functions.

For information about	See
Allocating a new statement handle	**SQLAllocStmt**
Setting the name of a cursor	**SQLSetCursorName**
Assigning storage for a column in the result set	**SQLBindCol**
Assigning storage areas for parameters, prior to execution	**SQLSetParam**
Executing a prepared statement	**SQLExecute**
Executing a statement directly	**SQLExecDirect**
Returning name, length, and data type information about a column in the result set	**SQLDescribeCol**
Returning the number of columns in a result set	**SQLNumResultCols**
Returning the number of rows in a result set	**SQLRowCount**

C Header Files

SQLPrimaryKeys

Extension Level 2

SQLPrimaryKeys returns the column name(s) that comprise the primary key for a table. The driver returns the information as a result set.

Syntax

RETCODE SQLPrimaryKeys(hstmt,szTableQualifier,cbTableQualifier,
szTableOwner,cbTableOwner,szTableName,cbTableName)

XE "SQLPrimaryKeys"§The **SQLPrimaryKeys** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle to retrieve results
UCHAR FAR *	szTableQualifier	Input	Qualifier name
SWORD	cbTableQualifier	Input	Length of szTableQualifier
UCHAR FAR *	szTableOwner	Input	String search pattern for owner name
SWORD	cbTableOwner	Input	Length of szTableOwner
UCHAR FAR *	szTableName	Input	String search pattern for table name
SWORD	cbTableName	Input	Length of szTableName

Returns

SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description

08	002	Connection in use
08	S01	Communication link failure
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	C00	Driver not capable
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

The szTableOwner and szTableName arguments constrain the search for table names. Each of these arguments supports a character string search pattern. The search pattern can contain the metacharacters underscore (_) and percent (%), as follows:

- The underscore character represents any single character.
- The percent character represents any sequence of zero or more characters.
- All other characters represent themselves.

For example, an szTableName equal to %A% returns all tables with names that contain the character 'A'. An szTableName equal to B__ ('B' followed by two underscores) returns all tables with names that are three characters long and start with the character 'B'.

If the pattern string contains no underscores or percent characters, then the driver compares characters from left to right. If the pattern is of different length than the string it is being compared to, then the driver equates the two strings by treating the shorter of the two strings as though it has trailing blank characters.

SQLPrimaryKeys returns results as a standard result set. The following table lists result columns.

Column Name	Contents
TABLE_QUALIFIER	Varchar(32)
TABLE_OWNER	Varchar(32)
TABLE_NAME	Varchar(32) not null
COLUMN_NAME	Varchar(32) not null
KEY_SEQ	Smallint not null; the sequence number of the column in a multipart key

The following table lists a related ODBC function.

For information about	See
Returning columns in a foreign key	**SQLForeignKeys**

SQLPutData

Extension Level 1

SQLPutData allows an application to send row data that corresponds to the current large data value parameter as indicated by a call to **SQLParamData**.

Syntax

RETCODE SQLPutData(hstmt,rgbValue,cbValueMax)

XE "SQLPutData"§The **SQLPutData** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
PTR	rgbValue	Input	A pointer to the buffer that contains the actual data for the parameter marker. The data must be in the form specified in the **SQLSetParam** call that the application used when specifying the parameter.
SDWORD	cbValue	Input	Length of rgbValue. Specifies the amount of data sent in a call to **SQLPutData**. The amount of data can vary with each call for a given parameter. The application can also specify SQL_NTS for cbValue.

Returns

SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_ERROR or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use
08	S01	Communication link failure

24	000	Invalid cursor state
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	010	ODBC function sequence error
80	LD0	No long data values pending
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

SQLPutData allows an application to send large data values to the data source. The data is stored in the current row.

SQLPutData works in conjunction with **SQLSetParam**, **SQLParamData**, **SQLExecute**, and **SQLExecDirect**. Prior to calling **SQLPutData**, an application must call **SQLSetParam** to define all parameters and indicate which parameters are associated with large data values.

The application then issues a single call to **SQLExecute** or **SQLExecDirect**. This call initiates execution of the SQL statement.

When **SQLExecute** or **SQLExecDirect** encounter a statement containing a large data value parameter, they return SQL_NEED_DATA. The application then uses **SQLParamData** and **SQLPutData** to process *all* remaining parameters.

Once the driver returns SQL_NEED_DATA, the application calls **SQLParamData** to retrieve a value associated with the next large data value parameter. The application then calls **SQLPutData** to supply data to the parameter. If the data value is larger than the buffer, the application continues to call **SQLPutData** until all data is sent.

When **SQLParamData** returns SQL_SUCCESS, processing is complete for all parameters in the current row.

The following table lists related ODBC functions.

For information about	See
Executing a statement	**SQLExecute**, **SQLExecDirect**
Allocating storage for parameters	**SQLSetParam**

SQLRowCount

Core function **SQLRowCount** returns the number of rows affected by an **UPDATE**, **INSERT**, or **DELETE** statement associated with the specified hstmt.

Syntax RETCODE SQLRowCount(hstmt,pcrow)

XE "SQLRowCount"§The **SQLRowCount** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle for the **UPDATE**, **INSERT**, or **DELETE** statement.
SDWORD FAR *	pcrow	Output	Number of rows affected by the operation.

Returns SQL_SUCCESS, SQL_ERROR or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
80	000	General ODBC error
80	001	Memory allocation error
80	007	Row count not available from the data source
80	009	Invalid argument value
80	010	ODBC function sequence error
DM	001	Driver does not support this function

Comments

SQLRowCount is valid only after an update, insert, or delete operation. If the last operation associated with hstmt was not **SQLExecute** or **SQLExecDirect** with an update, insert, or delete request, **SQLRowCount** sets pcrow to zero and returns SQL_ERROR with SQLSTATE equal to 80 007 (row count not available from the data source).

The following table lists related ODBC functions.

For information about	See
Executing an UPDATE, INSERT, or DELETE statement	**SQLExecute**, **SQLExecDirect**

SQLSetConnectOption

Extension Level 1

SQLSetConnectOption sets options that govern aspects of connections.

Syntax

RETCODE SQLSetConnectOption(hdbc,fOption,vParam)

XE "SQLSetConnectOption"§The **SQLSetConnectOption** function accepts the following arguments:

Type	Argument	Use	Description
HDBC	hdbc	Input	Database connection handle
UWORD	fOption	Input	Option to set, listed under "Comments," below.
UDWORD	vParam	Input	Value of the option

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use
08	003	Connection not open
08	S01	Communication link failure
80	000	General ODBC error
80	001	Memory allocation error

80	009	Invalid argument value
80	C00	Driver not capable
DM	001	Driver does not support this function

Comments

Depending on the option, an application may not need to establish a connection prior to calling **SQLSetConnectOption**. If a connection is required and the application has not established one prior to calling **SQLSetConnectOption**, the driver returns SQL_ERROR with a SQLSTATE value of 08 003 (connection not open).

Certain options, such as SQL_ASYNC_ENABLE, apply to connections as well as statements. When an option applies to a connection and associated statements, the driver sets the option for the hdbc as well as all hstmts associated with the hdbc.

SQLSetConnectOption returns 80 009 (invalid argument value) if hdbc is not allocated.

The driver can return SQL_SUCCESS_WITH_INFO to provide information about the result of setting an option. For example, setting SQL_AUTOCOMMIT during a transaction might cause the transaction to be committed. The driver could use SQL_SUCCESS_WITH_INFO–and information returned with **SQLError–**to inform the application of the commit action.

The currently-defined options are shown below; it is expected that more will be defined to take advantage of different data sources. Option numbers 1-999 are reserved by ODBC; driver writers can reserve values greater than 999 for driver-specific uses.

In addition, an application can call **SQLSetConnectOption** and include a statement option, in which case the driver sets the statement option for all hstmts associated with the specified hdbc. For a list of statement options, refer to **SQLSetStmtOption**.

fOption	Contents
SQL_ACCESS_MODE	SQL_READ_ONLY or SQL_READ_WRITE (the default).
SQL_AUTOCOMMIT	If vParam equals 1, each statement is implicitly committed: 0=Off 1=On (the default)

SQL_CHARSET	Character set used by the application: 1 = ANSI 2 = OEM (IBM PC).
SQL_LOCALE	Language (defined by the application): 1 = Danish 2 = Dutch 3 = English (American) 4 = English (International) 5 = Finnish 6 = French 7 = French Canadian 8 = Icelandic 9 = Italian 10 = Norwegian 11 = Portuguese 12 = Spanish 13 = Swedish
SQL_LOGIN_TIMEOUT	Number of seconds to wait for a login request to complete before returning to the application. The default is 15.
SQL_OPT_TRACE	A BOOLEAN value. If SQL_OPT_TRACE equals true, tracing is enabled. If SQL_OPT_TRACE equals false, tracing is not enabled. False is the default value.
SQL_OPT_TRACEFILE	Name of the trace file. If tracing is enabled, the driver writes to this file each time the application calls a function.
SQL_TXN_ISOLATION	One of the following values: 0 = Changes are immediately perceived by all transactions (Dirty read, nonrepeatable read and phantoms are possible) 1 = Row read by transaction 1 can be altered and committed by transaction 2 (nonrepeatable read and phantoms are possible) 2 = A transaction can add or remove rows matching the

search condition or a pending transaction (phantoms are possible)

3 = Data affected by pending transaction is not available to other transactions

4 = Equivalent of Oracle's Read Consistency isolation

The following table lists related ODBC functions.

For information about	See
Retrieving the current settings of connection options.	**SQLGetConnectOption**
Setting statement options	**SQLSetStmtOption**

SQLSetCursorName

Core function **SQLSetCursorName** associates a cursor name with an active hstmt. This function is optional; if omitted, the driver generates cursors as needed for SQL statement processing.

Syntax

```
RETCODE SQLSetCursorName (hstmt,szCursor,cbCursor)
```

XE "SQLSetCursorName"§The **SQLSetCursorName** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle
UCHAR FAR *	szCursor	Input	Cursor name
SWORD	cbCursor	Input	Length of szCursor

Returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
24	000	Invalid cursor state
34	000	Invalid cursor name
80	000	General ODBC error
80	001	Memory allocation error
80	009	Invalid argument value

DM	001	Driver does not support this function

Comments

The only ODBC operation that accepts a cursor name is a positioned update or delete (for example, "**UPDATE** *table-name* ...**WHERE CURRENT OF** *cursor-name*"). If the application does not supply a cursor name for a positioned update or delete, the driver generates one the first time a SELECT statement is executed on hstmt.

All cursor names within the hdbc must be unique. The maximum length of a cursor name is defined by the driver. For maximum interoperability, it is recommended that applications limit cursor names to 18 characters or less. If the cursor name exceeds the maximum length allowed by the driver or contains invalid characters, **SQLSetCursorName** returns SQLSTATE 34 000 (invalid cursor name).

If szCursor is null, the driver returns SQL_ERROR with SQLSTATE equal to 80 009 (invalid argument value).

The following table lists related ODBC functions.

For information about	See
Executing a prepared statement	**SQLExecute**
Executing a statement that was not previously prepared	**SQLExecDirect**
Setting options for a scrollable cursor	**SQLSetScrollOptions** (extension)
Retrieving a cursor name	**SQLGetCursorName**

SQLSetParam

Core function **SQLSetParam** allows an application to specify storage, data type, length, and storage location associated with a parameter marker in an SQL statement.

Syntax RETCODE SQLSetParam(hstmt,ipar,fCType,fSqlType,cbColDef,ibScale,rgbValue, pcbValue)

XE "SQLSetParam"§The **SQLSetParam** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle associated with the SQL statement.
UWORD	ipar	Input	Parameter number. Numbers indicate position of the parameter within the SQL statement, starting with 1 and increasing from left to right.
SWORD	fCType	Input	Describes the contents of the rgbValue as stored in the C program. fCType must be either a C data type or SQL_NO_CONVERT (if the value in the program is the C equivalent of fSqlType). Appendix D, "Data Type Definitions," describes data type mapping.
SWORD	fSqlType	Input	The data type defined for the column in the data source. fSqlType must be a valid SQL type code (listed in Appendix D, "Data Type Definitions"). This argument tells the driver how to convert the argument into a form acceptable by the data source.
UDWORD	cbColDe f	Input	The maximum length or precision of the column definition for the column or expression referenced; this differs depending on the class of data. (See Appendix D for more information.)
SWORD	ibScale	Input	The total number of digits to the right of the decimal point in the referenced column. This

			argument is valid only when fSqlType is SQL_DECIMAL, SQL_NUMERIC, or SQL_TIMESTAMP.
PTR	rgbValue	Input	A pointer to the location that, at run time, contains the actual data for the parameter marker. The data must be in the form specified by the fCType argument. If the parameter contains a large value and will be sent using **SQLPutData** to send large data values, rgbValue is passed to the application when accessing that parameter. (See "Comments," below, for additional information.) To send multiple values for a parameter, an application can set rgbValue to an array of parameter values. (See "Comments," below, for additional information.)

Type	Argument	Use	Description
SDWORD FAR *	pcbValue	Input	A pointer to a variable that contains the length of the parameter marker value in the value argument. This argument is ignored for numeric data types. If pcbValue equals null, the driver assumes that all parameters are non-null and all character strings are null-terminated. To specify a null parameter value, the application can set pcbValue to SQL_NULL_DATA. If the parameter contains large data values, the application should set pcbValue to SQL_LONG_DATA. (See "Comments," below, for more information.) When sending multiple values for a parameter, pcbValue points to an array of parameter value lengths.

Returns

SQL_SUCCESS, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
80	000	General ODBC error
80	001	Memory allocation error
80	003	fCType argument out of range
80	004	fSqlType argument out of range
80	005	Parameter number out of range

80	006	Invalid conversion
80	009	Invalid argument value
DM	001	Driver does not support this function

Comments

To embed a quote character (') in a character string for a parameter, an application must use two standard quote characters (''). The driver translates the single quotes into the form required by the underlying data source.

Parameter values for prepared statements remain in effect until the application calls **SQLSetParam** again or until the application calls **SQLFreeStmt** with the DROP or RESET_PARAMS option.

Parameter Data Types

Even though each parameter specified with **SQLSetParam** is defined using an SQL data type, the parameters in an SQL statement have no intrinsic data type. Therefore, parameter markers can be included in an SQL statement only if their data types can be inferred from another operand in the statement. For example, in an arithmetic expression such as '? + COLUMN1', the data type of the parameter can be inferred from the data type of the named column represented by COLUMN1. An application cannot use a parameter marker if the data type cannot be determined.

The following table describes how a data type is determined for several types of parameters.

Location of Parameter	Assumed Data Type
One operand of a binary arithmetic or comparison operator	Same as the other operand
The first operand in a BETWEEN clause	Same as the other operand
The second or third operand in a BETWEEN clause	Same as the first operand
An expression used with IN	Same as the first value or the result column of the subquery
A value used with IN	Same as the expression
A pattern value used with LIKE	VARCHAR
An update value used with UPDATE	Same as the update column

If fCType is SQL_C_CHAR and fSqlType is a numeric type, the driver converts rgbValue to the type specified by fSqlType. This conversion is performed within the driver; the value in rgbValue remains unchanged. Character strings must follow the formats specified in Appendix D, "Data Type Definitions."

Parameter Markers

An application cannot place parameter markers in the following locations:

- In a **SELECT** list.
- As both expressions in a comparison-predicate. (For a definition of these terms, refer to Appendix C, "SQL Grammar."
- As both operands of a binary operator.
- As both the first and second operands of a **BETWEEN** operation.

- As both the first and third operands of a **BETWEEN** operation.
- As both the expression and the first value of an **IN** operation.
- As the operand of a unary + or - operation.
- As the argument of a **SET** function.

For more information, refer to the ANSI SQL2 specification.

If an application includes parameter markers in the SQL statement, the application must call **SQLSetParam** to associate storage locations with parameter markers before it calls **SQLExecute** or **SQLExecDirect**. If the application calls **SQLPrepare**, the application can call **SQLSetParam** before or after it calls **SQLPrepare**.

The application can set parameter markers in any order. The driver buffers argument descriptors and sends the current values referenced by rgbValue for the associated parameter marker when the application calls **SQLExecute** or **SQLExecDirect**. It is the application's responsibility to ensure that all pointer arguments are valid at execution time.

Parameters with Large Data Values

An application can interleave large data parameters and other parameters. To specify a parameter for use with large data values, the application sets pcbValue to SQL_LONG_DATA. This feature is considered extended functionality.

To send large data values, the application calls **SQLParamData** and **SQLPutData**. The application can use rgbValue to associate an application-defined long data value with the parameter. Not all drivers support these functions; an application can call **SQLGetFunctions** to determine if the **SQLPutData** and **SQLParamData** functions are supported.

SQLExecute and, **SQLExecDirect** return SQL_NEED_DATA to indicate the need for large data values. An application then calls **SQLParamData** to begin processing the large data values in the row. **SQLParamData** returns the value referenced by rgbValue for each parameter. **SQLPutData** sends the data to the server.

Multiple Values for a Parameter

If the application calls **SQLParamOptions** to specify multiple values for a set of parameters, the application must use rgbValue to point to an array of parameter values. These values are processed with a single SQL statement. The application must also set pcbValue to point to an array of parameter value lengths. (This is considered extended functionality and is not supported by all drivers.)

The following table lists related ODBC functions.

For information about	See
Executing a statement	**SQLExecute**, **SQLExecDirect**

Retrieving the number of parameters in an SQL statement	**SQLNumParams** (extension)
Retrieving result rows	**SQLFetch**
Sending large data values	**SQLPutData** (extension) **SQLParamData** (extension)
Specifying multiple values for a parameter	**SQLParamOptions** (extension)

SQLSetPos

Extension Level 2

XE "SQLSetPos"§The **SQLSetPos** function positions a cursor within a fetched block of data.

Syntax

RETCODE SQLSetPos(hstmt,irow,fRefresh,fLock)

The **SQLSetPos** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
UWORD	irow	Input	Absolute position of the cursor within the retrieved block of data. If irow equals SQL_ENTIRE_ROWSET, **SQLSetPos** sets the cursor to the entire rowset retrieved by **SQLExtendedFetch**. Subsequent positioned operations affect values in the entire rowset. Otherwise, irow must be a value from 1 to the number of rows in the rowset. For additional information about rowset size, refer to **SQLSetScrollOptions**.
BOOL	fRefresh	Input	Specifies whether to refresh the buffer value for the positioned row. If TRUE, the driver refreshes the buffer value. If FALSE, the driver does not change the buffered value.
BOOL	fLock	Input	Specifies whether to lock the row for a subsequent update operation. If TRUE, the driver requests a lock for the row. If FALSE, the driver applies the form of concurrency control specified in a call to **SQLSetScrollOptions**.

Returns

SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
24	000	Invalid cursor state
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	010	Function sequence error
80	C00	Driver not capable
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

Before calling **SQLSetPos**, an application must call **SQLExtendedFetch** to retrieve a block of data. Locking works as follows:

- To guarantee that a row will not change, an application sets fRefresh to TRUE and fLock to TRUE.
- If the application does not call **SQLSetPos** to lock the rows, the driver guarantees a positioned update or delete operation only if the application specifies SQL_LOCK in a call to **SQLSetScrollOptions**.
- If the application requests SQL_OPT_TIMESTAMP or SQL_OPT_VALUES in a call to **SQLSetScrollOptions**, the driver compares timestamps or values and rejects the operation if the row has changed since the application fetched

the row.

- If the application requests SQL_READ_ONLY and did not lock the rows, the driver rejects any positioned update or delete operation.

The following table lists related ODBC functions.

For information about	See
Fetching a block of data from a result set	**SQLExtendedFetch**
Setting the number of rows received by **SQLExtendedFetch**	**SQLSetScrollOptions**

SQLSetScrollOptions

Extension Level 2

XE "SQLSetScrollOptions"§SQLSetScrollOptions sets options that control the behavior of cursors associated with an hstmt.

Syntax

RETCODE SQLSetScrollOptions(hstmt,fConcurrency,crowKeyset,crowRowset)

The **SQLSetScrollOptions** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle.
UWORD	fConcurrency	Input	Specifies concurrency control for the cursor and must be one of the values listed under "Comments," below.
SDWORD	crowKeyset	Input	Number of rows for which to buffer keys. This value must be greater than or equal to crowRowset or one of the following #defines: SQL_SCROLL_FORWARD_ONLY: the cursor behaves as a forward only scrolling cursor. The driver does not store any keys. The application cannot call **SQLSetPos** for the rowset. The only valid concurrency options are SQL_READ_ONLY and SQL_LOCK. Positioned operations can be performed only if crowRowset equals 1. SQL_SCROLL_KEYSET_DRIVEN: the driver keeps the key for every row retrieved. SQL_SCROLL_DYNAMIC: the driver sets crowKeyset to the value of crowRowset. If crowKeyset is a value greater than crowRowset, the value defines the number of rows in the keyset that are to be buffered by the driver. This reflects a mixed scrollable cursor; the cursor is keyset driven within the keyset and dynamic outside of the keyset.

UWORD	crowRowset	Input	Number of rows in a rowset. crowRowset defines the number of rows fetched by each call to **SQLExtendedFetch**; the number of rows that the application buffers.

Returns SQL_SUCCESS, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use
24	000	Invalid cursor state
80	000	General ODBC error
80	001	Memory allocation error
80	009	Invalid argument value
80	C00	Driver not capable
DM	001	Driver does not support this function

Comments

If an application calls **SQLSetScrollOptions**, the application must call **SQLSetScrollOptions** before it calls **SQLPrepare** or **SQLExecDirect**.

The application must specify a buffer in a call to **SQLBindCol** that is large enough to hold the number of rows specified in crowRowset.

If the application does not call **SQLSetScrollOptions**, crowRowset has a default value of 1, crowKeyset has a default value of 0, and fConcurrency equals SQL_READ_ONLY.

The following table lists valid concurrency control options.

Value	Description
SQL_READ_ONLY	Cursor is read only. No updates are allowed.
SQL_LOCK	Cursor uses intent-to-update locks.
SQL_OPT_TIMESTA	Cursor uses optimistic concurrency control, comparing

MP	timestamps.
SQL_OPT_VALUES	Cursor uses optimistic concurrency control, comparing values.

The following table lists related ODBC functions.

For information about	See
Setting the cursor position within a fetched array	**SQLSetPos**
Fetching one rowset of data from the result set	**SQLExtendedFetch**
Defining storage for a column in a result set	**SQLBindCol**

SQLSetStmtOption

Extension Level 1

SQLSetStmtOption sets options related to an hstmt. To set an option for all statements associated with a specific hdbc, an application can call **SQLSetConnectOption**.

Syntax

RETCODE SQLSetStmtOption(hstmt,fOption,vParam)

XE "SQLSetStmtOption"§The **SQLSetStmtOption** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle
UWORD	fOption	Input	Option to set, listed under "Comments," below.
UDWORD	vParam	Input	Value of the option

Returns

SQL_SUCCESS, SQL_INVALID_HANDLE, or SQL_ERROR.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use
08	S01	Communication link failure
80	000	General ODBC error
80	001	Memory allocation error
80	009	Invalid argument value

80	C00	Driver not capable
DM	001	Driver does not support this function

Comments

The following functions can be run asynchronously.

SQLColAttributes	**SQLExecDirect**	**SQLGetTypeInfo**	**SQLSetPos**
SQLColumns	**SQLExecute**	**SQLMoreResults**	**SQLSpecialColumns**
SQLColumnPrivileges	**SQLExtendedFetch**	**SQLNumResultCols**	**SQLStatistics**
SQLConnect	**SQLFetch**	**SQLParamData**	**SQLTablePrivileges**
SQLDescribeCol	**SQLForeignKeys**	**SQLPrepare**	**SQLTables**
SQLDescribeParam	**SQLGetData**	**SQLPrimaryKeys**	
SQLDriverConnect	**SQLGetInfo**	**SQLPutData**	

The currently-defined options are shown below; it is expected that more will be defined to take advantage of different data sources. Option numbers 1-999 are reserved by ODBC; driver writers can reserve values greater than 999 for driver-specific uses.

fOption	Contents
SQL_ASYNC_ENABLE	If vParam equals 1, the driver enables asynchronous behavior for function calls associated with the specified connection handle: 0=Off (the default) 1=On
SQL_HSTMT_USER_DATA	A four-byte data buffer associated with the hstmt.
SQL_MAX_LENGTH	Maximum amount of data returned with each call to **SQLGetData** or **SQLFetch** for columns with variable length data types (for example, VARCHAR, LONGVARCHAR, VARBINARY, and LONG VARBINARY).
SQL_NOSCAN	If TRUE, the driver does not scan SQL strings for escape clauses. Instead, the driver sends the statement directly to the data source. If FALSE (the default), the driver scans for escape clauses.
SQL_QUERY_TIMEOUT	Number of seconds to wait for an SQL statement to execute before returning to the application. If vParam equals 0 (the default), then there is no time out.
SQL_ROWCOUNT	Maximum number of rows to return to the application for a SELECT statement. If vParam equals 0 (the default), then the driver returns all rows.

The following table lists related ODBC functions.

For information about	See
Retrieving the current setting of a	**SQLGetConnectOption**

connection option.

Setting connection options or setting statement options for all statements associated with a specific connection.	**SQLSetConnectOption**
Retrieving the setting of a statement option.	**SQLGetStmtOption**

SQLSpecialColumns

Extension Level 1

SQLSpecialColumns retrieves the following information about columns within a specified table:

- The optimal set of columns that uniquely identifies a row in the table
- Columns that are automatically updated when any value in the row is updated by a transaction

Syntax

```
RETCODE  SQLSpecialColumns(hstmt,fColType,szTableQualifier,cbTableQualifier,szTableOwner,
    cbTableOwner,szTableName,cbTableName,fScope,fNullable)
```

XE "SQLSpecialColumns"§The **SQLSpecialColumns** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle for retrieving results.
UWORD	fColType	Input	Type of column to return: SQL_BEST_ROWID or SQL_ROWVER (described in "Comments," below).
UCHAR FAR *	szTableQualifier	Input	Qualifier name for the table.
SWORD	cbTableQualifier	Input	Length of szTableQualifier.
UCHAR FAR *	szTableOwner	Input	Owner name for the table.
SWORD	cbTableOwner	Input	Length of szTableOwner.
UCHAR FAR *	szTableName	Input	A single table name.

SWORD	cbTableName	Input	Length of szTableName.
UWORD	fScope	Input	Scope of the rowid: SQL_SCOPE_CURROW, SQL_SCOPE_TRANSACTION, or SQL_SCOPE_SESSION. (described in "Comments," below).
UWORD	fNullable	Input	Determines whether or not to return special columns that can have a NULL value. To exclude special columns that can have null values, set fNullable to SQL_NO_NULLS. To return special columns even if they can have null values, set fNullable to SQL_NULLABLE.

Returns

SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use
08	S01	Communication link failure
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	C00	Driver not capable
DM	001	Driver does not support this function

Comments

SQLSpecialColumns is provided so that applications can provide their own custom scrollable cursor functionality. The functionality is similar to that provided by **SQLExtendedFetch** and **SQLSetScrollOptions**.

The following table lists valid values for the fColType argument.

fColType	Description
SQL_BEST_ROWID	Returns the optimal column or set of columns that, by retrieving values from the column or columns, allows any row in the specified table to be uniquely identified. A column can be either a pseudocolumn specifically designed for this purpose (as in Oracle ROWID or Ingres TID) or the column or columns of any unique index for the table.

SQL_ROWVER	Returns the column or columns in the specified table, if any, that are automatically updated by the data source when any value in the row is updated by any transaction (as in SQLBase ROWID or Sybase TIMESTAMP).

The following table lists values for the fScope argument.

#define name	Description
SQL_SCOPE_CURROW	The ROWID is guaranteed to be valid only while positioned on that row. A later reselect using ROWID may not return a row if the row was updated or deleted by another transaction.
SQL_SCOPE_TRANSACTION	The ROWID is guaranteed to be valid for the duration of the current transaction.
SQL_SCOPE_SESSION	The ROWID is guaranteed to be valid for the duration of the session (across transaction boundaries).

SQLSpecialColumns returns results as a standard result set. The following table lists result columns.

Column Name	Data Type
SCOPE	SQL_SCOPE_CURROW, SQL_SCOPE_TRANSACTION, or SQL_SCOPE_SESSION.
COLUMN_NAME	Varchar(32) not null
DATA_TYPE	Smallint not null
TYPE_NAME	Varchar(32) not null (same as returned by **SQLGetTypeInfo**)
DATA_PRECISION	Int
LENGTH	Int not null
NUMERIC_SCALE	Smallint

Once the application retrieves values for SQL_BEST_ROWID, the application can use these values to reselect that row within the defined scope. The **SELECT** statement is guaranteed to return either no rows or one row.

If an application reselects a row based on the ROWID column or columns and the row is not found, then the application can assume that the row was deleted or updated. The opposite is not true: even if the ROWID has not changed, the other columns in the row may have changed.

Columns returned for type SQL_BEST_ROWID are useful for applications that need to scroll forwards and backwards within a result set to retrieve the most recent data from a set of rows. The column or columns of the ROWID are guaranteed not to change while positioned on that row.

The column or columns of the ROWID may remain valid even when the cursor is not positioned on the row; the application can determine this by checking the SCOPE column in the result set.

Columns returned for type SQL_ROWVER are useful for applications that need the ability to check if any columns in a given row have been updated while the row was reselected using the ROWID. For example, after reselecting a row using ROWID, the application can compare the previous ROWVER value to the one just fetched. If the row differs from the previous SQL_ROWVER value, the application can alert the user that data on the display has changed.

The following table lists related ODBC functions.

For information about	See
Retrieving a list of columns for the specified table	**SQLColumns**
Fetching a block of data from the result set	**SQLExtendedFetch**
Setting the number of rows received by **SQLExtendedFetch**	**SQLScrollOptions**

SQLStatistics

Extension Level 1

XE "SQLStatistics"§SQLStatistics retrieves a list of statistics about a single table and the indexes associated with the table. The driver returns the information as a result set.

Syntax

```
RETCODE  SQLStatistics(hstmt,szTableQualifier,cbTableQualifier,szTableOwner,
    cbTableOwner,szTableName,cbTableName,fUnique,fAccuracy)
```

The **SQLStatistics** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle for retrieving results.
UCHAR FAR *	szTableQualifier	Input	Qualifier name.
SWORD	cbTableQualifier	Input	Length of szTableQualifier.
UCHAR FAR *	szTableOwner	Input	Owner name.
SWORD	cbTableOwner	Input	Length of szTableOwner.
UCHAR FAR *	szTableName	Input	Table name.
SWORD	cbTableName	Input	Length of szTableName.
UWORD	fUnique	Input	Type of index: SQL_UNIQUE or SQL_NON_UNIQUE.
UWORD	fAccuracy	Input	The importance of the CARDINALITY and PAGES columns in the result set:

SQL_ENSURE requests that the driver unconditionally retrieve the statistics.

SQL_QUICK requests that the driver retrieve results only if they are readily available from the server. In this case, the driver does not ensure that the values are current.

Returns

SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use
08	S01	Communication link failure
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	C00	Driver not capable
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

SQLStatistics returns information about a single table. Results are returned as a standard result set. The result table combines statistics information for the table and each index.

The following table lists result columns; the driver sorts the result table in the following order:

- NON_UNIQUE
- TYPE
- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- INDEX_QUALIFIER
- INDEX_NAME
- SEQ_IN_INDEX

Column Name	Data Type
TABLE_QUALIFIER	Varchar(32)
TABLE_OWNER	Varchar(32)
TABLE_NAME	Varchar(32) not null
NON_UNIQUE	Smallint not null for index; 0=UNIQUE 1=NONUNIQUE
INDEX_QUALIFIER	Varchar(32)
INDEX_NAME	Varchar(32) not null for index

TYPE	Smallint not null; 0=statistic for table 1=clustered index 2=hashed index 3=other index
SEQ_IN_INDEX	Smallint not null for index; starting at 1
COLUMN_NAME	Varchar(32) not null for index
COLLATION	CHAR(1) not null for index; A=ASCENDING D=DESCENDING
CARDINALITY	Int; number of rows in the table or unique values in the index
PAGES	Int; number of pages used to store the index or table

If the row in the result set corresponds to a table, the driver sets TYPE to 0 and sets NON_UNIQUE, INDEX_QUALIFIER, INDEX_NAME, SEQ_IN_INDEX, COLUMN_NAME, and COLLATION to null. If CARDINALITY or PAGES are not available from the data source, the driver sets them to NULL.

The following table lists related ODBC functions.

For information about	See
Returning the list of columns that comprise a primary key	**SQLPrimaryKeys**
Returning the list of columns that comprise a foreign key	**SQLForeignKeys**

SQLTablePrivileges

Extension Level 2

SQLTablePrivileges returns a list of tables and the privileges associated with each table. The driver returns the information as a result set on the specified *hstmt*.

Syntax

```
RETCODE  SQLTablePrivileges(hstmt,szTableQualifier,cbTableQualifier,szTableOwner,
    cbTableOwner,szTableName,cbTableName)
```

XE "SQLTablePrivileges"§The **SQLTablePrivileges** function accepts the following arguments.

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle for retrieving results.
UCHAR FAR *	szTableQualifier	Input	Table qualifier.
SWORD	cbTableQualifier	Input	Length of szTableQualifier.
UCHAR FAR *	szTableOwner	Input	String search pattern for owner names.
SWORD	cbTableOwner	Input	Length of szTableOwner.
UCHAR FAR *	szTableName	Input	String search pattern for table names.
SWORD	cbTableName	Input	Length of szTableName.

Returns

SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description

08	002	Connection in use
08	S01	Communication link failure
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	C00	Driver not capable
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

The szTableOwner and szTableName arguments constrain the search for table names. Each of these arguments supports a character string search pattern. The search pattern can contain the metacharacters underscore (_) and percent (%), as follows:

- The underscore character represents any single character.
- The percent character represents any sequence of zero or more characters.
- All other characters represent themselves.

For example, an szTableName equal to %A% returns all tables with names that contain the character 'A'. An szTableName equal to B__ ('B' followed by two underscores) returns all tables with names that are three characters long and start with the character 'B'.

If the pattern string contains no underscores or percent characters, then the driver compares characters from left to right. If the pattern is of different length than the string it is being compared to, then the driver equates the two strings by treating the shorter of the two strings as though it has trailing blank characters.

SQLTablePrivileges returns the results as a standard result set. The following table lists result columns. For each of the privileges in the result set, a 0 indicates that the privilege is not granted for any column in the table, a 1 indicates that the privilege is granted for at least one column in the table, and a 2 indicates that the privilege is granted for all columns in the table.

For each of the grantable values in the result set, a 0 indicates that the privilege is not granted for any column in the table, a 1 indicates that the privilege is granted for at least one column in the table, a 2 indicates that the privilege is grantable for all columns.

If REFERENCES_PRIVILEGE and REFERENCES_GRANTABLE equal null, the privilege is not supported.

Column Name	Data Type
TABLE_QUALIFIER	Varchar(32)
TABLE_OWNER	Varchar(32)
TABLE_NAME	Varchar(32) not null
GRANTOR	Varchar(32) not null
GRANTEE	Varchar(32) not null
SELECT_PRIVILEGE	Smallint not null
SELECT_GRANTABLE	Smallint not null
INSERT_PRIVILEGE	Smallint not null
INSERT_GRANTABLE	Smallint not null
UPDATE_PRIVILEGE	Smallint not null
UPDATE_GRANTABLE	Smallint not null

DELETE_PRIVILEGE	Smallint not null
DELETE_GRANTABLE	Smallint not null
REFERENCES_PRIVILEGE	Int
REFERENCES_GRANTABLE	Int

The following table lists a related ODBC function.

For information about	See
Returning privileges for a specified column or columns	**SQLColumnPrivileges**

SQLTables

Extension Level 1

SQLTables returns the list of table names stored in a specific data source. The driver returns the information as a result set.

Syntax

RETCODE SQLTables(hstmt,szTableQualifier,cbTableQualifier,szTableOwner, cbTableOwner,szTableName,cbTableName,szTableType,cbTableType)

XE "SQLTables"§The **SQLTables** function accepts the following arguments:

Type	Argument	Use	Description
HSTMT	hstmt	Input	Statement handle for retrieved results.
UCHAR FAR *	szTableQualifier	Input	Table qualifier.
SWORD	cbTableQualifier	Input	Length of szTableQualifier.
UCHAR FAR *	szTableOwner	Input	String search pattern for owner names.
SWORD	cbTableOwner	Input	Length of szTableOwner.
UCHAR FAR *	szTableName	Input	String search pattern for table names.
SWORD	cbTableName	Input	Length of szTableName.
UCHAR FAR *	szTableType	Input	List of table types to match.
SWORD	cbTableType	Input	Length of szTableType.

Returns

SQL_SUCCESS, SQL_STILL_EXECUTING, SQL_ERROR or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	002	Connection in use
08	S01	Communication link failure
80	000	General ODBC error
80	001	Memory allocation error
80	008	Operation canceled
80	009	Invalid argument value
80	C00	Driver not capable
80	T00	Timeout expired
DM	001	Driver does not support this function

Comments

SQLTables lists all tables in the requested range. A user may or may not have **SELECT** privileges to any of these tables. To check accessibility, an application can:

- Call **SQLGetInfo** and check the SQL_ACCESSIBLE_TABLES result column.

  or

- Call **SQLTablePrivileges** to check the privileges for each table.

Otherwise, the application must be able to handle a situation where the user selects a table for which SELECT privileges are not granted.

The szTableOwner and szTableName arguments constrain the search for table names. Each of these arguments supports a character string search pattern. The search pattern can contain the metacharacters underscore (_) and percent (%), as follows:

- The underscore character represents any single character.
- The percent character represents any sequence of zero or more characters.
- All other characters represent themselves.

For example, an szTableName equal to %A% returns all tables with names that contain the character 'A'. An szTableName equal to B__ ('B' followed by two underscores) returns all tables with names that are three characters long and start with the character 'B'.

If the pattern string contains no underscores or percent characters, then the driver compares characters from left to right. If the pattern is of different length than the string it is being compared to, then the driver equates the two strings by treating the shorter of the two strings as though it has trailing blank characters.

If szTableType is not NULL, it must contain a list of comma-separated values for the types of interest (for example, "'TABLE' and 'VIEW'").

SQLTables returns results as a standard result set. The following table lists result columns.

Column Name	Data Type
TABLE_QUALIFIER	Varchar(32)
TABLE_OWNER	Varchar(32)
TABLE_NAME	Varchar(32) not null
TABLE_TYPE	Varchar(32) not null; one of {'Table', 'View', 'System Table','Alias', 'Synonym'}. Other values are DBMS specific
REMARKS	Varchar(254)

The following table lists related ODBC functions.

For information about	See

Returning a list of columns for a specified table or tables	**SQLColumns**
Returning privileges for a table	**SQLTablePrivileges**
Determining whether or not the user has access to all tables in the result set	**SQLGetInfo**

SQLTransact

Core function **SQLTransact** requests a commit or rollback operation for all update, insert, and delete transactions in progress on all hstmts associated with the connection.

Syntax

```
RETCODE SQLTransact(hdbc,fType)
```

XE "SQLTransact"§The **SQLTransact** function accepts the following arguments.

Type	Argument	Use	Description
HDBC	hdbc	Input	Connection handle associated with the transaction.
UWORD	fType	Input	One of the following two values: SQL_COMMIT SQL_ROLLBACK (These values are defined in the SQL.H file, listed in Appendix F.)

Returns

SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table lists possible SQLSTATE values.

SQLSTATE Class	Subclass	Description
08	003	Connection not open
80	000	General ODBC error
80	001	Memory allocation error
80	009	Invalid argument value
80	012	Invalid transaction operation code specified

80	C00	Driver not capable
DM	001	Driver does not support this function

Comments

SQLTransact operates on all hstmts associated with the hdbc. If fType equals SQL_COMMIT, **SQLTransact** issues a commit request for all active operations on all hstmts associated with the hdbc. If fType equals SQL_ROLLBACK, **SQLTransact** issues a rollback request for all active operations on all hstmts associated with the hdbc.

The rollback or commit request starts at connection or at the last call to **SQLTransact**, whichever is later.

An application can check SQL_CURSOR_ROLLBACK_BEHAVIOR and SQL_CURSOR_COMMIT_BEHAVIOR with **SQLGetInfo** to determine how these operations affect cursors.

If the cursor reset value equals zero, **SQLTransact** closes and deletes all open cursors associated with the hdbc and discards all pending results. **SQLTransact** leaves hstmts in an allocated state; the application can reuse them for subsequent SQL requests or can call **SQLFreeStmt** to deallocate them.

If the cursor reset value equals one, **SQLTransact** closes all open cursors associated with the hdbc. Cursors are positioned at the first row of the corresponding result set. **SQLTransact** leaves hstmts in an prepared state; the application can use the hstmts to call **SQLExecute** without first calling **SQLPrepare**.

If the cursor reset value equals two, **SQLTransact** does not affect open cursors associated with the hdbc. Cursors remain at the row they pointed to prior to the call to **SQLTransact**. **SQLTransact** leaves hstmts in a prepared state; the application can use the hstmts to call **SQLExecute** without first calling **SQLPrepare**, or can use the hstmts to call **SQLFetch** without first calling **SQLExecute**.

SQLTransact does not support two-phase commit operations.

The following table lists a related ODBC function.

For information about	See
Closing cursors	**SQLFreeStmt**

Appendix A ODBC Error Codes

XE "Errors:list"§**SQLError** returns SQLSTATE values as defined by the X/Open and SQL Access Group SQL CAE draft specification (1991). SQLSTATE values are strings that contain five characters. The strings are divided into two components, class and subclass. The following table lists class and subclass values that a driver can return for **SQLError**.

Class	Subclass	Description	**Can be returned from**
00	000	Success	**SQLError**
01	002	Disconnect error: Transaction rolled back.	**SQLDisconnect**
01	004	Data truncated.	**SQLDataSources** **SQLDescribeCol** **SQLDescribeParam** **SQLExtendedFetch** **SQLFetch** **SQLGetCursorName** **SQLGetData** **SQLGetInfo** **SQLNativeSql** **SQLPrepare**
07	000	Dynamic SQL error.	**SQLExecDirect** **SQLExecute** **SQLPrepare**
07	001	Wrong number of parameters.	**SQLExecDirect** **SQLExecute** **SQLPrepare**
07	002	Number of bound columns doesn't	**SQLExtendedFetch** **SQLFetch**

		match.	
08	001	Unable to connect to database server.	**SQLConnect** **SQLDriverConnect**

Class	Subclass	Description	**Can be obtained from**
08	002	Connection in use.	**SQLColAttributes** **SQLColumnPrivileges** **SQLColumns** **SQLConnect** **SQLDriverConnect** **SQLExecDirect** **SQLExecute** **SQLForeignKeys** **SQLFreeConnect** **SQLFreeEnv** **SQLGetInfo** **SQLGetTypeInfo** **SQLParamData** **SQLPrepare** **SQLPrimaryKeys** **SQLPutData** **SQLSetConnectOption** **SQLSetScrollOptions** **SQLSetStmtOption** **SQLSpecialColumns** **SQLStatistics** **SQLTablePrivileges** **SQLTables**
08	003	Connection not open.	**SQLAllocStmt** **SQLDisconnect** **SQLGetConnectOption** **SQLGetData** **SQLGetInfo** **SQLNativeSql** **SQLSetConnectOption** **SQLTransact**

Class	Subclass	Description	**Can be obtained from**
08	S01	Communication link failure	**SQLColAttributes** **SQLColumnPrivileges** **SQLColumns** **SQLConnect**

			SQLDriverConnect **SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLForeignKeys** **SQLFreeConnect** **SQLGetData** **SQLGetTypeInfo** **SQLParamData** **SQLPrepare** **SQLPrimaryKeys** **SQLPutData** **SQLSetConnectOption** **SQLSpecialColumns** **SQLStatistics** **SQLSetStmtOption** **SQLTables** **SQLTablePrivileges**
21	000	Cardinality violation.	**SQLExecDirect** **SQLExecute**
21	S01	Insert value list does not match column list.	**SQLExecDirect** **SQLExecute**
21	S02	Degree of derived table does not match column list.	**SQLExecDirect** **SQLExecute**
22	001	String data right truncation	**SQLExecDirect,** **SQLExecute**
22	003	Numeric value out of range.	**SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLPrepare**

Class	Subclass	Description	**Can be obtained from**
22	005	Error in assignment.	**SQLExecDirect** **SQLExecute** **SQLGetData** **SQLPrepare**
22	012	Division by zero.	**SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLPrepare**
23	000	Integrity constraint violation.	**SQLExecDirect** **SQLExecute**
24	000	Invalid cursor state.	**SQLColAttributes** **SQLExtendedFetch** **SQLFetch** **SQLGetData** **SQLMoreResults** **SQLParamData** **SQLPutData** **SQLSetCursorName** **SQLSetPos** **SQLSetScrollOptions**
28	000	Invalid user authorization specification.	**SQLConnect** **SQLDriverConnect**
34	000	Invalid cursor name.	**SQLSetCursorName**
37	000	Syntax error or access violation.	**SQLExecDirect** **SQLExecute** **SQLPrepare**
40	000	Serialization failure.	**SQLExecDirect**

			SQLExecute **SQLExtendedFetch** **SQLFetch** **SQLPrepare**
70	100	Server did not process the cancel request.	**SQLCancel**
80	000	General error.	All ODBC functions except: **SQLAllocEnv,** **SQLError**
80	001	Memory allocation error.	All ODBC functions except: **SQLError** **SQLFreeConnect** **SQLFreeEnv**
80	002	Invalid column number.	**SQLBindCol** **SQLColAttributes** **SQLDescribeCol** **SQLGetData**

Class	Subclass	Description	**Can be obtained from**
80	003	fCType argument out of range.	**SQLBindCol** **SQLSetParam**
80	004	fSqlType argument out of range.	**SQLSetParam**
80	005	Parameter number out of range.	**SQLDescribeParam** **SQLSetParam**
80	006	Invalid conversion specified.	**SQLFetch** **SQLExtendedFetch** **SQLGetData**

			SQLSetParam
80	007	Row count not available from the data source.	**SQLRowCount**
80	008	Operation canceled.	All ODBC functions that can be processed asynchronously. (Refer to Chapter 7 for a list.)
80	009	Invalid argument value.	All ODBC functions except: **SQLAllocEnv** **SQLCancel** **SQLDisconnect** **SQLError** **SQLFetch** **SQLFreeConnect** **SQLFreeEnv**
80	010	Function sequence error.	**SQLColAttributes** **SQLDataSources** **SQLDescribeCol** **SQLDescribeParam** **SQLExtendedFetch** **SQLFetch** **SQLFreeEnv** **SQLGetData** **SQLMoreResults** **SQLNumParams** **SQLNumResultCols** **SQLParamData** **SQLPutData** **SQLRowCount** **SQLSetPos**
80	012	Invalid transaction operation code specified.	**SQLTransact**
80	015	No cursor name available.	**SQLGetCursorName**

Class	Subclass	Description	**Can be obtained from**
80	C00	Driver not capable.	**SQLBindCol** **SQLColumnPrivileges** **SQLColumns** **SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLForeignKeys** **SQLGetInfo** **SQLPrimaryKeys** **SQLSetConnectOption** **SQLSetPos** **SQLSetScrollOptions** **SQLSetStmtOption** **SQLSpecialColumns** **SQLStatistics** **SQLTablePrivileges** **SQLTables** **SQLTransact**
80	LD0	No long data values pending	**SQLParamData** **SQLPutData**
80	T00	Timeout expired.	**SQLColumnPrivileges** **SQLColumns** **SQLConnect** **SQLDescribeCol** **SQLDescribeParam** **SQLDriverConnect** **SQLExecDirect** **SQLExecute** **SQLExtendedFetch** **SQLFetch** **SQLForeignKeys** **SQLGetData** **SQLGetInfo** **SQLGetTypeInfo** **SQLMoreResults** **SQLNumParams** **SQLNumResultCols** **SQLParamData**

Class	Subclass	Description	**Can be obtained from**
DM	001	Driver does not support this function.	All ODBC functions except: **SQLAllocConnect** **SQLAllocEnv** **SQLDataSources** **SQLError** **SQLFreeConnect** **SQLFreeEnv** **SQLGetFunctions**
DM	002	Data source name not found and no default driver specified.	**SQLConnect** **SQLDriverConnect**
DM	003	Driver specified by data source name could not be loaded.	**SQLConnect** **SQLDriverConnect**
DM	004	Driver's **SQLAllocEnv** failed.	**SQLConnect** **SQLDriverConnect**
DM	005	Driver's **SQLAllocConnect** failed.	**SQLConnect** **SQLDriverConnect**
DM	006	Driver's **SQLSetConnectOption** failed.	**SQLConnect** **SQLDriverConnect**
SG	000	Invalid table name.	**SQLExecDirect** **SQLExecute** **SQLPrepare**
SG	S00	Invalid table name; table not found.	**SQLExecDirect** **SQLExecute** **SQLPrepare**

SG	S01	Table already exists.	**SQLExecDirect** **SQLExecute** **SQLPrepare**
SH	000	Invalid view name.	**SQLExecDirect** **SQLExecute** **SQLPrepare**
SH	S00	Invalid view name; view not found.	**SQLExecDirect** **SQLExecute** **SQLPrepare**

Class	Subclass	Description	**Can be obtained from**
SH	S01	View already exists.	**SQLExecDirect** **SQLExecute** **SQLPrepare**
SI	000	Invalid index name.	**SQLExecDirect** **SQLExecute** **SQLPrepare**
SI	S00	Invalid index name; index not found.	**SQLExecDirect** **SQLExecute** **SQLPrepare**
SI	S01	Index already exists.	**SQLExecDirect** **SQLExecute** **SQLPrepare**
SJ	000	Invalid column name.	**SQLExecDirect** **SQLExecute** **SQLPrepare**
SJ	S00	Invalid column name; column not found.	**SQLExecDirect** **SQLExecute** **SQLPrepare**
SJ	S01	Column already exists.	**SQLExecDirect** **SQLExecute** **SQLPrepare**

Appendix B ODBC State Transition Table

The following tables show the effect of each ODBC statement on the states of the environment handle (henv), the connection handle (hdbc), and the statement handle (hstmt).

Each ODBC function is listed on the left-hand side of the following tables. The possible states of the handles (henv and hdbc in the first table, hstmt in the second), before application of the function, are listed horizontally at the top of each table. Each entry in the table is the result state, or set of result states of the handle after successful execution of the function. Unless noted, an error from an ODBC function causes no state transition.

Footnotes reference notes below each table.

The following functions do not cause state transitions, and are therefore not included in the state transition tables:

- **SQLCancel** causes the active function to return an error if the operation was canceled. For more information, refer to the **SQLCancel** function in Chapter 2, "ODBC API Reference."
- **SQLError** returns error and status information.

Environment and Connection State Transitions

The ODBC environment can be in one of four states:

S0 unallocated environment

S1 allocated environment

S2 allocated hdbc

S3 connected hdbc

The following table lists the next valid state for each function when called from a given state. The characters InvHndl indicate an INVALID_HANDLE return code.

Core Functions	S0 unallocated henv	S1 allocated henv	S2 allocated hdbc	S3 connected hdbc
SQLAllocEnv	S1	S1 (1)	S1 (1)	S1 (1)
SQLAllocConnect	InvHndl	S2	S2 (1)	S2 (1)
SQLConnect	InvHndl	InvHndl	S3	*08 002*
SQLDisconnect	InvHndl	InvHndl	*01 002* (3)	S2
SQLFreeConnect	InvHndl	InvHndl	S1	*08 002*
SQLFreeEnv	InvHndl	S0	*80 010*	*08 002*
SQLTransact	InvHndl	InvHndl	*08 003*	S3
Extended Functions	S0	S1	S2	S3
SQLDataSources	InvHndl	S1	S2	S3
SQLConnect	InvHndl	InvHndl	S3	*08 002*

SQLGetConnectOption	InvHndl	InvHndl	S2	S3
SQLGetFunctions	InvHndl	InvHndl	S2	S3
SQLGetInfo	InvHndl	InvHndl	S2 (2)	S3
SQLNativeSql	InvHndl	InvHndl	*08 003*	S3
SQLSetConnectOption	InvHndl	InvHndl	S2	S3

[1] Allocation functions should never be called for already-allocated handles, as the driver will lose any information associated with the handle and the handle will return to the allocated state (S1).

[2] Transition for information options SQL_DRIVER_NAME and SQL_ODBC_VER, which can be returned by the Driver Manager. Otherwise, the Driver Manager returns SLQ_ERROR with an SQLSTATE value equal to 08 003 (connection not open).

[3] **SQLDisconnect** returns SQLSTATE value 01 002 only if there is an uncommitted transaction associated with the hdbc. If there is no connection (for example, the application calls **SQLDisconnect** twice in a row), **SQLDisconnect** returns SQLSTATE value 08 003.

Statement Transitions

A statement handle (hstmt) can be in one of five states:

S0 Not allocated

S1 Allocated

S2 Prepared

S3 Executed, or cursor open but not positioned on a row

S4 Cursor positioned on a row

Prior to allocating a statement, an application must establish a successful connection.

The following table lists the next valid state for each function when called from a given state. Notes, indicated by parentheses, are listed on the following page. The letters InvHndl indicate an INVALID_HANDLE return code.

Core Functions	S0 not allocated	S1 allocated	S2 prepared	S3 executed	S4 cursor positioned
SQLAllocStmt	S1	S1 (1)	S1 (1)	S1 (1)	S1 (1)
SQLBindCol	InvHndl	S1	S2	S3	S4
SQLDescribeCol	InvHndl	*80 010*	S2	S3	S4
SQLDisconnect (9)	S0	S0	S0	S0(13)	S0(13)
SQLExecDirect (4)	InvHndl	S3 (2)	S3 (2)	S3 (5)	*08 002*
SQLExecute (4)	InvHndl	*80 010*	S3 (2)	S3 (5)	*08 002*
SQLFetch	InvHndl	*80 010*	*80 010*	S3 (8), S4	S3 (8),

					S4
SQLFreeStmt (6)	InvHndl	S1	S2	S2	S2
SQLFreeStmt (7)	InvHndl	S0	S0	S0	S0
SQLFreeStmt (15)	InvHndl	S1	S2	S3	S4
SQLGetCursorName (11)	InvHndl	*80 015*	*80 015*	S3	S4
SQLGetCursorName (12)	InvHndl	S1	S2	S3	S4
SQLNumResultCols	InvHndl	*80 010*	S2	S3	S4
SQLPrepare (3)	InvHndl	S2	S2	S2 (5)	*08 002*
SQLRowCount	InvHndl	*80 010*	*80 010*	S3 (10)	*80 007*
SQLSetCursorName	InvHndl	S1	S2	*24 000*	*24 000*
SQLSetParam	InvHndl	S1	S2	S3 (14)	S4 (14)
SQLTransact (9)	S0	S1	S1,S2	S1,S3	S1,S3,S4

		(18)	(18)	(18)

Extended Functions	S0 not allocated	S1 allocated	S2 prepared	S3 executed	S4 cursor positioned
SQLColAttributes	InvHndl	*80 010*	S2	S3	S4
SQLColumnPrivileges	InvHndl	S3	S3	S3 (5)	*08 002*
SQLColumns	InvHndl	S3	S3	S3 (5)	*08 002*
SQLDescribeParam	InvHndl	*80 010*	S2	S3	S4
SQLExtendedFetch	InvHndl	*08 010*	*80 010*	S3 (8), S4	S3 (8), S4
SQLForeignKeys	InvHndl	S3	S3	S3 (5)	*08 002*
SQLGetData	InvHndl	*08 010*	*08 010*	*24 000*	S4
SQLGetStmtOption	InvHndl	S1	S2	S3	S4
SQLGetTypeInfo	InvHndl	S3	S3	S3 (5)	*08 002*
SQLMoreResults	InvHndl	*80 010*	*80 010*	S3	S3
SQLNumParams	InvHndl	*80 010*	S2	S3	S4

SQLParamData	InvHndl	*80 010*	*80 010*	S3 (16)	*24 000*
SQLPrimaryKeys	InvHndl	S3	S3	S3 (5)	*08 002*
SQLPutData	InvHndl	*80 010*	*80 010*	S3 (17)	*24 000*
SQLSetPos	InvHndl	*80 010*	*80 010*	*24 000*	S4
SQLSetScrollOptions	InvHndl	S1	S2	S3 (5)	*24 000*
SQLSetStmtOption	InvHndl	S1	S2	S3	S4
SQLSpecialColumns	InvHndl	S3	S3	S3 (5)	*08 002*
SQLStatistics	InvHndl	S3	S3	S3 (5)	*08 002*
SQLTablePrivileges	InvHndl	S3	S3	S3 (5)	*08 002*
SQLTables	InvHndl	S3	S3	S3 (5)	*08 002*

Notes:

[1] Allocation functions should never be called for already-allocated handles, as

the driver will lose any information associated with the handle and the handle will return to the allocated state (S1).

[2]If an SQL statement contains no parameters and is not a **SELECT** statement, or when all parameters have been set and all selected columns have been bound; otherwise, a driver returns SQL_ERROR with SQLSTATE equal to 07 001.

[3]The hstmt indicated by cursor in WHERE CURRENT OF cursor must be in state S3 or S4 for **SQLPrepare**. After **SQLPrepare** the hstmt remains in the same state.

[4]The hstmt indicated by the cursor in "UPDATE WHERE CURRENT OF cursor" or "DELETE WHERE CURRENT OF cursor" must be in state S4, or the **SQLExecute** or **SQLExecDirect** function returns SQL_ERROR with an error value of 24 000 Following the positioned **UPDATE** or **DELETE**, the positioned hstmt is left in state S4.

[5]This transition is legal only if there are no open cursors on the hstmt . Any **SELECT** statement executed on the hstmt–or any function that returns a result set on an hstmt, such as **SQLColumns**–must be followed by a call to **SQLFreeStmt** with the SQL_CLOSE option specified. Otherwise, a driver returns SQL_ERROR with SQLSTATE equal to 08 002.

[6]Transition for end option equal to SQL_CLOSE.

[7]Transition for end option equal to SQL_DROP.

[8] Transition for SQL_NO_DATA_FOUND.

[9]Transition for all hstmts allocated with **SQLAllocStmt** for the same hdbc.

[10] Transition legal for **INSERT**, **UPDATE**, and **DELETE** statements only; otherwise, a driver returns SQL_ERROR with SQLSTATE equal to 80 007.

[11] Case where **SQLSetCursorName** has not been called.

[12] Case where **SQLSetCursorName** has been called.

[13] Any uncommitted work on the hdbc is rolled back.

[14] Resetting parameters has no effect on the already-executed statement.

[15] Transition for end option equal to SQL_UNBIND or SQL_RESET_PARAMS.

[16] Transition for an executed statement with one or more long value parameters remaining to be passed. If there are no long values waiting to be processed, the driver returns SQL_ERROR with SQLSTATE equal to 80 LD0 (no long data values pending). **SQLParamData** is the only function that can be called when **SQLExecute** or **SQLExecDirect** return SQL_NEED_DATA. **SQLPutData** or **SQLParamData** are the only functions that can be called after **SQLParamData** returns SQL_NEED_DATA. The statement is fully in state S3 only after **SQLParamData** returns SQL_SUCCESS.

[17] Transition for an executed statement where the final long value parameter is passed. If there are no long values waiting to be processes, the driver returns SQL_ERROR with SQLSTATE equal to 80 LD0 (no long data values pending). **SQLPutData** and **SQLParamData** are the only functions that can be called after **SQLPutData** returns SQL_NEED_DATA. The statement is fully in state S3 only after **SQLParamData** returns SQL_SUCCESS.

[18] Transitions for data sources that can retain cursor state across transaction boundaries. For more information, refer to **SQLGetInfo** options SQL_COMMIT_CURSOR_BEHAVIOR and SQL_ROLLBACK_CURSOR_BEHAVIOR.

Appendix C SQL Grammar

The following paragraphs list the constructs that are valid in a call to **SQLPrepare**, **SQLExecute**, or **SQLExecDirect**. To the left of each construct is an indicator that tells whether the construct is part of the core grammar, the minimum grammar, or both.

Elements that are part of Integrity Enhancement Facility (IEF) and are separate from the ANSI 1989 standard are presented in the following typeface and font, distinct from the rest of the grammar:

table-constraint-definition

The set of data types defined in this grammar is not necessarily supported by a specific data source; the use and syntax of each data type is database-dependent.

Element	Core	Mini-mum
alter-table-statement ::= ALTER TABLE *base-table-name* { ADD *column-identifier data-type* | ADD (*column-identifier data-type* [, *column-identifier data-type*]...) }	X	
create-index-statement ::= CREATE [UNIQUE] INDEX *index-name* ON *base-table-name* (*column-identifier* [ASC | DESC] [, *column-identifier* [ASC | DESC]]...)	X	

Element	Core	Mini- mum
create-table-statement ::= CREATE TABLE *base-table-name-1* (*column-element* [, *column-element*] ...) *column-element* ::= *column-definition* | *table-constraint-definition* *column-definition* ::= *column-identifier data-type* [DEFAULT *default-value*] [*column-constraint-definition* [, *column-constraint-definition*]...] *column-constraint-definition* ::= NOT NULL | [UNIQUE | PRIMARY KEY] (*column-identifier*[,*column-identifier*]...) | [REFERENCES *base-table-name-2 referenced-columns*] | [CHECK (*search-condition*)] *default-value* ::= *literal* | NULL | USER *table-constraint-definition* ::= UNIQUE (*column-identifier* [, *column-identifier*] ...) | PRIMARY KEY (*column-identifier* [, *column-identifier*] ...) | CHECK (*search-condition*) | FOREIGN KEY *referencing-columns* REFERENCES *base-table-name-2 referenced-columns*	X	
create-view-statement ::= CREATE VIEW *viewed-table-name* [(*column-identifier* [, *column-identifier*]...)] AS *query-specification*	X	
delete-statement-positioned ::= DELETE FROM *table-name* WHERE CURRENT OF *cursor-name*	X	
delete-statement-searched ::= DELETE FROM *table-name* [WHERE *search-condition*]	X	X

drop-index-statement ::= X
DROP INDEX *index-name*

drop-table-statement ::= X
DROP TABLE *base-table-name*
[{ CASCADE | RESTRICT }]

Element — Core — Minimum

drop-view-statement ::= — X

DROP VIEW *viewed-table-name*
[{ CASCADE | RESTRICT }]

grant-statement ::= — X

GRANT {ALL | *grant-privilege* [, *grant-privilege*]... }
ON *table-name*
TO {PUBLIC | *user-name* [, *user-name*]... }

grant-privilege ::=
DELETE
| INSERT
| SELECT
| UPDATE [(*column-identifier* [, *column-identifier*]...)]
| REFERENCES [(*column-identifier* [, *column-identifier*]...)]

insert-statement ::= — X

INSERT INTO *table-name* [(*column-identifier* [, *column-identifier*]...)]
{ *query-specification* | VALUES (*insert-value* [, *insert-value*]...)}

insert-value ::=
| *dynamic-parameter*
| *literal*
| NULL
| USER

revoke-statement ::= — X

REVOKE {ALL | *revoke-privilege* [, *revoke-privilege*]... }
ON *table-name*
FROM {PUBLIC | *user-name* [, *user-name*]... }
[{ CASCADE | RESTRICT }]

revoke-privilege ::=
DELETE
| INSERT
| SELECT

UPDATE

REFERENCES

Element	Core	Minimum
select-statement ::= SELECT [ALL | DISTINCT] *select-list* FROM *table-reference* [, *table-reference*]... [WHERE *search-condition*] [GROUP BY *column-name* [, *column-name*]...] [HAVING *search-condition*] [UNION *select-statement*]... [*order-by-clause*]	X	
select-statement ::= SELECT [ALL | DISTINCT] *select-list* FROM *table-reference* [, *table-reference*]... [WHERE *search-condition*] [*order-by-clause*]		X
select-for-update-statement ::= SELECT [ALL | DISTINCT] *select-list* FROM *table-reference* [, *table-reference*]... [WHERE *search-condition*] FOR UPDATE OF *column-name* [, *column-name*]...	X	
update-statement-positioned ::= UPDATE *table-name* SET *column-identifier* = {*expression* | NULL} [, *column-identifier* = {*expression* | NULL}]... WHERE CURRENT OF *cursor-name*	X	
update-statement-searched UPDATE *table-name* SET *column-identifier* = {*expression* | NULL } [, *column-identifier* = {*expression* | NULL}]... [WHERE *search-condition*]	X	X

Elements Used in SQL Statements

The following elements are used in the SQL statements listed previously .

Element	Core	Mini-mum
approximate-numeric-literal ::=*mantissa*E*exponent* *mantissa* ::= *exact-numeric-literal* *exponent* ::= [+|-] *unsigned-integer*	X	X
approximate-numeric-type ::= FLOAT | DOUBLE PRECISION | REAL	X	X
base-table-identifier ::= *user-defined-name*	X	X
base-table-name ::= [*user-name*.]*base-table-identifier*	X	
base-table-name ::=*base-table-identifier*		X
between-predicate ::= *expression* [NOT] BETWEEN *expression* AND *expression*	X	
binary-literal ::= {implementation defined}	X	X
binary-type ::= BINARY (*length*) | VARBINARY (*length*) | LONG VARBINARY(*length*)	X	X
character ::= {any character in the implementor's character set except the newline indication}	X	X
character-string-literal :: = '{*character*}...'	X	X

character-string-type ::= X X
CHARACTER(*length*)
| CHAR(*length*)
| CHARACTER VARYING(*length*)
| VARCHAR (*length*)
| LONG VARCHAR(*length*)

column-identifier ::= *user-defined-name* X X

column-name ::= [{*table-name* | *correlation-name*}.]*column-identifier* X

column-name ::= [*table-name*.]*column-identifier* X

comparison-operator ::= < | > | <= | >= | = | <> X X

comparison-predicate ::= *expression comparison-operator* X
{*expression* | (*sub-query*)}

comparison-predicate ::= X
expression comparison-operator expression

Element	Core	Minimum
correlation-name ::= *user-defined-name*	X	
cursor-name ::= *user-defined-name*	X	
data-type ::= *binary-type* | *character-string-type* | *date-type* | *exact-numeric-type* | *approximate-numeric-type* | *time-type* | *timestamp-type*	X	X
date-literal ::= *'date-value'*	X	X
date-separator ::= -		
date-type ::= DATE	X	X
date-value ::= *years-value date-separator months-value date-separator days-value*	X	X
days-value ::= *digit digit*	X	X
digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9	X	X
dynamic-parameter ::= ?	X	X
exact-numeric-literal ::= [+|-] { *unsigned-integer* [.*unsigned-integer*] | *unsigned-integer*.	X	X

| .*unsigned-integer* }

exact-numeric-type ::= DECIMAL(*precision*, *scale*) | INTEGER | SMALLINT | NUMERIC(*precision*, *scale*) | TINYINT | BIGINT | BIT *precision* ::= *unsigned-integer* *scale* ::= *unsigned-integer*	X	X
exists-predicate ::= EXISTS (*sub-query*)	X	

Element	Core	Minimum
expression ::= *term* | *expression* {+|-} *term*	X	X
term ::= *factor* | *term* {*|/} *factor*	X	X
factor ::= [+|-]*primary*	X	X
primary ::= *column-name* | *dynamic-parameter* | *literal* | *set-function-reference* | USER (*2*) | (*expression*)	X	
primary ::= *column-name* | *dynamic-parameter* | *literal* | (*expression*)		X
hours-value ::= *digit digit*	X	X
index-identifier ::= *user-defined-name*	X	
index-name ::= [*user-name*.]*index-identifier*	X	
in-predicate ::= *expression* [NOT] IN {(*value* {, *value*}...) | (*sub-query*)} *value* ::= *literal* | USER | *dynamic-parameter*	X	
join-condition ::= ON *search-condition*	X	X
keyword ::= (see list of reserved keywords)	X	X

length ::= *unsigned-integer*	X	X
letter ::= *lower-case-letter* | *upper-case-letter*	X	X
like-predicate ::= *column-name* [NOT] LIKE *pattern-value*	X	X
pattern-value ::= *character-string-literal* | *dynamic-parameter* | USER	X	
pattern-value ::= *character-string-literal* | *dynamic-parameter* (in character-string-literal, the percent character ('%') matches 0 or more of any character; the underscore character ('_') matches 1 or 0 characters)		X
literal ::= *character-string-literal* | *numeric-literal*	X	X
lower-case-letter ::= 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	X	X
minutes-value ::= *digit digit*	X	X
months-value ::= *digit digit*	X	X
null-predicate ::= *column-name* IS [NOT] NULL	X	X

Element	Core	Minimum
numeric-literal ::= *exact-numeric-literal* | *approximate-numeric-literal*	X	X
order-by-clause ::= ORDER BY *sort-specification* [, *sort-specification*]... *sort-specification* ::= {*unsigned-integer* | *column-name* } [ASC | DESC]	X	X
outer-join ::= *table-reference* LEFT OUTER JOIN *table-reference join-condition* *join\-condition* ::= ON *search-condition* (Notes: For outer joins, *search-condition* must contain only the join condition between the specified *table-reference*s. The outer-join syntax must be placed within an escape clause.)		
predicate ::= *between-predicate* | *comparison-predicate* | *exists-predicate* | *in-predicate* | *like-predicate* | *null-predicate* | *quantified-predicate*	X	
predicate ::= *comparison-predicate* | *like-predicate* | *null-predicate*		X
quantified-predicate ::= *expression comparison-operator* {ALL | ANY} (*sub-query*)	X	
referenced-columns ::= (*column-identifier* [, *column-identifier*]...)	X	
referencing-columns ::= (*column-identifier* [, *column-identifier*]...)	X	
search-condition ::= *boolean-term* [OR *search-condition*] *boolean-term* ::= *boolean-factor* [AND *boolean-term*] *boolean-factor* ::= [NOT] *boolean-primary*	X	X

boolean-primary ::= *predicate* | (*search-condition*)		
seconds-fraction ::= *digit digit digit [digit digit digit]*	X	X
seconds-value ::= *digit digit*	X	X
select-list ::= * | *select-sublist* [, *select-sublist*]...	X	X
select-sublist ::= *expression* | {*table-name* | *correlation-name*}.*	X	
select-sublist ::= *expression*		X
separator ::= The blank character or an implementation-defined end-of-line indicator.	X	X

<table>
<tr><th>Element</th><th>Core</th><th>Mini-
mum</th></tr>
<tr><td>set-function-reference ::= COUNT(*) | distinct-function | all-function
    distinct-function ::=
        {AVG | COUNT | MAX | MIN | SUM} (DISTINCT column-name)
    all-function ::=
        {AVG | MAX | MIN | SUM} (expression)</td><td>X</td><td></td></tr>
<tr><td>SQL-escape-clause ::=
    standard-SQL-escape-initiator extended-SQL-text standard-SQL-
        escape-terminator
    | extended-SQL-escape-prefix extended-SQL-text extended-SQL-
        escape-terminator
    standard-SQL-escape-initiator ::= standard-SQL-escape-prefix SQL-
        escape-identification,
    standard-SQL-escape-prefix ::= --*(
    extended-SQL-escape-prefix ::= {
    standard-SQL-escape-terminator ::= --*)
    extended-SQL-escape-terminator ::= }
    SQL-escape-identification ::= SQL-escape-vendor-clause
    SQL-escape-vendor-clause ::=
        VENDOR(Microsoft), PRODUCT(ODBC)</td><td>X</td><td>X</td></tr>
<tr><td>sub-query ::=
    SELECT [ALL | DISTINCT] select-list
    FROM table-reference [, table-reference]...
    [WHERE search-condition]
    [GROUP BY column-name [, column-name]...]
    [HAVING search-condition]</td><td>X</td><td></td></tr>
<tr><td>table-identifier ::= user-defined-name</td><td>X</td><td>X</td></tr>
<tr><td>table-name ::= [user-name.]table-identifier</td><td>X</td><td></td></tr>
</table>

table-name ::=*table-identifier*		X
table-reference ::= *table-name* [*correlation-name*]	X	
table-reference ::= *table-name*		X
time-literal ::='*time-value*'	X	X

<table>
<tr><th>Element</th><th>Core</th><th>Mini-
mum</th></tr>
<tr><td>time-separator ::= :</td><td></td><td></td></tr>
<tr><td>time-type ::= TIME</td><td>X</td><td>X</td></tr>
<tr><td>time-value ::=
hours-value time-separator minutes-value time-separator
seconds-value</td><td>X</td><td>X</td></tr>
<tr><td>timestamp-literal ::='date-value:time-value.[seconds-fraction]'</td><td>X</td><td>X</td></tr>
<tr><td>timestamp-type ::= TIMESTAMP</td><td>X</td><td>X</td></tr>
<tr><td>token ::= delimiter-token | non-delimiter-token
delimiter-token ::=
character-string-literal
| , | (|) | < | > | . | : | = | * | + | - | / | <> | >= | <= | ?
non-delimiter-token ::=
keyword
| numeric-literal
| user-defined-name</td><td>X</td><td>X</td></tr>
<tr><td>unsigned-integer ::= {digit}...</td><td>X</td><td>X</td></tr>
<tr><td>upper-case-letter ::=
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</td><td>X</td><td>X</td></tr>
<tr><td>user-defined-name ::=
letter[digit | letter | _]...</td><td>X</td><td>X</td></tr>
<tr><td>user-name ::= user-defined-name</td><td>X</td><td>X</td></tr>
</table>

viewed-table-identifier ::= *user-defined-name*	X	
viewed-table-name ::= [*user-name*.]*viewed-table-identifier*	X	
years-value ::= *digit digit digit digit*	X	X

List of Reserved Keywords

The following words are reserved for use in ODBC function calls. These words do not constrain the minimum SQL grammar; however, to ensure compatibility with drivers that support the core SQL grammar, applications should avoid using any of these keywords.

ABSOLUTE
ADA
ADD
ALL
ALLOCATE
ALTER
AND
ANY
ARE
AS
ASC
ASSERTION
AT
AUTHORIZATION
AVG
BEGIN
BETWEEN
BIT
BIT_LENGTH
BY
CASCADE
CASCADED
CASE
CAST
CATALOG
CHAR
CHAR_LENGTH
CHARACTER
CHARACTER_LENGTH
CHECK
CLOSE
COALESCE
COBOL
COLLATE
COLLATION
COLUMN
COMMIT
CONNECT
CONNECTION
CONSTRAINT
CONSTRAINTS
CONTINUE
CONVERT
CORRESPONDING
COUNT
CREATE
CURRENT
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURSOR
DATE
DAY
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFERRABLE
DEFERRED
DELETE
DESC
DESCRIBE
DESCRIPTOR
DIAGNOSTICS
DICTIONARY
DISCONNECT
DISPLACEMENT
DISTINCT
DOMAIN
DOUBLE
DROP
ELSE
END
END-EXEC
ESCAPE
EXCEPT
EXCEPTION
EXEC
EXECUTE
EXISTS
EXTERNAL
EXTRACT
FALSE
FETCH
FIRST
FLOAT
FOR
FOREIGN
FORTRAN
FOUND
FROM
FULL
GET
GLOBAL
GO
GOTO
GRANT
GROUP
HAVING
HOUR
IDENTITY
IGNORE
IMMEDIATE
IN
INCLUDE
INDEX
INDICATOR
INITIALLY
INNER

INPUT
INSENSITIVE
INSERT
INTEGER
INTERSECT
INTERVAL
INTO
IS
ISOLATION
JOIN
KEY
LANGUAGE
LAST
LEFT
LEVEL
LIKE
LOCAL
LOWER
MATCH
MAX
MIN
MINUTE
MODULE
MONTH
MUMPS
NAMES
NATIONAL
NCHAR
NEXT
NONE
NOT
NULL
NULLIF
NUMERIC
OCTET_LENGTH
OF
OFF
ON
ONLY
OPEN
OPTION
OR
ORDER
OUTER
OUTPUT
OVERLAPS
PARTIAL
PASCAL
PLI
POSITION
PRECISION
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVILEGES
PROCEDURE
PUBLIC
RESTRICT
REVOKE
RIGHT
ROLLBACK
ROWS
SCHEMA
SCROLL
SECOND
SECTION
SELECT
SEQUENCE
SET
SIZE
SMALLINT
SOME
SQL
SQLCA
SQLCODE
SQLERROR
SQLSTATE
SQLWARNING
SUBSTRING
SUM
SYSTEM
TABLE
TEMPORARY
THEN
TIME
TIMESTAMP
TIMEZONE_HOUR
TIMEZONE_MINUTE
TO
TRANSACTION
TRANSLATE
TRANSLATION
TRUE
UNION
UNIQUE
UNKNOWN
UPDATE
UPPER
USAGE
USER
USING
VALUE
VALUES
VARCHAR
VARYING
VIEW
WHEN
WHENEVER
WHERE
WITH
WORK
YEAR

Appendix D   Data Type Definitions

XE "Data types:definitions"§The ODBC interface defines two types of data types: core data types and extended data types. Most data sources support the core data types listed below. Applications should, however, rely on information obtained from SQLGetTypeInfo.

The following two subsections list core and extended data types. The last subsection lists data type conversion rules that apply in calls to **SQLBindCol**, **SQLGetData**, and **SQLSetParam**. If a data source is compliant with the X/Open and SQL Access Group CAE SQL draft specification, then all data types are defined as shown below.

Core Data Types

The first column of the following table lists data types that are found in SQL data sources.

The #define names in the second column describe ODBC data types that correspond to data source types. The descriptions in the third column are derived from Microsoft C. The fourth column lists the C type used in an ODBC call. The fifth column lists the C type defined in SQL.H.

Applications can use SQL data types in ODBC functions by using equivalent C types, listed in the following table. Names of SQL and C data types are provided in the C header file. (For more information about the header file, refer to Appendix F, “C Header Files.”)

#define Name for SQL Data Type	#define Name for C Data Type	C Type Used in ODBC Call	C Type in SQL.H File	Description
SQL_CHAR	SQL_C_CHAR	UCHAR *	unsigned char *	Fixed length character data of length n
SQL_NUMERIC	SQL_C_CHAR	UCHAR *	unsigned char *	Signed exact numeric value with a precision and scale. Precision determines the total number of decimal digits. Scale determines the number of decimal digits to the right of the decimal point.
SQL_DECIMAL	SQL_C_CHAR	UCHAR *	unsigned char *	Signed exact numeric value with a precision and scale. Precision determines the total number of decimal digits. Scale determines the number of decimal digits to the right of the decimal point.
SQL_INTEGER	SQL_C_LONG	SDWORD	long int	32-bit signed integer value
SQL_SMALLINT	SQL_C_SHORT	SWORD	short int	16-bit signed integer value
SQL_FLOAT	SQL_C_DOUBLE	SDOUBLE	double	64-bit value with 11-bit excess-1023 exponent, a 52-bit mantissa, and a high-order single bit (IEEE standard notation)
SQL_REAL	SQL_C_FLOAT	SFLOAT	float	32-bit value with 8-bit excess-127 exponent, and 23-bit mantissa (IEEE standard notation)
SQL_	SQL_C_DOUBL	SDOUBL	double	64-bit value with 11-

Applications can use the following data types in function arguments.

C Data Type Used in ODBC Call	Typical C Data Type	Usage
PTR	void *	Pointers to storage for data values and parameter marker values
UWORD	unsigned short	Column numbers and parameter marker numbers
UDWORD	unsigned long	Length of data buffers
HSTMT	void *	Internal data structure used by the driver for statement information
HDBC	void *	Internal data structure used by the driver for connection information
HENV	void *	Internal data structure used by the driver for environment information
RETCODE	short	Return code from ODBC functions

Limits on precision and scale follow the rules stated in The X/Open and SQL Access Group CAE SQL draft specification (1991). . Truncation of values to the right of the decimal point for numeric values returns a truncation error as a warning. Truncation to the left of the decimal point returns an error.

Extended Data Types

The following data types extend the set of data types defined in the X/Open and SQL Access Group SQL specification. The columns follow the same format as the core data type table.

#define Name for SQL Data Type	#define Name for C Data Type	C Type used in ODBC Call	C Type in SQLEXT. H File	Description
SQL_ LONGVARCHAR	SQL_C_CHAR	UCHAR *	unsigned char *	Long varying length characters of maximum length n
SQL_BINARY	SQL_C_BINARY	UCHAR *	unsigned char *	Fixed length binary data of maximum length n
SQL_VARBINARY	SQL_C_BINARY	UCHAR *	unsigned char *	Varying length binary data of maximum length n
SQL_ LONGVARBINARY	SQL_C_BINARY	UCHAR *	unsigned char *	Varying length binary data of maximum length n
SQL_BIT	SQL_C_BIT	UCHAR	unsigned char	8-bit unsigned character
SQL_TINYINT	SQL_C_TINYINT	SCHAR	signed char	8-bit signed character
SQL_BIGINT	SQL_C_CHAR	UCHAR *	unsigned char *	64-bit signed integer value, values between (-2 to the 63rd)-1 and (+2 to the 63rd)-1

Comments

SQL_DATE, SQL_TIME, and SQL_TIMESTAMP are defined as structures that contain a sequence of integer component values, as follows:

```
typedef struct tagDATE_STRUCT
  {
  SWORD year;
  UWORD month;
  UWORD day;
  } DATE_STRUCT;

typedef struct tagTIME_STRUCT
  {
  UWORD hour;
  UWORD minute;
  UWORD second;
  } TIME_STRUCT;

typedef struct tagTIMESTAMP_STRUCT
  {
  SWORD year;
  SWORD month;
  UWORD day;
  UWORD hour;
  UWORD minute;
  UWORD second;
  UDWORD fraction;
  } TIMESTAMP_STRUCT;
```

The following paragraphs list conversion results.

Conversion Guidelines

The following table lists conversions that are valid between data types. A plus sign (+) indicates that the conversion is supported according to the specified rules. An "S" indicates that the standard default transfer type applies as specified earlier in this appendix. A blank indicates that conversion is not supported between the two types.

SQL Data Type	ODBC Data Type	SQL_C_LONG	SQL_C_SHORT	SQL_C_TINYINT	SQL_C_BIT	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_CHAR	SQL_C_BINARY	SQL_C_DATE	SQL_C_TIME	SQL_C_TIMESTAMP
SQL_TINYINT		+	+	S	+	+	+	+				
SQL_SMALLINT		+	S	+	+	+	+	+				
SQL_INTEGER		S	+	+	+	+	+	+				
SQL_BIGINT		+	+	+	+	+	+	S				
SQL_BIT		+	+	S	S	+	+	+				
SQL_FLOAT		+	+	+	+	S	+	+				
SQL_REAL		+	+	+	+	S	+	+				
SQL_DOUBLE		+	+	+	+	+	S	+				

SQL_NUMERIC	+	+	+	+	+	+	S				
SQL_DECIMAL	+	+	+	+	+	+	S				
SQL_CHAR							S				
SQL_VARCHAR							S				
SQL_LONGVARCHAR							S				
SQL_BINARY							+	S			
SQL_VARBINARY							+	S			
SQL_LONGVARBINARY							+	S			
SQL_DATE							+		S		
SQL_TIME							+			S	
SQL_TIMESTAMP							+		+	+	S

An application can request data conversions in calls to **SQLBindCol**, **SQLGetData**, and **SQLSetParam**. **SQLBindCol** and **SQLGetData** allow the application to convert result column data types; **SQLSetParam** allows the application to convert parameter data types.

To request conversion for a column or parameter of a specific SQL data type, the application selects a C data type #define from the set of conversions that are valid for the SQL data type. The application then sets fCType equal to the C data type #define.

If requested in **SQLBindCol** or **SQLGetData**, the driver converts the column data to the specified C data type at fetch time. If the application specifies an invalid conversion, the driver returns an error.

If requested in **SQLSetParam**, the driver converts the parameter data to the SQL data type of the corresponding column at execute time. If the application specifies an invalid conversion, the driver returns an error.

If the application specifies fCType equal to SQL_NO_CONVERT in **SQLBindCol** or **SQLSetParam**, the driver uses the standard mapping between SQL data types and C data types (noted by "s" in the above table).

Conversion of Binary Data Types

Conversion of binary data types (SQL_BINARY, SQL_VARBINARY, and SQL_LONGVARBINARY) to character C data types result in each byte (8 bits) of source data represented in two characters as the ASCII hexadecimal equivalent. For example, a binary 00000001 converts to ASCII 31. The driver converts an even number of bytes.

Conversion of character SQL data types to binary SQL data types are performed as part of the **SQLFetch** function.

Conversion from a non-null value of a numeric SQL data type to an exact numeric C data type will be successful as long as the whole part of the number (the leading significant digits) is not truncated. The fractional part of the number can be truncated as necessary.

Conversion of Numeric Data Types

Conversion from a non-null value of a numeric SQL data type to an approximate numeric C data type will be successful as long as the number is within the range of magnitude of the destination data type. The result is an approximation of the SQL data type value with the precision of the C data type For example:

SQL_INTEGER 24,301 converts to SQL_C_FLOAT 24,301
SQL_FLOAT 2.176E+40 converts to SQL_C_FLOAT error
SQL_DOUBLE 2.176E_37 converts to SQL_C_FLOAT 2.176E+37

Conversion of Date, Time, and Timestamp Data Types

Conversion from a non-null value of a SQL date, time, or timestamp data type to a character C data type results in a date, time, or timestamp literal, respectively, as defined in Chapter 7, "Processing Extended Data Types." Truncation of the resulting string is handled in the same way as truncation of a non-null value of an SQL character data type.

Examples

Types NUMERIC, DECIMAL, and BIGINT translate to type SQL_C_CHAR (a character string). For example:

```
CREATE TABLE T (D DECIMAL (8,4))
INSERT INTO T VALUES (1234.5678)
SELECT * FROM T
```

produces:

```
"1234.5678"
```

Types DATE, TIME, and TIMESTAMP translate to type SQL_C_CHAR. For example:

"1986-12-01"	for date (yyyy-mm-dd format)
"10:15:30"	for time (hh:mm:ss format)
"1990-01-05 08:30:00.00500"	for timestamp (yyyy-mm-dd hh:mm:ss[.ffffff] format)

Types BINARY, VARBINARY, and LONG VARBINARY translate to type SQL_C_CHAR, with the value equal to the printable ASCII representation of the binary value. For example:

0x0 0x30	translates to "0"
0x1 0x31	translates to "1"
0xf 0x66	translates to "f"
"cd0534"	translates to 0x656430353334

Appendix E Comparison Between Embedded SQL and ODBC

This appendix compares ODBC to embedded SQL and embedded SQL to ODBC.

ODBC to Embedded SQL

XE "ODBC:comparing to embedded SQL"§The following table compares core ODBC functions to embedded SQL functions. This comparison is based on the X/Open and SQL Access Group and X/Open specification "Structured Query Language (SQL)," 1991.

ODBC uses a parameter marker in place of a host variable, wherever a host variable would occur in embedded SQL.

The SQL language is based on the X/Open and SQL Access Group/X/Open specification "Structured Query Language (SQL)" (XO/PRELIM/91/030).

ODBC Function	Statement	Comments
SQLAllocConnect	none	Client implementation memory allocation
SQLConnect	**CONNECT**	Association management
SQLDisconnect	**DISCONNECT**	Association management
SQLFreeConnect	none	Client implementation memory deallocation
SQLAllocStmt	none	Client implementation memory allocation

SQLPrepare	**PREPARE**	The prepared SQL string can contain any of the valid preparable functions as defined by the X/Open specification, including **ALTER**, **CREATE**, <cursor-specification>, searched **DELETE**, dynamic positioned **DELETE**, **DROP**,**GRANT**, **INSERT**, **REVOKE**, searched **UPDATE**, or dynamic positioned **UPDATE**.
SQLSetParam	**SET DESCRIPTOR**	Dynamic **ALLOCATE DESCRIPTOR** and dynamic SQL **SET DESCRIPTOR**. **ALLOCATE DESCRIPTOR** would normally be issued on the first call to **SQLSetParam** for an hstmt. Alternatively, **ALLOCATE DESCRIPTOR** can be called during **SQLAllocStmt**, although this call would be unneeded by SQL statements containing no embedded parameters. The descriptor name is generated by the ODBC implementation.
SQLSetCursorName	none	The specified cursor name is used in the **DECLARE CURSOR** statement generated by **SQLExecute orSQLExecDirect**.
SQLGetCursorName	none	Client cursor name management

ODBC Function	Statement	Comments
SQLExecute	**EXECUTE** or **DECLARE CURSOR** and **OPEN CURSOR**	Dynamic **SQL EXECUTE**. If the SQL statement requires a cursor, then a dynamic **DECLARE CURSOR** statement and a dynamic **OPEN** are issued at this time.
SQLExecDirect	**EXECUTE IMMEDIATE** or **DECLARE CURSOR** and **OPEN CURSOR**	The ODBC function call provides for support for a <cursor specification> and statements allowed in an **EXECUTE IMMEDIATE** dynamic SQL statement. In the case of a <cursor specification>, the call corresponds to static **DECLARE CURSOR** and **OPEN** statements.
SQLNumResultCols	**GET DESCRIPTOR**	COUNT form of dynamic **GET DESCRIPTOR**
SQLDescribeCol	**GET DESCRIPTOR**	VALUE form of dynamic **GET DESCRIPTOR** with <field-name> in {NAME, TYPE, LENGTH, PRECISION,SCALE,NULLABLE}.
SQLBindCol	none	This function establishes output buffers which correspond in usage to host variables for static **FETCH**, and to an SQL DESCRIPTOR for dynamic **FETCH** <cursor> **USING SQL DESCRIPTOR** <descriptor>.
SQLFetch	**FETCH**	Static or dynamic SQL **FETCH**. If the call is a dynamic **FETCH**, then the VALUE form of **GET DESCRIPTOR** is used, with <field-name> in {DATA, INDICATOR}. DATA and INDICATOR values are placed in output buffers specified in **SQLBindCol**.

SQLRowCount	**GET DIAGNOSTICS**	Requested field ROW_COUNT.
SQLFreeStmt (SQL_CLOSE option)	**CLOSE**	Dynamic SQL **CLOSE** .
SQLFreeStmt (SQL_DROP option)	none	Client memory deallocation.
SQLTransact	**COMMIT WORK** or **COMMIT ROLLBACK**	none
SQLCancel	none	none
SQLError	**GET DIAGNOSTICS**	**GET DIAGNOSTICS** retrieves information from the SQL diagnostics area that pertains to the most recently executed SQL statement. This information can be retrieved following execution and preceding the deallocation of the statement.

Embedded SQL to ODBC

The following tables list the relationship between the X/Open Embedded SQL language and corresponding ODBC functions. The section number shown in the first column of each table refers to the a section of the X/Open and SQL Access Group/X/Open specification "Structured Query Language (SQL)" (XO/PRELIM/91/030).

Declarative Statements

The following table lists declarative statements.

Section	SQL Statement	ODBC Function	Comments
4.3.1	Static **DECLARE CURSOR**	none	Issued implicitly by the ODBC driver if a <cursor specification> is passed to **SQLExecDirect**
4.3.2	Dynamic **DECLARE CURSOR**	none	Cursor is generated automatically by the ODBC implementation. To set a name for the cursor, use **SQLSetCursorName**. To retrieve a cursor name, use **SQLGetCursorName**.

Data Definition Statements

The following table lists data definition statements.

Section	SQL Statement	ODBC Function	Comments
5.1.2 5.1.3 5.1.4 5.1.5 5.1.6 5.1.7 5.1.8	**ALTER TABLE** **CREATE INDEX** **CREATE TABLE** **CREATE VIEW** **DROP INDEX** **DROP TABLE** **DROP VIEW**	**SQLPrepare**, **SQLExecute**, or **SQLExecDirect**	none

5.1.9 **GRANT REVOKE**

Data Manipulation Statements

The following table lists data manipulation statements.

Section	SQL Statement	ODBC Function	Comments
5.2.1	**CLOSE**	**SQLFreeStmt**(CLOSE option)	none
5.2.2	Positioned **DELETE**	**SQLExecDirect**(...,"**DELETE FROM** table-name **WHERE CURRENT OF** cursor-name);	Driver-generated <cursor-name> can be obtained by calling **SQLGetCursorName**.
5.2.3	Searched **DELETE**	**SQLExecDirect**(..., "**DELETE FROM** table-name **WHERE** search condition")	none
5.2.4	**FETCH**	**SQLFetch**	none
5.2.5	**INSERT**	**SQLExecDirect** (...,"**INSERT INTO** table-name ...")	Can also be invoked by **SQLPrepare** and **SQLExecute**
5.2.6	**OPEN**	none	Cursor is **OPEN**ed implicitly by **SQLExecute** or **SQLExecDirect** when a **SELECT** statement is specified.
5.2.7	**SELECT ...INTO**	none	not supported.
5.2.8	Positioned **UPDATE**	**SQLExecDirect**(...,"**UPDATE** table-name **SET** column-identifier = expression ...**WHERE**	Implementation-generated <cursor-name> can be obtained by calling

		CURRENT OF cursor-name);	**SQLGetCursorName**.
5.2.9	Searched **UPDATE**	**SQLExecDirect**(..., "**UPDATE** tablename **SET** column-identifier = expression ...**WHERE** search -condition")	none

Dynamic SQL Statements

The following table lists dynamic SQL statements.

Section	SQL Statement	ODBC Function	Comments
5.3 (see 5.2.1)	Dynamic **CLOSE**	**SQLFreeStmt**(**CLOSE** option)	none
5.3(see5.2.2)	Dynamic Positioned **DELETE**	**SQLExecDirect**(..., "**DELETE FROM** table-name **WHERE CURRENT OF** cursor-name);	may also be invoked by**SQLPrepare** or **SQLExecute**
5.3(see5.2.8)	Dynamic Positioned **UPDATE**	**SQLExecDirect**(..., "**UPDATE** table-name **SET** column-identifier = expression ...**WHERE CURRENT OF** cursor-name);	can also be invoked by **SQLPrepare** and **SQLExecute**
5.3.3	**ALLOCATE DESCRIPTOR**	none	Descriptor information is implicitly allocated and attached to the hstmt by the ODBC driver. Allocation occurs at either the first call to **SQLSetParam** or at **SQLExecute** or **SQLExecDirect** time
5.3.4	**DEALLOCATE DESCRIPTOR**	**SQLFreeStmt**(DROP Option)	none
5.3.5	**DESCRIBE**	none	none
5.3.6	**EXECUTE**	**SQLExecute**	none

5.3.7	**EXECUTE IMMEDIATE**	**SQLExecDirect**	none
5.3.8	Dynamic **FETCH**	**SQLFetch**	none
5.3.9	**GET DESCRIPTOR**	**SQLNumResultCols** **SQLDescribeCol**	COUNT form VALUE form with <field-name> in {NAME, TYPE, LENGTH, PRECISION, SCALE, NULLABLE}
5.3.10	Dynamic **OPEN**	**SQLExecute**	none
5.3.11	**PREPARE**	**SQLPrepare**	none
5.3.12	**SET DESCRIPTOR**	**SQLSetParam**	**SQLSetParam** is associated with only one hstmt where a descriptor is applied to any number of statements with USING SQL DESCRIPTOR.

Transaction Control Statements

The following table lists transaction control statements.

Section	SQL Statement	ODBC Function	Comments
5.4.1	**COMMIT WORK**	**SQLTransact** (SQL_COMMIT option)	none
	ROLLBACK WORK	**SQLTransact** (SQL_ROLLBACK option)	none

5.4.2

Association Management Statements

The following table lists association management statements.

Section	SQL Statement	ODBC Function	Comments
5.5.1	**CONNECT**	**SQLConnect**	none
5.5.2	**DISCONNECT**	**SQLDisconnect**	ODBC does not support **DISCONNECT ALL**
5.5.3	**SET CONNECTION**	none	Although multiple simultaneous connections can be established, only one connection may be active at one time. To enforce this restriction in ODBC, the client implementation tracks which connection is active, and will automatically switch to a different connection if a different connection handle is specified. However, the active connection must be in a state which allows the connection context to be switched, i.e. there must not be a transaction in progress on the current connection.

Diagnostic Statement

The following table lists the GET DIAGNOSTIC statement.

Section	SQL Statement	ODBC Function	Comments
5.6.1	**GET DIAGNOSTICS**	**SQLError** **SQLRowCount**	For **SQLError**, the following fields from the diagnostics area are available:

RETURNED_SQLSTATE,
MESSAGE_TEXT, and
MESSAGE_LENGTH. For
SQLRowCount, the
ROW_COUNT
field is available.

Appendix F C Header Files

XE "Header file, sample"§XE "SQL.H header file"§The following code contains declarations used in ODBC function calls.

Header File for Core Functions

```
/*****************************************************************
 SQL.H - This is the the main include for ODBC Core functions.

 preconditions:
  #include "windows.h"

 (C) Copyright 1990, 1991 By Microsoft Corp.
******************************************************************/

#ifndef __SQL
#define __SQL

/* generally useful constants */
#define SQL_NTS                -3  /* NTS = Null Terminated String  */
#define SQL_SQLSTATE_SIZE         5  /* size of SQLSTATE            */
#define SQL_MAX_MESSAGE_LENGTH  255  /* message buffer size          */

/* RETCODEs */
#define SQL_ERROR           -1
#define SQL_INVALID_HANDLE    -2
#define SQL_NEED_DATA        99
#define SQL_NO_DATA_FOUND    100
#define SQL_SUCCESS           0
#define SQL_SUCCESS_WITH_INFO  1

/* SQLFreeStmt defines */
#define SQL_CLOSE            0
#define SQL_DROP             1
#define SQL_UNBIND           2
#define SQL_RESET_PARAMS       3

/* SQLSetParam defines */
#define SQL_NO_CONVERT  99

/* SQLTransact defines */
#define SQL_COMMIT    0
#define SQL_ROLLBACK  1

/* Standard SQL datatypes, using ANSI type numbering */
#define SQL_CHAR      1
#define SQL_NUMERIC   2
#define SQL_DECIMAL   3
#define SQL_INTEGER   4
#define SQL_SMALLINT  5
#define SQL_FLOAT     6
#define SQL_REAL      7
#define SQL_DOUBLE    8
#define SQL_VARCHAR  12

#define SQL_TYPE_MIN   1
#define SQL_TYPE_NULL  0
#define SQL_TYPE_MAX  12

/* Standard precisions and scales */
#define SQL_PREC_SMALLINT  5
```

```
#define SQL_PREC_INTEGER  10
#define SQL_PREC_FLOAT    15
#define SQL_PREC_REAL     7

/* C datatype to SQL datatype mapping    SQL types
                                         ----------------------- */
#define SQL_C_CHAR    SQL_CHAR       /* CHAR, VARCHAR, DECIMAL, NUMERIC */
#define SQL_C_LONG    SQL_INTEGER    /* INTEGER         */
#define SQL_C_SHORT   SQL_SMALLINT   /* SMALLINT        */
#define SQL_C_FLOAT   SQL_REAL       /* REAL            */
#define SQL_C_DOUBLE  SQL_DOUBLE     /* FLOAT, DOUBLE   */

/* NULL status constants.  These are used in SQLDescribeCol and
SQLColumns to describe the nullablity of a column in a table.
SQL_NULLABLE_UNKNOWN can be returned only by SQLDescribeCol.  It is used
when the DBMS's meta-data does not contain this info.  */
#define SQL_NO_NULLS        0
#define SQL_NULLABLE        1
#define SQL_NULLABLE_UNKNOWN 2

/* Special length values */
#define SQL_NULL_DATA    -1
#define SQL_LONG_DATA    -2

/* SQLError defines */
#define SQL_NULL_HENV    0
#define SQL_NULL_HDBC    0
#define SQL_NULL_HSTMT   0

/* environment specific definitions */
#define SQL_API PASCAL FAR

/* SQL portable types for C */
typedef unsigned char       UCHAR;
typedef signed char         SCHAR;
typedef long int            SDWORD;
typedef short int           SWORD;
typedef unsigned long int   UDWORD;
typedef unsigned short int  UWORD;
typedef double              SDOUBLE;
typedef long double         LDOUBLE;
typedef float               SFLOAT;

typedef void FAR *          PTR;
typedef HANDLE              HENV;
typedef HANDLE              HDBC;
typedef HANDLE              HSTMT;
typedef int                 RETCODE;

/* DLL ordinals for SQL core functions */
#define SQL_DLL_SQLALLOCCONNECT   1
#define SQL_DLL_SQLALLOCENV       2
#define SQL_DLL_SQLALLOCSTMT      3
#define SQL_DLL_SQLBINDCOL        4
#define SQL_DLL_SQLCANCEL         5
#define SQL_DLL_SQLCONNECT        6
#define SQL_DLL_SQLDESCRIBECOL    7
#define SQL_DLL_SQLDISCONNECT     8
#define SQL_DLL_SQLERROR          9
#define SQL_DLL_SQLEXECDIRECT    10
#define SQL_DLL_SQLEXECUTE       11
#define SQL_DLL_SQLFETCH         12
#define SQL_DLL_SQLFREECONNECT   13
#define SQL_DLL_SQLFREEENV       14
#define SQL_DLL_SQLFREESTMT      15
#define SQL_DLL_SQLGETCURSORNAME 16
#define SQL_DLL_SQLNUMRESULTCOLS 17
#define SQL_DLL_SQLPREPARE       18
#define SQL_DLL_SQLROWCOUNT      19
#define SQL_DLL_SQLSETCURSORNAME 20
#define SQL_DLL_SQLSETPARAM      21
#define SQL_DLL_SQLTRANSACT      22
```

```
#define SQL_NUM_FUNCTIONS       22

/* Core Function Prototypes */

RETCODE SQL_API SQLAllocEnv(
    HENV FAR * phenv);

RETCODE SQL_API SQLFreeEnv(
    HENV    henv);

RETCODE SQL_API SQLAllocConnect(
    HENV    henv,
    HDBC FAR * phdbc);

RETCODE SQL_API SQLFreeConnect(
    HDBC    hdbc);

RETCODE SQL_API SQLConnect(
    HDBC    hdbc,
    UCHAR FAR * szDSN,
    SWORD cbDSN,
    UCHAR FAR * szUID,
    SWORD cbUID,
    UCHAR FAR * szAuthStr,
    SWORD cbAuthStr);

RETCODE SQL_API SQLDisconnect(
    HDBC    hdbc);

RETCODE SQL_API SQLAllocStmt(
    HDBC    hdbc,
    HSTMT FAR * phstmt);

RETCODE SQL_API SQLFreeStmt(
    HSTMT hstmt,
    UWORD fOption);

RETCODE SQL_API SQLPrepare(
    HSTMT hstmt,
    UCHAR FAR * szSqlStr,
    SDWORD   cbSqlStr);

RETCODE SQL_API SQLSetParam(
    HSTMT hstmt,
    UWORD ipar,
    SWORD fCType,
    SWORD fSqlType,
    UDWORD  cbColDef,
    SWORD ibScale,
    PTR   rgbValue,
    SDWORD FAR * pcbValue);

RETCODE SQL_API SQLSetCursorName(
    HSTMT hstmt,
    UCHAR FAR * szCursor,
    SWORD cbCursor);

RETCODE SQL_API SQLGetCursorName(
    HSTMT hstmt,
    UCHAR FAR * szCursor,
    SWORD cbCursorMax,
    SWORD FAR * pcbCursor);

RETCODE SQL_API SQLExecute(
    HSTMT hstmt);

RETCODE SQL_API SQLExecDirect(
    HSTMT hstmt,
    UCHAR FAR * szSqlStr,
    SDWORD   cbSqlStr);

RETCODE SQL_API SQLNumResultCols(
    HSTMT hstmt,
```

```
    SWORD FAR *    pccol);

RETCODE SQL_API SQLDescribeCol(
    HSTMT  hstmt,
    UWORD icol,
    UCHAR FAR *    szColName,
    SWORD cbColNameMax,
    SWORD FAR *    pcbColName,
    SWORD FAR *    pfSqlType,
    UDWORD FAR * pcbColDef,
    SWORD FAR *    pibScale,
    SWORD FAR *    pfNullable);

RETCODE SQL_API SQLBindCol(
    HSTMT  hstmt,
    UWORD icol,
    SWORD fCType,
    PTR   rgbValue,
    SDWORD   cbValueMax,
    SDWORD FAR * pcbValue);

RETCODE SQL_API SQLFetch(
    HSTMT  hstmt);

RETCODE SQL_API SQLRowCount(
    HSTMT  hstmt,
    SDWORD FAR * pcrow);

RETCODE SQL_API SQLCancel(
    HSTMT  hstmt);

RETCODE SQL_API SQLError(
    HENV    henv,
    HDBC    hdbc,
    HSTMT  hstmt,
    UCHAR FAR * szSqlState,
    UDWORD FAR * pfNativeError,
    UCHAR FAR * szErrorMsg,
    SWORD cbErrorMsgMax,
    SWORD FAR * pcbErrorMsg);

RETCODE SQL_API SQLTransact(
    HDBC    hdbc,
    UWORD fType);

#endif  /* #ifndef __SQL */
```

Header File for Extensions

```
/*
** SQLEXT.H - This is the include for applications using
**           the Microsoft SQL Extensions
**
** (C) Copyright 1990, 1991 By Microsoft Corp.
*/

#ifndef __SQLEXT
#define __SQLEXT

#ifndef __SQL
#include "sql.h"
#endif

/* options for SQLSetStmtOption/SQLGetStmtOption */
#define SQL_HSTMT_USER_DATA     0
#define SQL_QUERY_TIMEOUT       1
#define SQL_ROWCOUNT            2
#define SQL_NOSCAN              3
#define SQL_MAX_LENGTH          4
#define SQL_ASYNC_ENABLE        5

/* options for SQLSetConnectOption/SQLGetConnectOption */
#define SQL_LOCALE              101
#define SQL_CHARSET             102
#define SQL_LOGIN_TIMEOUT       103
#define SQL_AUTOCOMMIT          104
#define SQL_ACCESS_MODE         105
#define SQL_TXN_ISOLATION       106
#define SQL_CONCURRENCY_CONTROL 107

#define SQL_OPT_TRACE           200  /* debugging: trace flag */
#define SQL_OPT_TRACEFILE       201  /* debugging: trace file name */

/* option values for Connection option SQL_CONCURRENCY_CONTROL */
#define SQL_REPEATABLE_READ     0
#define SQL_CURSOR_STABILITY    1

/* option values for Connection option SQL_TXN_ISOLATION */
#define SQL_TXN_LEVEL_0         0
#define SQL_TXN_LEVEL_1         1
#define SQL_TXN_LEVEL_2         2
#define SQL_TXN_LEVEL_3         3
#define SQL_TXN_LEVEL_4         4

/* Options for SQLDriverConnect */
#define SQL_DRIVER_NOPROMPT     0
#define SQL_DRIVER_COMPLETE     1
#define SQL_DRIVER_PROMPT       2

/* Transaction mode and scroll types -- currently not implemented */
#define SQL_READ_WRITE    0
#define SQL_READ_ONLY     1
#define SQL_LOCK          2
#define SQL_OPT_TIMESTAMP 3
#define SQL_OPT_VALUES    4

/* Async status flag */
#define SQL_STILL_EXECUTING   2

/* SQL extended datatypes */
#define SQL_DATE            9
#define SQL_TIME           10
#define SQL_TIMESTAMP      11
#define SQL_LONGVARCHAR    -1
#define SQL_BINARY         -2
#define SQL_VARBINARY      -3
#define SQL_LONGVARBINARY  -4
```

```c
#define SQL_BIGINT        -5
#define SQL_TINYINT       -6
#define SQL_BIT           -7

/* C datatype to SQL datatype mapping */
#define SQL_C_DATE        SQL_DATE
#define SQL_C_TIME        SQL_TIME
#define SQL_C_TIMESTAMP   SQL_TIMESTAMP
#define SQL_C_BINARY      SQL_BINARY
#define SQL_C_BIT         SQL_BIT
#define SQL_C_TINYINT     SQL_TINYINT

/* SQL portable types for C */
/* transfer types for DATE, TIME, TIMESTAMP */
typedef struct tagDATE_STRUCT
 {
 SWORD year;
 UWORD month;
 UWORD day;
 } DATE_STRUCT;

typedef struct tagTIME_SRUCT
 {
 UWORD hour;
 UWORD minute;
 UWORD second;
 } TIME_STRUCT;

typedef struct tagTIMESTAMP_STRUCT
 {
 SWORD year;
 UWORD month;
 UWORD day;
 UWORD hour;
 UWORD minute;
 UWORD second;
 UDWORD fraction;
 } TIMESTAMP_STRUCT;

/* define that signals parameter data callback proc in SQLSetParam */
#define SQL_DATACALLBACK  0xFFFFFFFF

/* from SQL.h -- extended data types change this */
#undef  SQL_TYPE_MIN
#define SQL_TYPE_MIN     -7
#define SQL_ALL_TYPES     0

/* Extended SQL precisions and scales  */
#define SQL_PREC_BIT      1
#define SQL_PREC_TINYINT  3
#define SQL_PREC_REAL     7

/* Extended Column Attribute defines  */
#define   SQL_READWRITE_UNKNOWN 0x0000
#define   SQL_UNSIGNED         0x0001
#define   SQL_READONLY         0x0002
#define   SQL_WRITE            0x0004
#define   SQL_MONEY_TYPE       0x0008
#define   SQL_AUTO_INCREMENT   0x0010
#define   SQL_XNULLABLE        0x0020
#define   SQL_CASE_SENSITIVE   0x0040
#define   SQL_SEARCHABLE       0x0080
#define   SQL_UPDATABLE        (SQL_READONLY | SQL_WRITE)

/* SQLExtendedFetch "fFetchType" values */
#define SQL_FETCH_NEXT      1
#define SQL_FETCH_FIRST     2
#define SQL_FETCH_LAST      3
#define SQL_FETCH_PREV      4
#define SQL_FETCH_ABSOLUTE  5
#define SQL_FETCH_RELATIVE  6
#define SQL_FETCH_RESUME    7
```

```
/* SQLExtendedFetch "rgbRowStatus" element values */
#define   SQL_ROW_SUCCESS     0
#define   SQL_ROW_DELETED     1
#define   SQL_ROW_UPDATED     2
#define   SQL_ROW_NOROW       3

/* Defines for SQLStatistics */
#define SQL_UNIQUE     0
#define SQL_NON_UNIQUE 1
#define SQL_ENSURE     1
#define SQL_QUICK      0

/*  Column types and scopes in SQLSpecialColumns.  */
#define SQL_BEST_ROWID 1
#define SQL_ROWVER     2

#define SQL_SCOPE_CURROW       0
#define SQL_SCOPE_TRANSACTION  1
#define SQL_SCOPE_SESSION      2

/* Defines for SQLSetPos */

#define   SQL_ENTIRE_ROWSET      0

/* Info type defines for SQLGetInfo() */
#define SQL_DBMS_NAME                    0
#define SQL_DBMS_VER                     1
#define SQL_DRIVER_VER                   2
#define SQL_DRIVER_NAME                  3
#define SQL_ODBC_VER                     4
#define SQL_SERVER_NAME                  5
#define SQL_DATA_SOURCE_NAME             6
#define SQL_DATABASE_NAME                7
#define SQL_USER_NAME                    8
#define SQL_OWNER_TERM                   9
#define SQL_TABLE_TERM                   10
#define SQL_MAX_OWNER_NAME_LEN           11
#define SQL_MAX_TABLE_NAME_LEN           12
#define SQL_MAX_QUALIFIER_NAME_LEN       13
#define SQL_MAX_COLUMN_NAME_LEN          14
#define SQL_IDENTIFIER_CASE              15
#define SQL_DEFAULT_TXN_ISOLATION        16
#define SQL_COLLATION_SEQ                17
#define SQL_SAVEPOINT_SUPPORT            18
#define SQL_MULT_RESULT_SETS             19
#define SQL_ACCESSIBLE_TABLES            20
#define SQL_MULTIPLE_ACTIVE_TXN          21
#define SQL_OUTER_JOINS                  22
#define SQL_SAG_CLI_CONFORMANCE          23
#define SQL_ODBC_CONFORMANCE             24
#define   SQL_IDENTIFIER_QUOTE_CHAR      25
#define   SQL_COMMIT_CURSOR_BEHAVIOR     26
#define   SQL_ROLLBACK_CURSOR_BEHAVIOR   27
#define   SQL_EXPRESSIONS_IN_ORDER_BY    28
#define   SQL_SYNTAX_COMPATIBILITY       29
#define   SQL_ACTIVE_STATEMENTS          30
#define   SQL_ROW_UPDATES                31
#define   SQL_CONVERT_FUNCTIONS          32
#define   SQL_STRING_FUNCTIONS           33
#define   SQL_NUMERIC_FUNCTIONS          34
#define   SQL_TIMEDATE_FUNCTIONS         35
#define   SQL_SYSTEM_FUNCTIONS           36
#define   SQL_SCROLL_OPTIONS             37
#define   SQL_FETCH_DIRECTION            38
#define   SQL_SCROLL_CONCURRENCY         39
#define   SQL_DRIVER_HENV                40
#define   SQL_DRIVER_HDBC                41
#define   SQL_DRIVER_HSTMT               42
#define   SQL_CONVERT_CHAR               43
#define   SQL_CONVERT_NUMERIC            44
#define   SQL_CONVERT_DECIMAL            45
#define   SQL_CONVERT_INTEGER            46
#define   SQL_CONVERT_SMALLINT           47
```

```
#define   SQL_CONVERT_FLOAT               48
#define   SQL_CONVERT_REAL                49
#define   SQL_CONVERT_DOUBLE              50
#define   SQL_CONVERT_VARCHAR             51
#define   SQL_CONVERT_LONGVARCHAR         52
#define   SQL_CONVERT_BINARY              53
#define   SQL_CONVERT_VARBINARY           54
#define   SQL_CONVERT_BIT                 55
#define   SQL_CONVERT_TINYINT             56
#define   SQL_CONVERT_BIGINT              57
#define   SQL_CONVERT_DATE                58
#define   SQL_CONVERT_TIME                59
#define   SQL_CONVERT_TIMESTAMP           60

/* "SQL_CONVERT_" return value bitmasks */

#define   SQL_CVT_CHAR                    0x00000001
#define   SQL_CVT_NUMERIC                 0x00000002
#define   SQL_CVT_DECIMAL                 0x00000004
#define   SQL_CVT_INTEGER                 0x00000008
#define   SQL_CVT_SMALLINT                0x00000010
#define   SQL_CVT_FLOAT                   0x00000020
#define   SQL_CVT_REAL                    0x00000040
#define   SQL_CVT_DOUBLE                  0x00000080
#define   SQL_CVT_VARCHAR                 0x00000100
#define   SQL_CVT_LONGVARCHAR             0x00000200
#define   SQL_CVT_BINARY                  0x00000400
#define   SQL_CVT_VARBINARY               0x00000800
#define   SQL_CVT_BIT                     0x00001000
#define   SQL_CVT_TINYINT                 0x00002000
#define   SQL_CVT_BIGINT                  0x00004000
#define   SQL_CVT_DATE                    0x00008000
#define   SQL_CVT_TIME                    0x00010000
#define   SQL_CVT_TIMESTAMP               0x00020000

/* Conversion functions */
#define   SQL_FN_CVT_CONVERT              0x00000001

/* String functions */

#define   SQL_FN_STR_CONCAT               0x00000001
#define   SQL_FN_STR_INSERT               0x00000002
#define   SQL_FN_STR_LEFT                 0x00000004
#define   SQL_FN_STR_LTRIM                0x00000008
#define   SQL_FN_STR_LENGTH               0x00000010
#define   SQL_FN_STR_LOCATE               0x00000020
#define   SQL_FN_STR_LCASE                0x00000040
#define   SQL_FN_STR_REPEAT               0x00000080
#define   SQL_FN_STR_REPLACE              0x00000100
#define   SQL_FN_STR_RIGHT                0x00000200
#define   SQL_FN_STR_RTRIM                0x00000400
#define   SQL_FN_STR_SUBSTRING            0x00000800
#define   SQL_FN_STR_UCASE                0x00001000
#define   SQL_FN_STR_ASCII                0x00002000
#define   SQL_FN_STR_CHAR                 0x00004000

/* Numeric functions */

#define   SQL_FN_NUM_ABS                  0x00000001
#define   SQL_FN_NUM_ACOS                 0x00000002
#define   SQL_FN_NUM_ASIN                 0x00000004
#define   SQL_FN_NUM_ATAN                 0x00000008
#define   SQL_FN_NUM_ATAN2                0x00000010
#define   SQL_FN_NUM_CEILING              0x00000020
#define   SQL_FN_NUM_COS                  0x00000040
#define   SQL_FN_NUM_COT                  0x00000080
#define   SQL_FN_NUM_EXP                  0x00000100
#define   SQL_FN_NUM_FLOOR                0x00000200
#define   SQL_FN_NUM_LOG                  0x00000400
#define   SQL_FN_NUM_MOD                  0x00000800
#define   SQL_FN_NUM_SIGN                 0x00001000
#define   SQL_FN_NUM_SIN                  0x00002000
#define   SQL_FN_NUM_SQRT                 0x00004000
```

```
#define   SQL_FN_NUM_TAN                    0x00008000
#define   SQL_FN_NUM_PI                     0x00010000
#define   SQL_FN_NUM_RAND                   0x00020000

/* Time/date functions */

#define   SQL_FN_TD_NOW                     0x00000001
#define   SQL_FN_TD_CURDATE                 0x00000002
#define   SQL_FN_TD_DAYOFMONTH              0x00000004
#define   SQL_FN_TD_DAYOFWEEK               0x00000008
#define   SQL_FN_TD_DAYOFYEAR               0x00000010
#define   SQL_FN_TD_MONTH                   0x00000020
#define   SQL_FN_TD_QUARTER                 0x00000040
#define   SQL_FN_TD_WEEK                    0x00000080
#define   SQL_FN_TD_YEAR                    0x00000100
#define   SQL_FN_TD_CURTIME                 0x00000200
#define   SQL_FN_TD_HOUR                    0x00000400
#define   SQL_FN_TD_MINUTE                  0x00000800
#define   SQL_FN_TD_SECOND                  0x00001000

/* System functions */

#define   SQL_FN_SYS_USERNAME               0x00000001
#define   SQL_FN_SYS_DBNAME                 0x00000002
#define   SQL_FN_SYS_IFNULL                 0x00000004

/* Scroll option masks */

#define   SQL_SO_FORWARD_ONLY               0x0001
#define   SQL_SO_KEYSET_DRIVEN              0x0002
#define   SQL_SO_DYNAMIC                    0x0004
#define   SQL_SO_MIXED                      0x0008

/* Scroll concurrency option masks */

#define   SQL_SCCO_READ_ONLY                0x0001
#define   SQL_SCCO_LOCK                     0x0002
#define   SQL_SCCO_OPT_TIMESTAMP            0x0004
#define   SQL_SCCO_OPT_VALUES               0x0008

/* Fetch direction option masks */

#define   SQL_FD_FETCH_NEXT                 0x0001
#define   SQL_FD_FETCH_FIRST                0x0002
#define   SQL_FD_FETCH_LAST                 0x0004
#define   SQL_FD_FETCH_PREV                 0x0008
#define   SQL_FD_FETCH_ABSOLUTE             0x0010
#define   SQL_FD_FETCH_RELATIVE             0x0020
#define   SQL_FD_FETCH_RESUME               0x0040

/* Defines for SetScrollOptions */

#define   SQL_SCROLL_FORWARD_ONLY    0
#define   SQL_SCROLL_KEYSET_DRIVEN   1
#define   SQL_SCROLL_DYNAMIC         2

/* DLL ordinals for SQL extensions */

#define SQL_EXT_DLL_START         40

#define   SQL_DLL_SQLCOLATTRIBUTES   40
#define SQL_DLL_SQLCOLUMNPRIVILEGES 41
#define SQL_DLL_SQLCOLUMNS         42
#define SQL_DLL_SQLDATASOURCES     43
#define SQL_DLL_SQLDESCRIBEPARAM   44
#define SQL_DLL_SQLDRIVERCONNECT   45
#define SQL_DLL_SQLEXTENDEDFETCH   46
#define SQL_DLL_SQLFOREIGNKEYS     47
#define SQL_DLL_SQLGETCONNECTOPTION 48
#define SQL_DLL_SQLGETDATA         49
#define SQL_DLL_SQLGETFUNCTIONS    50
#define SQL_DLL_SQLGETINFO         51
#define SQL_DLL_SQLGETSTMTOPTION   52
```

```
#define SQL_DLL_SQLGETTYPEINFO      53
#define SQL_DLL_SQLMORERESULTS      54
#define SQL_DLL_SQLNATIVESQL        55
#define SQL_DLL_SQLNUMPARAMS        56
#define SQL_DLL_SQLPARAMDATA        57
#define SQL_DLL_SQLPARAMOPTIONS     58
#define SQL_DLL_SQLPRIMARYKEYS      59
#define SQL_DLL_SQLPUTDATA          60
#define SQL_DLL_SQLSETCONNECTOPTION 61
#define SQL_DLL_SQLSETPOS           62
#define SQL_DLL_SQLSETSCROLLOPTIONS 63
#define SQL_DLL_SQLSETSTMTOPTION    64
#define SQL_DLL_SQLSPECIALCOLUMNS   65
#define   SQL_DLL_SQLSTATISTICS       66
#define   SQL_DLL_SQLTABLEPRIVILEGES  67
#define   SQL_DLL_SQLTABLES           68

#define SQL_NUM_EXTENSIONS (68-SQL_EXT_DLL_START+1)

/* MS-Extended Function Prototypes */

RETCODE SQL_API SQLDriverConnect(
    HDBC    hdbc,
    HWND    hwnd,
    UCHAR FAR * szConnStrIn,
    SWORD cbConnStrIn,
    UCHAR FAR * szConnStrOut,
    SWORD cbConnStrOutMax,
    SWORD FAR * pcbConnStrOut,
    UWORD fDriverCompletion);

RETCODE SQL_API SQLGetFunctions(
    HDBC    hdbc,
    UWORD fFunction,
    UWORD FAR *pfExists);

RETCODE SQL_API SQLGetInfo(
    HDBC    hdbc,
    UWORD fInfoType,
    PTR    rgbInfoValue,
    SWORD cbInfoValueMax,
    SWORD FAR *pcbInfoValue);

RETCODE SQL_API SQLDataSources(
    HENV    henv,
    UWORD fDirection,
    UCHAR FAR * szDSN,
    SWORD cbDSNMax,
    SWORD FAR * pcbDSN,
    UCHAR FAR * szDescription,
    SWORD cbDescriptionMax,
    SWORD FAR * pcbDescription);

RETCODE SQL_API SQLGetTypeInfo(
    HSTMT  hstmt,
    SWORD fSqlType);

RETCODE SQL_API SQLTables(
    HSTMT  hstmt,
    UCHAR FAR * szTableQualifier,
    SWORD cbTableQualifier,
    UCHAR FAR * szTableOwner,
    SWORD cbTableOwner,
    UCHAR FAR * szTableName,
    SWORD cbTableName,
    UCHAR FAR * szTableType,
    SWORD cbTableType);

RETCODE SQL_API SQLColumns(
    HSTMT  hstmt,
    UCHAR FAR * szTableQualifier,
    SWORD cbTableQualifier,
    UCHAR FAR * szTableOwner,
```

```
    SWORD cbTableOwner,
    UCHAR FAR * szTableName,
    SWORD cbTableName,
    UCHAR FAR * szColumnName,
    SWORD cbColumnName);

RETCODE SQL_API SQLStatistics(
    HSTMT  hstmt,
    UCHAR FAR * szTableQualifier,
    SWORD cbTableQualifier,
    UCHAR FAR * szTableOwner,
    SWORD cbTableOwner,
    UCHAR FAR * szTableName,
    SWORD cbTableName,
    UWORDfUnique,
    UWORDfAccuracy);

RETCODE SQL_API SQLPrimaryKeys(
    HSTMT  hstmt,
    UCHAR FAR * szTableQualifier,
    SWORD cbTableQualifier,
    UCHAR FAR * szTableOwner,
    SWORD cbTableOwner,
    UCHAR FAR * szTableName,
    SWORD cbTableName);

RETCODE SQL_API SQLForeignKeys(
    HSTMT  hstmt,
    UCHAR FAR * szPkTableQualifier,
    SWORD cbPkTableQualifier,
    UCHAR FAR * szPkTableOwner,
    SWORD cbPkTableOwner,
    UCHAR FAR * szPkTableName,
    SWORD cbPkTableName,
    UCHAR FAR * szFkTableQualifier,
    SWORD cbFkTableQualifier,
    UCHAR FAR * szFkTableOwner,
    SWORD cbFkTableOwner,
    UCHAR FAR * szFkTableName,
    SWORD cbFkTableName);

RETCODE SQL_API SQLTablePrivileges(
    HSTMT  hstmt,
    UCHAR FAR * szTableQualifier,
    SWORD cbTableQualifier,
    UCHAR FAR * szTableOwner,
    SWORD cbTableOwner,
    UCHAR FAR * szTableName,
    SWORD cbTableName);

RETCODE SQL_API SQLColumnPrivileges(
    HSTMT  hstmt,
    UCHAR FAR * szTableQualifier,
    SWORD cbTableQualifier,
    UCHAR FAR * szTableOwner,
    SWORD cbTableOwner,
    UCHAR FAR * szTableName,
    SWORD cbTableName,
    UCHAR FAR * szColumnName,
    SWORD cbColumnName);

RETCODE SQL_API SQLNativeSql(
    HDBC    hdbc,
    UCHAR FAR * szSqlStrIn,
    SDWORD   cbSqlStrIn,
    UCHAR FAR * szSqlStr,
    SDWORD   cbSqlStrMax,
    SDWORD FAR * pcbSqlStr);

RETCODE SQL_API SQLDescribeParam(
    HSTMT  hstmt,
    UWORDipar,
    UCHAR FAR * szColName,
```

```
    SWORD cbColNameMax,
    SWORD FAR * pcbColName,
    SWORD FAR * pfSqlType,
    UDWORD FAR * pcbColDef,
    SWORD FAR * pibScale,
    SWORD FAR * pfNullable);

RETCODE SQL_API SQLGetData(
    HSTMT      hstmt,
    UWORD      icol,
    SWORD      fCType,
    PTR        rgbValue,
    SDWORD     cbValueMax,
    SDWORD FAR *pcbValue);

RETCODE SQL_API SQLExtendedFetch(
    HSTMT  hstmt,
    UWORDfFetchType,
    SDWORD   irow,
    UDWORD FAR * pcrow,
    UWORD FAR * rgfRowStatus);

RETCODE SQL_API SQLSetScrollOptions(
    HSTMT  hstmt,
    UWORDfConcurrency,
    SDWORD   crowKeyset,
    UWORDcrowRowset);

RETCODE SQL_API SQLSetPos(
    HSTMT  hstmt,
    UWORDirow,
    BOOL     fRefresh,
    BOOL     fLock);

RETCODE SQL_API SQLSetConnectOption(
    HDBC     hdbc,
    UWORDfOption,
    UDWORD   vParam);

RETCODE SQL_API SQLGetConnectOption(
    HDBC     hdbc,
    UWORDfOption,
    PTR   pvParam);

RETCODE SQL_API SQLSetStmtOption(
    HSTMT  hstmt,
    UWORDfOption,
    UDWORD   vParam);

RETCODE SQL_API SQLGetStmtOption(
    HSTMT  hstmt,
    UWORDfOption,
    PTR   pvParam);

RETCODE SQL_API SQLMoreResults(
    HSTMT  hstmt);

RETCODE SQL_API SQLSpecialColumns(
    HSTMT  hstmt,
    UWORDfColType,
    UCHAR FAR * szTableQualifier,
    SWORD cbTableQualifier,
    UCHAR FAR * szTableOwner,
    SWORD cbTableOwner,
    UCHAR FAR * szTableName,
    SWORD cbTableName,
    UWORDfScope,
    UWORDfNullable);

RETCODE SQL_API SQLColAttributes(
    HSTMT  hstmt,
    UWORDicol,
    UWORD FAR * pfAttributes,
```

```
    UCHAR FAR * szTypeName,
    SWORD cbTypeNameMax,
    SWORD FAR * pcbTypeName);

RETCODE SQL_API SQLPutData(
    HSTMT  hstmt,
    PTR   rgbValue,
    SDWORD   cbValue);

RETCODE SQL_API SQLParamData(
    HSTMT  hstmt,
    PTR   rgbValue);

RETCODE SQL_API SQLParamOptions(
    HSTMT  hstmt,
    UDWORD  crow,
    UDWORD FAR * pirow);

RETCODE SQL_API SQLNumParams(
    HSTMT  hstmt,
    SWORD FAR * pcpar);

#endif /* __SQLEXT */
```

Appendix G Canonical Functions

ODBC specifies five types of canonical functions:

- String functions
- Numeric functions
- Time and date functions
- System functions
- Data type conversion functions

The following sections list functions by function type. Descriptions include associated syntax.

This appendix also includes tables that map canonical functions to five DBMS products.

String Functions

The following table lists string manipulation functions.

Character string literals used as arguments to canonical functions must be bounded by single quotes.

Arguments denoted as *string_exp* can be the name of a column, a string literal, or the result of another canonical function, where the underlying data type can be represented as SQL_CHAR, SQL_VARCHAR or SQL_LONGVARCHAR.

Arguments denoted as *start*, *length* or *count* can be a numeric literal or the result of another canonical function, where the underlying data type can be represented as SQL_TINYINT, SQL_SMALLINT or SQL_INTEGER.

Function	Description
ASCII(*string_exp*)	Returns the ASCII code value of the leftmost character of *string_exp* as an integer.
CHAR(*code*)	Returns the character that has the ASCII code value specified by *code*. The value of *code* should be between 0 and 255; otherwise, the return value is DBMS-dependent.
CONCAT(*string_exp1*,*string_exp*	Returns a character string that is the result of concatenating *string_exp2* to *string_exp1*.

2)	The resulting string is database dependent. For example, if one of the strings were NULL, DB2 would return NULL, but SQL Server would return the non-NULL string.
INSERT(*string_exp1*,*start*,*length*, *string_exp2*)	Returns a character string where *length* characters have been deleted from *string_exp1* beginning at *start* and where *string_exp2* has been inserted into *string_exp*,beginning at *start*.
LEFT(*string_exp*,*count*)	Returns the leftmost *count* of characters of *string_exp*.

Function	Description
LTRIM(*string_exp*)	Returns the characters of *string_exp*, with leading blanks removed.
LENGTH(*string_exp*)	Returns the number of characters in *string_exp*, excluding trailing blanks and the string termination character.
LOCATE(*string_exp1*, *string_exp2[,start]*)	Returns the starting position of the first occurrence of *string_exp1* within *string_exp2*. The search for the first occurrence of *string_exp1* begins with the first character position in *string_exp2* unless the optional argument, *start*, is specified. If *start* is specified, the search begins with the character position indicated by the value of *start*. The first character position in *string_exp2* is indicated by the value 1. If *string_exp1* is not found within *string_exp2*, the value 0 is returned.
LCASE(*string_exp*)	Converts all upper case characters in *string_exp* to lower case.
REPEAT(*string_exp*,*count*)	Returns a character string composed of *string_exp* repeated *count* times.
REPLACE(*string_exp1*,*string_exp2*, *string_exp3*)	Replaces all occurrences of *string_exp2* in *string_exp1* with *string_exp3*.
RIGHT(*string_exp*,*count*)	Returns the rightmost *count* of characters of *string_exp*.
RTRIM(*string_exp*)	Returns the characters of *string_exp* with trailing blanks removed.

SUBSTRING(*string_exp*,*start*,*length*)	Returns a character string that is derived from *string_exp* beginning at the character position specified by *start* for *length* characters.
UCASE(*string_exp*)	Converts all lower case characters in *string_exp* to upper case.

Numeric Functions

The following table describes numeric functions that are included in the ODBC canonical function set.

Arguments denoted as *numeric_exp* can be the name of a column, the result of another canonical function, or a numeric literal, where the underlying data type could be represented as SQL_NUMERIC, SQL_DECIMAL, SQL_TINYINT, SQL_SMALLINT, SQL_INTEGER, SQL_BIGINT, SQL_FLOAT, SQL_REAL, SQL_DOUBLE.

Arguments denoted as *float_exp* can be the name of a column, the result of another canonical function, or a numeric literal, where the underlying data type can be represented as SQL_FLOAT.

Arguments denoted as *integer_exp* can be the name of a column, the result of another canonical function, or a numeric literal, where the underlying data type can be represented as SQL_TINYINT, SQL_SMALLINT, SQL_INTEGER or SQL_BIGINT.

Function	Description
ABS(*numeric_exp*)	Returns the absolute value of *numeric_exp*.
ACOS(f*loat_exp*)	Returns the arccosine of *float_exp* as an angle, expressed in radians.
ASIN(*float_exp*)	Returns the arcsine of *float_exp* as an angle, expressed in radians.
ATAN(*float_exp*)	Returns the arctangent of *float_exp* as an angle, expressed in radians.
ATAN2(*float_exp1*, *float_exp2*)	Returns the arctangent of the x and y coordinates, specified by *float_exp1* and *float_exp2*, respectively, as an angle, expressed in radians.
CEILING(*numeric_exp*)	Returns the smallest integer greater than or equal to *numeric_exp*.

COS(*float_exp*)	Returns the cosine of *float_exp* as an angle, expressed in radians.
COT(*float_exp*)	Returns the cotangent of *float_exp* as an angle, expressed in radians.
EXP(*float_exp*)	Returns the exponential value of *float_exp*.
FLOOR(*numeric_exp*)	Returns largest integer less than or equal to *numeric_exp*.
LOG(*float_exp*)	Returns the natural logarithm of *float_exp*.
MOD(*integer_exp1*,*integer_exp2*)	Returns the remainder (modulus) of *integer_exp1* divided by *integer_exp2*.
PI()	Returns the constant value of pi as a floating point value.
RAND([*integer_exp*])	Returns a random floating point value using *integer_exp* as the option seed value.

Function	Description
SIGN(*numeric_exp*)	Returns an indicator or the sign of *numeric_exp*. If *numeric_exp* is less than zero, -1 is returned. If *numeric_exp* equals zero, 0 is returned. If *numeric_exp* is greater than zero, 1 is returned.
SIN(*float_exp*)	Returns the sine of *float_exp* as an angle, expressed in radians.
SQRT(*float_exp*)	Returns the square root of *float_exp*.
TAN(*float_exp*)	Returns the tangent of *float_exp* as an angle, expressed in radians.

Time and Date Functions

The following table lists time and date functions that are included in the ODBC canonical function set.

Arguments denoted as *date_exp* can be the name of a column, the result of another canonical function or a date literal, where the underlying data type could be represented as SQL_CHAR, SQL_VARCHAR, SQL_DATE or SQL_TIMESTAMPSQL_DATETIME.

Arguments denoted as *time_exp* can be the name of a column, the result of another canonical function or a time literal, where the underlying data type could be represented as SQL_CHAR, SQL_VARCHAR, SQL_TIME or SQL_TIMESTAMPSQL_DATETIME.

Values returned are represented as ODBC data types.

Function	Description
NOW()	Returns current date and time as a timestamp value.
CURDATE()	Returns the current date as a date value.
DAYOFMONTH(*date_exp*)	Returns the day of the month in *date_exp* as an integer value in the range of 1 - 31.
DAYOFWEEK(*date_exp*)	Returns the day to the week in *date_exp* as an integer value in the range of 1 - 7, where 1 represents Sunday.
DAYOFYEAR(*date_exp*)	Returns the day of the year in *date_exp* as an integer value in the range of 1 - 366.
MONTH(*date_exp*)	Returns the month in *date_exp* as an integer value in the range of 1 - 12.
QUARTER(*date_exp*)	Returns the quarter in *date_exp* as an integer value in the range of 1 - 4.

WEEK(*date_exp*)	Returns the week of the year in *date_exp* as an integer value in the range of 1 - 53.
YEAR(*date_exp*)	Returns the year in *date_exp* as an integer value. The range is going to be DBMS dependent.
CURTIME()	Returns the current local time as a time value.
HOUR(*time_exp*)	Returns the hour in *time_exp* as an integer value in the range of 0 - 23.
MINUTE(*time_exp*)	Returns the minute in *time_exp* as an integer value in the range of 0 - 59.
SECOND(*time_exp*)	Returns the second in *time_exp* as an integer value in the range of 0 - 59.

System Functions

The following table lists system functions that are included in the ODBC canonical function set.

Arguments denoted as *exp* can be the name of a column, the result of another canonical function, or a literal, where the underlying data type could be represented as SQL_NUMERIC, SQL_DECIMAL, SQL_TINYINT, SQL_SMALLINT, SQL_INTEGER, SQL_BIGINT, SQL_FLOAT, SQL_REAL, SQL_DOUBLE. SQL_DATE, SQL_TIME or SQL_TIMESTAMP.

Arguments denoted as *value* can be a literal constant, where the underlying data type can be represented as SQL_NUMERIC, SQL_DECIMAL, SQL_TINYINT, SQL_SMALLINT, SQL_INTEGER, SQL_BIGINT, SQL_FLOAT, SQL_REAL, SQL_DOUBLE. SQL_DATE, SQL_TIME or SQL_TIMESTAMP.

Values returned are represented as ODBC data types.

Function	Description
USER()	Returns the user's authorization id. (The user's authorization id is also available via **SQLGetInfo** by specifying the information type: SQL_USER_NAME.)
DATABASE()	Returns the name of the database corresponding to the connection handle (hdbc). (The name of the database is also available via **SQLGetInfo** by specifying the information type: SQL_DATABASE_NAME.).
IFNULL(*exp*,*value*)	If *exp* is null, *value* is returned. If *exp* is not null, *exp* is returned. The possible data type(s) of *value* must be compatible with the data type of *exp*.

Explicit Data Type Conversion

Explicit data type conversion is specified in terms of ODBC SQL data type definitions.

The canonical form of the explicit data type conversion function does not restrict conversions. The validity of specific conversions of one data type to another data type will be determined by each driver specific implementation. The driver will as it translates the canonical form into the native form reject those conversions that although legal in the canonical form are not supported by the DBMS. The ODBC function **SQLGetTypeInfo** provides a way to inquire about conversions supported by the data source.

The format of the CONVERT function is:

CONVERT(value_exp,data_type)

The function returns the value specified by value_exp converted to the specified data_type.

The canonical form of the explicit data type conversion function does not support specification of conversion format. If specification of explicit formats is supported by the underlying data source, a driver must specify a default value or implement format specification.

The argument value_exp can be a column name, the result of another canonical function, or a numeric or string literal. For example:

```
--*( vendor(Microsoft),product(ODBC) fn (CONVERT--
*( vendor(Microsoft),product(ODBC), fn (CURDATE()--*,SQL_CHAR)--*)
```

for a numeric or string literal.

The argument data_type can be any one of the ODBC defined SQL data types specified in Appendix D; for example, SQL_CHAR, SQL_FLOAT, SQL_BIGINT, or SQL_DATE.

The following two examples illustrate the use of the CONVERT function. These examples assume there exists a table called EMPLOYEES, with an EMPNO column of type SQL_SMALLINT and an EMPNAME column of type SQL_CHAR.

If an application specifies the following:

```
SELECT EMPNO FROM EMPLOYEES WHERE
--
*(vendor(Microsoft),product(ODBC),fn(CONVERT(EMPNO,SQL_CHAR))--
*) LIKE '1%'
```

or its equivalent in shorthand form:

```
SELECT EMPNO FROM EMPLOYEES WHERE
 {fn CONVERT(EMPNO,SQL_CHAR)} LIKE '1%'
```

A driver that supports an ORACLE database would translate the request to:

```
SELECT EMPNO FROM EMPLOYEES WHERE to_char(EMPNO) LIKE
"1%"
```

A driver that supports a SQL Server database would translate the request to:

```
SELECT EMPNO FROM EMPLOYEES WHERE convert(char,EMPNO)
LIKE "1%"
```

If an application specifies the following:

```
SELECT
--*(vendor(Microsoft),product(ODBC),fn(ABS(EMPNO))--*),
--
*(vendor(Microsoft),product(ODBC),fn(CONVERT(EMPNAME,SQL_SMAL
LINT))--*)
FROM EMPLOYEES WHERE EMPNO <> 0
```

or its equivalent in shorthand form:

```
SELECT {fn ABS(EMPNO)}, {fn
CONVERT(EMPNAME,SQL_SMALLINT)}
```

```
FROM EMPLOYEES WHERE EMPNO <> 0
```

A driver that supports an ORACLE database would translate the request to:

```
SELECT abs(EMPNO), to number(EMPNAME) FROM EMPLOYEES
WHERE EMPNO <> 0
```

A driver that supports a SQL Server database would translate the request to:

```
SELECT abs(EMPNO), convert(smallint, EMPNAME) FROM EMPLOYEES
WHERE
EMPNO != 0
```

A driver that supports an Ingres database would translate the request to:

```
SELECT abs(EMPNO), int2(EMPNAME) FROM EMPLOYEES WHERE
EMPNO <> 0
```

Both ORACLE and SQL Server support an optional formatting argument that is not supported by ODBC.

Index

µ