ODBC API Help Contents

Function Summary

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.

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

Connecting to a Data Source

Function Name	Purpose
SQLAllocEnv	Obtains an environment handle. One environment handle is used for one or more connections.
SQLAllocConnect	Obtains a connection handle.
SQLConnect	Connects to a specific driver by data source name, user ID, and password.
SQLDriverConnect	Connects to a specific driver by connection string or requests that the Driver Manager and driver display connection dialog boxes for the user.
SQLBrowseConnect	Returns successive levels of connection attributes and valid attribute values. When a value has been specified for each connection attribute, connects to the data source.

Obtaining Information about a Driver and Data Source

Function Name	Purpose
SQLDataSources	Returns the list of available data sources.
SQLDrivers	Returns the list of installed drivers and their attributes.
SQLGetInfo	Returns information about a specific driver and data source.
SQLGetFunctions	Returns supported driver functions.
SQLGetTypeInfo	Returns information about supported data types.

Setting and Retrieving Driver Options

Function Name	Purpose
SQLSetConnectOption	Sets a connection option.
SQLGetConnectOption	Returns the value of a connection option.
SQLSetStmtOption	Sets a statement option.
SQLGetStmtOption	Returns the value of a statement option.

Preparing SQL Requests

Function Name	Purpose
SQLAllocStmt	Allocates a statement handle.
SQLPrepare	Prepares an SQL statement for later execution.
SQLBindParameter	Assigns storage for a parameter in an SQL statement.
SQLParamOptions	Specifies the use of multiple values for parameters.
SQLGetCursorName	Returns the cursor name associated with a statement handle.
SQLSetCursorName	Specifies a cursor name.

SQLSetScrollOptions	Sets options that control cursor behavior.

Submitting Requests

Function Name	**Purpose**
SQLExecute	Executes a prepared statement.
SQLExecDirect	Executes a statement.
SQLNativeSql	Returns the text of an SQL statement as translated by the driver.
SQLDescribeParam	Returns the description for a specific parameter in a statement.
SQLNumParams	Returns the number of parameters in a statement.
SQLParamData	Used in conjunction with **SQLPutData** to supply parameter data at execution time. (Useful for long data values.)
SQLPutData	Send part or all of a data value for a parameter. (Useful for long data values.)

Retrieving Results and Information about Results

Function Name	**Purpose**
SQLRowCount	Returns the number of rows affected by an insert, update, or delete request.
SQLNumResultCols	Returns the number of columns in the result set.
SQLDescribeCol	Describes a column in the result set.
SQLColAttributes	Describes attributes of a column in the result set.
SQLBindCol	Assigns storage for a result column and specifies the data type.
SQLFetch	Returns a result row.
SQLExtendedFetch	Returns multiple result rows.
SQLGetData	Returns part or all of one column of one row of a result set. (Useful for long data values.)
SQLSetPos	Positions a cursor within a fetched block of data.
SQLMoreResults	Determines whether there are more result sets available and, if so, initializes processing for the next result set.
SQLError	Returns additional error or status information.

Obtaining Information about the Data Source's System Tables (Catalog Functions)

Function Name	**Purpose**
SQLColumnPrivileges	Returns a list of columns and associated privileges for one or more tables.
SQLColumns	Returns the list of column names in specified tables.
SQLForeignKeys	Returns a list of column names that comprise foreign keys, if they exist for a specified table.
SQLPrimaryKeys	Returns the list of column name(s) that comprise the primary key for a table.
SQLProcedureColumns	Returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures.
SQLProcedures	Returns the list of procedure names stored in a

	specific data source.
SQLSpecialColumns	Returns information about the optimal set of columns that uniquely identifies a row in a specified table, or the columns that are automatically updated when any value in the row is updated by a transaction.
SQLStatistics	Returns statistics about a single table and the list of indexes associated with the table.
SQLTablePrivileges	Returns a list of tables and the privileges associated with each table.
SQLTables	Returns the list of table names stored in a specific data source.

Terminating a Statement

Function Name	**Purpose**
SQLFreeStmt	Ends statement processing and closes the associated cursor, discards pending results, and, optionally, frees all resources associated with the statement handle.
SQLCancel	Cancels an SQL statement.
SQLTransact	Commits or rolls back a transaction.

Terminating a Connection

Function Name	**Purpose**
SQLDisconnect	Closes the connection.
SQLFreeConnect	Releases the connection handle.
SQLFreeEnv	Releases the environment handle.

SQLAllocConnect (Core, ODBC 1.0)

Code Example Related Functions

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

Syntax

RETCODE **SQLAllocConnect**(*henv*, *phdbc*)

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_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

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

Diagnostics

When **SQLAllocConnect** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLAllocConnect** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	(DM) The Driver Manager was unable to allocate memory for the connection handle. The driver was unable to allocate memory for the connection handle.
S1009	Invalid argument value	(DM) The argument *phdbc* was a null pointer.

Comments

A connection handle references information such as the valid statement handles on the connection and whether a transaction is currently open. To request a connection handle, an application passes the address of an *hdbc* to **SQLAllocConnect**. The driver allocates memory for the connection information and stores the value of the associated handle in the *hdbc*. On operating systems that support multiple threads, applications can use the same *hdbc* on different threads and drivers must therefore support safe, multithreaded access to this information. 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**, **SQLBrowseConnect**, or **SQLDriverConnect**. (For more information, see the description of the **SQLConnect** function.)

If the application calls **SQLAllocConnect** with a pointer to a valid *hdbc*, the driver overwrites the *hdbc* without regard to its previous contents.

Code Example

See **SQLBrowseConnect** and **SQLConnect**.

Related Functions

SQLConnect

SQLFreeConnect

SQLAllocEnv (Core, ODBC 1.0)

Code Example Related Functions

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*)

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 **SQLAllocEnv** returns SQL_ERROR, it will set the *henv* referenced by *phenv* to SQL_NULL_HENV. In this case, the application can assume that the error was a memory allocation error.

Diagnostics

A driver cannot return SQLSTATE values directly after the call to **SQLAllocEnv**, since no valid handle will exist with which to call **SQLError**.

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**, **SQLBrowseConnect**, or **SQLDriverConnect**. If an error occurs in the driver-level **SQLAllocEnv** function, then the Driver Manager-level **SQLConnect**, **SQLBrowseConnect**, or **SQLDriverConnect** function returns SQL_ERROR. A subsequent call to **SQLError** with *henv*, SQL_NULL_HDBC, and SQL_NULL_HSTMT returns SQLSTATE IM004 (Driver's **SQLAllocEnv** failed), followed by one of the following errors from the driver:

- SQLSTATE S1000 (General error).
- A driver-specific SQLSTATE value, ranging from S1000 to S19ZZ. For example, SQLSTATE S1001 (Memory allocation failure) indicates that the Driver Manager's call to the driver-level **SQLAllocEnv** returned SQL_ERROR, and the Driver Manager's *henv* was set to SQL_NULL_HENV.

For additional information about the flow of function calls between the Driver Manager and a driver, see 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 an *henv* to **SQLAllocEnv**. The driver allocates memory for the environment information and stores the value of the associated handle in the *henv*. On operating systems that support multiple threads, applications can use the same *henv* on different threads and drivers must therefore support safe, multithreaded access to this information. The application passes the *henv* value in all subsequent calls that require an *henv*.

There should never be more than one *henv* allocated at one time and 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 *henv* without regard to its previous contents.

When the Driver Manager processes the **SQLAllocEnv** function, it checks the **Trace** keyword in the [ODBC] section of the ODBC.INI file or the ODBC subkey in the registry. If it is set to 1, the Driver Manager enables tracing for all applications on Windows 3.1 or for the current application on Windows NT.

Code Example

See **SQLBrowseConnect** and **SQLConnect**.

Related Functions

- SQLAllocConnect
- SQLConnect
- SQLFreeEnv

SQLAllocStmt (Core, ODBC 1.0)

Code Example Related Functions

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

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_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, or SQL_ERROR.

If **SQLAllocStmt** returns SQL_ERROR, it will set the *hstmt* referenced by *phstmt* to SQL_NULL_HSTMT. The application can then obtain additional information by calling **SQLError** with the *hdbc* and SQL_NULL_HSTMT.

Diagnostics

When **SQLAllocStmt** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLAllocStmt** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection not open	(DM) The connection specified by the *hdbc* argument was not open. The connection process must be completed successfully (and the connection must be open) for the driver to allocate an *hstmt*.
IM001	Driver does not support this function	(DM) The driver associated with the *hdbc* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	(DM) The Driver Manager was unable to allocate memory for the statement handle. The driver was unable to allocate memory for the statement handle.
S1009	Invalid argument value	(DM) The argument *phstmt* was a null pointer.

Comments

A statement handle references statement information, such as network information, SQLSTATE values and error messages, cursor name, number of result set columns, and status information for SQL statement processing.

To request a statement handle, an application connects to a data source and then passes the address of an *hstmt* to **SQLAllocStmt**. The driver allocates memory for the statement information and stores the value of the associated handle in the *hstmt*. On operating systems that support multiple threads, applications can use the same *hstmt* on different threads and drivers must therefore support safe, multithreaded access to this information. The application passes the *hstmt* value in all subsequent calls that require an *hstmt*.

If the application calls **SQLAllocStmt** with a pointer to a valid *hstmt*, the driver overwrites the *hstmt* without regard to its previous contents.

Code Example

See **SQLBrowseConnect**, **SQLConnect**, and **SQLSetCursorName**.

Related Functions

- SQLExecDirect
- SQLExecute
- SQLFreeStmt
- SQLPrepare

SQLBindCol (Core, ODBC 1.0)

Code Example Related Functions

SQLBindCol assigns the storage and data type 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 type conversion

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. A column number of 0 is used to retrieve a bookmark for the row; bookmarks are not supported by ODBC 1.0 drivers or by **SQLFetch**.
SWORD	*fCType*	Input	The C data type of the result data. This must be one of the following values: SQL_C_BINARY SQL_C_BIT SQL_C_BOOKMARK SQL_C_CHAR SQL_C_DATE SQL_C_DEFAULT SQL_C_DOUBLE SQL_C_FLOAT SQL_C_SLONG SQL_C_SSHORT SQL_C_STINYINT SQL_C_TIME SQL_C_TIMESTAMP SQL_C_ULONG SQL_C_USHORT SQL_C_UTINYINT SQL_C_DEFAULT specifies that data be transferred to its default C data type. **Note** Drivers must also support the following values of *fCType* from ODBC 1.0. Applications must use these values, rather than the ODBC 2.0 values, when calling an ODBC 1.0 driver: SQL_C_LONG SQL_C_SHORT SQL_C_TINYINT For more information, see

			ODBC 1.0 C Data Types. For information about how data is converted, see Converting Data from SQL to C Data Types.
PTR	*rgbValue*	Input	Pointer to storage for the data. If *rgbValue* is a null pointer, the driver unbinds the column. (To unbind all columns, an application calls **SQLFreeStmt** with the SQL_UNBIND option.) **Note** If a null pointer was passed for *rgbValue* in ODBC 1.0, the driver returned SQLSTATE S1009 (Invalid argument value); individual columns could not be unbound.
SDWORD	*cbValueMax*	Input	Maximum length of the *rgbValue* buffer. For character data, *rgbValue* must also include space for the null-termination byte. For more information about length, see Precision, Scale, Length, and Display Size.
SDWORD FAR *	*pcbValue*	Input	SQL_NULL_DATA or the number of bytes (excluding the null termination byte for character data) available to return in *rgbValue* prior to calling **SQLExtendedFetch** or **SQLFetch**, or SQL_NO_TOTAL if the number of available bytes cannot be determined. For character data, if the number of bytes available to return is SQL_NO_TOTAL or is greater than or equal to *cbValueMax*, the data in *rgbValue* is truncated to *cbValueMax* - 1 bytes and is null-terminated by the driver. For binary data, if the number of bytes available to return is SQL_NO_TOTAL or is greater than *cbValueMax*, the data in *rgbValue* is truncated to *cbValueMax* bytes. For all other data types, the value of *cbValueMax* is ignored and the driver assumes the size of *rgbValue* is the size of the C data type

specified with *fCType*.

For more information about the value returned in *pcbValue* for each *fCType*, see Converting Data from SQL to C Data Types.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLBindCol** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLBindCol** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1002	Invalid column number	The value specified for the argument *icol* was 0 and the driver was an ODBC 1.0 driver. The value specified for the argument *icol* exceeded the maximum number of columns supported by the data source.
S1003	Program type out of range	(DM) The argument *fCType* was not a valid data type or SQL_C_DEFAULT. The argument *icol* was 0 and the argument *fCType* was not SQL_C_BOOKMARK.
S1009	Invalid argument value	The driver supported ODBC 1.0 and the argument *rgbValue* was a null pointer.
S1010	Function sequence error	(DM) An asynchronously executing function was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value specified for the argument *cbValueMax* was less than 0.

S1C00	Driver not capable	The driver does not support the data type specified in the argument *fCType*. The argument *icol* was 0 and the driver does not support bookmarks. The driver only supports ODBC 1.0 and the argument *fCType* was one of the following: SQL_C_STINYINT SQL_C_UTINYINT SQL_C_SSHORT SQL_C_USHORT SQL_C_SLONG SQL_C_ULONG

Comments

The ODBC interface provides two ways to retrieve a column of data:

- **SQLBindCol** assigns the storage location for a column of data before the data is retrieved. When **SQLFetch** or **SQLExtendedFetch** is called, the driver places the data for all bound columns in the assigned locations.
- **SQLGetData** (an extended function) assigns a storage location for a column of data after **SQLFetch** or **SQLExtendedFetch** has been called. It also places the data for the requested column in the assigned location. Because it can retrieve data from a column in parts, **SQLGetData** can be used to retrieve long data values.

An application may choose to bind every column with **SQLBindCol**, to do no binding and retrieve data only with **SQLGetData**, or to use a combination of the two. However, unless the driver provides extended functionality, **SQLGetData** can only be used to retrieve data from columns that occur after the last bound column.

An application calls **SQLBindCol** to pass the pointer to the storage buffer for a column of data to the driver and to specify how or if the data will be converted. It is the application's responsibility to allocate enough storage for the data. If the buffer will contain variable length data, the application must allocate as much storage as the maximum length of the bound column or the data may be truncated. For a list of valid data conversion types, see Converting Data from SQL to C Data Types.

At fetch time, the driver processes the data for each bound column according to the arguments specified in **SQLBindCol**. First, it converts the data according to the argument *fCType*. Next, it fills the buffer pointed to by *rgbValue*. Finally, it stores the available number of bytes in *pcbValue*; this is the number of bytes available prior to calling **SQLFetch** or **SQLExtendedFetch**.

- If SQL_MAX_LENGTH has been specified with **SQLSetStmtOption** and the available number of bytes is greater than SQL_MAX_LENGTH, the driver stores SQL_MAX_LENGTH in *pcbValue*.
- If the data is truncated because of SQL_MAX_LENGTH, but the user's buffer was large enough for SQL_MAX_LENGTH bytes of data, SQL_SUCCESS is returned.

Note The SQL_MAX_LENGTH statement option is intended to reduce network traffic and may not be supported by all drivers. To guarantee that data is truncated, an application should allocate a buffer of the desired size and specify this size in the *cbValueMax* argument.

- If the user's buffer causes the truncation, the driver returns SQL_SUCCESS_WITH_INFO and SQLSTATE 01004 (Data truncated) for the fetch function.
- If the data value for a column is NULL, the driver sets *pcbValue* to SQL_NULL_DATA.
- If the number of bytes available to return cannot be determined in advance, the driver sets *pcbValue* to SQL_NO_TOTAL.

When an application uses **SQLExtendedFetch** to retrieve more than one row of data, it only needs to call **SQLBindCol** once for each column of the result set (just as when it binds a column in order to retrieve a single row of data with **SQLFetch**). The **SQLExtendedFetch** function coordinates the placement of each row of data into subsequent locations in the rowset buffers. For additional information about binding rowset buffers, see the "Comments" topic for **SQLExtendedFetch**.

An application can call **SQLBindCol** to bind a column to a new storage location, regardless of whether data has already been fetched. The new binding replaces the old binding. Note that the new binding does not apply to data already fetched; the next time data is fetched, the data will be placed in the new storage location.

To unbind a single bound column, an application calls **SQLBindCol** and specifies a null pointer for *rgbValue*; if *rgbValue* is a null pointer and the column is not bound, **SQLBindCol** returns SQL_SUCCESS. To unbind all bound columns, an application calls **SQLFreeStmt** with the SQL_UNBIND option.

▪

Code Example

In the following example, an application executes a **SELECT** statement to return a result set of the employee names, ages, and birthdays, which is sorted by birthday. It then calls **SQLBindCol** to bind the columns of data to local storage locations. Finally, the application fetches each row of data with **SQLFetch** and prints each employee's name, age, and birthday.

For more code examples, see **SQLColumns**, **SQLExtendedFetch**, and **SQLSetPos**.

```
#define NAME_LEN 30
#define BDAY_LEN 11

UCHAR  szName[NAME_LEN], szBirthday[BDAY_LEN];
SWORD  sAge;
SDWORD cbName, cbAge, cbBirthday;

retcode = SQLExecDirect(hstmt, "SELECT NAME, AGE, BIRTHDAY FROM EMPLOYEE
ORDER BY 3, 2, 1", SQL_NTS);

if (retcode == SQL_SUCCESS) {

  /* Bind columns 1, 2, and 3 */

  SQLBindCol(hstmt, 1, SQL_C_CHAR, szName, NAME_LEN, &cbName);
  SQLBindCol(hstmt, 2, SQL_C_SSHORT, &sAge, 0, &cbAge);
  SQLBindCol(hstmt, 3, SQL_C_CHAR, szBirthday, BDAY_LEN, &cbBirthday);

  /* Fetch and print each row of data.  On */
  /* an error, display a message and exit. */

  while (TRUE) {
    retcode = SQLFetch(hstmt);
    if (retcode == SQL_ERROR || retcode == SQL_SUCCESS_WITH_INFO) {
      show_error();
    }
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){
      fprintf(out, "%-*s %-2d %*s", NAME_LEN-1, szName, sAge, BDAY_LEN-1,
szBirthday);
    } else {
      break;
    }
  }
}
```

Related Functions

SQLDescribeCol
SQLExtendedFetch (extension)
SQLFetch
SQLFreeStmt
SQLGetData (extension)
SQLNumResultCols

SQLBindParameter (Level 1, ODBC 2.0)

Code Example Related Functions

SQLBindParameter binds a buffer to a parameter marker in an SQL statement.

Note This function replaces the ODBC 1.0 function **SQLSetParam**. For more information, see "Comments."

Syntax

RETCODE **SQLBindParameter**(*hstmt*, *ipar*, *fParamType*, *fCType*, *fSqlType*, *cbColDef*, *ibScale*, *rgbValue*, *cbValueMax*, *pcbValue*)

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

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.
UWORD	*ipar*	Input	Parameter number, ordered sequentially left to right, starting at 1.
SWORD	*fParamType*	Input	The type of the parameter. For more information, see fParamType Argument in "Comments."
SWORD	*fCType*	Input	The C data type of the parameter. For more information, see fCType Argument in "Comments."
SWORD	*fSqlType*	Input	The SQL data type of the parameter. For more information, see fSqlType Argument in "Comments."
UDWORD	*cbColDef*	Input	The precision of the column or expression of the corresponding parameter marker. For more information, see cbColDef Argument in "Comments."
SWORD	*ibScale*	Input	The scale of the column or expression of the corresponding parameter marker. For further information concerning scale, see Precision, Scale, Length, and Display Size.
PTR	*rgbValue*	Input/ Output	A pointer to a buffer for the parameter's data. For more information, see rgbValue Argument in "Comments."
SDWORD	*cbValueMax*	Input	Maximum length of the *rgbValue* buffer. For more information, see cbValueMax Argument in "Comments."
SDWORD FAR *	*pcbValue*	Input/ Output	A pointer to a buffer for the parameter's length. For more information, see pcbValue Argument in "Comments."

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLBindParameter** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLBindParameter** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
07006	Restricted data type attribute violation	The data value identified by the *fCType* argument cannot be converted to the data type identified by the *fSqlType* argument.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1003	Program type out of range	(DM) The value specified by the argument *fCType* was not a valid data type or SQL_C_DEFAULT.
S1004	SQL data type out of range	(DM) The value specified for the argument *fSqlType* was in the block of numbers reserved for ODBC SQL data type indicators but was not a valid ODBC SQL data type indicator.
S1009	Invalid argument value	(DM) The argument *rgbValue* was a null pointer, the argument *pcbValue* was a null pointer, and the argument *fParamType* was not SQL_PARAM_OUTPUT.
S1010	Function sequence error	(DM) An asynchronously executing function was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value specified for the argument *cbValueMax* was less than 0.
S1093	Invalid parameter number	(DM) The value specified for the argument *ipar* was less than 1. The value specified for the argument *ipar* was greater than the maximum

		number of parameters supported by the data source.
S1094	Invalid scale value	The value specified for the argument *ibScale* was outside the range of values supported by the data source for a column of the SQL data type specified by the *fSqlType* argument.
S1104	Invalid precision value	The value specified for the argument *cbColDef* was outside the range of values supported by the data source for a column of the SQL data type specified by the *fSqlType* argument.
S1105	Invalid parameter type	(DM) The value specified for the argument *fParamType* was invalid (see "Comments"). The value specified for the argument *fParamType* was SQL_PARAM_OUTPUT and the parameter did not mark a return value from a procedure or a procedure parameter. The value specified for the argument *fParamType* was SQL_PARAM_INPUT and the parameter marked the return value from a procedure.
S1C00	Driver not capable	The driver or data source does not support the conversion specified by the combination of the value specified for the argument *fCType* and the driver-specific value specified for the argument *fSqlType*. The value specified for the argument *fSqlType* was a valid ODBC SQL data type indicator for the version of ODBC supported by the driver, but was not supported by the driver or data source. The value specified for the argument *fSqlType* was in the range of numbers reserved for driver-specific SQL data type indicators, but was not supported by the driver or data source. The driver only supports ODBC 1.0 and the argument *fCType* was one of the following: SQL_C_STINYINT SQL_C_UTINYINT SQL_C_SSHORT SQL_C_USHORT SQL_C_SLONG SQL_C_ULONG

Comments

An application calls **SQLBindParameter** to bind each parameter marker in an SQL statement. Bindings remain in effect until the application calls **SQLBindParameter** again or until the application calls **SQLFreeStmt** with the SQL_DROP or SQL_RESET_PARAMS option.

fParamType Argument

The *fParamType* argument specifies the type of the parameter. All parameters in SQL statements that do not call procedures, such as **INSERT** statements, are input parameters. Parameters in procedure calls can be input, input/output, or output parameters. (An application calls **SQLProcedureColumns** to determine the type of a parameter in a procedure call; parameters in procedure calls whose type cannot be determined are assumed to be input parameters.)

The *fParamType* argument is one of the following values:

- SQL_PARAM_INPUT. The parameter marks a parameter in an SQL statement that does not call a procedure, such as an **INSERT** statement, or it marks an input parameter in a procedure; these are collectively known as *input parameters*. For example, the parameters in **INSERT INTO Employee VALUES (?, ?, ?)** and **{call AddEmp(?, ?, ?)}** are input parameters.

  When the statement is executed, the driver sends data for the parameter to the data source; the *rgbValue* buffer must contain a valid input value or the *pcbValue* buffer must contain SQL_NULL_DATA, SQL_DATA_AT_EXEC, or the result of the SQL_LEN_DATA_AT_EXEC macro.

  If an application cannot determine the type of a parameter in a procedure call, it sets *fParamType* to SQL_PARAM_INPUT; if the data source returns a value for the parameter, the driver discards it.

- SQL_PARAM_INPUT_OUTPUT. The parameter marks an input/output parameter in a procedure. For example, the parameter in **{call GetEmpDept(?)}** is an input/output parameter that accepts an employee's name and returns the name of the employee's department.

  When the statement is executed, the driver sends data for the parameter to the data source; the *rgbValue* buffer must contain a valid input value or the *pcbValue* buffer must contain SQL_NULL_DATA, SQL_DATA_AT_EXEC, or the result of the SQL_LEN_DATA_AT_EXEC macro. After the statement is executed, the driver returns data for the parameter to the application; if the data source does not return a value for an input/output parameter, the driver sets the *pcbValue* buffer to SQL_NULL_DATA.

  Note When an ODBC 1.0 application calls **SQLSetParam** in an ODBC 2.0 driver, the Driver Manager converts this to a call to **SQLBindParameter** in which the *fParamType* argument is set to SQL_PARAM_INPUT_OUTPUT.

- SQL_PARAM_OUTPUT. The parameter marks the return value of a procedure or an output parameter in a procedure; these are collectively known as *output parameters*. For example, the parameter in **{?=call GetNextEmpID}** is an output parameter that returns the next employee ID.

  After the statement is executed, the driver returns data for the parameter to the application, unless the *rgbValue* and *pcbValue* arguments are both null pointers, in which case the driver discards the output value. If the data source does not return a value for an output parameter, the driver sets the *pcbValue* buffer to SQL_NULL_DATA.

fCType Argument

The C data type of the parameter. This must be one of the following values:

SQL_C_BINARY
SQL_C_BIT
SQL_C_CHAR
SQL_C_DATE
SQL_C_DEFAULT
SQL_C_DOUBLE
SQL_C_FLOAT
SQL_C_SLONG
SQL_C_SSHORT
SQL_C_STINYINT
SQL_C_TIME
SQL_C_TIMESTAMP
SQL_C_ULONG
SQL_C_USHORT
SQL_C_UTINYINT

SQL_C_DEFAULT specifies that the parameter value be transferred from the default C data type for the SQL data type specified with *fSqlType*.

For more information, see Default C Data Types and Converting Data from C to SQL Data Types and Converting Data from SQL to C Data Types.

Note Drivers must also support the following values of *fCType* from ODBC 1.0. Applications must use these values, instead of the ODBC 2.0 values, when calling an ODBC 1.0 driver:

SQL_C_LONG
SQL_C_SHORT
SQL_C_TINYINT

For more information, see ODBC 1.0 C Data Types.

---

fSqlType Argument

This must be one of the following values:

SQL_BIGINT
SQL_BINARY
SQL_BIT
SQL_CHAR
SQL_DATE
SQL_DECIMAL
SQL_DOUBLE
SQL_FLOAT
SQL_INTEGER
SQL_LONGVARBINARY
SQL_LONGVARCHAR
SQL_NUMERIC
SQL_REAL
SQL_SMALLINT
SQL_TIME
SQL_TIMESTAMP
SQL_TINYINT
SQL_VARBINARY
SQL_VARCHAR

or a driver-specific value. Values greater than SQL_TYPE_DRIVER_START are reserved by ODBC; values less than or equal to SQL_TYPE_DRIVER_START are driver-specific.

For information about how data is converted, see Converting Data from C to SQL Data Types and Converting Data from SQL to C Data Types.

cbColDef Argument

The *cbColDef* argument specifies the precision of the column or expression corresponding to the parameter marker, unless all of the following are true:

- An ODBC 2.0 application calls **SQLBindParameter** in an ODBC 1.0 driver or an ODBC 1.0 application calls **SQLSetParam** in an ODBC 2.0 driver. (Note that the Driver Manager converts these calls.)
- The *fSqlType* argument is SQL_LONGVARBINARY or SQL_LONGVARCHAR.
- The data for the parameter will be sent with **SQLPutData**.

In this case, the *cbColDef* argument contains the total number of bytes that will be sent for the parameter. For more information, see Passing Parameter Values and SQL_DATA_AT_EXEC in pcbValue Argument.

rgbValue Argument

The *rgbValue* argument points to a buffer that, when **SQLExecute** or **SQLExecDirect** is called, contains the actual data for the parameter. The data must be in the form specified by the *fCType* argument.

If *rgbValue* points to a character string that contains a literal quote character ('), the driver ensures that each literal quote is translated into the form required by the data source. For example, if the data source required that embedded literal quotes be doubled, the driver would replace each quote character (') with two quote characters (' ').

If *pcbValue* is the result of the SQL_LEN_DATA_AT_EXEC(*length*) macro or SQL_DATA_AT_EXEC, then *rgbValue* is an application-defined 32-bit value that is associated with the parameter. It is returned to the application through **SQLParamData**. For example, *rgbValue* might be a token such as a parameter number, a pointer to data, or a pointer to a structure that the application used to bind input parameters. Note, however, that if the parameter is an input/output parameter, *rgbValue* must be a pointer to a buffer where the output value will be stored. If **SQLParamOptions** was called to specify multiple values for the parameter, the application can use the value of the *pirow* argument in **SQLParamOptions** in conjunction with the *rgbValue*. For example, *rgbValue* might point to an array of values   and the application might use *pirow* to retrieve the correct value from the array. For more information, see Passing Parameter Values.

If the *fParamType* argument is SQL_PARAM_INPUT_OUTPUT or SQL_PARAM_OUTPUT, *rgbValue* points to a buffer in which the driver returns the output value. If the procedure returns one or more result sets, the *rgbValue* buffer is not guaranteed to be set until all results have been fetched. (If *fParamType* is SQL_PARAM_OUTPUT and *rgbValue* and *pcbValue* are both null pointers, the driver discards the output value.)

If the application calls **SQLParamOptions** to specify multiple values for each parameter, *rgbValue* points to an array. A single SQL statement processes the entire array of input values for an input or input/output parameter and returns an array of output values for an input/output or output parameter.

cbValueMax Argument

For character and binary C data, the *cbValueMax* argument specifies the length of the *rgbValue* buffer (if it is a single element) or the length of an element in the *rgbValue* array (if the application calls **SQLParamOptions** to specify multiple values for each parameter). If the application specifies multiple values, *cbValueMax* is used to determine the location of values in the *rgbValue* array, both on input and on output. For input/output and output parameters, it is used to determine whether to truncate character and binary C data on output:

- For character C data, if the number of bytes available to return is greater than or equal to *cbValueMax*, the data in *rgbValue* is truncated to *cbValueMax* - 1 bytes and is null-terminated by the driver.
- For binary C data, if the number of bytes available to return is greater than *cbValueMax*, the data in *rgbValue* is truncated to *cbValueMax* bytes.

For all other types of C data, the *cbValueMax* argument is ignored. The length of the *rgbValue* buffer (if it is a single element) or the length of an element in the *rgbValue* array (if the application calls **SQLParamOptions** to specify multiple values for each parameter) is assumed to be the length of the C data type.

Note When an ODBC 1.0 application calls **SQLSetParam** in an ODBC 2.0 driver, the Driver Manager converts this to a call to **SQLBindParameter** in which the *cbValueMax* argument is always SQL_SETPARAM_VALUE_MAX. Because the Driver Manager returns an error if an ODBC 2.0 application sets *cbValueMax* to SQL_SETPARAM_VALUE_MAX, an ODBC 2.0 driver can use this to determine when it is called by an ODBC 1.0 application.

When an ODBC 2.0 application calls **SQLBindParameter** in an ODBC 1.0 driver, the Driver Manager converts this to a call to **SQLSetParam** and discards the *cbValueMax* argument.

In **SQLSetParam**, the way in which an application specifies the length of the *rgbValue* buffer so that the driver can return character or binary data and the way in which an application sends an array of character or binary parameter values to the driver are driver-defined. If an ODBC 2.0 application uses this functionality in an ODBC 1.0 driver, it must use the semantics defined by that driver. If an ODBC 2.0 driver supported this functionality as an ODBC 1.0 driver, it must continue to support this functionality for ODBC 1.0 applications.

---

pcbValue Argument

The *pcbValue* argument points to a buffer that, when **SQLExecute** or **SQLExecDirect** is called, contains one of the following:

- The length of the parameter value stored in *rgbValue*. This is ignored except for character or binary C data.
- SQL_NTS. The parameter value is a null-terminated string.
- SQL_NULL_DATA. The parameter value is NULL.
- SQL_DEFAULT_PARAM. A procedure is to use the default value of a parameter, rather than a value retrieved from the application. This value is valid only in a procedure call, and then only if the *fParamType* argument is SQL_PARAM_INPUT or SQL_PARAM_INPUT_OUTPUT. When *pcbValue* is SQL_DEFAULT_PARAM, the *fCType*, *fSqlType*, *cbColDef*, *ibScale*, *cbValueMax* and *rgbValue* arguments are ignored for input parameters and are used only to define the output parameter value for input/output parameters.

Note This value was introduced in ODBC 2.0.

- The result of the SQL_LEN_DATA_AT_EXEC(*length*) macro. The data for the parameter will be sent with **SQLPutData**. If the *fSqlType* argument is SQL_LONGVARBINARY, SQL_LONGVARCHAR, or a long, data source-specific data type and the driver returns "Y" for the SQL_NEED_LONG_DATA_LEN information type in **SQLGetInfo**, *length* is the number of bytes of data to be sent for the parameter; otherwise, *length* must be a nonnegative value and is ignored. For more information, see "Passing Parameter Values."

  For example, to specify that 10,000 bytes of data will be sent with **SQLPutData** for an SQL_LONGVARCHAR parameter, an application sets *pcbValue* to SQL_LEN_DATA_AT_EXEC(10000).

Note This macro was introduced in ODBC 2.0.

- SQL_DATA_AT_EXEC. The data for the parameter will be sent with **SQLPutData**. This value is used by ODBC 2.0 applications when calling ODBC 1.0 drivers and by ODBC 1.0 applications when calling ODBC 2.0 drivers. For more information, see Passing Parameter Values.

If *pcbValue* is a null pointer, the driver assumes that all input parameter values are non-NULL and that character and binary data are null-terminated. If *fParamType* is SQL_PARAM_OUTPUT and *rgbValue* and *pcbValue* are both null pointers, the driver discards the output value.

Note Application developers are strongly discouraged from specifying a null pointer for *pcbValue* when the data type of the parameter is SQL_C_BINARY. For SQL_C_BINARY data, a driver sends only the data preceding an occurrence of the null-termination character, 0x00. To ensure that a driver does not unexpectedly truncate SQL_C_BINARY data, *pcbValue* should contain a pointer to a valid length value.

If the *fParamType* argument is SQL_PARAM_INPUT_OUTPUT or SQL_PARAM_OUTPUT, *pcbValue* points to a buffer in which the driver returns SQL_NULL_DATA, the number of bytes available to return in *rgbValue* (excluding the null termination byte of character data), or SQL_NO_TOTAL if the number of bytes available to return cannot be determined. If the procedure returns one or more result sets, the *pcbValue* buffer is not guaranteed to be set until all results have been fetched.

If the application calls **SQLParamOptions** to specify multiple values for each parameter, *pcbValue* points to an array of SDWORD values. These can be any of the values listed earlier in this section and are processed with a single SQL statement.

Passing Parameter Values

An application can pass the value for a parameter either in the *rgbValue* buffer or with one or more calls to **SQLPutData**. Parameters whose data is passed with **SQLPutData** are known as *data-at-execution* parameters. These are commonly used to send data for SQL_LONGVARBINARY and SQL_LONGVARCHAR parameters and can be mixed with other parameters.

To pass parameter values, an application:

1. Calls **SQLBindParameter** for each parameter to bind buffers for the parameter's value (*rgbValue* argument) and length (*pcbValue* argument). For data-at-execution parameters, *rgbValue* is an application-defined 32-bit value such as a parameter number or a pointer to data. The value will be

returned later and can be used to identify the parameter.

2. Places values for input and input/output parameters in the *rgbValue* and *pcbValue* buffers:

- For normal parameters, the application places the parameter value in the *rgbValue* buffer and the length of that value in the *pcbValue* buffer.
- For data-at-execution parameters, the application places the result of the SQL_LEN_DATA_AT_EXEC(*length*) macro (when calling an ODBC 2.0 driver) or SQL_DATA_AT_EXEC (when calling an ODBC 1.0 driver) in the *pcbValue* buffer.

3. Calls **SQLExecute** or **SQLExecDirect** to execute the SQL statement.

- If there are no data-at-execution parameters, the process is complete.
- If there are any data-at-execution parameters, the function returns SQL_NEED_DATA.

4. Calls **SQLParamData** to retrieve the application-defined value specified in the *rgbValue* argument for the first data-at-execution parameter to be processed.

Note Although data-at-execution parameters are similar to data-at-execution columns, the value returned by **SQLParamData** is different for each.

Data-at-execution parameters are parameters in an SQL statement for which data will be sent with **SQLPutData** when the statement is executed with **SQLExecDirect** or **SQLExecute**. They are bound with **SQLBindParameter**. The value returned by **SQLParamData** is a 32-bit value passed to **SQLBindParameter** in the *rgbValue* argument.

Data-at-execution columns are columns in a rowset for which data will be sent with **SQLPutData** when a row is updated or added with **SQLSetPos**. They are bound with **SQLBindCol**. The value returned by **SQLParamData** is the address of the row in the *rgbValue* buffer that is being processed.

---

5. Calls **SQLPutData** one or more times to send data for the parameter. More than one call is needed if the data value is larger than the *rgbValue* buffer specified in **SQLPutData**; note that multiple calls to **SQLPutData** for the same parameter are allowed only when sending character C data to a column with a character, binary, or data source-specific data type or when sending binary C data to a column with a character, binary, or data source-specific data type.

6. Calls **SQLParamData** again to signal that all data has been sent for the parameter.

- If there are more data-at-execution parameters, **SQLParamData** returns SQL_NEED_DATA and the application-defined value for the next data-at-execution parameter to be processed. The application repeats steps 5 and 6.
- If there are no more data-at-execution parameters, the process is complete. If the statement was successfully executed, **SQLParamData** returns SQL_SUCCESS or SQL_SUCCESS_WITH_INFO; if the execution failed, it returns SQL_ERROR. at this point, **SQLParamData** can return any SQLSTATE that can be returned by the function used to execute the statement (**SQLExecDirect** or **SQLExecute**).

Output values for any input/output or output parameters will be available in the *rgbValue* and *pcbValue* buffers after the application retrieves any result sets generated by the statement.

After **SQLExecute** or **SQLExecDirect** returns SQL_NEED_DATA, and before data is sent for all data-at-execution parameters, the statement is canceled, or an error occurs in **SQLParamData** or **SQLPutData**, the application can only call **SQLCancel**, **SQLGetFunctions**, **SQLParamData**, or **SQLPutData** with the *hstmt* or the *hdbc* associated with the *hstmt*. If it calls any other function with the *hstmt* or the *hdbc* associated with the *hstmt*, the function returns SQL_ERROR and SQLSTATE S1010 (Function sequence error).

If the application calls **SQLCancel** while the driver still needs data for data-at-execution parameters, the driver cancels statement execution; the application can then call **SQLExecute** or **SQLExecDirect** again. If the application calls **SQLParamData** or **SQLPutData** after canceling the statement, the function returns SQL_ERROR and SQLSTATE S1008 (Operation canceled).

Conversion of Calls to and from SQLSetParam

When an ODBC 1.0 application calls **SQLSetParam** in an ODBC 2.0 driver, the ODBC 2.0 Driver Manager maps the call as follows:

Call by ODBC 1.0 Application	Call to ODBC 2.0 Driver
`SQLSetParam(`	`SQLBindParameter(`
`  hstmt, ipar,`	`  hstmt, ipar,`

```
  fCType, fSqlType,          SQL_PARAM_INPUT_OUTPUT,
cbColDef, ibScale,             fCType, fSqlType,
  rgbValue,                  cbColDef, ibScale,
  pcbValue);                   rgbValue,
                             SQL_SETPARAM_VALUE_MAX,
                               pcbValue);
```

When an ODBC 2.0 application calls **SQLBindParameter** in an ODBC 1.0 driver, the ODBC 2.0 Driver Manager maps the calls as follows:

Call by ODBC 2.0 Application	Call to ODBC 1.0 Driver

`SQLBindParameter(`
`  hstmt, ipar, fParamType,`
`  fCType, fSqlType,`
`cbColDef,`
`ibScale,`
`  rgbValue, cbValueMax,`
`pcbValue);`
	`SQLSetParam(`
	`  hstmt, ipar,`
	`  fCType, fSqlType,`
	`cbColDef,`
	`ibScale,`
	`  rgbValue, pcbValue);`

▪

Code Example

In the following example, an application prepares an SQL statement to insert data into the EMPLOYEE table. The SQL statement contains parameters for the NAME, AGE, and BIRTHDAY columns. For each parameter in the statement, the application calls **SQLBindParameter** to specify the ODBC C data type and the SQL data type of the parameter and to bind a buffer to each parameter. For each row of data, the application assigns data values to each parameter and calls **SQLExecute** to execute the statement.

For more code examples, see **SQLParamOptions**, **SQLProcedures**, **SQLPutData**, and **SQLSetPos**.

```
#define NAME_LEN 30

UCHAR       szName[NAME_LEN];
SWORD       sAge;
SDWORD      cbName = SQL_NTS, cbAge = 0, cbBirthday = 0;
DATE_STRUCT dsBirthday;

retcode = SQLPrepare(hstmt, "INSERT INTO EMPLOYEE (NAME, AGE, BIRTHDAY)
VALUES (?, ?, ?)", SQL_NTS);

if (retcode == SQL_SUCCESS) {

  /* Specify data types and buffers.          */
  /* for Name, Age, Birthday parameter data.  */

  SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, NAME_LEN,
0, szName, 0, &cbName);
  SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_SSHORT, SQL_SMALLINT, 0,
0, &sAge, 0, &cbAge);
  SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_DATE, SQL_DATE, 0, 0,
&dsBirthday, 0, &cbBirthday);

  strcpy(szName, "Smith, John D.");    /* Specify first row of   */
  sAge = 40;                           /* parameter data         */
  dsBirthday.year = 1952;
  dsBirthday.month = 2;
  dsBirthday.day = 29;
  retcode = SQLExecute(hstmt);         /* Execute statement with */
                                       /* first row              */

  strcpy(szName, "Jones, Bob K.");     /* Specify second row of  */
  sAge = 52;                           /* parameter data         */
  dsBirthday.year = 1940;
  dsBirthday.month = 3;
  dsBirthday.day = 31;
  SQLExecute(hstmt);                   /* Execute statement with  */
                                       /* second row              */
}
```

Related Functions

SQLDescribeParam (extension)
SQLExecDirect
SQLExecute
SQLNumParams (extension)
SQLParamData (extension)
SQLParamOptions (extension)
SQLPutData (extension)

SQLBrowseConnect (Level 2, ODBC 1.0)

Code Example Related Functions

SQLBrowseConnect supports an iterative method of discovering and enumerating the attributes and attribute values required to connect to a data source. Each call to **SQLBrowseConnect** returns successive levels of attributes and attribute values. When all levels have been enumerated, a connection to the data source is completed and a complete connection string is returned by **SQLBrowseConnect**. A return code of SQL_SUCCESS or SQL_SUCCESS_WITH_INFO indicates that all connection information has been specified and the application is now connected to the data source.

Syntax

RETCODE **SQLBrowseConnect**(*hdbc*, *szConnStrIn*, *cbConnStrIn*, *szConnStrOut*, *cbConnStrOutMax*, *pcbConnStrOut*)

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

Type	Argument	Use	Description
HDBC	*hdbc*	Input	Connection handle.
UCHAR FAR *	*szConnStrIn*	Input	Browse request connection string (see szConnStrIn Argument in "Comments").
SWORD	*cbConnStrIn*	Input	Length of *szConnStrIn*.
UCHAR FAR *	*szConnStrOut*	Output	Pointer to storage for the browse result connection string (see szConnStrOut Argument in "Comments").
SWORD	*cbConnStrOutMax*	Input	Maximum length of the *szConnStrOut* buffer.
SWORD FAR *	*pcbConnStrOut*	Output	The total number of bytes (excluding the null termination byte) available to return in *szConnStrOut*. If the number of bytes available to return is greater than or equal to *cbConnStrOutMax*, the connection string in *szConnStrOut* is truncated to *cbConnStrOutMax* - 1 bytes.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLBrowseConnect** returns SQL_ERROR, SQL_SUCCESS_WITH_INFO, or SQL_NEED_DATA, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLBrowseConnect** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The buffer *szConnStrOut* was not large enough to return entire browse result connection string, so the string was truncated. The argument *pcbConnStrOut* contains the length of

		the untruncated browse result connection string. (Function returns SQL_SUCCESS_WITH_INFO.)
01S00	Invalid connection string attribute	An invalid attribute keyword was specified in the browse request connection string (*szConnStrIn*). (Function returns SQL_NEED_DATA.) An attribute keyword was specified in the browse request connection string (*szConnStrIn*) that does not apply to the current connection level. (Function returns SQL_NEED_DATA.)
08001	Unable to connect to data source	The driver was unable to establish a connection with the data source.
08002	Connection in use	(DM) The specified *hdbc* had already been used to establish a connection with a data source and the connection was open.
08004	Data source rejected establishment of connection	The data source rejected the establishment of the connection for implementation defined reasons.
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was attempting to connect failed before the function completed processing.
28000	Invalid authorization specification	Either the user identifier or the authorization string or both as specified in the browse request connection string (*szConnStrIn*) violated restrictions defined by the data source.
IM001	Driver does not support this function	(DM) The driver corresponding to the specified data source name does not support the function.
IM002	Data source not found and no default driver specified	(DM) The data source name specified in the browse request connection string (*szConnStrIn*) was not found in the ODBC.INI file or registry nor was there a default driver specification. (DM) The ODBC.INI file could not be found.
IM003	Specified driver could not be loaded	(DM) The driver listed in the data source specification in the ODBC.INI file or registry, or specified by the **DRIVER** keyword was not found or could not be loaded for some other reason.
IM004	Driver's SQLAllocEnv failed	(DM) During **SQLBrowseConnect**, the Driver Manager called the driver's **SQLAllocEnv** function and the driver returned an error.
IM005	Driver's SQLAllocConnect failed	(DM) During **SQLBrowseConnect**, the Driver Manager called the driver's **SQLAllocConnect** function and the driver returned an error.
IM006	Driver's	(DM) During **SQLBrowseConnect**, the

	SQLSetConnect-Option failed	Driver Manager called the driver's **SQLSetConnectOption** function and the driver returned an error.
IM009	Unable to load translation DLL	The driver was unable to load the translation DLL that was specified for the data source or for the connection.
IM010	Data source name too long	(DM) The attribute value for the DSN keyword was longer than SQL_MAX_DSN_LENGTH characters.
IM011	Driver name too long	(DM) The attribute value for the DRIVER keyword was longer than 255 characters.
IM012	DRIVER keyword syntax error	(DM) The keyword-value pair for the DRIVER keyword contained a syntax error.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	(DM) The Driver Manager was unable to allocate memory required to support execution or completion of the function. The driver was unable to allocate memory required to support execution or completion of the function.
S1090	Invalid string or buffer length	(DM) The value specified for argument *cbConnStrIn* was less than 0 and was not equal to SQL_NTS. (DM) The value specified for argument *cbConnStrOutMax* was less than 0.
S1T00	Timeout expired	The timeout period expired before the connection to the data source completed. The timeout period is set through **SQLSetConnectOption**, SQL_LOGIN_TIMEOUT.

Comments

szConnStrIn Argument

A browse request connection string has the following syntax:

connection-string ::= *attribute*[;] | *attribute*; *connection-string*
attribute ::= *attribute-keyword*=*attribute-value* | DRIVER={*attribute-value*}
(The braces are literal; the application must specify them.)
attribute-keyword ::= DSN | UID | PWD
| *driver-defined-attribute-keyword*
attribute-value ::= *character-string*
driver-defined-attribute-keyword ::= *identifier*

where *character-string* has zero or more characters; *identifier* has one or more characters; *attribute-keyword* is case insensitive; *attribute-value* may be case sensitive; and the value of the **DSN** keyword does not consist solely of blanks. Because of connection string and initialization file grammar, keywords and attribute values that contain the characters **[]{}(),;?*=!@** should be avoided. Because of the registry grammar, keywords and data source names cannot contain the backslash (\) character.

Note The **DRIVER** keyword was introduced in ODBC 2.0 and is not supported by ODBC 1.0 drivers.

If any keywords are repeated in the browse request connection string, the driver uses the value associated with the first occurrence of the keyword. If the **DSN** and **DRIVER** keywords are included in the same browse request connection string, the Driver Manager and driver use whichever keyword appears first.

szConnStrOut Argument

The browse result connection string is a list of connection attributes. A connection attribute consists of an attribute keyword and a corresponding attribute value. The browse result connection string has the following syntax:

connection-string ::= *attribute*[;] | *attribute*; *connection-string*
attribute ::= [*]*attribute-keyword=attribute-value*
attribute-keyword ::= *ODBC-attribute-keyword*
| *driver-defined-attribute-keyword*
ODBC-attribute-keyword = {UID | PWD}[:*localized-identifier*]
driver-defined-attribute-keyword ::= *identifer*[:*localized-identifier*]
attribute-value ::= {*attribute-value-list*} | ?
(The braces are literal; they are returned by the driver.)
attribute-value-list ::= *character-string* | *character-string*, *attribute-value-list*

where *character-string* has zero or more characters; *identifier* and *localized-identifier* have one or more characters;*attribute-keyword* is case insensitive; and *attribute-value* may be case sensitive. Because of connection string and initialization file grammar, keywords, localized identifiers, and attribute values that contain the characters **[]{}(),;?*=!@** should be avoided. Because of the registry grammar, keywords and data source names cannot contain the backslash (\) character.

The browse result connection string syntax is used according to the following semantic rules:

- If an asterisk (*) precedes an *attribute-keyword*, the *attribute* is optional, and may be omitted in the next call to **SQLBrowseConnect**.
- The attribute keywords **UID** and **PWD** have the same meaning as defined in **SQLDriverConnect**.
- A *driver-defined-attribute-keyword* names the kind of attribute for which an attribute value may be supplied. For example, it might be **SERVER**, **DATABASE**, **HOST**, or **DBMS**.
- *ODBC-attribute-keywords* and *driver-defined-attribute-keywords* include a localized or user-friendly version of the keyword. This might be used by applications as a label in a dialog box. However, **UID**, **PWD**, or the *identifier* alone must be used when passing a browse request string to the driver.
- The {*attribute-value-list*} is an enumeration of actual values valid for the corresponding *attribute-keyword*. Note that the braces ({}) do not indicate a list of choices; they are returned by the driver. For example, it might be a list of server names or a list of database names.
- If the *attribute-value* is a single question mark (?), a single value corresponds to the *attribute-keyword*. For example, UID=JohnS; PWD=Sesame.
- Each call to **SQLBrowseConnect** returns only the information required to satisfy the next level of the connection process. The driver associates state information with the connection handle so that the context can always be determined on each call.

Using SQLBrowseConnect

SQLBrowseConnect requires an allocated *hdbc*. The Driver Manager loads the driver that was specified in or that corresponds to the data source name specified in the initial browse request connection string; for information on when this occurs, see the "Comments" section in **SQLConnect**. It may establish a connection with the data source during the browsing process. If **SQLBrowseConnect** returns SQL_ERROR, outstanding connections are terminated and the *hdbc* is returned to an unconnected state.

When **SQLBrowseConnect** is called for the first time on an *hdbc*, the browse request connection string must contain the **DSN** keyword or the **DRIVER** keyword. If the browse request connection string contains the **DSN** keyword, the Driver Manager locates a corresponding data source specification in the ODBC.INI file or registry:

- If the Driver Manager finds the corresponding data source specification, it loads the associated driver DLL; the driver can retrieve information about the data source from the ODBC.INI file or registry.
- If the Driver Manager cannot find the corresponding data source specification, it locates the default data source specification and loads the associated driver DLL; the driver can retrieve information

about the default data source from the ODBC.INI file or registry.

- If the Driver Manager cannot find the corresponding data source specification and there is no default data source specification, it returns SQL_ERROR with SQLSTATE IM002 (Data source not found and no default driver specified).

If the browse request connection string contains the **DRIVER** keyword, the Driver Manager loads the specified driver; it does not attempt to locate a data source in the ODBC.INI file or registry. Because the **DRIVER** keyword does not use information from the ODBC.INI file or registry, the driver must define enough keywords so that a driver can connect to a data source using only the information in the browse request connection strings.

On each call to **SQLBrowseConnect**, the application specifies the connection attribute values in the browse request connection string. The driver returns successive levels of attributes and attribute values in the browse result connection string; it returns SQL_NEED_DATA as long as there are connection attributes that have not yet been enumerated in the browse request connection string. The application uses the contents of the browse result connection string to build the browse request connection string for the next call to **SQLBrowseConnect**. Note that the application cannot use the contents of previous browse result connection strings when building the current browse request connection string; that is, it cannot specify different values for attributes set in previous levels.

When all levels of connection and their associated attributes have been enumerated, the driver returns SQL_SUCCESS, the connection to the data source is complete, and a complete connection string is returned to the application. The connection string is suitable to use in conjunction with **SQLDriverConnect** with the SQL_DRIVER_NOPROMPT option to establish another connection.

SQLBrowseConnect also returns SQL_NEED_DATA if there are recoverable, nonfatal errors during the browse process, for example, an invalid password supplied by the application or an invalid attribute keyword supplied by the application. When SQL_NEED_DATA is returned and the browse result connection string is unchanged, an error has occurred and the application must call **SQLError** to return the SQLSTATE for browse-time errors. This permits the application to correct the attribute and continue the browse.

An application may terminate the browse process at any time by calling **SQLDisconnect**. The driver will terminate any outstanding connections and return the *hdbc* to an unconnected state.

If a driver supports **SQLBrowseConnect**, the driver keyword section of the ODBC.INF file for the driver must contain the **ConnectFunctions** keyword with the third character set to "Y".

▪

Code Example

In the following example, an application calls **SQLBrowseConnect** repeatedly. Each time **SQLBrowseConnect** returns SQL_NEED_DATA, it passes back information about the data it needs in *szConnStrOut*. The application passes *szConnStrOut* to its routine **GetUserInput** (not shown). **GetUserInput** parses the information, builds and displays a dialog box, and returns the information entered by the user in *szConnStrIn*. The application passes the user's information to the driver in the next call to **SQLBrowseConnect**. After the application has provided all necessary information for the driver to connect to the data source, **SQLBrowseConnect** returns SQL_SUCCESS and the application proceeds.

For example, to connect to the data source My Source, the following actions might occur. First, the application passes the following string to **SQLBrowseConnect**:

```
"DSN=My Source"
```

The Driver Manager loads the driver associated with the data source My Source. It then calls the driver's **SQLBrowseConnect** function with the same arguments it received from the application. The driver returns the following string in *szConnStrOut*.

```
"HOST:Server={red,blue,green};UID:ID=?;PWD:Password=?"
```

The application passes this string to its **GetUserInput** routine, which builds a dialog box that asks the user to select the red, blue, or green server, and to enter a user ID and password. The routine passes the following user-specified information back in *szConnStrIn*, which the application passes to **SQLBrowseConnect**:

```
"HOST=red;UID=Smith;PWD=Sesame"
```

SQLBrowseConnect uses this information to connect to the red server as Smith with the password Sesame, then returns the following string in *szConnStrOut*:

```
"*DATABASE:Database={master,model,empdata}"
```

The application passes this string to its **GetUserInput** routine, which builds a dialog box that asks the user to select a database. The user selects empdata and the application calls **SQLBrowseConnect** a final time with the string:

```
"DATABASE=empdata"
```

This is the final piece of information the driver needs to connect to the data source; **SQLBrowseConnect** returns SQL_SUCCESS and *szConnStrOut* contains the completed connection string:

```
"DSN=My Source;HOST=red;UID=Smith;PWD=Sesame;DATABASE=empdata"
```

```
#define BRWS_LEN 100
HENV    henv;
HDBC    hdbc;
HSTMT   hstmt;
RETCODE retcode;
UCHAR   szConnStrIn[BRWS_LEN], szConnStrOut[BRWS_LEN];
SWORD   cbConnStrOut;

retcode = SQLAllocEnv(&henv);                 /* Environment handle */
if (retcode == SQL_SUCCESS) {
  retcode = SQLAllocConnect(henv, &hdbc);    /* Connection handle  */
  if (retcode == SQL_SUCCESS) {

    /* Call SQLBrowseConnect until it returns a value other than */
    /* SQL_NEED_DATA (pass the data source name the first time). */
    /* If SQL_NEED_DATA is returned, call GetUserInput (not      */
    /* shown) to build a dialog from the values in szConnStrOut. */
    /* The user-supplied values are returned in szConnStrIn,     */
    /* which is passed in the next call to SQLBrowseConnect.     */

    lstrcpy(szConnStrIn, "DSN=MyServer");
    do {
      retcode = SQLBrowseConnect(hstmt, szConnStrIn, SQL_NTS, szConnStrOut,
BRWS_LEN, &cbConnStrOut)
      if (retcode == SQL_NEED_DATA)
        GetUserInput(szConnStrOut, szConnStrIn);
    } while (retcode == SQL_NEED_DATA);

    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){

      /* Process data after successful connection */

      retcode = SQLAllocStmt(hdbc, &hstmt);
      if (retcode == SQL_SUCCESS) {
        ...;
        ...;
        ...;
        SQLFreeStmt(hstmt, SQL_DROP);
      }
      SQLDisconnect(hdbc);
    }
  }
  SQLFreeConnect(hdbc);
}
SQLFreeEnv(henv);
```

Related Functions

SQLAllocConnect
SQLConnect
SQLDisconnect
SQLDriverConnect (extension)
SQLDrivers (extension)
SQLFreeConnect

SQLCancel (Core, ODBC 1.0)

Related Functions

SQLCancel cancels the processing on an *hstmt*.

Syntax

RETCODE **SQLCancel**(*hstmt*)

The **SQLCancel** function accepts the following argument.

Type	**Argument**	**Use**	**Description**
HSTMT	*hstmt*	Input	Statement handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLCancel** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLCancel** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	**Error**	**Description**
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
70100	Operation aborted	The data source was unable to process the cancel request.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.

Comments

SQLCancel can cancel the following types of processing on an *hstmt*:

- A function running asynchronously on the *hstmt*.
- A function on an *hstmt* that needs data.
- A function running on the *hstmt* on another thread.

If an application calls **SQLCancel** when no processing is being done on the *hstmt*, **SQLCancel** has the same effect as **SQLFreeStmt** with the SQL_CLOSE option; this behavior is defined only for completeness and applications should call **SQLFreeStmt** to close cursors.

Canceling Asynchronous Processing

After an application calls a function asynchronously, it calls the function repeatedly to determine whether it has finished processing. If the function is still processing, it returns SQL_STILL_EXECUTING. If the function has finished processing, it returns a different code.

After any call to the function that returns SQL_STILL_EXECUTING, an application can call **SQLCancel** to cancel the function. If the cancel request is successful, the driver returns SQL_SUCCESS. This message does not indicate that the function was actually canceled; it indicates that the cancel request was processed. When or if the function is actually canceled is driver- and data source-dependent. The application must continue to call the original function until the return code is not

SQL_STILL_EXECUTING. If the function was successfully canceled, the return code is SQL_ERROR and SQLSTATE S1008 (Operation canceled). If the function completed its normal processing, the return code is SQL_SUCCESS or SQL_SUCCESS_WITH_INFO if the function succeeded or SQL_ERROR and a SQLSTATE other than S1008 (Operation canceled) if the function failed.

Canceling Functions that Need Data

After **SQLExecute** or **SQLExecDirect** returns SQL_NEED_DATA and before data has been sent for all data-at-execution parameters, an application can call **SQLCancel** to cancel the statement execution. After the statement has been canceled, the application can call **SQLExecute** or **SQLExecDirect** again. For more information, see **SQLBindParameter**.

After **SQLSetPos** returns SQL_NEED_DATA and before data has been sent for all data-at-execution columns, an application can call **SQLCancel** to cancel the operation. After the operation has been canceled, the application can call **SQLSetPos** again; canceling does not affect the cursor state or the current cursor position. For more information, see **SQLSetPos**.

Canceling Functions in Multithreaded Applications

In a multithreaded application, the application can cancel a function that is running synchronously on an *hstmt*. To cancel the function, the application calls **SQLCancel** with the same *hstmt* as that used by the target function, but on a different thread. As in canceling a function running asynchronously, the return code of the **SQLCancel** only indicates whether the driver processed the request successfully. The return code of the original function indicates whether it completed normally or was canceled.

Related Functions

- **SQLBindParameter**
- **SQLExecDirect**
- **SQLExecute**
- **SQLFreeStmt**
- **SQLSetPos** (extension)
- **SQLParamData** (extension)
- **SQLPutData** (extension)

SQLColAttributes (Core, ODBC 1.0)

Related Functions

SQLColAttributes returns descriptor information for a column in a result set; it cannot be used to return information about the bookmark column (column 0). Descriptor information is returned as a character string, a 32-bit descriptor-dependent value, or an integer value.

Syntax

RETCODE **SQLColAttributes**(*hstmt*, *icol*, *fDescType*, *rgbDesc*, *cbDescMax*, *pcbDesc*, *pfDesc*)

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

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.
UWORD	*icol*	Input	Column number of result data, ordered sequentially from left to right, starting at 1. Columns may be described in any order.
UWORD	*fDescType*	Input	A valid descriptor type (see "Comments").
PTR	*rgbDesc*	Output	Pointer to storage for the descriptor information. The format of the descriptor information returned depends on the *fDescType*.
SWORD	*cbDescMax*	Input	Maximum length of the *rgbDesc* buffer.
SWORD FAR *	*pcbDesc*	Output	Total number of bytes (excluding the null termination byte for character data) available to return in *rgbDesc*. For character data, if the number of bytes available to return is greater than or equal to *cbDescMax*, the descriptor information in *rgbDesc* is truncated to *cbDescMax* - 1 bytes and is null-terminated by the driver. For all other types of data, the value of *cbValueMax* is ignored and the driver assumes the size of *rgbValue* is 32 bits.
SDWORD FAR *	*pfDesc*	Output	Pointer to an integer value to contain descriptor information for numeric descriptor types, such as SQL_COLUMN_LENGTH.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLColAttributes** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLColAttributes** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)

01004	Data truncated	The buffer *rgbDesc* was not large enough to return the entire string value, so the string value was truncated. The argument *pcbDesc* contains the length of the untruncated string value. (Function returns SQL_SUCCESS_WITH_INFO.)
24000	Invalid cursor state	The statement associated with the *hstmt* did not return a result set. There were no columns to describe.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1002	Invalid column number	(DM) The value specified for the argument *icol* was 0 and the argument *fDescType* was not SQL_COLUMN_COUNT. The value specified for the argument *icol* was greater than the number of columns in the result set and the argument *fDescType* was not SQL_COLUMN_COUNT.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) The function was called prior to calling **SQLPrepare** or **SQLExecDirect** for the *hstmt*. (DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value specified for the argument *cbDescMax* was less than 0.
S1091	Descriptor type out	(DM) The value specified for the

	of range	argument *fDescType* was in the block of numbers reserved for ODBC descriptor types but was not valid for the version of ODBC supported by the driver (see "Comments").
S1C00	Driver not capable	The value specified for the argument *fDescType* was in the range of numbers reserved for driver-specific descriptor types but was not supported by the driver.
S1T00	Timeout expired	The timeout period expired before the data source returned the requested information. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

SQLColAttributes can return any SQLSTATE that can be returned by **SQLPrepare** or **SQLExecute** when called after **SQLPrepare** and before **SQLExecute** depending on when the data source evaluates the SQL statement associated with the *hstmt*.

Comments

SQLColAttributes returns information either in *pfDesc* or in *rgbDesc*. Integer information is returned in *pfDesc* as a 32-bit, signed value; all other formats of information are returned in *rgbDesc*. When information is returned in *pfDesc*, the driver ignores *rgbDesc*, *cbDescMax*, and *pcbDesc*. When information is returned in *rgbDesc*, the driver ignores *pfDesc*.

The currently defined descriptor types, the version of ODBC in which they were introduced, and the arguments in which information is returned for them are shown below; it is expected that more descriptor types will be defined to take advantage of different data sources. Descriptor types from 0 to 999 are reserved by ODBC; driver developers must reserve values greater than or equal to SQL_COLUMN_DRIVER_START for driver-specific use.

A driver must return a value for each of the descriptor types defined in the following table. If a descriptor type does not apply to a driver or data source, then, unless otherwise stated, the driver returns 0 in *pcbDesc* or an empty string in *rgbDesc*.

fDescType	**Information returned in**	**Description**
SQL_COLUMN_AUTO_INCREMENT (ODBC 1.0)	*pfDesc*	TRUE if the column is autoincrement. FALSE if the column is not autoincrement or is not numeric. Auto increment is valid for numeric data type columns only. An application can insert values into an autoincrement column, but cannot update values in the column.
SQL_COLUMN_CASE_SENSITIVE (ODBC 1.0)	*pfDesc*	TRUE if the column is treated as case sensitive for collations and comparisons. FALSE if the column is not treated as case sensitive for collations and comparisons or is noncharacter.
SQL_COLUMN_COUNT (ODBC 1.0)	*pfDesc*	Number of columns available in the result set. The *icol*

		argument is ignored.
SQL_COLUMN_DISPLAY_SIZE (ODBC 1.0)	*pfDesc*	Maximum number of characters required to display data from the column. For more information on display size, see Precision, Scale, Length, and Display Size.
SQL_COLUMN_LABEL (ODBC 2.0)	*rgbDesc*	The column label or title. For example, a column named EmpName might be labeled Employee Name. If a column does not have a label, the column name is returned. If the column is unlabeled and unnamed, an empty string is returned.
SQL_COLUMN_LENGTH (ODBC 1.0)	*pfDesc*	The length in bytes of data transferred on an **SQLGetData** or **SQLFetch** operation if SQL_C_DEFAULT is specified. For numeric data, this size may be different than the size of the data stored on the data source. For more length information, see Precision, Scale, Length, and Display Size.
SQL_COLUMN_MONEY (ODBC 1.0)	*pfDesc*	TRUE if the column is money data type. FALSE if the column is not money data type.
SQL_COLUMN_NAME (ODBC 1.0)	*rgbDesc*	The column name. If the column is unnamed, an empty string is returned.
SQL_COLUMN_NULLABLE (ODBC 1.0)	*pfDesc*	SQL_NO_NULLS if the column does not accept NULL values. SQL_NULLABLE if the column accepts NULL values. SQL_NULLABLE_UNKNOWN if it is not known if the column accepts NULL values.
SQL_COLUMN_OWNER_NAME (ODBC 2.0)	*rgbDesc*	The owner of the table that contains the column. The returned value is implementation-defined if the column is an expression or if the column is part of a view. If the data source does not support owners or the owner name cannot be determined, an empty string is returned.
SQL_COLUMN_PRECISION (ODBC 1.0)	*pfDesc*	The precision of the column on the data source. For more information on precision, see Precision, Scale, Length, and

		Display Size.
SQL_COLUMN_QUALIFIER_NAME (ODBC 2.0)	*rgbDesc*	The qualifier of the table that contains the column. The returned value is implementation-defined if the column is an expression or if the column is part of a view. If the data source does not support qualifiers or the qualifier name cannot be determined, an empty string is returned.
SQL_COLUMN_SCALE (ODBC 1.0)	*pfDesc*	The scale of the column on the data source. For more information on scale, see Precision, Scale, Length, and Display Size.
SQL_COLUMN_SEARCHABLE (ODBC 1.0)	*pfDesc*	SQL_UNSEARCHABLE if the column cannot be used in a **WHERE** clause. SQL_LIKE_ONLY if the column can be used in a **WHERE** clause only with the **LIKE** predicate. SQL_ALL_EXCEPT_LIKE if the column can be used in a **WHERE** clause with all comparison operators except **LIKE**. SQL_SEARCHABLE if the column can be used in a **WHERE** clause with any comparison operator. Columns of type SQL_LONGVARCHAR and SQL_LONGVARBINARY usually return SQL_LIKE_ONLY.
SQL_COLUMN_TABLE_NAME (ODBC 2.0)	*rgbDesc*	The name of the table that contains the column. The returned value is implementation-defined if the column is an expression or if the column is part of a view. If the table name cannot be determined, an empty string is returned.
SQL_COLUMN_TYPE (ODBC 1.0)	*pfDesc*	SQL data type. This can be an ODBC SQL data type or a driver-specific SQL data type. For a list of valid ODBC SQL data types, see SQL Data Types. For information about driver-specific SQL data types, see the driver's documentation.
SQL_COLUMN_TYPE_NAME	*rgbDesc*	Data source-dependent data

(ODBC 1.0)		type name; for example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINARY", or "CHAR ( ) FOR BIT DATA". If the type is unknown, an empty string is returned.
SQL_COLUMN_UNSIGNED (ODBC 1.0)	*pfDesc*	TRUE if the column is unsigned (or not numeric). FALSE if the column is signed.
SQL_COLUMN_UPDATABLE (ODBC 1.0)	*pfDesc*	Column is described by the values for the defined constants: SQL_ATTR_READONLY SQL_ATTR_WRITE SQL_ATTR_READWRITE_UNKNOWN SQL_COLUMN_UPDATABLE describes the updatability of the column in the result set. Whether a column is updatable can be based on the data type, user privileges, and the definition of the result set itself. If it is unclear whether a column is updatable, SQL_ATTR_READWRITE_UNKNOWN should be returned.

This function is an extensible alternative to **SQLDescribeCol**. **SQLDescribeCol** returns a fixed set of descriptor information based on ANSI-89 SQL. **SQLColAttributes** allows access to the more extensive set of descriptor information available in ANSI SQL-92 and DBMS vendor extensions.

Related Functions

SQLBindCol
SQLCancel
SQLDescribeCol
SQLExtendedFetch (extension)
SQLFetch

SQLColumnPrivileges (Extension Level 2, ODBC 1.0)

Code Example Related Functions

SQLColumnPrivileges returns a list of columns and associated privileges for the specified table. 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*)

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

Type	**Argument**	**Use**	**Description**
HSTMT	*hstmt*	Input	Statement handle.
UCHAR FAR *	*szTableQualifier*	Input	Table qualifier. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	*cbTableQualifier*	Input	Length of *szTableQualifier*.
UCHAR FAR *	*szTableOwner*	Input	Owner name. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.
SWORD	*cbTableOwner*	Input	Length of *szTableOwner*.
UCHAR FAR *	*szTableName*	Input	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_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLColumnPrivileges** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLColumnPrivileges** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	**Error**	**Description**
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
24000	Invalid cursor state	(DM) A cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch**

		had been called.
		A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*.
		The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called.
		(DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the name length arguments was less than 0, but not equal to SQL_NTS.
		The value of one of the name length arguments exceeded the maximum length value for the corresponding qualifier or name (see "Comments").
S1C00	Driver not capable	A table qualifier was specified and the driver or data source does not support qualifiers.
		A table owner was specified and the driver or data source does not support owners.
		A string search pattern was specified for the column name and the data source does not support search patterns for that argument.
		The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver

		or data source.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

SQLColumnPrivileges returns the results as a standard result set, ordered by TABLE_QUALIFIER, TABLE_OWNER, TABLE_NAME, COLUMN_NAME, and PRIVILEGE. The following table lists the columns in the result set.

Note **SQLColumnPrivileges** might not return privileges for all columns. For example, a driver might not return information about privileges for pseudo-columns, such as Oracle ROWID. Applications can use any valid column, regardless of whether it is returned by **SQLColumnPrivileges**.

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the TABLE_QUALIFIER, TABLE_OWNER, TABLE_NAME, and COLUMN_NAME columns, an application can call **SQLGetInfo** with the SQL_MAX_QUALIFIER_NAME_LEN, SQL_MAX_OWNER_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN options.

Column Name	**Data Type**	**Comments**
TABLE_QUALIFIER	Varchar(128)	Table qualifier identifier; NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have qualifiers.
TABLE_OWNER	Varchar(128)	Table owner identifier; NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.
TABLE_NAME	Varchar(128) not NULL	Table identifier.
COLUMN_NAME	Varchar(128) not NULL	Column identifier.
GRANTOR	Varchar(128)	Identifier of the user who granted the privilege; NULL if not applicable to the data source.
GRANTEE	Varchar(128) not NULL	Identifier of the user to whom the privilege was granted.
PRIVILEGE	Varchar(128) not NULL	Identifies the column privilege. May be one of the following or others supported by the data source when implementation-defined: SELECT: The grantee is permitted to retrieve data for the column. INSERT: The grantee is permitted to provide data for the column in new rows that are inserted into the associated table.

		UPDATE: The grantee is permitted to update data in the column. REFERENCES: The grantee is permitted to refer to the column within a constraint (for example, a unique, referential, or table check constraint).
IS_GRANTABLE	Varchar(3)	Indicates whether the grantee is permitted to grant the privilege to other users; "YES", "NO", or NULL if unknown or not applicable to the data source.

The *szColumnName* argument accepts a search pattern.

Code Example

For a code example of a similar function, see **SQLColumns**.

Related Functions

SQLBindCol
SQLCancel
SQLColumns (extension)
SQLExtendedFetch (extension)
SQLFetch
SQLTablePrivileges (extension)
SQLTables (extension)

SQLColumns (Extension Level 1, ODBC 1.0)

Code Example Related Functions

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*)

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

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.
UCHAR FAR *	*szTableQualifier*	Input	Qualifier name. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	*cbTableQualifier*	Input	Length of *szTableQualifier*.
UCHAR FAR *	*szTableOwner*	Input	String search pattern for owner names. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.
SWORD	*cbTableOwner*	Input	Length of *szTableOwner*.
UCHAR FAR *	*szTableName*	Input	String search pattern for table names.
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_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLColumns** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLColumns** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.

24000	Invalid cursor state	(DM) A cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch** had been called. A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the name length arguments was less than 0, but not equal to SQL_NTS. The value of one of the name length arguments exceeded the maximum length value for the corresponding qualifier or name. The maximum length of each qualifier or name may be obtained by calling **SQLGetInfo** with the *fInfoType* values (see "Comments").
S1C00	Driver not capable	A table qualifier was specified and the driver or data source does not support qualifiers. A table owner was specified and the driver or data source does not support owners. A string search pattern was specified for the table owner, table name, or column name and the data source does not

		support search patterns for one or more of those arguments. The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

This function is typically used before statement execution to retrieve information about columns for a table or tables from the data source's catalog. Note by contrast, that the functions **SQLColAttributes** and **SQLDescribeCol** describe the columns in a result set and that the function **SQLNumResultCols** returns the number of columns in a result set.

Note **SQLColumns** might not return all columns. For example, a driver might not return information about pseudo-columns, such as Oracle ROWID. Applications can use any valid column, regardless of whether it is returned by **SQLColumns**.

SQLColumns returns the results as a standard result set, ordered by TABLE_QUALIFIER, TABLE_OWNER, and TABLE_NAME. The following table lists the columns in the result set. Additional columns beyond column 12 (REMARKS) can be defined by the driver.

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the TABLE_QUALIFIER, TABLE_OWNER, TABLE_NAME, and COLUMN_NAME columns, an application can call **SQLGetInfo** with the SQL_MAX_QUALIFIER_NAME_LEN, SQL_MAX_OWNER_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN options.

Column Name	**Data Type**	**Comments**
TABLE_QUALIFIER	Varchar(128)	Table qualifier identifier; NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have qualifiers.
TABLE_OWNER	Varchar(128)	Table owner identifier; NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.
TABLE_NAME	Varchar(128) not NULL	Table identifier.
COLUMN_NAME	Varchar(128) not NULL	Column identifier.
DATA_TYPE	Smallint not NULL	SQL data type. This can be an ODBC SQL data type or a driver-specific SQL data type. For a list of valid ODBC SQL data types, see SQL Data Types. For information about driver-

		specific SQL data types, see the driver's documentation.
TYPE_NAME	Varchar(128) not NULL	Data source-dependent data type name; for example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINARY", or "CHAR ( ) FOR BIT DATA".
PRECISION	Integer	The precision of the column on the data source. For precision information, see Precision, Scale, Length, and Display Size.
LENGTH	Integer	The length in bytes of data transferred on an **SQLGetData** or **SQLFetch** operation if SQL_C_DEFAULT is specified. For numeric data, this size may be different than the size of the data stored on the data source. This value is the same as the PRECISION column for character or binary data. For more information about length, see Precision, Scale, Length, and Display Size.
SCALE	Smallint	The scale of the column on the data source. For more scale information, see Precision, Scale, Length, and Display Size. NULL is returned for data types where scale is not applicable.
RADIX	Smallint	For numeric data types, either 10 or 2. If it is 10, the values in PRECISION and SCALE give the number of decimal digits allowed for the column. For example, a DECIMAL(12,5) column would return a RADIX of 10, a PRECISION of 12, and a SCALE of 5; A FLOAT column could return a RADIX of 10, a PRECISION of 15 and a SCALE of NULL. If it is 2, the values in PRECISION and SCALE give the number of bits allowed in the column. For example, a FLOAT column could return a RADIX of 2, a PRECISION of 53, and a SCALE of NULL. NULL is returned for data types where radix is not applicable.
NULLABLE	Smallint not NULL	SQL_NO_NULLS if the column does not accept NULL values. SQL_NULLABLE if the column accepts NULL values. SQL_NULLABLE_UNKNOWN if it is not known if the column accepts NULL values.
REMARKS	Varchar(254)	A description of the column.

The *szTableOwner*, *szTableName*, and *szColumnName* arguments accept search patterns.

▪

Code Example

In the following example, an application declares storage locations for the result set returned by **SQLColumns**. It calls **SQLColumns** to return a result set that describes each column in the EMPLOYEE table. It then calls **SQLBindCol** to bind the columns in the result set to the storage locations. Finally, the application fetches each row of data with **SQLFetch** and processes it.

```
#define STR_LEN 128+1
#define REM_LEN 254+1

/* Declare storage locations for result set data */

UCHAR  szQualifier[STR_LEN], szOwner[STR_LEN];
UCHAR  szTableName[STR_LEN], szColName[STR_LEN];
UCHAR  szTypeName[STR_LEN], szRemarks[REM_LEN];
SDWORD Precision, Length;
SWORD  DataType, Scale, Radix, Nullable;

/* Declare storage locations for bytes available to return */

SDWORD cbQualifier, cbOwner, cbTableName, cbColName;
SDWORD cbTypeName, cbRemarks, cbDataType, cbPrecision;
SDWORD cbLength, cbScale, cbRadix, cbNullable;

/* All qualifiers, all owners, EMPLOYEE table, all columns */

retcode = SQLColumns(hstmt, NULL, 0, NULL, 0, "EMPLOYEE", SQL_NTS, NULL, 0);

if (retcode == SQL_SUCCESS) {

  /* Bind columns in result set to storage locations */

  SQLBindCol(hstmt, 1, SQL_C_CHAR, szQualifier, STR_LEN,&cbQualifier);
  SQLBindCol(hstmt, 2, SQL_C_CHAR, szOwner, STR_LEN, &cbOwner);
  SQLBindCol(hstmt, 3, SQL_C_CHAR, szTableName, STR_LEN,&cbTableName);
  SQLBindCol(hstmt, 4, SQL_C_CHAR, szColName, STR_LEN, &cbColName);
  SQLBindCol(hstmt, 5, SQL_C_SSHORT, &DataType, 0, &cbDataType);
  SQLBindCol(hstmt, 6, SQL_C_CHAR, szTypeName, STR_LEN, &cbTypeName);
  SQLBindCol(hstmt, 7, SQL_C_SLONG, &Precision, 0, &cbPrecision);
  SQLBindCol(hstmt, 8, SQL_C_SLONG, &Length, 0, &cbLength);
  SQLBindCol(hstmt, 9, SQL_C_SSHORT, &Scale, 0, &cbScale);
  SQLBindCol(hstmt, 10, SQL_C_SSHORT, &Radix, 0, &cbRadix);
  SQLBindCol(hstmt, 11, SQL_C_SSHORT, &Nullable, 0, &cbNullable);
  SQLBindCol(hstmt, 12, SQL_C_CHAR, szRemarks, REM_LEN, &cbRemarks);

  while(TRUE) {
    retcode = SQLFetch(hstmt);
    if (retcode == SQL_ERROR || retcode == SQL_SUCCESS_WITH_INFO) {
      show_error( );
    }
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){
      ...;  /* Process fetched data */
    } else {
      break;
    }
  }
}
```

Related Functions

- **SQLBindCol**
- **SQLCancel**
- **SQLColumnPrivileges** (extension)
- **SQLExtendedFetch** (extension)
- **SQLFetch**
- **SQLStatistics** (extension)
- **SQLTables** (extension)
- **SQLTablePrivileges** (extension)

SQLConnect (Core, ODBC 1.0)

Code Example Related Functions

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*)

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

Type	Argument	Use	Description
HDBC	*hdbc*	Input	Connection handle.
UCHAR FAR *	*szDSN*	Input	Data source name.
SWORD	*cbDSN*	Input	Length of *szDSN*.
UCHAR FAR *	*szUID*	Input	User identifier.
SWORD	*cbUID*	Input	Length of *szUID*.
UCHAR FAR *	*szAuthStr*	Input	Authentication string (typically the password).
SWORD	*cbAuthStr*	Input	Length of *szAuthStr*.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLConnect** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLConnect** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08001	Unable to connect to data source	The driver was unable to establish a connection with the data source.
08002	Connection in use	(DM) The specified *hdbc* had already been used to establish a connection with a data source and the connection was still open.
08004	Data source rejected establishment of connection	The data source rejected the establishment of the connection for implementation-defined reasons.
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was attempting to connect failed before the function completed processing.
28000	Invalid authorization specification	The value specified for the argument *szUID* or the value specified for the argument *szAuthStr* violated restrictions defined by the data source.
IM001	Driver does not support this function	(DM) The driver specified by the data source name does not support the function.
IM002	Data source not	(DM) The data source name specified in

	found and no default driver specified	the argument *szDSN* was not found in the ODBC.INI file or registry, nor was there a default driver specification. (DM) The ODBC.INI file could not be found.
IM003	Specified driver could not be loaded	(DM) The driver listed in the data source specification in the ODBC.INI file or registry was not found or could not be loaded for some other reason.
IM004	Driver's SQLAllocEnv failed	(DM) During **SQLConnect**, the Driver Manager called the driver's **SQLAllocEnv** function and the driver returned an error.
IM005	Driver's SQLAllocConnect failed	(DM) During **SQLConnect**, the Driver Manager called the driver's **SQLAllocConnect** function and the driver returned an error.
IM006	Driver's SQLSetConnect-Option failed	(DM) During **SQLConnect**, the Driver Manager called the driver's **SQLSetConnectOption** function and the driver returned an error. (Function returns SQL_SUCCESS_WITH_INFO).
IM009	Unable to load translation DLL	The driver was unable to load the translation DLL that was specified for the data source.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	(DM) The Driver Manager was unable to allocate memory required to support execution or completion of the function. The driver was unable to allocate memory required to support execution or completion of the function.
S1090	Invalid string or buffer length	(DM) The value specified for argument *cbDSN* was less than 0, but not equal to SQL_NTS. (DM) The value specified for argument *cbDSN* exceeded the maximum length for a data source name. (DM) The value specified for argument *cbUID* was less than 0, but not equal to SQL_NTS. (DM) The value specified for argument *cbAuthStr* was less than 0, but not equal to SQL_NTS.
S1T00	Timeout expired	The timeout period expired before the connection to the data source completed. The timeout period is set through **SQLSetConnectOption**, SQL_LOGIN_TIMEOUT.

Comments

The Driver Manager does not load a driver until the application calls a function (**SQLConnect**, **SQLDriverConnect**, or **SQLBrowseConnect**) to connect to the driver. Until that point, the Driver Manager works with its own handles and manages connection information. When the application calls a connection function, the Driver Manager checks if a driver is currently loaded for the specified *hdbc*:

- If a driver is not loaded, the Driver Manager loads the driver and calls **SQLAllocEnv**, **SQLAllocConnect**, **SQLSetConnectOption** (if the application specified any connection options), and the connection function in the driver. The Driver Manager returns SQLSTATE IM006 (Driver's SQLSetConnectOption failed) and SQL_SUCCESS_WITH_INFO for the connection function if the driver returned an error for **SQLSetConnectOption**.
- If the specified driver is already loaded on the *hdbc*, the Driver Manager only calls the connection function in the driver. In this case, the driver must ensure that all connection options for the *hdbc* maintain their current settings.
- If a different driver is loaded, the Driver Manager calls **SQLFreeConnect** and **SQLFreeEnv** in the loaded driver and then unloads that driver. It then performs the same operations as when a driver is not loaded.

The driver then allocates handles and initializes itself.

Note To resolve the addresses of the ODBC functions exported by the driver, the Driver Manager checks if the driver exports a dummy function with the ordinal 199. If it does not, the Driver Manager resolves the addresses by name. If it does, the Driver Manager resolves the addresses of the ODBC functions by ordinal, which is faster. The ordinal values of the ODBC functions must match the values of the *fFunction* argument in **SQLGetFunctions**; all other exported functions (such as **WEP**) must have ordinal values outside the range 1-199.

When the application calls **SQLDisconnect**, the Driver Manager calls **SQLDisconnect** in the driver. However, it does not unload the driver. This keeps the driver in memory for applications that repeatedly connect to and disconnect from a data source. When the application calls **SQLFreeConnect**, the Driver Manager calls **SQLFreeConnect** and **SQLFreeEnv** in the driver and then unloads the driver.

An ODBC application can establish more than one connection.

Driver Manager Guidelines

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

- If *szDSN* contains a valid data source name, the Driver Manager locates the corresponding data source specification in the ODBC.INI file or registry and loads the associated driver DLL. The Driver Manager passes each **SQLConnect** argument to the driver.
- If the data source name cannot be found or *szDSN* is a null pointer, the Driver Manager locates the default data source specification and loads the associated driver DLL. The Driver Manager passes each **SQLConnect** argument to the driver.
- If the data source name cannot be found or *szDSN* is a null pointer, and the default data source specification does not exist, the Driver Manager returns SQL_ERROR with SQLSTATE IM002 (Data source name not found and no default driver specified).

After being loaded by the Driver Manager, a driver can locate its corresponding data source specification in the ODBC.INI file or registry and use driver-specific information from the specification to complete its set of required connection information.

If a default translation DLL is specified in the ODBC.INI file or registry for the data source, the driver loads it. A different translation DLL can be loaded by calling **SQLSetConnectOption** with the SQL_TRANSLATE_DLL option. A translation option can be specified by calling **SQLSetConnectOption** with the SQL_TRANSLATE_OPTION option.

If a driver supports **SQLConnect**, the driver keyword section of the ODBC.INF file for the driver must contain the **ConnectFunctions** keyword with the first character set to "Y".

▪

Code Example

In the following example, an application allocates environment and connection handles. It then connects to the EmpData data source with the user ID JohnS and the password Sesame and processes data. When it has finished processing data, it disconnects from the data source and frees the handles.

```
HENV    henv;
HDBC    hdbc;
HSTMT   hstmt;
RETCODE retcode;

retcode = SQLAllocEnv(&henv);              /* Environment handle */
if (retcode == SQL_SUCCESS) {
  retcode = SQLAllocConnect(henv, &hdbc); /* Connection handle */
  if (retcode == SQL_SUCCESS) {

    /* Set login timeout to 5 seconds. */

    SQLSetConnectOption(hdbc, SQL_LOGIN_TIMEOUT, 5);

    /* Connect to data source */

    retcode = SQLConnect(hdbc, "EmpData", SQL_NTS, "JohnS", SQL_NTS, 
"Sesame", SQL_NTS);
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){

      /* Process data after successful connection */

      retcode = SQLAllocStmt(hdbc, &hstmt); /* Statement handle */
      if (retcode == SQL_SUCCESS) {
        ...;
        ...;
        ...;
        SQLFreeStmt(hstmt, SQL_DROP);
      }
      SQLDisconnect(hdbc);
    }
    SQLFreeConnect(hdbc);
  }
  SQLFreeEnv(henv);
}
```

Related Functions

SQLAllocConnect
SQLAllocStmt
SQLBrowseConnect (extension)
SQLDisconnect
SQLDriverConnect (extension)
SQLGetConnectOption (extension)
SQLSetConnectOption (extension)

SQLDataSources (Extension Level 2, ODBC 1.0)

Related Functions

SQLDataSources lists data source names. This function is implemented solely by the Driver Manager.

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 Manager fetches the next data source name in the list (SQL_FETCH_NEXT) or whether the search starts from the beginning of the list (SQL_FETCH_FIRST).
UCHAR FAR *	*szDSN*	Output	Pointer to storage for the data source name.
SWORD	*cbDSNMax*	Input	Maximum length of the *szDSN* buffer; this does not need to be longer than SQL_MAX_DSN_LENGTH + 1.
SWORD FAR *	*pcbDSN*	Output	Total number of bytes (excluding the null termination byte) available to return in *szDSN*. If the number of bytes available to return is greater than or equal to *cbDSNMax*, the data source name in *szDSN* is truncated to *cbDSNMax* - 1 bytes.
UCHAR FAR *	*szDescription*	Output	Pointer to storage for the description of the driver associated with the data source. For example, dBASE or SQL Server.
SWORD	*cbDescriptionMax*	Input	Maximum length of the *szDescription* buffer; this should be at least 255 bytes.
SWORD FAR *	*pcbDescription*	Output	Total number of bytes (excluding the null termination byte) available to return in *szDescription*. If the number of bytes available to return is greater than or equal to *cbDescriptionMax*, the driver description in *szDescription* is truncated to *cbDescriptionMax* - 1 bytes.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA_FOUND, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLDataSources** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLDataSources** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	**Error**	**Description**
01000	General warning	(DM) Driver Manager-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	(DM) The buffer *szDSN* was not large enough to return the entire data source name, so the name was truncated. The argument *pcbDSN* contains the length of the entire data source name. (Function returns SQL_SUCCESS_WITH_INFO.)
		(DM) The buffer *szDescription* was not large enough to return the entire driver description, so the description was truncated. The argument *pcbDescription* contains the length of the untruncated data source description. (Function returns SQL_SUCCESS_WITH_INFO.)
S1000	General error	(DM) An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	(DM) The Driver Manager was unable to allocate memory required to support execution or completion of the function.
S1090	Invalid string or buffer length	(DM) The value specified for argument *cbDSNMax* was less than 0.
		(DM) The value specified for argument *cbDescriptionMax* was less than 0.
S1103	Direction option out of range	(DM) The value specified for the argument *fDirection* was not equal to SQL_FETCH_FIRST or SQL_FETCH_NEXT.

Comments

Because **SQLDataSources** is implemented in the Driver Manager, it is supported for all drivers regardless of a particular driver's conformance level.

An application can call **SQLDataSources** multiple times to retrieve all data source names. The Driver Manager retrieves this information from the ODBC.INI file or the registry. When there are no more data source names, the Driver Manager returns SQL_NO_DATA_FOUND. If **SQLDataSources** is called with SQL_FETCH_NEXT immediately after it returns SQL_NO_DATA_FOUND, it will return the first data source name.

If SQL_FETCH_NEXT is passed to **SQLDataSources** the very first time it is called, it will return the first data source name.

The driver determines how data source names are mapped to actual data sources.

Related Functions

SQLBrowseConnect (extension)
SQLConnect
SQLDriverConnect (extension)
SQLDrivers (extension)

SQLDescribeCol (Core, ODBC 1.0)

Related Functions

SQLDescribeCol returns the result descriptor − column name, type, precision, scale, and nullability − for one column in the result set; it cannot be used to return information about the bookmark column (column 0).

Syntax

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

The **SQLDescribeCol** 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.
UCHAR FAR *	*szColName*	Output	Pointer to storage for the column name. If the column is unnamed or the column name cannot be determined, the driver returns an empty string.
SWORD	*cbColNameMax*	Input	Maximum length of the *szColName* buffer.
SWORD FAR *	*pcbColName*	Output	Total number of bytes (excluding the null termination byte) available to return in *szColName*. If the number of bytes available to return is greater than or equal to *cbColNameMax*, the column name in *szColName* is truncated to *cbColNameMax* - 1 bytes.
SWORD FAR *	*pfSqlType*	Output	The SQL data type of the column. This must be one of the following values: SQL_BIGINT SQL_BINARY SQL_BIT SQL_CHAR SQL_DATE SQL_DECIMAL SQL_DOUBLE SQL_FLOAT SQL_INTEGER SQL_LONGVARBINARY SQL_LONGVARCHAR SQL_NUMERIC SQL_REAL SQL_SMALLINT SQL_TIME SQL_TIMESTAMP SQL_TINYINT SQL_VARBINARY SQL_VARCHAR or a driver-specific SQL data type. If the data type cannot be determined, the driver returns 0.

			For more information, see SQL Data Types. For information about driver-specific SQL data types, see the driver's documentation.
UDWORD FAR *	*pcbColDef*	Output	The precision of the column on the data source. If the precision cannot be determined, the driver returns 0. For more information on precision, see Precision, Scale, Length, and Display Size.
SWORD FAR *	*pibScale*	Output	The scale of the column on the data source. If the scale cannot be determined or is not applicable, the driver returns 0. For more information on scale, see Precision, Scale, Length, and Display Size.
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.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLDescribeCol** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLDescribeCol** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	**Error**	**Description**
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The buffer *szColName* was not large enough to return the entire column name, so the column name was truncated. The argument *pcbColName* contains the length of the untruncated column name. (Function returns SQL_SUCCESS_WITH_INFO.)
24000	Invalid cursor state	The statement associated with the *hstmt* did not return a result set. There were no columns to describe.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no

		implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1002	Invalid column number	(DM) The value specified for the argument *icol* was 0. The value specified for the argument *icol* was greater than the number of columns in the result set.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) The function was called prior to calling **SQLPrepare** or **SQLExecDirect** for the *hstmt*. (DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value specified for argument *cbColNameMax* was less than 0.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

SQLDescribeCol can return any SQLSTATE that can be returned by **SQLPrepare** or **SQLExecute** when called after **SQLPrepare** and before **SQLExecute** depending on when the data source evaluates the SQL statement associated with the *hstmt*.

Comments

An application typically calls **SQLDescribeCol** 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 the X/Open and SQL Access Group Call Level Interface specification does not specify the option for **SQLDescribeCol**.

Related Functions

- SQLBindCol
- SQLCancel
- SQLColAttributes
- SQLFetch
- SQLNumResultCols
- SQLPrepare

SQLDescribeParam (Extension Level 2, ODBC 1.0)

Related Functions

SQLDescribeParam returns the description of a parameter marker associated with a prepared SQL statement.

Syntax

RETCODE **SQLDescribeParam**(*hstmt*, *ipar*, *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 ordered sequentially left to right, starting at 1.
SWORD FAR *	*pfSqlType*	Output	The SQL data type of the parameter. This must be one of the following values: SQL_BIGINT SQL_BINARY SQL_BIT SQL_CHAR SQL_DATE SQL_DECIMAL SQL_DOUBLE SQL_FLOAT SQL_INTEGER SQL_LONGVARBINARY SQL_LONGVARCHAR SQL_NUMERIC SQL_REAL SQL_SMALLINT SQL_TIME SQL_TIMESTAMP SQL_TINYINT SQL_VARBINARY SQL_VARCHAR or a driver-specific SQL data type. For more information, see SQL Data Types. For information about driver-specific SQL data types, see the driver's documentation.
UDWORD FAR *	*pcbColDef*	Output	The precision of the column or expression of the corresponding parameter marker as defined by the data source. For further information concerning precision, see Precision, Scale, Length, and Display Size.
SWORD FAR *	*pibScale*	Output	The scale of the column or expression of the corresponding parameter as defined by the data source. For more information on scale, see Precision, Scale, Length, and Display Size.
SWORD FAR *	*pfNullable*	Output	Indicates whether the parameter allows NULL values. One of the

following:

SQL_NO_NULLS: The parameter does not allow NULL values (this is the default value).

SQL_NULLABLE: The parameter allows NULL values.

SQL_NULLABLE_UNKNOWN: The driver cannot determine if the parameter allows NULL values.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLDescribeParam** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLDescribeParam** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	**Error**	**Description**
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) The function was called prior to calling **SQLPrepare** or **SQLExecDirect** for the *hstmt*. (DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This

		function was called before data was sent for all data-at-execution parameters or columns.
S1093	Invalid parameter number	(DM) The value specified for the argument *ipar* was 0. The value specified for the argument *ipar* was greater than the number of parameters in the associated SQL statement.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

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

SQLDescribeParam does not return the type (input, input/output, or output) of a parameter in an SQL statement. Except in calls to procedures, all parameters in SQL statements are input parameters. To determine the type of each parameter in a call to a procedure, an application calls **SQLProcedureColumns**.

Related Functions

- SQLCancel
- SQLExecute
- SQLPrepare
- SQLBindParameter

SQLDisconnect (Core, ODBC 1.0)

Code Example Related Functions

SQLDisconnect closes the connection associated with a specific connection handle.

Syntax

RETCODE **SQLDisconnect**(*hdbc*)

The **SQLDisconnect** function accepts the following argument.

Type	Argument	Use	Description
HDBC	*hdbc*	Input	Connection handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLDisconnect** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLDisconnect** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01002	Disconnect error	An error occurred during the disconnect. However, the disconnect succeeded. (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection not open	(DM) The connection specified in the argument *hdbc* was not open.
25000	Invalid transaction state	There was a transaction in process on the connection specified by the argument *hdbc*. The transaction remains active.
IM001	Driver does not support this function	(DM) The driver associated with the *hdbc* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1010	Function sequence error	(DM) An asynchronously executing function was called for an *hstmt* associated with the *hdbc* and was still executing when **SQLDisconnect** was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for an *hstmt* associated with the *hdbc* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.

Comments

If an application calls **SQLDisconnect** after **SQLBrowseConnect** returns SQL_NEED_DATA and before it returns a different return code, the driver cancels the connection browsing process and returns the *hdbc* to an unconnected state.

If an application calls **SQLDisconnect** while there is an incomplete transaction associated with the connection handle, the driver returns SQLSTATE 25000 (Invalid transaction state), indicating that the transaction is unchanged and the connection is open. An incomplete transaction is one that has not been committed or rolled back with **SQLTransact**.

If an application calls **SQLDisconnect** before it has freed all *hstmts* associated with the connection, the driver frees those *hstmts* after it successfully disconnects from the data source. However, if one or more of the *hstmts* associated with the connection are still executing asynchronously, **SQLDisconnect** will return SQL_ERROR with a SQLSTATE value of S1010 (Function sequence error).

Code Example

See **SQLBrowseConnect** and **SQLConnect**.

Related Functions

SQLAllocConnect
SQLConnect
SQLDriverConnect (extension)
SQLFreeConnect
SQLTransact

SQLDriverConnect (Extension Level 1, ODBC 1.0)

Related Functions

SQLDriverConnect is an alternative to **SQLConnect**. It supports data sources that require more connection information than the three arguments in **SQLConnect**; dialog boxes to prompt the user for all connection information; and data sources that are not defined in the ODBC.INI file or registry.

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.
- Establish a connection to a data source that is not defined in the ODBC.INI file or registry. If the application supplies a partial connection string, the driver can 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.

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	Window handle. The application can pass the handle of the parent window, if applicable, or a null pointer if either the window handle is not applicable or if **SQLDriverConnect** will not present any dialog boxes.
UCHAR FAR *	*szConnStrIn*	Input	A full connection string (see the syntax in "Comments"), a partial connection string, or an empty string.
SWORD	*cbConnStrIn*	Input	Length of *szConnStrIn*.
UCHAR FAR *	*szConnStrOut*	Output	Pointer to storage for the completed connection string. 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	Maximum length of the *szConnStrOut* buffer.
SWORD FAR *	*pcbConnStrOut*	Output	Pointer to the total number of bytes (excluding the null termination byte) available to return in *szConnStrOut*. If the number of bytes available to return is greater than or equal to *cbConnStrOutMax*, the completed connection string in *szConnStrOut* is truncated to *cbConnStrOutMax* - 1 bytes.
UWORD	*fDriverCompletion*	Input	Flag which indicates whether

Driver Manager or driver must prompt for more connection information:

SQL_DRIVER_PROMPT, SQL_DRIVER_COMPLETE, SQL_DRIVER_COMPLETE_REQUIRED, or SQL_DRIVER_NOPROMPT.

(See "Comments," for additional information.)

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA_FOUND, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLDriverConnect** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLDriverConnect** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	**Error**	**Description**
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The buffer *szConnStrOut* was not large enough to return the entire connection string, so the connection string was truncated. The argument *pcbConnStrOut* contains the length of the untruncated connection string. (Function returns SQL_SUCCESS_WITH_INFO.)
01S00	Invalid connection string attribute	An invalid attribute keyword was specified in the connection string (*szConnStrIn*) but the driver was able to connect to the data source anyway. (Function returns SQL_SUCCESS_WITH_INFO.)
08001	Unable to connect to data source	The driver was unable to establish a connection with the data source.
08002	Connection in use	(DM) The specified *hdbc* had already been used to establish a connection with a data source and the connection was still open.
08004	Data source rejected establishment of connection	The data source rejected the establishment of the connection for implementation-defined reasons.
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was attempting to connect failed before the function completed processing.
28000	Invalid authorization specification	Either the user identifier or the authorization string or both as specified in the connection string (*szConnStrIn*)

		violated restrictions defined by the data source.
IM001	Driver does not support this function	(DM) The driver corresponding to the specified data source name does not support the function.
IM002	Data source not found and no default driver specified	(DM) The data source name specified in the connection string (*szConnStrIn*) was not found in the ODBC.INI file or registry and there was no default driver specification. (DM) The ODBC.INI file could not be found.
IM003	Specified driver could not be loaded	(DM) The driver listed in the data source specification in the ODBC.INI file or registry, or specified by the **DRIVER** keyword, was not found or could not be loaded for some other reason.
IM004	Driver's SQLAllocEnv failed	(DM) During **SQLDriverConnect**, the Driver Manager called the driver's **SQLAllocEnv** function and the driver returned an error.
IM005	Driver's SQLAllocConnect failed	(DM) During **SQLDriverConnect**, the Driver Manager called the driver's **SQLAllocConnect** function and the driver returned an error.
IM006	Driver's SQLSetConnect-Option failed	(DM) During **SQLDriverConnect**, the Driver Manager called the driver's **SQLSetConnectOption** function and the driver returned an error.
IM007	No data source or driver specified; dialog prohibited	No data source name or driver was specified in the connection string and *fDriverCompletion* was SQL_DRIVER_NOPROMPT.
IM008	Dialog failed	(DM) The Driver Manager attempted to display the SQL Data Sources dialog box and failed. The driver attempted to display its login dialog box and failed.
IM009	Unable to load translation DLL	The driver was unable to load the translation DLL that was specified for the data source or for the connection.
IM010	Data source name too long	(DM) The attribute value for the DSN keyword was longer than SQL_MAX_DSN_LENGTH characters.
IM011	Driver name too long	(DM) The attribute value for the DRIVER keyword was longer than 255 characters.
IM012	DRIVER keyword syntax error	(DM) The keyword-value pair for the DRIVER keyword contained a syntax error.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.

S1001	Memory allocation failure	The Driver Manager was unable to allocate memory required to support execution or completion of the function. The driver was unable to allocate memory required to support execution or completion of the function.
S1090	Invalid string or buffer length	(DM) The value specified for argument *cbConnStrIn* was less than 0 and was not equal to SQL_NTS. (DM) The value specified for argument *cbConnStrOutMax* was less than 0.
S1110	Invalid driver completion	(DM) The value specified for the argument *fDriverCompletion* was not equal to SQL_DRIVER_PROMPT, SQL_DRIVER_COMPLETE, SQL_DRIVER_COMPLETE_REQUIRED or SQL_DRIVER_NOPROMPT.
S1T00	Timeout expired	The timeout period expired before the connection to the data source completed. The timeout period is set through **SQLSetConnectOption**, SQL_LOGIN_TIMEOUT.

Comments

Connection Strings

A connection string has the following syntax:

connection-string ::= *empty-string*[;] | *attribute*[;] | *attribute*; *connection-string*
empty-string ::=
attribute ::= *attribute-keyword*=*attribute-value* | DRIVER={*attribute-value*}
(The braces ({}) are literal; the application must specify them.)
attribute-keyword ::= DSN | UID | PWD
| *driver-defined-attribute-keyword*
attribute-value ::= *character-string*
driver-defined-attribute-keyword ::= *identifier*

where *character-string* has zero or more characters; *identifier* has one or more characters; *attribute-keyword* is case insensitive; *attribute-value* may be case sensitive; and the value of the **DSN** keyword does not consist solely of blanks. Because of connection string and initialization file grammar, keywords and attribute values that contain the characters **[]{}(),;?*=!@** should be avoided. Because of the registry grammar, keywords and data source names cannot contain the backslash (\) character.

Note The **DRIVER** keyword was introduced in ODBC 2.0 and is not supported by ODBC 1.0 drivers.

The connection string may include any number of driver-defined keywords. Because the **DRIVER** keyword does not use information from the ODBC.INI file or registry, the driver must define enough keywords so that a driver can connect to a data source using only the information in the connection string. (For more information, see "Driver Guidelines," later in this section.) The driver defines which keywords are required in order to connect to the data source.

If any keywords are repeated in the connection string, the driver uses the value associated with the first occurrence of the keyword. If the **DSN** and **DRIVER** keywords are included in the same connection string, the Driver Manager and the driver use whichever keyword appears first. The following table describes the attribute values of the **DSN**, **DRIVER**, **UID**, and **PWD** keywords.

Keyword	**Attribute value description**
DSN	Name of a data source as returned by **SQLDataSources** or the data sources dialog box of **SQLDriverConnect**.
DRIVER	Description of the driver as returned by the

	SQLDrivers function. For example, Rdb or SQL Server.
UID	A user ID.
PWD	The password corresponding to the user ID, or an empty string if there is no password for the user ID (PWD=;).

Driver Manager Guidelines

The Driver Manager constructs a connection string to pass to the driver in the *szConnStrIn* argument of the driver's **SQLDriverConnect** function. Note that the Driver Manager does not modify the *szConnStrIn* argument passed to it by the application.

If the connection string specified by the application contains the **DSN** keyword or does not contain either the **DSN** or **DRIVER** keywords, the action of the Driver Manager is based on the value of the *fDriverCompletion* argument:

- SQL_DRIVER_PROMPT: The Driver Manager displays the Data Sources dialog box. It constructs a connection string from the data source name returned by the dialog box and any other keywords passed to it by the application. If the data source name returned by the dialog box is empty, the Driver Manager specifies the keyword-value pair DSN=Default.
- SQL_DRIVER_COMPLETE or SQL_DRIVER_COMPLETE_REQUIRED: If the connection string specified by the application includes the **DSN** keyword, the Driver Manager copies the connection string specified by the application. Otherwise, it takes the same actions as it does when *fDriverCompletion* is SQL_DRIVER_PROMPT.
- SQL_DRIVER_NOPROMPT: The Driver Manager copies the connection string specified by the application.

If the connection string specified by the application contains the **DRIVER** keyword, the Driver Manager copies the connection string specified by the application.

Using the connection string it has constructed, the Driver Manager determines which driver to use, loads that driver, and passes the connection string it has constructed to the driver; for more information about the interaction of the Driver Manager and the driver, see the "Comments" section in **SQLConnect**. If the connection string contains the **DSN** keyword or does not contain either the **DSN** or the **DRIVER** keyword, the Driver Manager determines which driver to use as follows:

1. If the connection string contains the **DSN** keyword, the Driver Manager retrieves the driver associated with the data source from the ODBC.INI file or registry.
2. If the connection string does not contain the **DSN** keyword or the data source is not found, the Driver Manager retrieves the driver associated with the Default data source from the ODBC.INI file or registry. However, the Driver Manager does not change the value of the **DSN** keyword in the connection string.
3. If the data source is not found and the Default data source is not found, the Driver Manager returns SQL_ERROR with SQLSTATE IM002 (Data source not found and no default driver specified).

Driver Guidelines

The driver checks if the connection string passed to it by the Driver Manager contains the **DSN** or **DRIVER** keyword. If the connection string contains the **DRIVER** keyword, the driver cannot retrieve information about the data source from the ODBC.INI file or registry. If the connection string contains the **DSN** keyword or does not contain either the **DSN** or the **DRIVER** keyword, the driver can retrieve information about the data source from the ODBC.INI file or registry as follows:

1. If the connection string contains the **DSN** keyword, the driver retrieves the information for the specified data source.
2. If the connection string does not contain the **DSN** keyword or the specified data source is not found, the driver retrieves the information for the Default data source.

The driver uses any information it retrieves from the ODBC.INI file or registry to augment the information passed to it in the connection string. If the information in the ODBC.INI file or registry duplicates information in the connection string, the driver uses the information in the connection string.

Based on the value of *fDriverCompletion*, the driver prompts the user for connection information, such as the user ID and password, and connects to the data source:

- SQL_DRIVER_PROMPT: The driver displays a dialog box, using the values from the connection string and ODBC.INI file or registry (if any) as initial values. When the user exits the dialog box, the driver connects to the data source. It also constructs a connection string from the value of the **DSN** or **DRIVER** keyword in *szConnStrIn* and the information returned from the dialog box. It places this connection string in the buffer referenced by *szConnStrOut*.
- SQL_DRIVER_COMPLETE or SQL_DRIVER_COMPLETE_REQUIRED: If the connection string contains enough information, and that information is correct, the driver connects to the data source and copies *szConnStrIn* to *szConnStrOut*. If any information is missing or incorrect, the driver takes the same actions as it does when *fDriverCompletion* is SQL_DRIVER_PROMPT, except that if *fDriverCompletion* is SQL_DRIVER_COMPLETE_REQUIRED, the driver disables the controls for any information not required to connect to the data source.
- SQL_DRIVER_NOPROMPT: If the connection string contains enough information, the driver connects to the data source and copies *szConnStrIn* to *szConnStrOut*. Otherwise, the driver returns SQL_ERROR for **SQLDriverConnect**.

On successful connection to the data source, the driver also sets *pcbConnStrOut* to the length of *szConnStrOut*.

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

For information about how the Driver Manager and the driver interact during the connection process, see **SQLConnect**.

If a driver supports **SQLDriverConnect**, the driver keyword section of the ODBC.INF file for the driver must contain the **ConnectFunctions** keyword with the second character set to "Y".

Connection Options

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, a waiting period for each login request begins after the user has dismissed each dialog box.

The driver opens the connection in SQL_MODE_READ_WRITE access mode by default. To set the access mode to SQL_MODE_READ_ONLY, the application must call **SQLSetConnectOption** with the SQL_ACCESS_MODE option prior to calling **SQLDriverConnect**.

If a default translation DLL is specified in the ODBC.INI file or registry for the data source, the driver loads it. A different translation DLL can be loaded by calling **SQLSetConnectOption** with the SQL_TRANSLATE_DLL option. A translation option can be specified by calling **SQLSetConnectOption** with the SQL_TRANSLATE_OPTION option.

Related Functions

SQLAllocConnect
SQLBrowseConnect (extension)
SQLConnect
SQLDisconnect
SQLDrivers (extension)
SQLFreeConnect
SQLSetConnectOption (extension)

SQLDrivers (Extension Level 2, ODBC 2.0)

Related Functions

SQLDrivers lists driver descriptions and driver attribute keywords. This function is implemented solely by the Driver Manager.

Syntax

RETCODE **SQLDrivers**(*henv*, *fDirection*, *szDriverDesc*, *cbDriverDescMax*, *pcbDriverDesc*, *szDriverAttributes*, *cbDrvrAttrMax*, *pcbDrvrAttr*)

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

Type	**Argument**	**Use**	**Description**
HENV	*henv*	Input	Environment handle.
UWORD	*fDirection*	Input	Determines whether the Driver Manager fetches the next driver description in the list (SQL_FETCH_NEXT) or whether the search starts from the beginning of the list (SQL_FETCH_FIRST).
UCHAR FAR *	*szDriverDesc*	Output	Pointer to storage for the driver description.
SWORD	*cbDriverDescMax*	Input	Maximum length of the *szDriverDesc* buffer.
SWORD FAR *	*pcbDriverDesc*	Output	Total number of bytes (excluding the null termination byte) available to return in *szDriverDesc*. If the number of bytes available to return is greater than or equal to *cbDriverDescMax*, the driver description in *szDriverDesc* is truncated to *cbDriverDescMax* - 1 bytes.
UCHAR FAR *	*szDriverAttributes*	Output	Pointer to storage for the list of driver attribute value pairs (see "Comments").
SWORD	*cbDrvrAttrMax*	Input	Maximum length of the *szDriverAttributes* buffer.
SWORD FAR *	*pcbDrvrAttr*	Output	Total number of bytes (excluding the null termination byte) available to return in *szDriverAttributes*. If the number of bytes available to return is greater than or equal to *cbDrvrAttrMax*, the list of attribute value pairs in *szDriverAttributes* is truncated to *cbDrvrAttrMax* - 1 bytes.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA_FOUND, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLDrivers** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLDrivers** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	**Error**	**Description**
01000	General warning	(DM) Driver Manager-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	(DM) The buffer *szDriverDesc* was not large enough to return the entire driver description, so the description was truncated. The argument *pcbDriverDesc* contains the length of the entire driver description. (Function returns SQL_SUCCESS_WITH_INFO.)
		(DM) The buffer *szDriverAttributes* was not large enough to return the entire list of attribute value pairs, so the list was truncated. The argument *pcbDrvrAttr* contains the length of the untruncated list of attribute value pairs. (Function returns SQL_SUCCESS_WITH_INFO.)
S1000	General error	(DM) An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	(DM) The Driver Manager was unable to allocate memory required to support execution or completion of the function.
S1090	Invalid string or buffer length	(DM) The value specified for argument *cbDriverDescMax* was less than 0.
		(DM) The value specified for argument *cbDrvrAttrMax* was less than 0 or equal to 1.
S1103	Direction option out of range	(DM) The value specified for the argument *fDirection* was not equal to SQL_FETCH_FIRST or SQL_FETCH_NEXT.

Comments

SQLDrivers returns the driver description in the *szDriverDesc* argument. It returns additional information about the driver in the *szDriverAttributes* argument as a list of keyword-value pairs. Each pair is terminated with a null byte, and the entire list is terminated with a null byte (that is, two null bytes mark the end of the list). For example, a dBASE driver might return the following list of attributes ("\0" represents a null byte):

```
FileUsage=1\0FileExtns=*.dbf\0\0
```

If *szDriverAttributes* is not large enough to hold the entire list, the list is truncated, **SQLDrivers** returns SQLSTATE 01004 (Data truncated), and the length of the list (excluding the final null termination byte) is returned in *pcbDrvrAttr*.

Driver attribute keywords are added from the ODBC.INF file when the driver is installed.

An application can call **SQLDrivers** multiple times to retrieve all driver descriptions. The Driver Manager retrieves this information from the ODBCINST.INI file or the registry. When there are no more driver descriptions, **SQLDrivers** returns SQL_NO_DATA_FOUND. If **SQLDrivers** is called with SQL_FETCH_NEXT immediately after it returns SQL_NO_DATA_FOUND, it returns the first driver description.

If SQL_FETCH_NEXT is passed to **SQLDrivers** the very first time it is called, **SQLDrivers** returns the first data source name.

Because **SQLDrivers** is implemented in the Driver Manager, it is supported for all drivers regardless of a particular driver's conformance level.

Related Functions

SQLBrowseConnect (extension)
SQLConnect
SQLDataSources (extension)
SQLDriverConnect (extension)

SQLError (Core, ODBC 1.0)

SQLError returns error or status information.

Syntax

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

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	SQLSTATE as null-terminated string.
SDWORD FAR *	*pfNativeError*	Output	Native error code (specific to the data source).
UCHAR FAR *	*szErrorMsg*	Output	Pointer to storage for the error message text.
SWORD	*cbErrorMsgMax*	Input	Maximum length of the *szErrorMsg* buffer. This must be less than or equal to SQL_MAX_MESSAGE_ LENGTH - 1.
SWORD FAR *	*pcbErrorMsg*	Output	Pointer to the total number of bytes (excluding the null termination byte) available to return in *szErrorMsg*. If the number of bytes available to return is greater than or equal to *cbErrorMsgMax*, the error message text in *szErrorMsg* is truncated to *cbErrorMsgMax* - 1 bytes.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA_FOUND, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

SQLError does not post error values for itself. **SQLError** returns SQL_NO_DATA_FOUND when it is unable to retrieve any error information, (in which case *szSqlState* equals 00000). 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

An application typically calls **SQLError** when a previous call to an ODBC function returns SQL_ERROR or SQL_SUCCESS_WITH_INFO. However, any ODBC function can post zero or more errors each time it is called, so an application can call **SQLError** after any ODBC function call.

SQLError retrieves an error from the data structure associated with the rightmost non-null 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 non-null handle, **SQLError** returns SQL_NO_DATA_FOUND. In this case, *szSqlState* equals 00000 (Success), *pfNativeError* is undefined, *pcbErrorMsg* equals 0, and *szErrorMsg* contains a single null termination byte (unless *cbErrorMsgMax* equals 0).

The Driver Manager stores error information in its *henv*, *hdbc*, and *hstmt* structures. Similarly, the driver stores error information in its *henv*, *hdbc*, and *hstmt* structures. When the application calls **SQLError**, the Driver Manager checks if there are any errors in its structure for the specified handle. If there are errors for the specified handle, it returns the first error; if there are no errors, it calls **SQLError** in the driver.

The Driver Manager can store up to 64 errors with an *henv* and its associated *hdbcs* and *hstmts*. When this limit is reached, the Driver Manager discards any subsequent errors posted on the Driver Manager's *henv*, *hdbcs*, or *hstmts*. The number of errors that a driver can store is driver-dependent.

An error is removed from the structure associated with a handle when **SQLError** is called for that handle and returns that error. All errors stored for a given handle are removed when that handle is used in a subsequent function call. For example, errors on an *hstmt* that were returned by **SQLExecDirect** are removed when **SQLExecDirect** or **SQLTables** is called with that *hstmt*. The errors stored on a given handle are not removed as the result of a call to a function using an associated handle of a different type. For example, errors on an *hdbc* that were returned by **SQLNativeSql** are not removed when **SQLError** or **SQLExecDirect** is called with an *hstmt* associated with that *hdbc*.

SQLExecDirect (Core, ODBC 1.0)

Code Example Related Functions

SQLExecDirect executes a preparable statement, using the current values of the parameter marker variables if any parameters exist in the statement. **SQLExecDirect** is the fastest way to submit an SQL statement for one-time execution.

Syntax

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

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 *szSqlStr*.

Returns

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

Diagnostics

When **SQLExecDirect** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLExecDirect** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The argument *szSqlStr* contained an SQL statement that contained a character or binary parameter or literal and the value exceeded the maximum length of the associated table column. The argument *szSqlStr* contained an SQL statement that contained a numeric parameter or literal and the fractional part of the value was truncated. The argument *szSqlStr* contained an SQL statement that contained a date or time parameter or literal and a timestamp value was truncated.
01006	Privilege not revoked	The argument *szSqlStr* contained a **REVOKE** statement and the user did not have the specified privilege. (Function returns SQL_SUCCESS_WITH_INFO.)
01S03	No rows updated or deleted	The argument *szSqlStr* contained a positioned update or delete statement and no rows were updated or deleted. (Function returns SQL_SUCCESS_WITH_INFO.)
01S04	More than one row updated or deleted	The argument *szSqlStr* contained a positioned update or delete statement and more than one row was updated or deleted. (Function returns

		SQL_SUCCESS_WITH_INFO.)
07001	Wrong number of parameters	The number of parameters specified in **SQLBindParameter** was less than the number of parameters in the SQL statement contained in the argument *szSqlStr*.
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
21S01	Insert value list does not match column list	The argument *szSqlStr* contained an **INSERT** statement and the number of values to be inserted did not match the degree of the derived table.
21S02	Degree of derived table does not match column list	The argument *szSqlStr* contained a **CREATE VIEW** statement and the number of names specified is not the same degree as the derived table defined by the query specification.
22003	Numeric value out of range	The argument *szSqlStr* contained an SQL statement which contained a numeric parameter or literal and the value caused the whole (as opposed to fractional) part of the number to be truncated when assigned to the associated table column.
22005	Error in assignment	The argument *szSqlStr* contained an SQL statement that contained a parameter or literal and the value was incompatible with the data type of the associated table column.
22008	Datetime field overflow	The argument *szSqlStr* contained an SQL statement that contained a date, time, or timestamp parameter or literal and the value was, respectively, an invalid date, time, or timestamp.
22012	Division by zero	The argument *szSqlStr* contained an SQL statement which contained an arithmetic expression which caused division by zero.
23000	Integrity constraint violation	The argument *szSqlStr* contained an SQL statement which contained a parameter or literal. The parameter value was NULL for a column defined as NOT NULL in the associated table column, a duplicate value was supplied for a column constrained to contain only unique values, or some other integrity constraint was violated.
24000	Invalid cursor state	(DM) A cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch** had been called. A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called. The argument *szSqlStr* contained a positioned update or delete statement

		and the cursor was positioned before the start of the result set or after the end of the result set.
34000	Invalid cursor name	The argument *szSqlStr* contained a positioned update or delete statement and the cursor referenced by the statement being executed was not open.
37000	Syntax error or access violation	The argument *szSqlStr* contained an SQL statement that was not preparable or contained a syntax error.
40001	Serialization failure	The transaction to which the SQL statement contained in the argument *szSqlStr* belonged was terminated to prevent deadlock.
42000	Syntax error or access violation	The user did not have permission to execute the SQL statement contained in the argument *szSqlStr*.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S0001	Base table or view already exists	The argument *szSqlStr* contained a **CREATE TABLE** or **CREATE VIEW** statement and the table name or view name specified already exists.
S0002	Table or view not found	The argument *szSqlStr* contained a **DROP TABLE** or a **DROP VIEW** statement and the specified table name or view name did not exist.
		The argument *szSqlStr* contained an **ALTER TABLE** statement and the specified table name did not exist.
		The argument *szSqlStr* contained a **CREATE VIEW** statement and a table name or view name defined by the query specification did not exist.
		The argument *szSqlStr* contained a **CREATE INDEX** statement and the specified table name did not exist.
		The argument *szSqlStr* contained a **GRANT** or **REVOKE** statement and the specified table name or view name did not exist.
		The argument *szSqlStr* contained a **SELECT** statement and a specified table name or view name did not exist.
		The argument *szSqlStr* contained a **DELETE**, **INSERT**, or **UPDATE** statement and the specified table name did not exist.
		The argument *szSqlStr* contained a **CREATE TABLE** statement and a table specified in a constraint (referencing a table other than the one being created) did not exist.
S0011	Index already exists	The argument *szSqlStr* contained a **CREATE INDEX** statement and the specified index name already existed.

S0012	Index not found	The argument *szSqlStr* contained a **DROP INDEX** statement and the specified index name did not exist.
S0021	Column already exists	The argument *szSqlStr* contained an **ALTER TABLE** statement and the column specified in the **ADD** clause is not unique or identifies an existing column in the base table.
S0022	Column not found	The argument *szSqlStr* contained a **CREATE INDEX** statement and one or more of the column names specified in the column list did not exist. The argument *szSqlStr* contained a **GRANT** or **REVOKE** statement and a specified column name did not exist. The argument *szSqlStr* contained a **SELECT**, **DELETE**, **INSERT**, or **UPDATE** statement and a specified column name did not exist. The argument *szSqlStr* contained a **CREATE TABLE** statement and a column specified in a constraint (referencing a table other than the one being created) did not exist.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1009	Invalid argument value	(DM) The argument *szSqlStr* was a null pointer.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or	(DM) The argument *cbSqlStr* was less

	buffer length	than or equal to 0, but not equal to SQL_NTS. A parameter value, set with **SQLBindParameter**, was a null pointer and the parameter length value was not 0, SQL_NULL_DATA, SQL_DATA_AT_EXEC, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET. A parameter value, set with **SQLBindParameter**, was not a null pointer and the parameter length value was less than 0, but was not SQL_NTS, SQL_NULL_DATA, SQL_DATA_AT_EXEC, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.
S1109	Invalid cursor position	The argument *szSqlStr* contained a positioned update or delete statement and the cursor was positioned (by **SQLSetPos** or **SQLExtendedFetch**) on a row for which the value in the *rgfRowStatus* array in **SQLExtendedFetch** was SQL_ROW_DELETED or SQL_ROW_ERROR.
S1C00	Driver not capable	The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

The application calls **SQLExecDirect** to send an SQL statement to the data source. The driver modifies the statement to use the form of SQL used by the data source, then submits it to the data source. In particular, the driver modifies the escape clauses used to define ODBC-specific SQL.

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 statement at the appropriate position.

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

If the data source is in manual-commit mode (requiring explicit transaction initiation), and a transaction has not already been initiated, the driver initiates a transaction before it sends the SQL statement.

If an application uses **SQLExecDirect** to submit a **COMMIT** or **ROLLBACK** statement, it will not be interoperable between DBMS products. To commit or roll back a transaction, call **SQLTransact**.

If **SQLExecDirect** encounters a data-at-execution parameter, it returns SQL_NEED_DATA. The application sends the data using **SQLParamData** and **SQLPutData**. See **SQLBindParameter**, **SQLParamOptions**, **SQLParamData**, and **SQLPutData** for more information.

Code Example

See **SQLBindCol**, **SQLExtendedFetch**, **SQLGetData**, and **SQLProcedures**.

Related Functions

- **SQLBindCol**
- **SQLCancel**
- **SQLExecute**
- **SQLExtendedFetch** (extension)
- **SQLFetch**
- **SQLGetCursorName**
- **SQLGetData** (extension)
- **SQLParamData** (extension)
- **SQLPrepare**
- **SQLPutData** (extension)
- **SQLSetCursorName**
- **SQLSetStmtOption** (extension)
- **SQLTransact**

SQLExecute (Core, ODBC 1.0)

Code Example Related Functions

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*)

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_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLExecute** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLExecute** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The prepared statement associated with the *hstmt* contained a character or binary parameter or literal and the value exceeded the maximum length of the associated table column. The prepared statement associated with the *hstmt* contained a numeric parameter or literal and the fractional part of the value was truncated. The prepared statement associated with the *hstmt* contained a date or time parameter or literal and a timestamp value was truncated.
01006	Privilege not revoked	The prepared statement associated with the *hstmt* was **REVOKE** and the user did not have the specified privilege. (Function returns SQL_SUCCESS_WITH_INFO.)
01S03	No rows updated or deleted	The prepared statement associated with the *hstmt* was a positioned update or delete statement and no rows were updated or deleted. (Function returns SQL_SUCCESS_WITH_INFO.)
01S04	More than one row updated or deleted	The prepared statement associated with the *hstmt* was a positioned update or delete statement and more than one row was updated or deleted. (Function returns SQL_SUCCESS_WITH_INFO.)
07001	Wrong number of parameters	The number of parameters specified in **SQLBindParameter** was less than the number of parameters in the prepared

		statement associated with the *hstmt*.
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
22003	Numeric value out of range	The prepared statement associated with the *hstmt* contained a numeric parameter and the parameter value caused the whole (as opposed to fractional) part of the number to be truncated when assigned to the associated table column.
22005	Error in assignment	The prepared statement associated with the *hstmt* contained a parameter and the value was incompatible with the data type of the associated table column.
22008	Datetime field overflow	The prepared statement associated with the *hstmt* contained a date, time, or timestamp parameter or literal and the value was, respectively, an invalid date, time, or timestamp.
22012	Division by zero	The prepared statement associated with the *hstmt* contained an arithmetic expression which caused division by zero.
23000	Integrity constraint violation	The prepared statement associated with the *hstmt* contained a parameter. The parameter value was NULL for a column defined as NOT NULL in the associated table column, a duplicate value was supplied for a column constrained to contain only unique values, or some other integrity constraint was violated.
24000	Invalid cursor state	(DM) A cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch** had been called. A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called. The prepared statement associated with the *hstmt* contained a positioned update or delete statement and the cursor was positioned before the start of the result set or after the end of the result set.
40001	Serialization failure	The transaction to which the prepared statement associated with the *hstmt* belonged was terminated to prevent deadlock.
42000	Syntax error or access violation	The user did not have permission to execute the prepared statement associated with the *hstmt*.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was

		defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns. (DM) The *hstmt* was not prepared. Either the *hstmt* was not in an executed state, or a cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch** had been called. The *hstmt* was not prepared. It was in an executed state and either no result set was associated with the *hstmt* or **SQLFetch** or **SQLExtendedFetch** had not been called.
S1090	Invalid string or buffer length	A parameter value, set with **SQLBindParameter**, was a null pointer and the parameter length value was not 0, SQL_NULL_DATA, SQL_DATA_AT_EXEC, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET. A parameter value, set with **SQLBindParameter**, was not a null pointer and the parameter length value was less than 0, but was not SQL_NTS, SQL_NULL_DATA, or SQL_DATA_AT_EXEC, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.
S1109	Invalid cursor position	The prepared statement was a positioned update or delete statement and the cursor was positioned (by **SQLSetPos** or **SQLExtendedFetch**) on a row for which the value in the *rgfRowStatus* array in

		SQLExtendedFetch was SQL_ROW_DELETED or SQL_ROW_ERROR.
S1C00	Driver not capable	The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

SQLExecute can return any SQLSTATE that can be returned by **SQLPrepare** based on when the data source evaluates the SQL statement associated with the *hstmt*.

Comments

SQLExecute executes a statement prepared by **SQLPrepare**. 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 must call **SQLFreeStmt** with the SQL_CLOSE parameter before reissuing the **SELECT** statement.

If the data source is in manual-commit mode (requiring explicit transaction initiation), and a transaction has not already been initiated, the driver initiates a transaction before it sends the SQL statement.

If an application uses **SQLPrepare** to prepare and **SQLExecute** to submit a **COMMIT** or **ROLLBACK** statement, it will not be interoperable between DBMS products. To commit or roll back a transaction, call **SQLTransact**.

If **SQLExecute** encounters a data-at-execution parameter, it returns SQL_NEED_DATA. The application sends the data using **SQLParamData** and **SQLPutData**. See **SQLBindParameter**, **SQLParamOptions**, **SQLParamData**, and **SQLPutData** for more information.

Code Example

See **SQLBindParameter**, **SQLParamOptions**, **SQLPutData**, and **SQLSetPos**.

Related Functions

SQLBindCol
SQLCancel
SQLExecDirect
SQLExtendedFetch (extension)
SQLFetch
SQLFreeStmt
SQLGetCursorName
SQLGetData (extension)
SQLParamData (extension)
SQLPrepare
SQLPutData (extension)
SQLSetCursorName
SQLSetStmtOption (extension)
SQLTransact

SQLExtendedFetch (Extension Level 2, ODBC 1.0)

Code Examples Related Functions

SQLExtendedFetch extends the functionality of **SQLFetch** in the following ways:

- It returns rowset data (one or more rows), in the form of an array, for each bound column.
- It scrolls through the result set according to the setting of a scroll-type argument.

SQLExtendedFetch works in conjunction with **SQLSetStmtOption**.

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. For more information, see the "Comments" section.
SDWORD	*irow*	Input	Number of the row to fetch. For more information, see the "Comments" section.
UDWORD FAR *	*pcrow*	Output	Number of rows actually fetched.
UWORD FAR *	*rgfRowStatus*	Output	An array of status values. For more information, see the "Comments" section.

Returns

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

Diagnostics

When **SQLExtendedFetch** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLExtendedFetch** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The data returned for one or more columns was truncated. String values are right truncated. For numeric values, the fractional part of number was truncated. (Function returns SQL_SUCCESS_WITH_INFO.)
01S01	Error in row	An error occurred while fetching one or more rows. (Function returns SQL_SUCCESS_WITH_INFO.)
07006	Restricted data type attribute violation	A data value could not be converted to the C data type specified by *fCType* in **SQLBindCol**.
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the

		function completed processing.
22003	Numeric value out of range	Returning the numeric value (as numeric or string) for one or more columns would have caused the whole (as opposed to fractional) part of the number to be truncated. Returning the binary value for one or more columns would have caused a loss of binary significance. For more information, see Appendix D, Data Types.
22012	Division by zero	A value from an arithmetic expression was returned which resulted in division by zero.
24000	Invalid cursor state	The *hstmt* was in an executed state but no result set was associated with the *hstmt*.
40001	Serialization failure	The transaction in which the fetch was executed was terminated to prevent deadlock.
IM001	Driver does not support this function	(DM) The driver associated with the *hdbc* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1002	Invalid column number	A column number specified in the binding for one or more columns was greater than the number of columns in the result set. Column 0 was bound with **SQLBindCol** and the SQL_USE_BOOKMARKS statement option was set to SQL_UB_OFF.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) The specified *hstmt* was not in an executed state. The function was called without first calling **SQLExecDirect**, **SQLExecute**, or a catalog function.. (DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this

		function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns. (DM) **SQLExtendedFetch** was called for an *hstmt* after **SQLFetch** was called and before **SQLFreeStmt** was called with the SQL_CLOSE option.
S1106	Fetch type out of range	(DM) The value specified for the argument *fFetchType* was invalid (see "Comments"). The value of the SQL_CURSOR_TYPE statement option was SQL_CURSOR_FORWARD_ONLY and the value of argument *fFetchType* was not SQL_FETCH_NEXT.
S1107	Row value out of range	The value specified with the SQL_CURSOR_TYPE statement option was SQL_CURSOR_KEYSET_DRIVEN, but the value specified with the SQL_KEYSET_SIZE statement option was greater than 0 and less than the value specified with the SQL_ROWSET_SIZE statement option.
S1111	Invalid bookmark value	The argument *fFetchType* was SQL_FETCH_BOOKMARK and the bookmark specified in the *irow* argument was not valid.
S1C00	Driver not capable	Driver or data source does not support the specified fetch type. The driver or data source does not support the conversion specified by the combination of the *fCType* in **SQLBindCol** and the SQL data type of the corresponding column. This error only applies when the SQL data type of the column was mapped to a driver-specific SQL data type. The argument *fFetchType* was SQL_FETCH_RESUME and the driver supports ODBC 2.0.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

SQLExtendedFetch returns one rowset of data to the application. An application cannot mix calls to **SQLExtendedFetch** and **SQLFetch** for the same cursor.

An application specifies the number of rows in the rowset by calling **SQLSetStmtOption** with the SQL_ROWSET_SIZE statement option.

Binding

If any columns in the result set have been bound with **SQLBindCol**, the driver converts the data for the bound columns as necessary and stores it in the locations bound to those columns. The result set can be bound in a column-wise (the default) or row-wise fashion.

Column-Wise Binding

To bind a result set in column-wise fashion, an application specifies SQL_BIND_BY_COLUMN for the SQL_BIND_TYPE statement option. (This is the default value.) For each column to be bound, the application:

1. Allocates an array of data storage buffers. The array has as many elements as there are rows in the rowset, plus an additional element if the application will search for key values or append new rows of data. Each buffer's size is the maximum size of the C data that can be returned for the column. For example, when the C data type is SQL_C_DEFAULT, each buffer's size is the column length. When the C data type is SQL_C_CHAR, each buffer's size is the display size of the data. For more information, see Converting Data from SQL to C Data Types and Precision, Scale, Length, and Display Size.
2. Allocates an array of SDWORDs to hold the number of bytes available to return for each row in the column. The array has as many elements as there are rows in the rowset.
3. Calls **SQLBindCol**:

- The *rgbValue* argument specifies the address of the data storage array.
- The *cbValueMax* argument specifies the size of each buffer in the data storage array.
- The *pcbValue* argument specifies the address of the number-of-bytes array.

When the application calls **SQLExtendedFetch**, the driver retrieves the data and the number of bytes available to return and stores them in the buffers allocated by the application:

- For each bound column, the driver stores the data in the *rgbValue* buffer bound to the column. It stores the first row of data at the start of the buffer and each subsequent row of data at an offset of *cbValueMax* bytes from the data for the previous row.
- For each bound column, the driver stores the number of bytes available to return in the *pcbValue* buffer bound to the column. This is the number of bytes available prior to calling **SQLExtendedFetch**. (If the number of bytes available to return cannot be determined in advance, the driver sets *pcbValue* to SQL_NO_TOTAL. If the data for the column is NULL, the driver sets *pcbValue* to SQL_NULL_DATA.) It stores the number of bytes available to return for the first row at the start of the buffer and the number of bytes available to return for each subsequent row at an offset of **sizeof(SDWORD)** from the value for the previous row.

Row-Wise Binding

To bind a result set in row-wise fashion, an application:

1. Declares a structure that can hold a single row of retrieved data and the associated data lengths. For each bound column, the structure contains one field for the data and one SDWORD field for the number of bytes available to return. The data field's size is the maximum size of the C data that can be returned for the column.
2. Calls **SQLSetStmtOption** with *fOption* set to SQL_BIND_TYPE and *vParam* set to the size of the structure.
3. Allocates an array of these structures. The array has as many elements as there are rows in the rowset, plus an additional element if the application will search for key values or append new rows of data.
4. Calls **SQLBindCol** for each column to be bound:

- The *rgbValue* argument specifies the address of the column's data field in the first array element.
- The *cbValueMax* argument specifies the size of the column's data field.
- The *pcbValue* argument specifies the address of the column's number-of-bytes field in the first array element.

When the application calls **SQLExtendedFetch**, the driver retrieves the data and the number of bytes available to return and stores them in the buffers allocated by the application:

- For each bound column, the driver stores the first row of data at the address specified by *rgbValue* for the column and each subsequent row of data at an offset of *vParam* bytes from the data for the previous row.

- For each bound column, the driver stores the number of bytes available to return for the first row at the address specified by *pcbValue* and the number of bytes available to return for each subsequent row at an offset of *vParam* bytes from the value for the previous row. This is the number of bytes available prior to calling **SQLExtendedFetch**. (If the number of bytes available to return cannot be determined in advance, the driver sets *pcbValue* to SQL_NO_TOTAL. If the data for the column is NULL, the driver sets *pcbValue* to SQL_NULL_DATA.)

Positioning the Cursor

The following operations require a cursor position:

- Positioned update and delete statements.
- Calls to **SQLGetData**.
- Calls to **SQLSetPos** with the SQL_DELETE, SQL_REFRESH, and SQL_UPDATE options.

An application can specify a cursor position when it calls **SQLSetPos**. Before it executes a positioned update or delete statement or calls **SQLGetData**, the application must position the cursor by calling **SQLExtendedFetch** to retrieve a rowset; the cursor points to the first row in the rowset. To position the cursor to a different row in the rowset, the application calls **SQLSetPos**.

The following table shows the rowset and return code returned when the application requests different rowsets.

Requested Rowset	Return Code	Cursor Position	Returned Rowset
Before start of result set	SQL_NO_DATA_FOUND	Before start of result set	None. The contents of the rowset buffers are undefined.
Overlaps start of result set	SQL_SUCCESS	Row 1 of rowset	First rowset in result set.
Within result set	SQL_SUCCESS	Row 1 of rowset	Requested rowset.
Overlaps end of result set	SQL_SUCCESS	Row 1 of rowset	For rows in the rowset that overlap the result set, data is returned. For rows in the rowset outside the result set, the contents of the *rgbValue* and *pcbValue* buffers are undefined and the *rgfRowStatus* array contains SQL_ROW_NOROW.
After end of result set	SQL_NO_DATA_FOUND	After end of result set	None. The contents of the rowset buffers are undefined.

For example, suppose a result set has 100 rows and the rowset size is 5. The following table shows the rowset and return code returned by **SQLExtendedFetch** for different values of *irow* when the fetch type is SQL_FETCH_RELATIVE:

Current Rowset	*irow*	Return Code	New Rowset
1 to 5	-5	SQL_NO_DATA_FOUND	None.
1 to 5	-3	SQL_SUCCESS	1 to 5
96 to 100	5	SQL_NO_DATA_FOUND	None.
96 to 100	3	SQL_SUCCESS	99 and 100. For rows 3, 4, and 5 in the rowset, the *rgfRowStatusArray* is set to SQL_ROW_NOROW.

Before **SQLExtendedFetch** is called the first time, the cursor is positioned before the start of the result set.

For the purpose of moving the cursor, deleted rows (that is, rows with an entry in the *rgfRowStatus* array of SQL_ROW_DELETED) are treated no differently than other rows. For example, calling **SQLExtendedFetch** with *fFetchType* set to SQL_FETCH_ABSOLUTE and *irow* set to 15 returns the rowset starting at row 15, even if the *rgfRowStatus* array for row 15 is SQL_ROW_DELETED.

Processing Errors

If an error occurs that pertains to the entire rowset, such as SQLSTATE S1T00 (Timeout expired), the driver returns SQL_ERROR and the appropriate SQLSTATE. The contents of the rowset buffers are undefined and the cursor position is unchanged.

If an error occurs that pertains to a single row, the driver:

- Sets the element in the *rgfRowStatus* array for the row to SQL_ROW_ERROR.
- Posts SQLSTATE 01S01 (Error in row) in the error queue.
- Posts zero or more additional SQLSTATEs for the error after SQLSTATE 01S01 (Error in row) in the error queue.

After it has processed the error or warning, the driver continues the operation for the remaining rows in the rowset and returns SQL_SUCCESS_WITH_INFO. Thus, for each error that pertains to a single row, the error queue contains SQLSTATE 01S01 (Error in row) followed by zero or more additional SQLSTATEs.

After it has processed the error, the driver fetches the remaining rows in the rowset and returns SQL_SUCCESS_WITH_INFO. Thus, for each row that returned an error, the error queue contains SQLSTATE 01S01 (Error in row) followed by zero or more additional SQLSTATEs.

If the rowset contains rows that have already been fetched, the driver is not required to return SQLSTATEs for errors that occurred when the rows were first fetched. It is, however, required to return SQLSTATE 01S01 (Error in row) for each row in which an error originally occurred and to return SQL_SUCCESS_WITH_INFO. For example, a static cursor that maintains a cache might cache row status information (so it can determine which rows contain errors) but might not cache the SQLSTATE associated with those errors.

Error rows do not affect relative cursor movements. For example, suppose the result set size is 100 and the rowset size is 10. If the current rowset is rows 11 through 20 and the element in the *rgfRowStatus* array for row 11 is SQL_ROW_ERROR, calling **SQLExtendedFetch** with the SQL_FETCH_NEXT fetch type still returns rows 21 through 30.

If the driver returns any warnings, such as SQLSTATE 01004 (Data truncated), it returns warnings that apply to the entire rowset or to unknown rows in the rowset before it returns error information applying to specific rows. It returns warnings for specific rows along with any other error information about those rows.

fFetchType Argument

The *fFetchType* argument specifies how to move through the result set. It is one of the following values:

SQL_FETCH_NEXT

SQL_FETCH_FIRST

SQL_FETCH_LAST

SQL_FETCH_PRIOR

SQL_FETCH_ABSOLUTE

SQL_FETCH_RELATIVE

SQL_FETCH_BOOKMARK

If the value of the SQL_CURSOR_TYPE statement option is SQL_CURSOR_FORWARD_ONLY, the *fFetchType* argument must be SQL_FETCH_NEXT.

Note In ODBC 1.0, **SQLExtendedFetch** supported the SQL_FETCH_RESUME fetch type. In ODBC 2.0, SQL_FETCH_RESUME is obsolete and the Driver Manager returns SQLSTATE S1C00 (Driver not capable) if an application specifies it for an ODBC 2.0 driver.

The SQL_FETCH_BOOKMARK fetch type was introduced in ODBC 2.0; the Driver Manager returns SQLSTATE S1106 (Fetch type out of range) if it is specified for an ODBC 1.0 driver.

Moving by Row Position

SQLExtendedFetch supports the following values of the *fFetchType* argument to move relative to the current rowset:

***fFetchType* Argument**	**Action**
SQL_FETCH_NEXT	The driver returns the next rowset. If the cursor is positioned before the start of the result set, this is equivalent to SQL_FETCH_FIRST.
SQL_FETCH_PRIOR	The driver returns the prior rowset. If the cursor is positioned after the end of the result set, this is equivalent to SQL_FETCH_LAST.
SQL_FETCH_RELATIVE	The driver returns the rowset *irow* rows from the start of the current rowset. If *irow* equals 0, the driver refreshes the current rowset. If the cursor is positioned before the start of the result set and *irow* is greater than 0 or if the cursor is positioned after the end of the result set and *irow* is less than 0, this is equivalent to SQL_FETCH_ABSOLUTE.

It supports the following values of the *fFetchType* argument to move to an absolute position in the result set:

***fFetchType* Argument**	**Action**
SQL_FETCH_FIRST	The driver returns the first rowset in the result set.
SQL_FETCH_LAST	The driver returns the last complete rowset in the result set.
SQL_FETCH_ABSOLUTE	If *irow* is greater than 0, the driver returns the rowset starting at row *irow*. If *irow* equals 0, the driver returns SQL_NO_DATA_FOUND and the cursor is positioned before the start of the result set. If *irow* is less than 0, the driver returns the rowset starting at row *n*+*irow*+1, where *n* is the number of rows in the result set. For example, if *irow* is -1, the driver returns the rowset starting at the last row in the result set. If the result set size is 10 and *irow* is -10, the driver returns the rowset starting at the first row in the result set.

Positioning to a Bookmark

When an application calls **SQLExtendedFetch** with the SQL_FETCH_BOOKMARK fetch type, the driver retrieves the rowset starting with the row specified by the bookmark in the *irow* argument.

To inform the driver that it will use bookmarks, the application calls **SQLSetStmtOption** with the SQL_USE_BOOKMARKS option before opening the cursor. To retrieve the bookmark for a row, the application either positions the cursor on the row and calls **SQLGetStmtOption** with the SQL_GET_BOOKMARK option, or retrieves the bookmark from column 0 of the result set. If the application retrieves a bookmark from column 0 of the result set, it must set *fCType* in **SQLBindCol** or **SQLGetData** to SQL_C_BOOKMARK. The application stores the bookmarks for those rows in each rowset to which it will return later.

Bookmarks are 32-bit binary values; if a bookmark requires more than 32 bits, such as when it is a key value, the driver maps the bookmarks requested by the application to 32-bit binary values. The 32-bit binary values are then returned to the application. Because this mapping may require considerable memory, applications should only bind column 0 of the result set if they will actually use bookmarks for most rows. Otherwise, applications should call **SQLGetStmtOption** with the SQL_GET_BOOKMARK statement option or call **SQLGetData** for column 0.

irow Argument

For the SQL_FETCH_ABSOLUTE fetch type, **SQLExtendedFetch** returns the rowset starting at the row number specified by the *irow* argument.

For the SQL_FETCH_RELATIVE fetch type, **SQLExtendedFetch** returns the rowset starting *irow* rows from the first row in the current rowset.

For the SQL_FETCH_BOOKMARK fetch type, the *irow* argument specifies the bookmark that marks the first row in the requested rowset.

The *irow* argument is ignored for the SQL_FETCH_NEXT, SQL_FETCH_PRIOR, SQL_FETCH_FIRST, and SQL_FETCH_LAST, fetch types.

rgfRowStatus Argument

In the *rgfRowStatus* array, **SQLExtendedFetch** returns any changes in status to each row since it was last retrieved from the data source. Rows may be unchanged (SQL_ROW_SUCCESS), updated (SQL_ROW_UPDATED), deleted (SQL_ROW_DELETED), added (SQL_ROW_ADDED), or were unretrievable due to an error (SQL_ROW_ERROR). For static cursors, this information is available for all rows. For keyset, mixed, and dynamic cursors, this information is only available for rows in the keyset; the driver does not save data outside the keyset and therefore cannot compare the newly retrieved data to anything.

Note Some drivers cannot detect changes to data. To determine whether a driver can detect changes to refetched rows, an application calls **SQLGetInfo** with the SQL_ROW_UPDATES option.

The number of elements must equal the number of rows in the rowset (as defined by the SQL_ROWSET_SIZE statement option). 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.

When an application calls **SQLSetPos** with *fOption* set to SQL_DELETE or SQL_UPDATE, **SQLSetPos** changes the *rgfRowStatus* array for the changed row to SQL_ROW_DELETED or SQL_ROW_UPDATED.

Note For keyset, mixed, and dynamic cursors, if a key value is updated, the row of data is considered to have been deleted and a new row added.

▪

Code Example

The following two examples show how an application could use column-wise or row-wise binding to bind storage locations to the same result set.

For more code examples, see **SQLSetPos**.

Column-Wise Binding

In the following example, an application declares storage locations for column-wise bound data and the returned numbers of bytes. Because column-wise binding is the default, there is no need, as in the row-wise binding example, to request column-wise binding with **SQLSetStmtOption**. However, the application does call **SQLSetStmtOption** to specify the number of rows in the rowset.

The application then executes a **SELECT** statement to return a result set of the employee names and birthdays, which is sorted by birthday. It calls **SQLBindCol** to bind the columns of data, passing the addresses of storage locations for both the data and the returned numbers of bytes. Finally, the application fetches the rowset data with **SQLExtendedFetch** and prints each employee's name and birthday.

```
#define ROWS 100
#define NAME_LEN 30
#define BDAY_LEN 11

UCHAR     szName[ROWS][NAME_LEN], szBirthday[ROWS][BDAY_LEN];
SWORD     sAge[ROWS];
SDWORD    cbName[ROWS], cbAge[ROWS], cbBirthday[ROWS];

UDWORD    crow, irow;
UWORD     rgfRowStatus[ROWS];

SQLSetStmtOption(hstmt, SQL_CONCURRENCY, SQL_CONCUR_READ_ONLY);
SQLSetStmtOption(hstmt, SQL_CURSOR_TYPE, SQL_CURSOR_KEYSET_DRIVEN);
SQLSetStmtOption(hstmt, SQL_ROWSET_SIZE, ROWS);
retcode = SQLExecDirect(hstmt, "SELECT NAME, AGE, BIRTHDAY FROM EMPLOYEE
ORDER BY 3, 2, 1", SQL_NTS);

if (retcode == SQL_SUCCESS) {
  SQLBindCol(hstmt, 1, SQL_C_CHAR, szName, NAME_LEN, cbName);
  SQLBindCol(hstmt, 2, SQL_C_SSHORT, sAge, 0, cbAge);
  SQLBindCol(hstmt, 3, SQL_C_CHAR, szBirthday, BDAY_LEN, cbBirthday);

  /* Fetch the rowset data and print each row. */
  /* On an error, display a message and exit.  */

  while (TRUE) {
    retcode = SQLExtendedFetch(hstmt, SQL_FETCH_NEXT, 1, &crow,
rgfRowStatus);
    if (retcode == SQL_ERROR || retcode == SQL_SUCCESS_WITH_INFO) {
      show_error();
    }
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){
      for (irow = 0; irow < crow; irow++) {
        if (rgfRowStatus[irow] != SQL_ROW_DELETED && rgfRowStatus[irow] !=
SQL_ROW_ERROR)
          fprintf(out, "%-*s  %-2d  %*s", NAME_LEN-1, szName[irow],
sAge[irow], BDAY_LEN-1, szBirthday[irow]);
      }
    } else {
      break;
    }
  }
}
```

Row-Wise Binding

In the following example, an application declares an array of structures to hold row-wise bound data and the returned numbers of bytes. Using **SQLSetStmtOption**, it requests row-wise binding and passes the size of the structure to the driver. The driver will use this size to find successive storage locations in the array of structures. Using **SQLSetStmtOption**, it specifies the size of the rowset.

The application then executes a **SELECT** statement to return a result set of the employee names and birthdays, which is sorted by birthday. It calls **SQLBindCol** to bind the columns of data, passing the addresses of storage locations for both the data and the returned numbers of bytes. Finally, the application fetches the rowset data with **SQLExtendedFetch** and prints each employee's name and birthday.

```
#define ROWS 100
#define NAME_LEN 30
#define BDAY_LEN 11

typedef struct {
  UCHAR     szName[NAME_LEN];
  SDWORD    cbName;
  SWORD     sAge;
  SDWORD    cbAge;
  UCHAR     szBirthday[BDAY_LEN];
  SDWORD    cbBirthday;
  } EmpTable;

EmpTable rget[ROWS];
UDWORD   crow, irow;
UWORD    rgfRowStatus[ROWS];

SQLSetStmtOption(hstmt, SQL_BIND_TYPE, sizeof(EmpTable));
SQLSetStmtOption(hstmt, SQL_CONCURRENCY, SQL_CONCUR_READ_ONLY);
SQLSetStmtOption(hstmt, SQL_CURSOR_TYPE, SQL_CURSOR_KEYSET_DRIVEN);
SQLSetStmtOption(hstmt, SQL_ROWSET_SIZE, ROWS);
retcode = SQLExecDirect(hstmt, "SELECT NAME, AGE, BIRTHDAY FROM EMPLOYEE
ORDER BY 3, 2, 1", SQL_NTS);

if (retcode == SQL_SUCCESS) {
  SQLBindCol(hstmt, 1, SQL_C_CHAR, rget[0].szName, NAME_LEN,
&rget[0].cbName);
  SQLBindCol(hstmt, 2, SQL_C_SSHORT, &rget[0].sAge, 0, &rget[0].cbAge);
  SQLBindCol(hstmt, 3, SQL_C_CHAR, rget[0].szBirthday, BDAY_LEN,
&rget[0].cbBirthday);
  /* Fetch the rowset data and print each row. */
  /* On an error, display a message and exit.  */

  while (TRUE) {
    retcode = SQLExtendedFetch(hstmt, SQL_FETCH_NEXT, 1, &crow,
rgfRowStatus);
    if (retcode == SQL_ERROR || retcode == SQL_SUCCESS_WITH_INFO) {
      show_error();
    }
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){
      for (irow = 0; irow < crow; irow++) {
        if (rgfRowStatus[irow] != SQL_ROW_DELETED && rgfRowStatus[irow] !=
SQL_ROW_ERROR)
          fprintf(out, "%-*s  %-2d  %*s", NAME_LEN-1, rget[irow].szName,
rget[irow].sAge, BDAY_LEN-1, rget[irow].szBirthday);
      }
    } else {
      break;
    }
  }
}
```

Related Functions

SQLBindCol
SQLCancel
SQLDescribeCol
SQLExecDirect
SQLExecute
SQLNumResultCols
SQLSetPos (extension)
SQLSetStmtOption (extension)

SQLFetch (Core, ODBC 1.0)

Code Example Related Functions

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*)

The **SQLFetch** function accepts the following argument.

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.

Returns

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

Diagnostics

When **SQLFetch** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLFetch** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The data returned for one or more columns was truncated. String values are right truncated. For numeric values, the fractional part of number was truncated. (Function returns SQL_SUCCESS_WITH_INFO.)
07006	Restricted data type attribute violation	The data value could not be converted to the data type specified by *fCType* in **SQLBindCol**.
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
22003	Numeric value out of range	Returning the numeric value (as numeric or string) for one or more columns would have caused the whole (as opposed to fractional) part of the number to be truncated. Returning the binary value for one or more columns would have caused a loss of binary significance. For more information, see Converting Data from SQL to C Data Types.
22012	Division by zero	A value from an arithmetic expression was returned which resulted in division by zero.
24000	Invalid cursor state	The *hstmt* was in an executed state but no result set was associated with the *hstmt*.
40001	Serialization failure	The transaction in which the fetch was executed was terminated to prevent

		deadlock.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1002	Invalid column number	A column number specified in the binding for one or more columns was greater than the number of columns in the result set. A column number specified in the binding for a column was 0; **SQLFetch** cannot be used to retrieve bookmarks.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) The specified *hstmt* was not in an executed state. The function was called without first calling **SQLExecDirect**, **SQLExecute**, or a catalog function.. (DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns. (DM) **SQLExtendedFetch** was called for an *hstmt* after **SQLFetch** was called and before **SQLFreeStmt** was called with the SQL_CLOSE option.
S1C00	Driver not capable	The driver or data source does not support the conversion specified by the combination of the *fCType* in **SQLBindCol** and the SQL data type of the corresponding column. This error only applies when the SQL data type of the column was mapped to a driver-specific SQL data type.
S1T00	Timeout expired	The timeout period expired before the

data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

SQLFetch positions the cursor on the next row of the result set. Before **SQLFetch** is called the first time, the cursor is positioned before the start of the result set. When the cursor is positioned on the last row of the result set, **SQLFetch** returns SQL_NO_DATA_FOUND and the cursor is positioned after the end of the result set. An application cannot mix calls to **SQLExtendedFetch** and **SQLFetch** for the same cursor.

If the application called **SQLBindCol** to bind columns, **SQLFetch** stores data into the locations specified by the calls to **SQLBindCol**. If the application does not call **SQLBindCol** to bind any columns, **SQLFetch** doesn't return any data; it just moves the cursor to the next row. An application can call **SQLGetData** to retrieve data that is not bound to a storage location.

The driver manages cursors during the fetch operation and places each value of a bound column into the associated storage. 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 number of bytes available to return for the column. This is the number of bytes available prior to calling **SQLFetch**. If the number of bytes available to return cannot be determined in advance, the driver sets *pcbValue* to SQL_NO_TOTAL. (If SQL_MAX_LENGTH has been specified with **SQLSetStmtOption** and the number of bytes available to return is greater than SQL_MAX_LENGTH, *pcbValue* contains SQL_MAX_LENGTH.)

Note The SQL_MAX_LENGTH statement option is intended to reduce network traffic and may not be supported by all drivers. To guarantee that data is truncated, an application should allocate a buffer of the desired size and specify this size in the *cbValueMax* argument.

- 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 *cbValueMax*, then truncation occurred.
- If the data value for the column is NULL, the driver stores SQL_NULL_DATA in *pcbValue*.

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

For information about conversions allowed by **SQLBindCol** and **SQLGetData**, see Converting Data from SQL to C Data Types.

Code Example

See **SQLBindCol**, **SQLColumns**, **SQLGetData**, and **SQLProcedures**.

Related Functions

SQLBindCol
SQLCancel
SQLDescribeCol
SQLExecDirect
SQLExecute
SQLExtendedFetch (extension)
SQLFreeStmt
SQLGetData (extension)
SQLNumResultCols
SQLPrepare

SQLForeignKeys (Extension Level 2, ODBC 1.0)

Code Example Related Functions

SQLForeignKeys can return:

- A list of foreign keys in the specified table (columns in the specified table that refer to primary keys in other tables).
- A list of foreign keys in other tables that refer to the primary key in the specified table.

The driver returns each 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*)

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

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.
UCHAR FAR *	*szPkTableQualifier*	Input	Primary key table qualifier. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	*cbPkTableQualifier*	Input	Length of *szPkTableQualifier*.
UCHAR FAR *	*szPkTableOwner*	Input	Primary key owner name. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.
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. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	*cbFkTableQualifier*	Input	Length of *szFkTableQualifier*.
UCHAR FAR *	*szFkTableOwner*	Input	Foreign key owner name. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have

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

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLForeignKeys** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLForeignKeys** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	**Error**	**Description**
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
24000	Invalid cursor state	(DM) A cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch** had been called. A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1009	Invalid argument value	(DM) The arguments *szPkTableName* and *szFkTableName* were both null pointers.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this

		function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the name length arguments was less than 0, but not equal to SQL_NTS. The value of one of the name length arguments exceeded the maximum length value for the corresponding qualifier or name (see "Comments").
S1C00	Driver not capable	A table qualifier was specified and the driver or data source does not support qualifiers. A table owner was specified and the driver or data source does not support owners. The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

If *szPkTableName* contains a table name, **SQLForeignKeys** returns a result set containing the primary key of the specified table and all of the foreign keys that refer to it.

If *szFkTableName* contains a table name, **SQLForeignKeys** returns a result set containing all of the foreign keys in the specified table and the primary keys (in other tables) to which they refer.

If both *szPkTableName* and *szFkTableName* contain table names, **SQLForeignKeys** returns the foreign keys in the table specified in *szFkTableName* that refer to the primary key of the table specified in *szPkTableName*. This should be one key at most.

SQLForeignKeys returns results as a standard result set. If the foreign keys associated with a primary key are requested, the result set is ordered by FKTABLE_QUALIFIER, FKTABLE_OWNER, FKTABLE_NAME, and KEY_SEQ. If the primary keys associated with a foreign key are requested, the result set is ordered by PKTABLE_QUALIFIER, PKTABLE_OWNER, PKTABLE_NAME, and KEY_SEQ. The following table lists the columns in the result set.

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the TABLE_QUALIFIER, TABLE_OWNER, TABLE_NAME, and COLUMN_NAME columns, an application can call **SQLGetInfo** with the SQL_MAX_QUALIFIER_NAME_LEN, SQL_MAX_OWNER_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN options.

Column Name	**Data Type**	**Comments**
PKTABLE_QUALIFIER	Varchar(128)	Primary key table qualifier identifier; NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an

		empty string ("") for those tables that do not have qualifiers.
PKTABLE_OWNER	Varchar(128)	Primary key table owner identifier; NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.
PKTABLE_NAME	Varchar(128) not NULL	Primary key table identifier.
PKCOLUMN_NAME	Varchar(128) not NULL	Primary key column identifier.
FKTABLE_QUALIFIER	Varchar(128)	Foreign key table qualifier identifier; NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have qualifiers.
FKTABLE_OWNER	Varchar(128)	Foreign key table owner identifier; NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.
FKTABLE_NAME	Varchar(128) not NULL	Foreign key table identifier.
FKCOLUMN_NAME	Varchar(128) not NULL	Foreign key column identifier.
KEY_SEQ	Smallint not NULL	Column sequence number in key (starting with 1).
UPDATE_RULE	Smallint	Action to be applied to the foreign key when the SQL operation is **UPDATE**: SQL_CASCADE SQL_RESTRICT SQL_SET_NULL NULL if not applicable to the data source.
DELETE_RULE	Smallint	Action to be applied to the foreign key when the SQL operation is **DELETE**: SQL_CASCADE SQL_RESTRICT SQL_SET_NULL NULL if not applicable to the data source.
FK_NAME	Varchar(128)	Foreign key identifier. NULL if not applicable to the data source.
PK_NAME	Varchar(128)	Primary key identifier. NULL if not applicable to the data source.

Note The FK_NAME and PK_NAME columns were added in ODBC 2.0. ODBC 1.0 drivers may return different, driver-specific columns with the same column numbers.

▪

Code Example

This example uses four tables:

SALES_ORDER	**SALES_LINE**	**CUSTOMER**	**EMPLOYEE**
SALES_ID	SALES_ID	CUSTOMER_ID	EMPLOYEE_ID
CUSTOMER_ID	LINE_NUMBER	CUST_NAME	NAME
EMPLOYEE_ID	PART_ID	ADDRESS	AGE
TOTAL_PRICE	QUANTITY	PHONE	BIRTHDAY
	PRICE		

In the SALES_ORDER table, CUSTOMER_ID identifies the customer to whom the sale has been made. It is a foreign key that refers to CUSTOMER_ID in the CUSTOMER table. EMPLOYEE_ID identifies the employee who made the sale. It is a foreign key that refers to EMPLOYEE_ID in the EMPLOYEE table.

In the SALES_LINE table, SALES_ID identifies the sales order with which the line item is associated. It is a foreign key that refers to SALES_ID in the SALES_ORDER table.

This example calls **SQLPrimaryKeys** to get the primary key of the SALES_ORDER table. The result set will have one row and the significant columns are:

TABLE_NAME	**COLUMN_NAME**	**KEY_SEQ**
SALES_ORDER	SALES_ID	1

Next, the example calls **SQLForeignKeys** to get the foreign keys in other tables that reference the primary key of the SALES_ORDER table. The result set will have one row and the significant columns are:

PKTABLE_ NAME	**PKCOLUMN_ NAME**	**FKTABLE_ NAME**	**FKCOLUMN_ NAME**	**KEY_SEQ**
SALES_ORDER	SALES_ID	SALES_LINE	SALES_ID	1

Finally, the example calls **SQLForeignKeys** to get the foreign keys in the SALES_ORDER table the refer to the primary keys of other tables. The result set will have two rows and the significant columns are:

PKTABLE_ NAME	**PKCOLUMN_ NAME**	**FKTABLE_ NAME**	**FKCOLUMN_ NAME**	**KEY_SEQ**
CUSTOMER	CUSTOMER_ID	SALES_ORDER	CUSTOMER_ID	1
EMPLOYEE	EMPLOYEE_ID	SALES_ORDER	EMPLOYEE_ID	1

```
#define TAB_LEN SQL_MAX_TABLE_NAME_LEN + 1
#define COL_LEN SQL_MAX_COLUMN_NAME_LEN + 1

LPSTR szTable;              /* Table to display       */

UCHAR szPkTable[TAB_LEN];   /* Primary key table name */
UCHAR szFkTable[TAB_LEN];   /* Foreign key table name */
UCHAR szPkCol[COL_LEN];     /* Primary key column     */
UCHAR szFkCol[COL_LEN];     /* Foreign key column     */

HSTMT   hstmt;
SDWORD    cbPkTable, cbPkCol, cbFkTable, cbFkCol, cbKeySeq;
SWORD   iKeySeq;
RETCODE   retcode;
```

```
/* Bind the columns that describe the primary and foreign keys.  */
/* Ignore the table owner, name, and qualifier for this example. */

SQLBindCol(hstmt, 3, SQL_C_CHAR, szPkTable, TAB_LEN, &cbPkTable);
SQLBindCol(hstmt, 4, SQL_C_CHAR, szPkCol, COL_LEN, &cbPkCol);
SQLBindCol(hstmt, 5, SQL_C_SSHORT, &iKeySeq, TAB_LEN, &cbKeySeq);
SQLBindCol(hstmt, 7, SQL_C_CHAR, szFkTable, TAB_LEN, &cbFkTable);
SQLBindCol(hstmt, 8, SQL_C_CHAR, szFkCol, COL_LEN, &cbFkCol);

strcpy(szTable, "SALES_ORDER");

/* Get the names of the columns in the primary key.              */

retcode = SQLPrimaryKeys(hstmt, NULL, 0, NULL, 0, szTable, SQL_NTS);

while ((retcode == SQL_SUCCESS) || (retcode == SQL SUCCESS_WITH_INFO)) {

  /* Fetch and display the result set. This will be a list of the */
  /* columns in the primary key of the SALES_ORDER table.         */

  retcode = SQLFetch(hstmt);
  if (retcode == SQL_SUCCESS || retcode != SQL_SUCCESS_WITH_INFO)
    fprintf(out, "Column: %s    Key Seq: %hd \n", szPkCol, iKeySeq);
}

/* Close the cursor (the hstmt is still allocated).                */

SQLFreeStmt(hstmt, SQL_CLOSE);

/* Get all the foreign keys that refer to SALES_ORDER primary key. */

retcode = SQLForeignKeys(hstmt, NULL, 0, NULL, 0, szTable, SQL_NTS, NULL, 0,
NULL, 0,  NULL, 0);

while ((retcode == SQL_SUCCESS) || (retcode == SQL_SUCCESS_WITH_INFO)) {

  /* Fetch and display the result set. This will be all of the    */
  /* foreign keys in other tables that refer to the SALES_ORDER   */
  /* primary key.                                                  */

  retcode = SQLFetch(hstmt);
  if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
    fprintf(out, "%-s ( %-s ) <-- %-s ( %-s )\n", szPkTable, szPkCol,
szFkTable, szFkCol);
}
```

```
/* Close the cursor (the hstmt is still allocated).                 */

SQLFreeStmt(hstmt, SQL_CLOSE);

/* Get all the foreign keys in the SALES_ORDER table.               */

retcode = SQLForeignKeys(hstmt, NULL, 0, NULL, 0, NULL, 0, NULL, 0, NULL, 0,
szTable, SQL_NTS);

while ((retcode == SQL_SUCCESS) || (retcode == SQL_SUCCESS_WITH_INFO)) {

  /* Fetch and display the result set. This will be all of the    */
  /* primary keys in other tables that are referred to by foreign */
  /* keys in the SALES_ORDER table.                               */

  retcode = SQLFetch(hstmt);
  if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
    fprintf(out, "%-s ( %-s )--> %-s ( %-s )\n", szFkTable, szFkCol,
            szPkTable, szPkCol);
}

/* Free the hstmt. */

SQLFreeStmt(hstmt, SQL_DROP);
```

Related Functions

- **SQLBindCol**
- **SQLCancel**
- **SQLExtendedFetch** (extension)
- **SQLFetch**
- **SQLPrimaryKeys** (extension)
- **SQLStatistics** (extension)

SQLFreeConnect (Core, ODBC 1.0)

Code Example Related Functions

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

Syntax

RETCODE **SQLFreeConnect**(*hdbc*)

The **SQLFreeConnect** function accepts the following argument.

Type	Argument	Use	Description
HDBC	*hdbc*	Input	Connection handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLFreeConnect** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLFreeConnect** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1010	Function sequence error	(DM) The function was called prior to calling **SQLDisconnect** for the *hdbc*.

Comments

Prior to calling **SQLFreeConnect,** an application must call **SQLDisconnect** for the *hdbc.* Otherwise, **SQLFreeConnect** returns SQL_ERROR and the *hdbc* remains valid. Note that **SQLDisconnect** automatically drops any *hstmts* open on the *hdbc*.

Code Example

See **SQLBrowseConnect** and **SQLConnect**.

Related Functions

SQLAllocConnect
SQLConnect
SQLDisconnect
SQLDriverConnect (extension)
SQLFreeEnv
SQLFreeStmt

SQLFreeEnv (Core, ODBC 1.0)

Code Example Related Functions

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

Syntax

RETCODE **SQLFreeEnv**(*henv*)

The **SQLFreeEnv** function accepts the following argument.

Type	**Argument**	**Use**	**Description**
HENV	*henv*	Input	Environment handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLFreeEnv** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLFreeEnv** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	**Error**	**Description**
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1010	Function sequence error	(DM) There was at least one *hdbc* in an allocated or connected state. Call **SQLDisconnect** and **SQLFreeConnect** for each *hdbc* before calling **SQLFreeEnv**.

Comments

Prior to calling **SQLFreeEnv**, an application must call **SQLFreeConnect** for any *hdbc* allocated under the *henv*. Otherwise, **SQLFreeEnv** returns SQL_ERROR and the *henv* and any active *hdbc* remains valid.

When the Driver Manager processes the **SQLFreeEnv** function, it checks the **TraceAutoStop** keyword in the [ODBC] section of the ODBC.INI file or the ODBC subkey of the registry. If it is set to 1, the Driver Manager disables tracing for all applications and sets the **Trace** keyword in the [ODBC] section of the ODBC.INI file or the ODBC subkey of the registry to 0.

Code Example

See **SQLBrowseConnect** and **SQLConnect**.

Related Functions

SQLAllocEnv

SQLFreeConnect

SQLFreeStmt (Core, ODBC 1.0)

Code Example Related Functions

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*)

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 with the same or different parameter values. If no cursor is open, this option has no effect for the application. 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*. The *hstmt* must be reallocated to be reused. SQL_UNBIND: Release all column buffers bound by **SQLBindCol** for the given *hstmt*. SQL_RESET_PARAMS: Release all parameter buffers set by **SQLBindParameter** for the given *hstmt*.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLFreeStmt** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLFreeStmt** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was

		no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1010	Function sequence error	(DM) An asynchronously executing function was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1092	Option type out of range	(DM) The value specified for the argument *fOption* was not: SQL_CLOSE SQL_DROP SQL_UNBIND SQL_RESET_PARAMS

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.

Code Example

See **SQLBrowseConnect** and **SQLConnect**.

Related Functions

<u>SQLAllocStmt</u>
<u>SQLCancel</u>
<u>SQLSetCursorName</u>

SQLGetConnectOption (Extension Level 1, ODBC 1.0)

Related Functions

SQLGetConnectOption returns the current setting of a connection option.

Syntax

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

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

Type	Argument	Use	Description
HDBC	*hdbc*	Input	Connection handle.
UWORD	*fOption*	Input	Option to retrieve.
PTR	*pvParam*	Output	Value associated with *fOption*. Depending on the value of *fOption*, a 32-bit integer value or a pointer to a null-terminated character string will be returned in *pvParam*.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA_FOUND, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLGetConnectOption** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLGetConnectOption** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection not open	(DM) An *fOption* value was specified that required an open connection.
IM001	Driver does not support this function	(DM) The driver corresponding to the *hdbc* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1010	Function sequence error	(DM) **SQLBrowseConnect** was called for the *hdbc* and returned SQL_NEED_DATA. This function was called before **SQLBrowseConnect** returned SQL_SUCCESS_WITH_INFO or SQL_SUCCESS.
S1092	Option type out of range	(DM) The value specified for the argument *fOption* was in the block of numbers reserved for ODBC connection and statement options, but was not valid for the version of ODBC supported by the driver.

S1C00	Driver not capable	The value specified for the argument *fOption* was a valid ODBC connection option for the version of ODBC supported by the driver, but was not supported by the driver. The value specified for the argument *fOption* was in the block of numbers reserved for driver-specific connection and statement options, but was not supported by the driver.

Comments

For a list of options, see **SQLSetConnectOption**. Note that if *fOption* specifies an option that returns a string, *pvParam* must be a pointer to storage for the string. The maximum length of the string will be SQL_MAX_OPTION_STRING_LENGTH bytes (excluding the null termination byte).

Depending on the option, an application does not need to establish a connection prior to calling **SQLGetConnectOption**. However, if **SQLGetConnectOption** is called and the specified option does not have a default and has not been set by a prior call to **SQLSetConnectOption**, **SQLGetConnnectOption** will return SQL_NO_DATA_FOUND.

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.

Related Functions

SQLGetStmtOption (extension)
SQLSetConnectOption (extension)
SQLSetStmtOption (extension)

SQLGetCursorName (Core, ODBC 1.0)

Related Functions

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

Syntax

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

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

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.
UCHAR FAR *	*szCursor*	Output	Pointer to storage for the cursor name.
SWORD	*cbCursorMax*	Input	Length of *szCursor*.
SWORD FAR *	*pcbCursor*	Output	Total number of bytes (excluding the null termination byte) available to return in *szCursor*. If the number of bytes available to return is greater than or equal to *cbCursorMax*, the cursor name in *szCursor* is truncated to *cbCursorMax* - 1 bytes.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLGetCursorName** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLGetCursorName** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The buffer *szCursor* was not large enough to return the entire cursor name, so the cursor name was truncated. The argument *pcbCursor* contains the length of the untruncated cursor name. (Function returns SQL_SUCCESS_WITH_INFO.)
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1010	Function sequence error	(DM) An asynchronously executing function was called for the *hstmt* and was

		still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1015	No cursor name available	(DM) There was no open cursor on the *hstmt* and no cursor name had been set with **SQLSetCursorName**.
S1090	Invalid string or buffer length	(DM) The value specified in the argument *cbCursorMax* was less than 0.

Comments

The only ODBC SQL statements that use a cursor name are positioned update and delete (for example, **UPDATE** *table-name* ...**WHERE CURRENT OF** *cursor-name*). If the application does not call **SQLSetCursorName** to define a cursor name, on execution of a **SELECT** statement the driver generates a name that begins with the letters SQL_CUR and does not exceed 18 characters in length.

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

A cursor name that is set either explicitly or implicitly remains set until the *hstmt* with which it is associated is dropped, using **SQLFreeStmt** with the SQL_DROP option.

Related Functions

SQLExecDirect
SQLExecute
SQLPrepare
SQLSetCursorName
SQLSetScrollOptions (extension)

SQLGetData (Extension Level 1, ODBC 1.0)

Code Example Related Functions

SQLGetData returns result data for a single unbound column in the current row. The application must call **SQLFetch**, or **SQLExtendedFetch** and (optionally) **SQLSetPos** to position the cursor on a row of data before it calls **SQLGetData**. It is possible to use **SQLBindCol** for some columns and use **SQLGetData** for others within the same row. This function can be used to retrieve character or binary data values in parts from a column with a character, binary, or data source-specific data type (for example, data from SQL_LONGVARBINARY or SQL_LONGVARCHAR columns).

Syntax

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

The **SQLGetData** 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. A column number of 0 is used to retrieve a bookmark for the row; bookmarks are not supported by ODBC 1.0 drivers or **SQLFetch**.
SWORD	*fCType*	Input	The C data type of the result data. This must be one of the following values: SQL_C_BINARY SQL_C_BIT SQL_C_BOOKMARK SQL_C_CHAR SQL_C_DATE SQL_C_DEFAULT SQL_C_DOUBLE SQL_C_FLOAT SQL_C_SLONG SQL_C_SSHORT SQL_C_STINYINT SQL_C_TIME SQL_C_TIMESTAMP SQL_C_ULONG SQL_C_USHORT SQL_C_UTINYINT SQL_C_DEFAULT specifies that data be converted to its default C data type. **Note** Drivers must also support the following values of *fCType* from ODBC 1.0. Applications must use these values, rather than the ODBC 2.0 values, when calling an ODBC 1.0 driver: SQL_C_LONG SQL_C_SHORT SQL_C_TINYINT For information about how

			data is converted, see Converting Data from SQL to C Data Types.
PTR	*rgbValue*	Output	Pointer to storage for the data.
SDWORD	*cbValueMax*	Input	Maximum length of the *rgbValue* buffer. For character data, *rgbValue* must also include space for the null-termination byte. For character and binary C data, *cbValueMax* determines the amount of data that can be received in a single call to **SQLGetData**. For all other types of C data, *cbValueMax* is ignored; the driver assumes that the size of *rgbValue* is the size of the C data type specified with *fCType* and returns the entire data value. For more information about length, see Precision, Scale, Length, and Display Size.
SDWORD FAR *	*pcbValue*	Output	SQL_NULL_DATA, the total number of bytes (excluding the null termination byte for character data) available to return in *rgbValue* prior to the current call to **SQLGetData**, or SQL_NO_TOTAL if the number of available bytes cannot be determined. For character data, if *pcbValue* is SQL_NO_TOTAL or is greater than or equal to *cbValueMax*, the data in *rgbValue* is truncated to *cbValueMax* - 1 bytes and is null-terminated by the driver. For binary data, if *pcbValue* is SQL_NO_TOTAL or is greater than *cbValueMax*, the data in *rgbValue* is truncated to *cbValueMax* bytes. For all other data types, the value of *cbValueMax* is ignored and the driver assumes the size of *rgbValue* is the size of the C data type specified with *fCType*.

Returns

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

Diagnostics

When **SQLGetData** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated

SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLGetData** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	All of the data for the specified column, *icol*, could not be retrieved in a single call to the function. The argument *pcbValue* contains the length of the data remaining in the specified column prior to the current call to **SQLGetData**. (Function returns SQL_SUCCESS_WITH_INFO.) For more information on using multiple calls to **SQLGetData** for a single column, see "Comments."
07006	Restricted data type attribute violation	The data value cannot be converted to the C data type specified by the argument *fCType*.
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
22003	Numeric value out of range	Returning the numeric value (as numeric or string) for the column would have caused the whole (as opposed to fractional) part of the number to be truncated. Returning the binary value for the column would have caused a loss of binary significance. For more information, see Appendix D, Data Types.
22005	Error in assignment	The data for the column was incompatible with the data type into which it was to be converted. For more information, see Appendix D, Data Types.
22008	Datetime field overflow	The data for the column was not a valid date, time, or timestamp value. For more information, see Appendix D, Data Types.
24000	Invalid cursor state	(DM) The *hstmt* was in an executed state but no result set was associated with the *hstmt*. (DM) A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called. A cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch** had been called, but the cursor was positioned before the start of the result set or after the end of the result set.

IM001	Driver does not support this function	(DM) The driver corresponding to the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1002	Invalid column number	The value specified for the argument *icol* was 0 and the driver was an ODBC 1.0 driver. The value specified for the argument *icol* was 0 and **SQLFetch** was used to fetch the data. The value specified for the argument *icol* was 0 and the SQL_USE_BOOKMARKS statement option was set to SQL_UB_OFF. The specified column was greater than the number of result columns. The specified column was bound through a call to **SQLBindCol**. This description does not apply to drivers that return the SQL_GD_BOUND bitmask for the SQL_GETDATA_EXTENSIONS option in **SQLGetInfo**. The specified column was at or before the last bound column specified through **SQLBindCol**. This description does not apply to drivers that return the SQL_GD_ANY_COLUMN bitmask for the SQL_GETDATA_EXTENSIONS option in **SQLGetInfo**. The application has already called **SQLGetData** for the current row. The column specified in the current call was before the column specified in the preceding call. This description does not apply to drivers that return the SQL_GD_ANY_ORDER bitmask for the SQL_GETDATA_EXTENSIONS option in **SQLGetInfo**.
S1003	Program type out of range	(DM) The argument *fCType* was not a valid data type or SQL_C_DEFAULT. The argument *icol* was 0 and the argument *fCType* was not SQL_C_BOOKMARK.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*.

		The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1009	Invalid argument value	(DM) The argument *rgbValue* was a null pointer.
S1010	Function sequence error	(DM) The specified *hstmt* was not in an executed state. The function was called without first calling **SQLExecDirect**, **SQLExecute**, or a catalog function. (DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value specified for argument *cbValueMax* was less than 0.
S1109	Invalid cursor position	The cursor was positioned (by **SQLSetPos** or **SQLExtendedFetch**) on a row for which the value in the *rgfRowStatus* array in **SQLExtendedFetch** was SQL_ROW_DELETED or SQL_ROW_ERROR.
S1C00	Driver not capable	The driver or data source does not support use of **SQLGetData** with multiple rows in **SQLExtendedFetch**. This description does not apply to drivers that return the SQL_GD_BLOCK bitmask for the SQL_GETDATA_EXTENSIONS option in **SQLGetInfo**. The driver or data source does not support the conversion specified by the combination of the *fCType* argument and the SQL data type of the corresponding column. This error only applies when the SQL data type of the column was mapped to a driver-specific SQL data type. The argument *icol* was 0 and the driver does not support bookmarks. The driver only supports ODBC 1.0 and the argument *fCType* was one of the following: SQL_C_STINYINT SQL_C_UTINYINT SQL_C_SSHORT SQL_C_USHORT SQL_C_SLONG SQL_C_ULONG
S1T00	Timeout expired	The timeout period expired before the

data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

With each call, the driver sets *pcbValue* to the number of bytes that were available in the result column prior to the current call to **SQLGetData**. (If SQL_MAX_LENGTH has been set with **SQLSetStmtOption**, and the total number of bytes available on the first call is greater than SQL_MAX_LENGTH, the available number of bytes is set to SQL_MAX_LENGTH. Note that the SQL_MAX_LENGTH statement option is intended to reduce network traffic and may not be supported by all drivers. To guarantee that data is truncated, an application should allocate a buffer of the desired size and specify this size in the *cbValueMax* argument.) If the total number of bytes in the result column cannot be determined in advance, the driver sets *pcbValue* to SQL_NO_TOTAL. If the data value for the column is NULL, the driver stores SQL_NULL_DATA in *pcbValue*.

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 Converting Data from SQL to C Data Types.

If more than one call to **SQLGetData** is required to retrieve data from a single column with a character, binary, or data source-specific data type, the driver returns SQL_SUCCESS_WITH_INFO. A subsequent call to **SQLError** returns SQLSTATE 01004 (Data truncated). The application can then use the same column number to retrieve subsequent parts of the data until **SQLGetData** returns SQL_SUCCESS, indicating that all data for the column has been retrieved. **SQLGetData** will return SQL_NO_DATA_FOUND when it is called for a column after all of the data has been retrieved and before data is retrieved for a subsequent column. The application can ignore excess data by proceeding to the next result column.

Note An application can use **SQLGetData** to retrieve data from a column in parts only when retrieving character C data from a column with a character,binary, or data source-specific data type or when retrieving binary C data from a column with a character, binary, or data source-specific data type. If **SQLGetData** is called more than one time in a row for a column under any other conditions, it returns SQL_NO_DATA_FOUND for all calls after the first.

For maximum interoperability, applications should call **SQLGetData** only for unbound columns with numbers greater than the number of the last bound column. Within a single row of data, the column number in each call to **SQLGetData** should be greater than or equal to the column number in the previous call (that is, data should be retrieved in increasing order of column number). As extended functionality, drivers can return data through **SQLGetData** from bound columns, from columns before the last bound column, or from columns in any order. To determine whether a driver supports these extensions, an application calls **SQLGetInfo** with the SQL_GETDATA_EXTENSIONS option.

Furthermore, applications that use **SQLExtendedFetch** to retrieve data should call **SQLGetData** only when the rowset size is 1. As extended functionality, drivers can return data through **SQLGetData** when the rowset size is greater than 1. The application calls **SQLSetPos** to position the cursor on a row and calls **SQLGetData** to retrieve data from an unbound column. To determine whether a driver supports this extension, an application calls **SQLGetInfo** with the SQL_GETDATA_EXTENSIONS option.

▪

Code Example

In the following example, an application executes a **SELECT** statement to return a result set of the employee names, ages, and birthdays sorted by birthday, age, and name. For each row of data, it calls **SQLFetch** to position the cursor to the next row. It calls **SQLGetData** to retrieve the fetched data; the storage locations for the data and the returned number of bytes are specified in the call to **SQLGetData**. Finally, it prints each employee's name, age, and birthday.

```
#define NAME_LEN 30
#define BDAY_LEN 11

UCHAR     szName[NAME_LEN], szBirthday[BDAY_LEN];
SWORD     sAge;
SDWORD    cbName, cbAge, cbBirthday;

retcode = SQLExecDirect(hstmt, "SELECT NAME, AGE, BIRTHDAY FROM EMPLOYEE
ORDER BY 3, 2, 1", SQL_NTS);

if (retcode == SQL_SUCCESS) {
  while (TRUE) {
    retcode = SQLFetch(hstmt);
    if (retcode == SQL_ERROR || retcode == SQL_SUCCESS_WITH_INFO) {
      show_error();
    }
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){

      /* Get data for columns 1, 2, and 3 */
      /* Print the row of data            */

      SQLGetData(hstmt, 1, SQL_C_CHAR, szName, NAME_LEN, &cbName);
      SQLGetData(hstmt, 2, SQL_C_SSHORT, &sAge, 0, &cbAge);
      SQLGetData(hstmt, 3, SQL_C_CHAR, szBirthday, BDAY_LEN, &cbBirthday);

      fprintf(out, "%-*s %-2d %*s", NAME_LEN-1, szName, sAge, BDAY_LEN-1,
szBirthday);
    } else {
      break;
    }
  }
}
```

Related Functions

SQLBindCol
SQLCancel
SQLExecDirect
SQLExecute
SQLExtendedFetch (extension)
SQLFetch
SQLPutData (extension)

SQLGetFunctions (Extension Level 1, ODBC 1.0)

Code Example Related Functions

SQLGetFunctions returns information about whether a driver supports a specific ODBC function. This function is implemented in the Driver Manager; it can also be implemented in drivers. If a driver implements **SQLGetFunctions**, the Driver Manager calls the function in the driver. Otherwise, it executes the function itself.

Syntax

RETCODE **SQLGetFunctions**(*hdbc*, *fFunction*, *pfExists*)

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

Type	Argument	Use	Description
HDBC	*hdbc*	Input	Connection handle.
UWORD	*fFunction*	Input	SQL_API_ALL_FUNCTIONS or a **#define** value that identifies the ODBC function of interest. For a list of **#define** values that identify ODBC functions, see the tables in "Comments."
UWORD FAR *	*pfExists*	Output	If *fFunction* is SQL_API_ALL_FUNCTIONS, *pfExists* points to a UWORD array with 100 elements. The array is indexed by **#define** values used by *fFunction* to identify each ODBC function; some elements of the array are unused and reserved for future use. An element is TRUE if it identifies an ODBC function supported by the driver. It is FALSE if it identifies an ODBC function not supported by the driver or does not identify an ODBC function. **Note** The *fFunction* value SQL_API_ALL_FUNCTIONS was added in ODBC 2.0. If *fFunction* identifies a single ODBC function, *pfExists* points to single UWORD. *pfExists* is TRUE if the specified function is supported by the driver; otherwise, it is FALSE.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLGetFunctions** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLGetFunctions** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1010	Function sequence error	(DM) **SQLGetFunctions** was called before **SQLConnect**, **SQLBrowseConnect**, or **SQLDriverConnect**. (DM) **SQLBrowseConnect** was called for the *hdbc* and returned SQL_NEED_DATA. This function was called before **SQLBrowseConnect** returned SQL_SUCCESS_WITH_INFO or SQL_SUCCESS.
S1095	Function type out of range	(DM) An invalid *fFunction* value was specified.

Comments

SQLGetFunctions always returns that **SQLGetFunctions**, **SQLDataSources**, and **SQLDrivers** are supported. It does this because these functions are implemented in the Driver Manager.

The following table lists valid values for *fFunction* for ODBC core functions.

SQL_API_SQLALLOCCONNECT	SQL_API_SQLFETCH
SQL_API_SQLALLOCENV	SQL_API_SQLFREECONNECT
SQL_API_SQLALLOCSTMT	SQL_API_SQLFREEENV
SQL_API_SQLBINDCOL	SQL_API_SQLFREESTMT
SQL_API_SQLCANCEL	SQL_API_SQLGETCURSORNAME
SQL_API_SQLCOLATTRIBUTES	SQL_API_SQLNUMRESULTCOLS
SQL_API_SQLCONNECT	SQL_API_SQLPREPARE
SQL_API_SQLDESCRIBECOL	SQL_API_SQLROWCOUNT
SQL_API_SQLDISCONNECT	SQL_API_SQLSETCURSORNAME
SQL_API_SQLERROR	SQL_API_SQLSETPARAM
SQL_API_SQLEXECDIRECT	SQL_API_SQLTRANSACT
SQL_API_SQLEXECUTE	

Note For ODBC 1.0 drivers, **SQLGetFunctions** returns TRUE in *pfExists* if *fFunction* is SQL_API_SQLBINDPARAMETER or SQL_API_SQLSETPARAM and the driver supports **SQLSetParam**. For ODBC 2.0 drivers, **SQLGetFunctions** returns TRUE in *pfExists* if *fFunction* is SQL_API_SQLSETPARAM or SQL_API_SQLBINDPARAMETER and the driver supports **SQLBindParameter**.

The following table lists valid values for *fFunction* for ODBC extension level 1 functions.

SQL_API_SQLBINDPARAMETER	SQL_API_SQLGETTYPEINFO
SQL_API_SQLCOLUMNS	SQL_API_SQLPARAMDATA
SQL_API_SQLDRIVERCONNECT	SQL_API_SQLPUTDATA

SQL_API_SQLGETCONNECT-OPTION	SQL_API_SQLSETCONNECTOPTION
SQL_API_SQLGETDATA	SQL_API_SQLSETSTMTOPTION
SQL_API_SQLGETFUNCTIONS	SQL_API_SQLSPECIALCOLUMNS
SQL_API_SQLGETINFO	SQL_API_SQLSTATISTICS
SQL_API_SQLGETSTMTOPTION	SQL_API_SQLTABLES

The following table lists valid values for *fFunction* for ODBC extension level 2 functions.

SQL_API_SQLBROWSECONNECT	SQL_API_SQLNUMPARAMS
SQL_API_SQLCOLUMN-PRIVILEGES	SQL_API_SQLPARAMOPTIONS
SQL_API_SQLDATASOURCES	SQL_API_SQLPRIMARYKEYS
SQL_API_SQLDESCRIBEPARAM	SQL_API_SQLPROCEDURECOLUMNS
SQL_API_SQLDRIVERS	SQL_API_SQLPROCEDURES
SQL_API_SQLEXTENDEDFETCH	SQL_API_SQLSETPOS
SQL_API_SQLFOREIGNKEYS	SQL_API_SQLSETSCROLLOPTIONS
SQL_API_SQLMORERESULTS	SQL_API_SQLTABLEPRIVILEGES
SQL_API_SQLNATIVESQL	

▪

Code Example

The following two examples show how an application uses **SQLGetFunctions** to determine if a driver supports **SQLTables**, **SQLColumns**, and **SQLStatistics**. If the driver does not support these functions, the application disconnects from the driver. The first example calls **SQLGetFunctions** once for each function.

```
UWORD TablesExists, ColumnsExists, StatisticsExists;

SQLGetFunctions(hdbc, SQL_API_SQLTABLES, &TablesExists);
SQLGetFunctions(hdbc, SQL_API_SQLCOLUMNS, &ColumnsExists);
SQLGetFunctions(hdbc, SQL_API_SQLSTATISTICS, &StatisticsExists);

if (TablesExists && ColumnsExists && StatisticsExists) {

  /* Continue with application */

}

SQLDisconnect(hdbc);
```

The second example calls **SQLGetFunctions** a single time and passes it an array in which **SQLGetFunctions** returns information about all ODBC functions.

```
UWORD fExists[100];

SQLGetFunctions(hdbc, SQL_API_ALL_FUNCTIONS, fExists);

if (fExists[SQL_API_SQLTABLES] && fExists[SQL_API_SQLCOLUMNS] &&
fExists[SQL_API_SQLSTATISTICS]) {

  /* Continue with application */

}

SQLDisconnect(hdbc);
```

Related Functions

SQLGetConnectOption (extension)
SQLGetInfo (extension)
SQLGetStmtOption (extension)

SQLGetInfo (Extension Level 1, ODBC 1.0)

Code Example Related Functions

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").
PTR	*rgbInfoValue*	Output	Pointer to storage for the information. Depending on the *fInfoType* requested, the information returned will be one of the following: a null-terminated character string, a 16-bit integer value, a 32-bit flag, or a 32-bit binary value.
SWORD	*cbInfoValueMax*	Input	Maximum length of the *rgbInfoValue* buffer.
SWORD FAR *	*pcbInfoValue*	Output	The total number of bytes (excluding the null termination byte for character data) available to return in *rgbInfoValue*. For character data, if the number of bytes available to return is greater than or equal to *cbInfoValueMax*, the information in *rgbInfoValue* is truncated to *cbInfoValueMax* - 1 bytes and is null-terminated by the driver. For all other types of data, the value of *cbValueMax* is ignored and the driver assumes the size of *rgbValue* is 32 bits.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLGetInfo** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLGetInfo** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The buffer *rgbInfoValue* was not large

		enough to return all of the requested information, so the information was truncated. The argument *pcbInfoValue* contains the length of the requested information in its untruncated form. (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection not open	(DM) The type of information requested in *fInfoType* requires an open connection. Of the information types reserved by ODBC, only SQL_ODBC_VER can be returned without an open connection.
22003	Numeric value out of range	Returning the requested information would have caused a loss of numeric or binary significance.
IM001	Driver does not support this function	(DM) The driver corresponding to the *hdbc* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1009	Invalid argument value	(DM) The *fInfoType* was SQL_DRIVER_HSTMT, and the value pointed to by *rgbInfoValue* was not a valid statement handle.
S1090	Invalid string or buffer length	(DM) The value specified for argument *cbInfoValueMax* was less than 0.
S1096	Information type out of range	(DM) The value specified for the argument *fOption* was in the block of numbers reserved for ODBC information types, but was not valid for the version of ODBC supported by the driver.
S1C00	Driver not capable	The value specified for the argument *fOption* was in the range of numbers reserved for driver-specific information types, but was not supported by the driver.
S1T00	Timeout expired	The timeout period expired before the data source returned the requested information. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

The currently defined information types are shown below; it is expected that more will be defined to take advantage of different data sources. Information types from 0 to 999 are reserved by ODBC; driver developers must reserve values greater than or equal to SQL_INFO_DRIVER_START for driver-specific use.

The format of the information returned in *rgbInfoValue* depends on the *fInfoType* requested. **SQLGetInfo** will return information in one of five different formats:

- A null-terminated character string,
- A 16-bit integer value,
- A 32-bit bitmask,
- A 32-bit integer value,
- Or a 32-bit binary value.

The format of each of the following information types is noted in the type's description. The application must cast the value returned in *rgbInfoValue* accordingly. For an example of how an application could retrieve data from a 32-bit bitmask, see "Code Example."

A driver must return a value for each of the information types defined in the following tables. If an information type does not apply to the driver or data source, then the driver returns one of the following values:

Format of *rgbInfoValue*	**Returned value**
Character string ("Y" or "N")	"N"
Character string (not "Y" or "N")	Empty string
16-bit integer	0
32-bit bitmask or 32-bit binary value	0L

For example, if a data source does not support procedures, **SQLGetInfo** returns the following values for the values of *fInfoType* that are related to procedures:

fInfoType	**Returned value**
SQL_PROCEDURES	"N"
SQL_ACCESSIBLE_PROCEDURES	"N"
SQL_MAX_PROCEDURE_NAME_LEN	0
SQL_PROCEDURE_TERM	Empty string

SQLGetInfo returns SQLSTATE S1096 (Invalid argument value) for values of *fInfoType* that are in the range of information types reserved for use by ODBC but are not defined by the version of ODBC supported by the driver. To determine what version of ODBC a driver conforms to, an application calls **SQLGetInfo** with the SQL_DRIVER_ODBC_VER information type. **SQLGetInfo** returns SQLSTATE S1C00 (Driver not capable) for values of *fInfoType* that are in the range of information types reserved for driver-specific use but are not supported by the driver.

Note Application developers should be aware that ODBC 1.0 drivers might return SQL_ERROR and SQLSTATE S1C00 (Driver not capable) for values of *fInfoType* that were defined in ODBC 1.0 but do not apply to the driver or the data source.

Information Types

This section lists the information types supported by **SQLGetInfo**. Information types are grouped categorically and listed alphabetically.

Driver Information

The following values of *fInfoType* return information about the ODBC driver, such as the number of active statements, the data source name, and the API conformance levels.

SQL_ACTIVE_CONNECTIONS

SQL_ACTIVE_STATEMENTS

SQL_DATA_SOURCE_NAME

SQL_DRIVER_HDBC

SQL_DRIVER_HENV

SQL_DRIVER_HLIB

SQL_DRIVER_HSTMT

SQL_DRIVER_NAME

SQL_DRIVER_ODBC_VER

SQL_DRIVER_VER

SQL_FETCH_DIRECTION

SQL_FILE_USAGE

SQL_GETDATA_EXTENSIONS

SQL_LOCK_TYPES

SQL_ODBC_API_CONFORMANCE

SQL_ODBC_SAG_CLI_CONFORMANCE

SQL_ODBC_VER

SQL_POS_OPERATIONS

SQL_ROW_UPDATES

SQL_SEARCH_PATTERN_ESCAPE

SQL_SERVER_NAME

DBMS Product Information

The following values of *fInfoType* return information about the DBMS product, such as the DBMS name and version.

SQL_DATABASE_NAME

SQL_DBMS_NAME

SQL_DBMS_VER

Data Source Information

The following values of *fInfoType* return information about the data source, such as cursor characteristics and transaction capabilities.

SQL_ACCESSIBLE_PROCEDURES

SQL_ACCESSIBLE_TABLES

SQL_BOOKMARK_PERSISTENCE

SQL_CONCAT_NULL_BEHAVIOR

SQL_CURSOR_COMMIT_BEHAVIOR

SQL_CURSOR_ROLLBACK_BEHAVIOR

SQL_DATA_SOURCE_READ_ONLY

SQL_DEFAULT_TXN_ISOLATION

SQL_MULT_RESULT_SETS

SQL_MULTIPLE_ACTIVE_TXN

SQL_NEED_LONG_DATA_LEN

SQL_NULL_COLLATION

SQL_OWNER_TERM

SQL_PROCEDURE_TERM

SQL_QUALIFIER_TERM

SQL_SCROLL_CONCURRENCY

SQL_SCROLL_OPTIONS

SQL_STATIC_SENSITIVITY

SQL_TABLE_TERM

SQL_TXN_CAPABLE

SQL_TXN_ISOLATION_OPTION

SQL_USER_NAME

Supported SQL

The following values of *fInfoType* return information about the SQL statements supported by the data source. These information types do not exhaustively describe the entire ODBC SQL grammar. Instead, they describe those parts of the grammar for which data sources commonly offer different levels of support.

Applications should determine the general level of supported grammar from the SQL_ODBC_SQL_CONFORMANCE information type and use the other information types to determine variations from the stated conformance level.

SQL_ALTER_TABLE

SQL_COLUMN_ALIAS

SQL_CORRELATION_NAME

SQL_EXPRESSIONS_IN_ORDERBY

SQL_GROUP_BY

SQL_IDENTIFIER_CASE

SQL_IDENTIFIER_QUOTE_CHAR

SQL_KEYWORDS

SQL_LIKE_ESCAPE_CLAUSE

SQL_NON_NULLABLE_COLUMNS

SQL_ODBC_SQL_CONFORMANCE

SQL_ODBC_SQL_OPT_IEF

SQL_ORDER_BY_COLUMNS_IN_SELECT

SQL_OUTER_JOINS

SQL_OWNER_USAGE

SQL_POSITIONED_STATEMENTS

SQL_PROCEDURES

SQL_QUALIFIER_LOCATION

SQL_QUALIFIER_NAME_SEPARATOR

SQL_QUALIFIER_USAGE

SQL_QUOTED_IDENTIFIER_CASE

SQL_SPECIAL_CHARACTERS

SQL_SUBQUERIES

SQL_UNION

SQL Limits

The following values of *fInfoType* return information about the limits applied to identifiers and clauses in SQL statements, such as the maximum lengths of identifiers and the maximum number of columns in a select list. Limitations may be imposed by either the driver or the data source.

SQL_MAX_BINARY_LITERAL_LEN

SQL_MAX_CHAR_LITERAL_LEN

SQL_MAX_COLUMN_NAME_LEN

SQL_MAX_COLUMNS_IN_GROUP_BY

SQL_MAX_COLUMNS_IN_ORDER_BY

SQL_MAX_COLUMNS_IN_INDEX

SQL_MAX_COLUMNS_IN_SELECT

SQL_MAX_COLUMNS_IN_TABLE

SQL_MAX_CURSOR_NAME_LEN

SQL_MAX_INDEX_SIZE

SQL_MAX_OWNER_NAME_LEN

SQL_MAX_PROCEDURE_NAME_LEN

SQL_MAX_QUALIFIER_NAME_LEN

SQL_MAX_ROW_SIZE

SQL_MAX_ROW_SIZE_INCLUDES_LONG

SQL_MAX_STATEMENT_LEN

SQL_MAX_TABLE_NAME_LEN

SQL_MAX_TABLES_IN_SELECT

SQL_MAX_USER_NAME_LEN

Scalar Function Information

The following values of *fInfoType* return information about the scalar functions supported by the data source and the driver.

SQL_CONVERT_FUNCTIONS

SQL_NUMERIC_FUNCTIONS

SQL_STRING_FUNCTIONS

SQL_SYSTEM_FUNCTIONS

SQL_TIMEDATE_ADD_INTERVALS

SQL_TIMEDATE_DIFF_INTERVALS

SQL_TIMEDATE_FUNCTIONS

Conversion Information

The following values of *fInfoType* return a list of the SQL data types to which the data source can convert the specified SQL data type with the **CONVERT** scalar function.

SQL_CONVERT_BIGINT

SQL_CONVERT_BINARY

SQL_CONVERT_BIT

SQL_CONVERT_CHAR

SQL_CONVERT_DATE

SQL_CONVERT_DECIMAL

SQL_CONVERT_DOUBLE

SQL_CONVERT_FLOAT

SQL_CONVERT_INTEGER

SQL_CONVERT_LONGVARBINARY

SQL_CONVERT_LONGVARCHAR

SQL_CONVERT_NUMERIC

SQL_CONVERT_REAL

SQL_CONVERT_SMALLINT

SQL_CONVERT_TIME

SQL_CONVERT_TIMESTAMP

SQL_CONVERT_TINYINT

SQL_CONVERT_VARBINARY

SQL_CONVERT_VARCHAR

Information Type Descriptions

The following table alphabetically lists each information type, the version of ODBC in which it was introduced, and its description.

InfoType	Returns
SQL_ACCESSIBLE_PROCEDURES (ODBC 1.0)	A character string: "Y" if the user can execute all procedures returned by **SQLProcedures**, "N" if there may be procedures returned that the user cannot execute.
SQL_ACCESSIBLE_TABLES (ODBC 1.0)	A character string: "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_CONNECTIONS (ODBC 1.0)	A 16-bit integer value specifying the maximum number of active *hdbcs* that the driver can support. This value can reflect a limitation imposed by either the driver or the data source. If there is no specified limit or the limit is unknown, this value is set to zero.
SQL_ACTIVE_STATEMENTS (ODBC 1.0)	A 16-bit integer value specifying 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 or the limit is unknown, this value is set to zero.
SQL_ALTER_TABLE (ODBC 2.0)	A 32-bit bitmask enumerating the clauses in the **ALTER TABLE** statement supported by the data source. The following bitmask is used to determine which clauses are supported: SQL_AT_ADD_COLUMN SQL_AT_DROP_COLUMN
SQL_BOOKMARK_PERSISTENCE (ODBC 2.0)	A 32-bit bitmask enumerating the operations through which bookmarks persist. The following bitmasks are used in conjunction with the flag to determine through which options bookmarks persist: SQL_BP_CLOSE = Bookmarks are valid after an application calls **SQLFreeStmt** with the SQL_CLOSE option to close the cursor associated with an *hstmt*. SQL_BP_DELETE = The bookmark for a row is valid after that row has been deleted. SQL_BP_DROP = Bookmarks are valid after an *hstmt* an application calls **SQLFreeStmt** with the SQL_DROP option to drop an *hstmt*. SQL_BP_SCROLL = Bookmarks are valid after any scrolling operation (call to **SQLExtendedFetch**). Because all bookmarks must remain valid after **SQLExtendedFetch** is called, this value can be used by applications to determine whether bookmarks are supported. SQL_BP_TRANSACTION = Bookmarks are valid after an application commits or rolls back a transaction. SQL_BP_UPDATE = The bookmark for a row is valid after any column in that row has been updated, including key columns. SQL_BP_OTHER_HSTMT = A bookmark associated with one *hstmt* can be used with another *hstmt*.

SQL_COLUMN_ALIAS (ODBC 2.0)	A character string: "Y" if the data source supports column aliases; otherwise, "N".
SQL_CONCAT_NULL_BEHAVIOR (ODBC 1.0)	A 16-bit integer value indicating how the data source handles the concatenation of NULL valued character data type columns with non-NULL valued character data type columns: SQL_CB_NULL = Result is NULL valued. SQL_CB_NON_NULL = Result is concatenation of non-NULL valued column or columns.
SQL_CONVERT_BIGINT SQL_CONVERT_BINARY SQL_CONVERT_BIT SQL_CONVERT_CHAR SQL_CONVERT_DATE SQL_CONVERT_DECIMAL SQL_CONVERT_DOUBLE SQL_CONVERT_FLOAT SQL_CONVERT_INTEGER SQL_CONVERT_LONGVARBINAR Y SQL_CONVERT_LONGVARCHAR SQL_CONVERT_NUMERIC SQL_CONVERT_REAL SQL_CONVERT_SMALLINT SQL_CONVERT_TIME SQL_CONVERT_TIMESTAMP SQL_CONVERT_TINYINT SQL_CONVERT_VARBINARY SQL_CONVERT_VARCHAR (ODBC 1.0)	A 32-bit bitmask. The bitmask indicates the conversions supported by the data source with the CONVERT scalar function for data of the type named in the *fInfoType*. If the bitmask equals zero, the data source does not support any conversions for data of the named type, including conversion to the same data type. For example, to find out if a data source supports the conversion of SQL_INTEGER data to the SQL_BIGINT data type, an application calls **SQLGetInfo** with the *fInfoType* of SQL_CONVERT_INTEGER. The application ANDs the returned bitmask with SQL_CVT_BIGINT. If the resulting value is nonzero, the conversion is supported. The following bitmasks are used to determine which conversions are supported: 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_LONGVARBINARY 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
SQL_CONVERT_FUNCTIONS (ODBC 1.0)	A 32-bit bitmask enumerating the scalar conversion functions supported by the driver and associated data source. The following bitmask is used to determine which conversion functions are supported: SQL_FN_CVT_CONVERT
SQL_CORRELATION_NAME (ODBC 1.0)	A 16-bit integer indicating if table correlation names are supported: SQL_CN_NONE = Correlation names are not supported. SQL_CN_DIFFERENT = Correlation names are supported, but must differ from the names of the tables they represent. SQL_CN_ANY = Correlation names are supported and can be any valid user-defined name.
SQL_CURSOR_COMMIT _BEHAVIOR (ODBC 1.0)	A 16-bit integer value indicating how a **COMMIT** operation affects cursors and prepared statements in the data source: SQL_CB_DELETE = Close cursors and delete prepared

	statements. To use the cursor again, the application must reprepare and reexecute the *hstmt*. SQL_CB_CLOSE = Close cursors. For prepared statements, the application can call **SQLExecute** on the *hstmt* without calling **SQLPrepare** again. SQL_CB_PRESERVE = Preserve cursors in the same position as before the **COMMIT** operation. The application can continue to fetch data or it can close the cursor and reexecute the *hstmt* without repreparing it.
SQL_CURSOR_ROLLBACK_ BEHAVIOR (ODBC 1.0)	A 16-bit integer value indicating how a **ROLLBACK** operation affects cursors and prepared statements in the data source: SQL_CB_DELETE = Close cursors and delete prepared statements. To use the cursor again, the application must reprepare and reexecute the *hstmt*. SQL_CB_CLOSE = Close cursors. For prepared statements, the application can call **SQLExecute** on the *hstmt* without calling **SQLPrepare** again. SQL_CB_PRESERVE = Preserve cursors in the same position as before the **ROLLBACK** operation. The application can continue to fetch data or it can close the cursor and reexecute the *hstmt* without repreparing it.
SQL_DATA_SOURCE_NAME (ODBC 1.0)	A character string with the data source name used during connection. If the application called **SQLConnect**, this is the value of the *szDSN* argument. If the application called **SQLDriverConnect** or **SQLBrowseConnect**, this is the value of the DSN keyword in the connection string passed to the driver. If the connection string did not contain the DSN keyword (such as when it contains the DRIVER keyword), this is an empty string.
SQL_DATA_SOURCE_READ_ONLY (ODBC 1.0)	A character string. "Y" if the data source is set to READ ONLY mode, "N" if it is otherwise. This characteristic pertains only to the data source itself, it is not a characteristic of the driver that enables access to the data source.
SQL_DATABASE_NAME (ODBC 1.0)	A character string with the name of the current database in use, if the data source defines a named object called "database." **Note** In ODBC 2.0, this value of *fInfoType* has been replaced by the SQL_CURRENT_QUALIFIER connection option. ODBC 2.0 drivers should continue to support the SQL_DATABASE_NAME information type, and ODBC 2.0 applications should only use it with ODBC 1.0 drivers.
SQL_DBMS_NAME (ODBC 1.0)	A character string with the name of the DBMS product accessed by the driver.
SQL_DBMS_VER (ODBC 1.0)	A character string indicating the version of the DBMS product accessed by the driver. The version is of the form ##.##.####, where the first two digits are the major version, the next two digits are the minor version, and the last four digits are the release version. The driver must render the DBMS product version in this form, but can also append the DBMS product-specific version as well. For example, "04.01.0000 Rdb 4.1".
SQL_DEFAULT_TXN_ISOLATION (ODBC 1.0)	A 32-bit integer that indicates the default transaction isolation level supported by the driver or data source, or zero if the

	data source does not support transactions. The following terms are used to define transaction isolation levels: **Dirty Read** Transaction 1 changes a row. Transaction 2 reads the changed row before transaction 1 commits the change. If transaction 1 rolls back the change, transaction 2 will have read a row that is considered to have never existed. **Nonrepeatable Read** Transaction 1 reads a row. Transaction 2 updates or deletes that row and commits this change. If transaction 1 attempts to reread the row, it will receive different row values or discover that the row has been deleted. **Phantom** Transaction 1 reads a set of rows that satisfy some search criteria. Transaction 2 inserts a row that matches the search criteria. If transaction 1 reexecutes the statement that read the rows, it receives a different set of rows. If the data source supports transactions, the driver returns one of the following bitmasks: SQL_TXN_READ_UNCOMMITTED = Dirty reads, nonrepeatable reads, and phantoms are possible. SQL_TXN_READ_COMMITTED = Dirty reads are not possible. Nonrepeatable reads and phantoms are possible. SQL_TXN_REPEATABLE_READ = Dirty reads and nonrepeatable reads are not possible. Phantoms are possible. SQL_TXN_SERIALIZABLE = Transactions are serializable. Dirty reads, nonrepeatable reads, and phantoms are not possible. SQL_TXN_VERSIONING = Transactions are serializable, but higher concurrency is possible than with SQL_TXN_SERIALIZABLE. Dirty reads are not possible. Typically, SQL_TXN_SERIALIZABLE is implemented by using locking protocols that reduce concurrency and SQL_TXN_VERSIONING is implemented by using a non-locking protocol such as record versioning. Oracle's Read Consistency isolation level is an example of SQL_TXN_VERSIONING.
SQL_DRIVER_HDBC SQL_DRIVER_HENV (ODBC 1.0)	A 32-bit value, the driver's environment handle or connection handle, determined by the argument *hdbc*. These information types are implemented by the Driver Manager alone.
SQL_DRIVER_HLIB (ODBC 2.0)	A 32-bit value, the library handle returned to the Driver Manager when it loaded the driver DLL. The handle is only valid for the *hdbc* specified in the call to **SQLGetInfo**. This information type is implemented by the Driver Manager alone.
SQL_DRIVER_HSTMT (ODBC 1.0)	A 32-bit value, the driver's statement handle determined by the Driver Manager statement handle, which must be passed on input in *rgbInfoValue* from the application. Note that in this case, *rgbInfoValue* is both an input and an output argument. The input *hstmt* passed in *rgbInfoValue* must have been an *hstmt* allocated on the argument *hdbc*. This information type is implemented by the Driver Manager alone.
SQL_DRIVER_NAME	A character string with the filename of the driver used to

(ODBC 1.0)	access the data source.
SQL_DRIVER_ODBC_VER (ODBC 2.0)	A character string with the version of ODBC that the driver supports. The version is of the form ##.##, where the first two digits are the major version and the next two digits are the minor version. SQL_SPEC_MAJOR and SQL_SPEC_MINOR define the major and minor version numbers. For the version of ODBC described in this manual, these are 2 and 0, and the driver should return "02.00". If a driver supports **SQLGetInfo** but does not support this value of the *fInfoType* argument, the Driver Manager returns "01.00".
SQL_DRIVER_VER (ODBC 1.0)	A character string with the version of the 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, the next two digits are the minor version, and the last four digits are the release version.
SQL_EXPRESSIONS_IN_ORDERBY (ODBC 1.0)	A character string: "Y" if the data source supports expressions in the **ORDER BY** list; "N" if it does not.
SQL_FETCH_DIRECTION (ODBC 1.0)	A 32-bit bitmask enumerating the supported fetch direction options.
The information type was introduced in ODBC 1.0; each bitmask is labeled with the version in which it was introduced.	The following bitmasks are used in conjunction with the flag to determine which options are supported: SQL_FD_FETCH_NEXT (ODBC 1.0) SQL_FD_FETCH_FIRST (ODBC 1.0) SQL_FD_FETCH_LAST (ODBC 1.0) SQL_FD_FETCH_PRIOR (ODBC 1.0) SQL_FD_FETCH_ABSOLUTE (ODBC 1.0) SQL_FD_FETCH_RELATIVE (ODBC 1.0) SQL_FD_FETCH_RESUME (ODBC 1.0) SQL_FD_FETCH_BOOKMARK (ODBC 2.0)
SQL_FILE_USAGE (ODBC 2.0)	A 16-bit integer value indicating how a single-tier driver directly treats files in a data source: SQL_FILE_NOT_SUPPORTED = The driver is not a single-tier driver. For example, an ORACLE driver is a two-tier driver. SQL_FILE_TABLE = A single-tier driver treats files in a data source as tables. For example, an Xbase driver treats each Xbase file as a table. SQL_FILE_QUALIFIER = A single-tier driver treats files in a data source as a qualifier. For example, a Microsoft Access driver treats each Microsoft Access file as a complete database. An application might use this to determine how users will select data. For example, Xbase users often think of data as stored in files, while ORACLE and Microsoft Access users generally think of data as stored in tables. When a user selects an Xbase data source, the application could display the Windows File Open common dialog box; when the user selects a Microsoft Access or ORACLE data source, the application could display a custom Select Table dialog box.
SQL_GETDATA_EXTENSIONS (ODBC 2.0)	A 32-bit bitmask enumerating extensions to **SQLGetData**. The following bitmasks are used in conjunction with the flag to determine what common extensions the driver supports for **SQLGetData**:

	SQL_GD_ANY_COLUMN = **SQLGetData** can be called for any unbound column, including those before the last bound column. Note that the columns must be called in order of ascending column number unless SQL_GD_ANY_ORDER is also returned. SQL_GD_ANY_ORDER = **SQLGetData** can be called for unbound columns in any order. Note that **SQLGetData** can only be called for columns after the last bound column unless SQL_GD_ANY_COLUMN is also returned. SQL_GD_BLOCK = **SQLGetData** can be called for an unbound column in any row in a block (more than one row) of data after positioning to that row with **SQLSetPos**. SQL_GD_BOUND = **SQLGetData** can be called for bound columns as well as unbound columns. A driver cannot return this value unless it also returns SQL_GD_ANY_COLUMN. **SQLGetData** is only required to return data from unbound columns that occur after the last bound column, are called in order of increasing column number, and are not in a row in a block of rows.
SQL_GROUP_BY (ODBC 2.0)	A 16-bit integer value specifying the relationship between the columns in the **GROUP BY** clause and the non-aggregated columns in the select list: SQL_GB_NOT_SUPPORTED = **GROUP BY** clauses are not supported. SQL_GB_GROUP_BY_EQUALS_SELECT = The **GROUP BY** clause must contain all non-aggregated columns in the select list. It cannot contain any other columns. For example, **SELECT DEPT, MAX(SALARY) FROM EMPLOYEE GROUP BY DEPT**. SQL_GB_GROUP_BY_CONTAINS_SELECT = The **GROUP BY** clause must contain all non-aggregated columns in the select list. It can contain columns that are not in the select list. For example, **SELECT DEPT, MAX(SALARY) FROM EMPLOYEE GROUP BY DEPT, AGE**. SQL_GB_NO_RELATION = The columns in the **GROUP BY** clause and the select list are not related. The meaning of non-grouped, non-aggregated columns in the select list is data source-dependent. For example, **SELECT DEPT, SALARY FROM EMPLOYEE GROUP BY DEPT, AGE**.
SQL_IDENTIFIER_CASE (ODBC 1.0)	A 16-bit integer value as follows: SQL_IC_UPPER = Identifiers in SQL are case insensitive and are stored in upper case in system catalog. SQL_IC_LOWER = Identifiers in SQL are case insensitive and are stored in lower case in system catalog. SQL_IC_SENSITIVE = Identifiers in SQL are case sensitive and are stored in mixed case in system catalog. SQL_IC_MIXED = Identifiers in SQL are case insensitive and are stored in mixed case in system catalog.
SQL_IDENTIFIER_QUOTE_CHAR (ODBC 1.0)	The character string used as the starting and ending delimiter of a quoted (delimited) identifiers in SQL statements. (Identifiers passed as arguments to ODBC functions do not need to be quoted.) If the data source does not support quoted identifiers, a blank is returned.
SQL_KEYWORDS (ODBC 2.0)	A character string containing a comma-separated list of all data source-specific keywords. This list does not contain keywords specific to ODBC or keywords used by both the

	data source and ODBC. The **#define** value SQL_ODBC_KEYWORDS contains a comma-separated list of ODBC keywords.
SQL_LIKE_ESCAPE_CLAUSE (ODBC 2.0)	A character string: "Y" if the data source supports an escape character for the percent character (%) and underscore character (_) in a **LIKE** predicate and the driver supports the ODBC syntax for defining a **LIKE** predicate escape character; "N" otherwise.
SQL_LOCK_TYPES (ODBC 2.0)	A 32-bit bitmask enumerating the supported lock types for the *fLock* argument in **SQLSetPos**. The following bitmasks are used in conjunction with the flag to determine which lock types are supported: SQL_LCK_NO_CHANGE SQL_LCK_EXCLUSIVE SQL_LCK_UNLOCK
SQL_MAX_BINARY_LITERAL_LEN (ODBC 2.0)	A 32-bit integer value specifying the maximum length (number of hexadecimal characters, excluding the literal prefix and suffix returned by **SQLGetTypeInfo**) of a binary literal in an SQL statement. For example, the binary literal 0xFFAA has a length of 4. If there is no maximum length or the length is unknown, this value is set to zero.
SQL_MAX_CHAR_LITERAL_LEN (ODBC 2.0)	A 32-bit integer value specifying the maximum length (number of characters, excluding the literal prefix and suffix returned by **SQLGetTypeInfo**) of a character literal in an SQL statement. If there is no maximum length or the length is unknown, this value is set to zero.
SQL_MAX_COLUMN_NAME_LEN (ODBC 1.0)	A 16-bit integer value specifying the maximum length of a column name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.
SQL_MAX_COLUMNS_IN_GROUP_BY (ODBC 2.0)	A 16-bit integer value specifying the maximum number of columns allowed in a **GROUP BY** clause. If there is no specified limit or the limit is unknown, this value is set to zero.
SQL_MAX_COLUMNS_IN_INDEX (ODBC 2.0)	A 16-bit integer value specifying the maximum number of columns allowed in an index. If there is no specified limit or the limit is unknown, this value is set to zero.
SQL_MAX_COLUMNS_IN_ORDER_BY (ODBC 2.0)	A 16-bit integer value specifying the maximum number of columns allowed in an **ORDER BY** clause. If there is no specified limit or the limit is unknown, this value is set to zero.
SQL_MAX_COLUMNS_IN_SELECT (ODBC 2.0)	A 16-bit integer value specifying the maximum number of columns allowed in a select list. If there is no specified limit or the limit is unknown, this value is set to zero.
SQL_MAX_COLUMNS_IN_TABLE (ODBC 2.0)	A 16-bit integer value specifying the maximum number of columns allowed in a table. If there is no specified limit or the limit is unknown, this value is set to zero.
SQL_MAX_CURSOR_NAME_LEN (ODBC 1.0)	A 16-bit integer value specifying the maximum length of a cursor name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.
SQL_MAX_INDEX_SIZE (ODBC 2.0)	A 32-bit integer value specifying the maximum number of bytes allowed in the combined fields of an index. If there is no specified limit or the limit is unknown, this value is set to zero.
SQL_MAX_OWNER_NAME_LEN (ODBC 1.0)	A 16-bit integer value specifying the maximum length of an owner name in the data source. If there is no maximum

	length or the length is unknown, this value is set to zero.
SQL_MAX_PROCEDURE_NAME_LEN (ODBC 1.0)	A 16-bit integer value specifying the maximum length of a procedure name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.
SQL_MAX_QUALIFIER_NAME_LEN (ODBC 1.0)	A 16-bit integer value specifying the maximum length of a qualifier name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.
SQL_MAX_ROW_SIZE (ODBC 2.0)	A 32-bit integer value specifying the maximum length of a single row in a table. If there is no specified limit or the limit is unknown, this value is set to zero.
SQL_MAX_ROW_SIZE_INCLUDES_LONG (ODBC 2.0)	A character string: "Y" if the maximum row size returned for the SQL_MAX_ROW_SIZE information type includes the length of all SQL_LONGVARCHAR and SQL_LONGVARBINARY columns in the row; "N" otherwise.
SQL_MAX_STATEMENT_LEN (ODBC 2.0)	A 32-bit integer value specifying the maximum length (number of characters, including white space) of an SQL statement. If there is no maximum length or the length is unknown, this value is set to zero.
SQL_MAX_TABLE_NAME_LEN (ODBC 1.0)	A 16-bit integer value specifying the maximum length of a table name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.
SQL_MAX_TABLES_IN_SELECT (ODBC 2.0)	A 16-bit integer value specifying the maximum number of tables allowed in the **FROM** clause of a **SELECT** statement. If there is no specified limit or the limit is unknown, this value is set to zero.
SQL_MAX_USER_NAME_LEN (ODBC 2.0)	A 16-bit integer value specifying the maximum length of a user name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.
SQL_MULT_RESULT_SETS (ODBC 1.0)	A character string: "Y" if the data source supports multiple result sets, "N" if it does not.
SQL_MULTIPLE_ACTIVE_TXN (ODBC 1.0)	A character string: "Y" if active transactions on multiple connections are allowed, "N" if only one connection at a time can have an active transaction.
SQL_NEED_LONG_DATA_LEN (ODBC 2.0)	A character string: "Y" if the data source needs the length of a long data value (the data type is SQL_LONGVARCHAR, SQL_LONGVARBINARY, or a long, data source-specific data type) before that value is sent to the data source, "N" if it does not. For more information, see **SQLBindParameter** and **SQLSetPos**.
SQL_NON_NULLABLE_COLUMNS (ODBC 1.0)	A 16-bit integer specifying whether the data source supports non-nullable columns: SQL_NNC_NULL = All columns must be nullable. SQL_NNC_NON_NULL = Columns may be non-nullable (the data source supports the **NOT NULL** column constraint in **CREATE TABLE** statements).
SQL_NULL_COLLATION (ODBC 2.0)	A 16-bit integer value specifying where NULLs are sorted in a list: SQL_NC_END = NULLs are sorted at the end of the list, regardless of the sort order. SQL_NC_HIGH = NULLs are sorted at the high end of the list. SQL_NC_LOW = NULLs are sorted at the low end of the list. SQL_NC_START = NULLs are sorted at the start of the list, regardless of the sort order.
SQL_NUMERIC_FUNCTIONS	A 32-bit bitmask enumerating the scalar numeric functions

(ODBC 1.0) The information type was introduced in ODBC 1.0; each bitmask is labeled with the version in which it was introduced.	supported by the driver and associated data source. The following bitmasks are used to determine which numeric functions are supported: SQL_FN_NUM_ABS (ODBC 1.0) SQL_FN_NUM_ACOS (ODBC 1.0) SQL_FN_NUM_ASIN (ODBC 1.0) SQL_FN_NUM_ATAN (ODBC 1.0) SQL_FN_NUM_ATAN2 (ODBC 1.0) SQL_FN_NUM_CEILING (ODBC 1.0) SQL_FN_NUM_COS (ODBC 1.0) SQL_FN_NUM_COT (ODBC 1.0) SQL_FN_NUM_DEGREES (ODBC 2.0) SQL_FN_NUM_EXP (ODBC 1.0) SQL_FN_NUM_FLOOR (ODBC 1.0) SQL_FN_NUM_LOG (ODBC 1.0) SQL_FN_NUM_LOG10 (ODBC 2.0) SQL_FN_NUM_MOD (ODBC 1.0) SQL_FN_NUM_PI (ODBC 1.0) SQL_FN_NUM_POWER (ODBC 2.0) SQL_FN_NUM_RADIANS (ODBC 2.0) SQL_FN_NUM_RAND (ODBC 1.0) SQL_FN_NUM_ROUND (ODBC 2.0) SQL_FN_NUM_SIGN (ODBC 1.0) SQL_FN_NUM_SIN (ODBC 1.0) SQL_FN_NUM_SQRT (ODBC 1.0) SQL_FN_NUM_TAN (ODBC 1.0) SQL_FN_NUM_TRUNCATE (ODBC 2.0)

SQL_ODBC_API_CONFORMANCE
(ODBC 1.0)
	A 16-bit integer value indicating the level of ODBC conformance:
	SQL_OAC_NONE = None
	SQL_OAC_LEVEL1 = Level 1 supported
	SQL_OAC_LEVEL2 = Level 2 supported

SQL_ODBC_SAG_CLI_ CONFORMANCE (ODBC 1.0)	A 16-bit integer value indicating compliance to the functions of the SAG specification: SQL_OSCC_NOT_COMPLIANT = Not SAG-compliant; one or more core functions are not supported SQL_OSCC_COMPLIANT = SAG-compliant

SQL_ODBC_SQL_CONFORMANC
E
(ODBC 1.0)
	A 16-bit integer value indicating SQL grammar supported by the driver:
	SQL_OSC_MINIMUM = Minimum grammar supported
	SQL_OSC_CORE = Core grammar supported
	SQL_OSC_EXTENDED = Extended grammar supported

SQL_ODBC_SQL_OPT_IEF (ODBC 1.0)	A character string: "Y" if the data source supports the optional Integrity Enhancement Facility; "N" if it does not.
SQL_ODBC_VER (ODBC 1.0)	A character string with the version of ODBC to which the Driver Manager conforms. The version is of the form ##.##, where the first two digits are the major version and the next two digits are the minor version. This is implemented solely in the Driver Manager.
SQL_ORDER_BY_COLUMNS_IN_ SELECT (ODBC 2.0)	A character string: "Y" if the columns in the **ORDER BY** clause must be in the select list; otherwise, "N".
SQL_OUTER_JOINS (ODBC 1.0) The information type was introduced	A character string: "N" = No. The data source does not support outer joins. (ODBC 1.0)

in ODBC 1.0; each return value is labeled with the version in which it was introduced.	"Y" = Yes. The data source supports two-table outer joins, and the driver supports the ODBC outer join syntax except for nested outer joins. However, columns on the left side of the comparison operator in the ON clause must come from the left-hand table in the outer join, and columns on the right side of the comparison operator must come from the right-hand table. (ODBC 1.0) "P" = Partial. The data source partially supports nested outer joins, and the driver supports the ODBC outer join syntax. However, columns on the left side of the comparison operator in the ON clause must come from the left-hand table in the outer join and columns on the right side of the comparison operator must come from the right-hand table. Also, the right-hand table of an outer join cannot be included in an inner join. (ODBC 2.0) "F" = Full. The data source fully supports nested outer joins, and the driver supports the ODBC outer join syntax. (ODBC 2.0)
SQL_OWNER_TERM (ODBC 1.0)	A character string with the data source vendor's name for an owner; for example, "owner", "Authorization ID", or "Schema".
SQL_OWNER_USAGE (ODBC 2.0)	A 32-bit bitmask enumerating the statements in which owners can be used: SQL_OU_DML_STATEMENTS = Owners are supported in all Data Manipulation Language statements: **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and, if supported, **SELECT FOR UPDATE** and positioned update and delete statements. SQL_OU_PROCEDURE_INVOCATION = Owners are supported in the ODBC procedure invocation statement. SQL_OU_TABLE_DEFINITION = Owners are supported in all table definition statements: **CREATE TABLE**, **CREATE VIEW**, **ALTER TABLE**, **DROP TABLE**, and **DROP VIEW**. SQL_OU_INDEX_DEFINITION = Owners are supported in all index definition statements: **CREATE INDEX** and **DROP INDEX**. SQL_OU_PRIVILEGE_DEFINITION = Owners are supported in all privilege definition statements: **GRANT** and **REVOKE**.
SQL_POS_OPERATIONS (ODBC 2.0)	A 32-bit bitmask enumerating the supported operations in **SQLSetPos**. The following bitmasks are used to in conjunction with the flag to determine which options are supported: SQL_POS_POSITION SQL_POS_REFRESH SQL_POS_UPDATE SQL_POS_DELETE SQL_POS_ADD
SQL_POSITIONED_STATEMENTS (ODBC 2.0)	A 32-bit bitmask enumerating the supported positioned SQL statements. The following bitmasks are used to determine which statements are supported: SQL_PS_POSITIONED_DELETE SQL_PS_POSITIONED_UPDATE SQL_PS_SELECT_FOR_UPDATE
SQL_PROCEDURE_TERM	A character string with the data source vendor's name for a procedure; for example, "database procedure", "stored

(ODBC 1.0)	procedure", or "procedure".
SQL_PROCEDURES (ODBC 1.0)	A character string: "Y" if the data source supports procedures and the driver supports the ODBC procedure invocation syntax; "N" otherwise.
SQL_QUALIFIER_LOCATION (ODBC 2.0)	A 16-bit integer value indicating the position of the qualifier in a qualified table name: SQL_QL_START SQL_QL_END For example, an Xbase driver returns SQL_QL_START because the directory (qualifier) name is at the start of the table name, as in \EMPDATA\EMP.DBF. An ORACLE Server driver returns SQL_QL_END, because the qualifier is at the end of the table name, as in ADMIN.EMP@EMPDATA.
SQL_QUALIFIER_NAME _SEPARATOR (ODBC 1.0)	A character string: the character or characters that the data source defines as the separator between a qualifier name and the qualified name element that follows it.
SQL_QUALIFIER_TERM (ODBC 1.0)	A character string with the data source vendor's name for a qualifier; for example, "database" or "directory".
SQL_QUALIFIER_USAGE (ODBC 2.0)	A 32-bit bitmask enumerating the statements in which qualifiers can be used. The following bitmasks are used to determine where qualifiers can be used: SQL_QU_DML_STATEMENTS = Qualifiers are supported in all Data Manipulation Language statements: **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and, if supported, **SELECT FOR UPDATE** and positioned update and delete statements. SQL_QU_PROCEDURE_INVOCATION = Qualifiers are supported in the ODBC procedure invocation statement. SQL_QU_TABLE_DEFINITION = Qualifiers are supported in all table definition statements: **CREATE TABLE**, **CREATE VIEW**, **ALTER TABLE**, **DROP TABLE**, and **DROP VIEW**. SQL_QU_INDEX_DEFINITION = Qualifiers are supported in all index definition statements: **CREATE INDEX** and **DROP INDEX**. SQL_QU_PRIVILEGE_DEFINITION = Qualifiers are supported in all privilege definition statements: **GRANT** and **REVOKE**.
SQL_QUOTED_IDENTIFIER_CASE (ODBC 2.0)	A 16-bit integer value as follows: SQL_IC_UPPER = Quoted identifiers in SQL are case insensitive and are stored in upper case in system catalog. SQL_IC_LOWER = Quoted identifiers in SQL are case insensitive and are stored in lower case in system catalog. SQL_IC_SENSITIVE = Quoted identifiers in SQL are case sensitive and are stored in mixed case in system catalog. SQL_IC_MIXED = Quoted identifiers in SQL are case insensitive and are stored in mixed case in system catalog.
SQL_ROW_UPDATES (ODBC 1.0)	A character string: "Y" if a keyset-driven or mixed cursor maintains row versions or values for all fetched rows and therefore can detect any changes made to a row by any user since the row was last fetched; otherwise, "N".
SQL_SCROLL_CONCURRENCY (ODBC 1.0)	A 32-bit bitmask enumerating the concurrency control options supported for scrollable cursors. The following bitmasks are used to determine which options are supported:

	SQL_SCCO_READ_ONLY = Cursor is read only. No updates are allowed. SQL_SCCO_LOCK = Cursor uses the lowest level of locking sufficient to ensure that the row can be updated. SQL_SCCO_OPT_ROWVER = Cursor uses optimistic concurrency control, comparing row versions, such as SQLBase® ROWID or Sybase TIMESTAMP. SQL_SCCO_OPT_VALUES = Cursor uses optimistic concurrency control, comparing values.
SQL_SCROLL_OPTIONS (ODBC 1.0)	A 32-bit bitmask enumerating the scroll options supported for scrollable cursors.
The information type was introduced in ODBC 1.0; each bitmask is labeled with the version in which it was introduced.	The following bitmasks are used to determine which options are supported: SQL_SO_FORWARD_ONLY = The cursor only scrolls forward. (ODBC 1.0) SQL_SO_STATIC = The data in the result set is static. (ODBC 2.0) SQL_SO_KEYSET_DRIVEN = The driver saves and uses the keys for every row in the result set. (ODBC 1.0) SQL_SO_DYNAMIC = The driver keeps the keys for every row in the rowset (the keyset size is the same as the rowset size). (ODBC 1.0) SQL_SO_MIXED = The driver keeps the keys for every row in the keyset, and the keyset size is greater than the rowset size. The cursor is keyset-driven inside the keyset and dynamic outside the keyset. (ODBC 1.0)
SQL_SEARCH_PATTERN_ESCAPE (ODBC 1.0)	A character string specifying what the driver supports as an escape character that permits the use of the pattern match metacharacters underscore (_) and percent (%) as valid characters in search patterns. This escape character applies only for those catalog function arguments that support search strings. If this string is empty, the driver does not support a search-pattern escape character. This *fInfoType* is limited to catalog functions.
SQL_SERVER_NAME (ODBC 1.0)	A character string with the actual data source-specific server name; useful when a data source name is used during **SQLConnect**, **SQLDriverConnect**, and **SQLBrowseConnect**.
SQL_SPECIAL_CHARACTERS (ODBC 2.0)	A character string containing all special characters (that is, all characters except a through z, A through Z, 0 through 9, and underscore) that can be used in an object name, such as a table, column, or index name, on the data source. For example, "#$^".
SQL_STATIC_SENSITIVITY (ODBC 2.0)	A 32-bit bitmask enumerating whether changes made by an application to a static or keyset-driven cursor through **SQLSetPos** or positioned update or delete statements can be detected by that application: SQL_SS_ADDITIONS = Added rows are visible to the cursor; the cursor can scroll to these rows. Where these rows are added to the cursor is driver-dependent. SQL_SS_DELETIONS = Deleted rows are no longer available to the cursor and do not leave a "hole" in the result set; after the cursor scrolls from a deleted row, it cannot return to that row. SQL_SS_UPDATES = Updates to rows are visible to the

	cursor; if the cursor scrolls from and returns to an updated row, the data returned by the cursor is the updated data, not the original data. Because updating key values in a keyset-driven cursor is considered to be deleting the existing row and adding a new row, this value is always returned for keyset-driven cursors. Whether an application can detect changes made to the result set by other users, including other cursors in the same application, depends on the cursor type.
SQL_STRING_FUNCTIONS (ODBC 1.0) The information type was introduced in ODBC 1.0; each bitmask is labeled with the version in which it was introduced.	A 32-bit bitmask enumerating the scalar string functions supported by the driver and associated data source. The following bitmasks are used to determine which string functions are supported: SQL_FN_STR_ASCII (ODBC 1.0) SQL_FN_STR_CHAR (ODBC 1.0) SQL_FN_STR_CONCAT (ODBC 1.0) SQL_FN_STR_DIFFERENCE (ODBC 2.0) SQL_FN_STR_INSERT (ODBC 1.0) SQL_FN_STR_LCASE (ODBC 1.0) SQL_FN_STR_LEFT (ODBC 1.0) SQL_FN_STR_LENGTH (ODBC 1.0) SQL_FN_STR_LOCATE (ODBC 1.0) SQL_FN_STR_LOCATE_2 (ODBC 2.0) SQL_FN_STR_LTRIM (ODBC 1.0) SQL_FN_STR_REPEAT (ODBC 1.0) SQL_FN_STR_REPLACE (ODBC 1.0) SQL_FN_STR_RIGHT (ODBC 1.0) SQL_FN_STR_RTRIM (ODBC 1.0) SQL_FN_STR_SOUNDEX (ODBC 2.0) SQL_FN_STR_SPACE (ODBC 2.0) SQL_FN_STR_SUBSTRING (ODBC 1.0) SQL_FN_STR_UCASE (ODBC 1.0) If an application can call the LOCATE scalar function with the *string_exp1*, *string_exp2*, and *start* arguments, the driver returns the SQL_FN_STR_LOCATE bitmask. If an application can call the LOCATE scalar function with only the *string_exp1* and *string_exp2* arguments, the driver returns the SQL_FN_STR_LOCATE_2 bitmask. Drivers that fully support the LOCATE scalar function return both bitmasks.
SQL_SUBQUERIES (ODBC 2.0)	A 32-bit bitmask enumerating the predicates that support subqueries: SQL_SQ_CORRELATED_SUBQUERIES SQL_SQ_COMPARISON SQL_SQ_EXISTS SQL_SQ_IN SQL_SQ_QUANTIFIED The SQL_SQ_CORRELATED_SUBQUERIES bitmask indicates that all predicates that support subqueries support correlated subqueries.
SQL_SYSTEM_FUNCTIONS (ODBC 1.0)	A 32-bit bitmask enumerating the scalar system functions supported by the driver and associated data source. The following bitmasks are used to determine which system functions are supported: SQL_FN_SYS_DBNAME SQL_FN_SYS_IFNULL SQL_FN_SYS_USERNAME
SQL_TABLE_TERM	A character string with the data source vendor's name for a

(ODBC 1.0)	table; for example, "table" or "file".
SQL_TIMEDATE_ADD_INTERVALS (ODBC 2.0)	A 32-bit bitmask enumerating the timestamp intervals supported by the driver and associated data source for the TIMESTAMPADD scalar function. The following bitmasks are used to determine which intervals are supported: SQL_FN_TSI_FRAC_SECOND SQL_FN_TSI_SECOND SQL_FN_TSI_MINUTE SQL_FN_TSI_HOUR SQL_FN_TSI_DAY SQL_FN_TSI_WEEK SQL_FN_TSI_MONTH SQL_FN_TSI_QUARTER SQL_FN_TSI_YEAR
SQL_TIMEDATE_DIFF_INTERVALS (ODBC 2.0)	A 32-bit bitmask enumerating the timestamp intervals supported by the driver and associated data source for the TIMESTAMPDIFF scalar function. The following bitmasks are used to determine which intervals are supported: SQL_FN_TSI_FRAC_SECOND SQL_FN_TSI_SECOND SQL_FN_TSI_MINUTE SQL_FN_TSI_HOUR SQL_FN_TSI_DAY SQL_FN_TSI_WEEK SQL_FN_TSI_MONTH SQL_FN_TSI_QUARTER SQL_FN_TSI_YEAR
SQL_TIMEDATE_FUNCTIONS (ODBC 1.0) The information type was introduced in ODBC 1.0; each bitmask is labeled with the version in which it was introduced.	A 32-bit bitmask enumerating the scalar date and time functions supported by the driver and associated data source. The following bitmasks are used to determine which date and time functions are supported: SQL_FN_TD_CURDATE (ODBC 1.0) SQL_FN_TD_CURTIME (ODBC 1.0) SQL_FN_TD_DAYNAME (ODBC 2.0) SQL_FN_TD_DAYOFMONTH (ODBC 1.0) SQL_FN_TD_DAYOFWEEK (ODBC 1.0) SQL_FN_TD_DAYOFYEAR (ODBC 1.0) SQL_FN_TD_HOUR (ODBC 1.0) SQL_FN_TD_MINUTE (ODBC 1.0) SQL_FN_TD_MONTH (ODBC 1.0) SQL_FN_TD_MONTHNAME (ODBC 2.0) SQL_FN_TD_NOW (ODBC 1.0) SQL_FN_TD_QUARTER (ODBC 1.0) SQL_FN_TD_SECOND (ODBC 1.0) SQL_FN_TD_TIMESTAMPADD (ODBC 2.0)

	SQL_FN_TD_TIMESTAMPDIFF (ODBC 2.0) SQL_FN_TD_WEEK (ODBC 1.0) SQL_FN_TD_YEAR (ODBC 1.0)
SQL_TXN_CAPABLE (ODBC 1.0) The information type was introduced in ODBC 1.0; each return value is labeled with the version in which it was introduced	A 16-bit integer value describing the transaction support in the driver or data source: SQL_TC_NONE = Transactions not supported. (ODBC 1.0) SQL_TC_DML = Transactions can only contain Data Manipulation Language (DML) statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**). Data Definition Language (DDL) statements encountered in a transaction cause an error. (ODBC 1.0) SQL_TC_DDL_COMMIT = Transactions can only contain DML statements. DDL statements (**CREATE TABLE**, **DROP INDEX**, an so on) encountered in a transaction cause the transaction to be committed. (ODBC 2.0) SQL_TC_DDL_IGNORE = Transactions can only contain DML statements. DDL statements encountered in a transaction are ignored. (ODBC 2.0) SQL_TC_ALL = Transactions can contain DDL statements and DML statements in any order. (ODBC 1.0)
SQL_TXN_ISOLATION_OPTION (ODBC 1.0)	A 32-bit bitmask enumerating the transaction isolation levels available from the driver or data source. The following bitmasks are used in conjunction with the flag to determine which options are supported: SQL_TXN_READ_UNCOMMITTED SQL_TXN_READ_COMMITTED SQL_TXN_REPEATABLE_READ SQL_TXN_SERIALIZABLE SQL_TXN_VERSIONING For descriptions of these isolation levels, see the description of SQL_DEFAULT_TXN_ISOLATION.
SQL_UNION (ODBC 2.0)	A 32-bit bitmask enumerating the support for the **UNION** clause: SQL_U_UNION = The data source supports the **UNION** clause. SQL_U_UNION_ALL = The data source supports the **ALL** keyword in the **UNION** clause. (**SQLGetInfo** returns both SQL_U_UNION and SQL_U_UNION_ALL in this case.)
SQL_USER_NAME (ODBC 1.0)	A character string with the name used in a particular database, which can be different than login name.

▪

Code Example

SQLGetInfo returns lists of supported options as a 32-bit bitmask in *rgbInfoValue*. The bitmask for each option is used in conjunction with the flag to determine whether the option is supported.

For example, an application could use the following code to determine whether the SUBSTRING scalar function is supported by the driver associated with the *hdbc*:

```
UDWORD   fFuncs;


SQLGetInfo(hdbc, SQL_STRING_FUNCTIONS, (PTR)&fFuncs, sizeof(fFuncs), NULL);


if (fFuncs & SQL_FN_STR_SUBSTRING) /* SUBSTRING supported */
   ...;
else                               /* SUBSTRING not supported */
   ...;
```

Related Functions

SQLGetConnectOption (extension)
SQLGetFunctions (extension)
SQLGetStmtOption (extension)
SQLGetTypeInfo (extension)

SQLGetStmtOption (Extension Level 1, ODBC 1.0)

Related Functions

SQLGetStmtOption returns the current setting of a statement option.

Syntax

RETCODE **SQLGetStmtOption**(*hstmt*, *fOption*, *pvParam*)

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 associated with *fOption*. Depending on the value of *fOption*, a 32-bit integer value or a pointer to a null-terminated character string will be returned in *pvParam*.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLGetStmtOption** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLGetStmtOption** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
24000	Invalid cursor state	The argument *fOption* was SQL_ROW_NUMBER or SQL_GET_BOOKMARK and the cursor was not open, or the cursor was positioned before the start of the result set or after the end of the result set.
IM001	Driver does not support this function	(DM) The driver corresponding to the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1010	Function sequence error	(DM) An asynchronously executing function was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters

		or columns.
S1011	Operation invalid at this time	The *fOption* argument was SQL_GET_BOOKMARK and the value of the SQL_USE_BOOKMARKS statement option was SQL_UB_OFF.
S1092	Option type out of range	(DM) The value specified for the argument *fOption* was in the block of numbers reserved for ODBC connection and statement options, but was not valid for the version of ODBC supported by the driver.
S1109	Invalid cursor position	The *fOption* argument was SQL_GET_BOOKMARK or SQL_ROW_NUMBER and the value in the *rgfRowStatus* array in **SQLExtendedFetch** for the current row was SQL_ROW_DELETED or SQL_ROW_ERROR.
S1C00	Driver not capable	The value specified for the argument *fOption* was a valid ODBC statement option for the version of ODBC supported by the driver, but was not supported by the driver. The value specified for the argument *fOption* was in the block of numbers reserved for driver-specific connection and statement options, but was not supported by the driver.

Comments

The following table lists statement options for which corresponding values can be returned, but not set. The table also lists the version of ODBC in which they were introduced. For a list of options that can be set and retrieved, see **SQLSetStmtOption**. If *fOption* specifies an option that returns a string, *pvParam* must be a pointer to storage for the string. The maximum length of the string will be SQL_MAX_OPTION_STRING_LENGTH bytes (excluding the null termination byte).

fOption	***pvParam* contents**
SQL_GET_BOOKMARK (ODBC 2.0)	A 32-bit integer value that is the bookmark for the current row. Before using this option, an application must set the SQL_USE_BOOKMARKS statement option to SQL_UB_ON, create a result set, and call **SQLExtendedFetch**. To return to the rowset starting with the row marked by this bookmark, an application calls **SQLExtendedFetch** with the SQL_FETCH_BOOKMARK fetch type and *irow* set to this value. Bookmarks are also returned as column 0 of the result set.
SQL_ROW_NUMBER (ODBC 2.0)	A 32-bit integer value that specifies the number of the current row in the entire result set. If the number of the current row cannot be determined or there is no current row, the driver returns 0.

Related Functions

SQLGetConnectOption (extension)
SQLSetConnectOption (extension)
SQLSetStmtOption (extension)

SQLGetTypeInfo (Extension Level 1, ODBC 1.0)

Related Functions

SQLGetTypeInfo returns information about data types supported by the data source. The driver returns the information in the form of an SQL result set.

Important Applications must use the type names returned in the TYPE_NAME column in **ALTER TABLE** and **CREATE TABLE** statements. **SQLGetTypeInfo** may return more than one row with the same value in the DATA_TYPE column.

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	The SQL data type. This must be one of the following values: SQL_BIGINT SQL_BINARY SQL_BIT SQL_CHAR SQL_DATE SQL_DECIMAL SQL_DOUBLE SQL_FLOAT SQL_INTEGER SQL_LONGVARBINARY SQL_LONGVARCHAR SQL_NUMERIC SQL_REAL SQL_SMALLINT SQL_TIME SQL_TIMESTAMP SQL_TINYINT SQL_VARBINARY SQL_VARCHAR or a driver-specific SQL data type. SQL_ALL_TYPES specifies that information about all data types should be returned. For information about ODBC SQL data types, see SQL Data Types. For information about driver-specific SQL data types, see the driver's documentation.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLGetTypeInfo** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLGetTypeInfo** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description

01000	General warning	Driver specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
24000	Invalid cursor state	(DM) A cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch** had not been called. A result set was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called.
IM001	Driver does not support this function	(DM) The driver corresponding to the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1004	SQL data type out of range	(DM) The value specified for the argument *fSqlType* was in the block of numbers reserved for ODBC SQL data type indicators but was not a valid ODBC SQL data type indicator.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*, then the function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1C00	Driver not capable	The value specified for the argument *fSqlType* was in the range of numbers reserved for driver-specific SQL data type indicators, but was not supported by the driver or data source. The combination of the current settings of the SQL_CONCURRENCY and

		SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

SQLGetTypeInfo returns the results as a standard result set, ordered by DATA_TYPE and TYPE_NAME. The following table lists the columns in the result set.

Note **SQLGetTypeInfo** might not return all data types. For example, a driver might not return user-defined data types. Applications can use any valid data type, regardless of whether it is returned by **SQLGetTypeInfo**.

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source.

Column Name	**Data Type**	**Comments**
TYPE_NAME	Varchar(128) not NULL	Data source-dependent data type name; for example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINARY", or "CHAR ( ) FOR BIT DATA". Applications must use this name in **CREATE TABLE** and **ALTER TABLE** statements.
DATA_TYPE	Smallint not NULL	SQL data type. This can be an ODBC SQL data type or a driver-specific SQL data type. For a list of valid ODBC SQL data types, see SQL Data Types. For information about driver-specific SQL data types, see the driver's documentation.
PRECISION	Integer	The maximum precision of the data type on the data source. NULL is returned for data types where precision is not applicable. For more information on precision, see Precision, Scale, Length, and Display Size.
LITERAL_PREFIX	Varchar(128)	Character or characters used to prefix a literal; for example, a single quote (') for character data types or 0x for binary data types; NULL is returned for data types where a literal prefix is not applicable.
LITERAL_SUFFIX	Varchar(128)	Character or characters used to terminate a literal; for example, a single quote (') for character data types; NULL is returned for data types where a literal suffix is not applicable.
CREATE_PARAMS	Varchar(128)	Parameters for a data type definition. For example, CREATE_PARAMS for DECIMAL would be "precision,scale"; CREATE_PARAMS for VARCHAR would equal "max length"; NULL is returned if there are no parameters for

		the data type definition, for example INTEGER. The driver supplies the CREATE_PARAMS text in the language of the country where it is used.
NULLABLE	Smallint not NULL	Whether the data type accepts a NULL value: SQL_NO_NULLS if the data type does not accept NULL values. SQL_NULLABLE if the data type accepts NULL values. SQL_NULLABLE_UNKNOWN if it is not known if the column accepts NULL values.
CASE_SENSITIVE	Smallint not NULL	Whether a character data type is case sensitive in collations and comparisons: TRUE if the data type is a character data type and is case sensitive. FALSE if the data type is not a character data type or is not case sensitive.
SEARCHABLE	Smallint not NULL	How the data type is used in a **WHERE** clause: SQL_UNSEARCHABLE if the data type cannot be used in a **WHERE** clause. SQL_LIKE_ONLY if the data type can be used in a **WHERE** clause only with the **LIKE** predicate. SQL_ALL_EXCEPT_LIKE if the data type can be used in a **WHERE** clause with all comparison operators except **LIKE**. SQL_SEARCHABLE if the data type can be used in a **WHERE** clause with any comparison operator.
UNSIGNED_ATTRIBUTE	Smallint	Whether the data type is unsigned: TRUE if the data type is unsigned. FALSE if the data type is signed. NULL is returned if the attribute is not applicable to the data type or the data type is not numeric.
MONEY	Smallint not NULL	Whether the data type is a money data type: TRUE if it is a money data type. FALSE if it is not.
AUTO_INCREMENT	Smallint	Whether the data type is autoincrementing: TRUE if the data type is autoincrementing. FALSE if the data type is not autoincrementing.

		NULL is returned if the attribute is not applicable to the data type or the data type is not numeric. An application can insert values into a column having this attribute, but cannot update the values in the column.
LOCAL_TYPE_NAME	Varchar(128)	Localized version of the data source-dependent name of the data type. NULL is returned if a localized name is not supported by the data source. This name is intended for display only, such as in dialog boxes.
MINIMUM_SCALE	Smallint	The minimum scale of the data type on the data source. If a data type has a fixed scale, the MINIMUM_SCALE and MAXIMUM_SCALE columns both contain this value. For example, an SQL_TIMESTAMP column might have a fixed scale for fractional seconds. NULL is returned where scale is not applicable. For more information, see Precision, Scale, Length, and Display Size.
MAXIMUM_SCALE	Smallint	The maximum scale of the data type on the data source. NULL is returned where scale is not applicable. If the maximum scale is not defined separately on the data source, but is instead defined to be the same as the maximum precision, this column contains the same value as the PRECISION column. For more information, see Precision, Scale, Length, and Display Size.

Note The MINIMUM_SCALE and MAXIMUM_SCALE columns were added in ODBC 2.0. ODBC 1.0 drivers may return different, driver-specific columns with the same column numbers.

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.

Related Functions

SQLBindCol
SQLCancel
SQLColAttributes
SQLExtendedFetch (extension)
SQLFetch
SQLGetInfo (extension)

SQLMoreResults (Extension Level 2, ODBC 1.0)

Related Functions

SQLMoreResults determines whether there are more results available on an *hstmt* containing **SELECT**, **UPDATE**, **INSERT**, or **DELETE** statements and, if so, initializes processing for those results.

Syntax

RETCODE **SQLMoreResults**(*hstmt*)

The **SQLMoreResults** function accepts the following argument:

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.

Returns

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

Diagnostics

When **SQLMoreResults** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLMoreResults** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.

S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

SELECT statements return result sets. **UPDATE**, **INSERT**, and **DELETE** statements return a count of affected rows. If any of these statements are batched, submitted with arrays of parameters, or in procedures, they can return multiple result sets or counts.

If another result set or count is available, **SQLMoreResults** returns SQL_SUCCESS and initializes the result set or count for additional processing. After calling **SQLMoreResults** for **SELECT** statements, an application can call functions to determine the characteristics of the result set and to retrieve data from the result set. After calling **SQLMoreResults** for **UPDATE**, **INSERT**, or **DELETE** statements, an application can call **SQLRowCount**.

If all results have been processed, **SQLMoreResults** returns SQL_NO_DATA_FOUND.

Note that if there is a current result set with unfetched rows, **SQLMoreResults** discards that result set and makes the next result set or count available.

If a batch of statements or a procedure mixes other SQL statements with **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements, these other statements do not affect **SQLMoreResults**.

Related Functions

SQLCancel
SQLExtendedFetch (extension)
SQLFetch
SQLGetData (extension)

SQLNativeSql (Extension Level 2, ODBC 1.0)

SQLNativeSql returns the SQL string as translated by the driver.

Syntax

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

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

Type	Argument	Use	Description
HDBC	*hdbc*	Input	Connection handle.
UCHAR FAR *	*szSqlStrIn*	Input	SQL text string to be translated.
SDWORD	*cbSqlStrIn*	Input	Length of *szSqlStrIn* text string.
UCHAR FAR *	*szSqlStr*	Output	Pointer to storage for the translated SQL string.
SDWORD	*cbSqlStrMax*	Input	Maximum length of the *szSqlStr* buffer.
SDWORD FAR *	*pcbSqlStr*	Output	The total number of bytes (excluding the null termination byte) available to return in *szSqlStr*. If the number of bytes available to return is greater than or equal to *cbSqlStrMax*, the translated SQL string in *szSqlStr* is truncated to *cbSqlStrMax* - 1 bytes.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLNativeSql** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLNativeSql** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The buffer *szSqlStr* was not large enough to return the entire SQL string, so the SQL string was truncated. The argument *pcbSqlStr* contains the length of the untruncated SQL string. (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection not open	The *hdbc* was not in a connected state.
37000	Syntax error or access violation	The argument *szSqlStrIn* contained an SQL statement that was not preparable or contained a syntax error.
IM001	Driver does not support this function	(DM) The driver associated with the *hdbc* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was

		defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1009	Invalid argument value	(DM) The argument *szSqlStrIn* was a null pointer.
S1090	Invalid string or buffer length	(DM) The argument *cbSqlStrIn* was less than 0, but not equal to SQL_NTS. (DM) The argument *cbSqlStrMax* was less than 0 and the argument *szSqlStr* was not a null pointer.

Comments

The following are examples of what **SQLNativeSql** might return for the following input SQL string containing the scalar function CONVERT. Assume that the column empid is of type INTEGER in the data source:

```
SELECT { fn CONVERT (empid, SQL_SMALLINT) } FROM employee
```

A driver for SQL Server might return the following translated SQL string:

```
SELECT convert (smallint, empid) FROM employee
```

A driver for ORACLE Server might return the following translated SQL string:

```
SELECT to_number (empid) FROM employee
```

A driver for Ingres might return the following translated SQL string:

```
SELECT int2 (empid) FROM employee
```

SQLNumParams (Extension Level 2, ODBC 1.0)

Related Functions

SQLNumParams returns the number of parameters in an SQL statement.

Syntax

RETCODE **SQLNumParams**(*hstmt*, *pcpar*)

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_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLNumParams** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLNumParams** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) The function was called prior to calling **SQLPrepare** or **SQLExecDirect** for the *hstmt*. (DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt*

		and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

SQLNumParams can only be called after **SQLPrepare** has been called.

If the statement associated with *hstmt* does not contain parameters, **SQLNumParams** sets *pcpar* to 0.

Related Functions

SQLDescribeParam (extension)
SQLBindParameter

SQLNumResultCols (Core, ODBC 1.0)

Related Functions

SQLNumResultCols returns the number of columns in a result set.

Syntax

RETCODE **SQLNumResultCols**(*hstmt*, *pccol*)

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_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLNumResultCols** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLNumResultCols** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) The function was called prior to calling **SQLPrepare** or **SQLExecDirect** for the *hstmt*. (DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt*

		and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

SQLNumResultCols can return any SQLSTATE that can be returned by **SQLPrepare** or **SQLExecute** when called after **SQLPrepare** and before **SQLExecute** depending on when the data source evaluates the SQL statement associated with the *hstmt*.

Comments

SQLNumResultCols can be called successfully only when the *hstmt* is in the prepared, executed, or positioned state.

If the statement associated with *hstmt* does not return columns, **SQLNumResultCols** sets *pccol* to 0.

Related Functions

SQLBindCol
SQLCancel
SQLColAttributes
SQLDescribeCol
SQLExtendedFetch (extension)
SQLFetch
SQLGetData (extension)
SQLSetScrollOptions (extension)

SQLParamData (Extension Level 1, ODBC 1.0)

Code Example Related Functions

SQLParamData is used in conjunction with **SQLPutData** to supply parameter data at statement execution time.

Syntax

RETCODE **SQLParamData**(*hstmt*, *prgbValue*)

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

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.
PTR FAR *	*prgbValue*	Output	Pointer to storage for the value specified for the *rgbValue* argument in **SQLBindParameter** (for parameter data) or the address of the *rgbValue* buffer specified in **SQLBindCol** (for column data).

Returns

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

Diagnostics

When **SQLParamData** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLParamData** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
22026	String data, length mismatch	The SQL_NEED_LONG_DATA_LEN information type in **SQLGetInfo** was "Y" and less data was sent for a long parameter (the data type was SQL_LONGVARCHAR, SQL_LONGVARBINARY, or a long, data source-specific data type) than was specified with the *pcbValue* argument in **SQLBindParameter**. The SQL_NEED_LONG_DATA_LEN information type in **SQLGetInfo** was "Y" and less data was sent for a long column (the data type was SQL_LONGVARCHAR, SQL_LONGVARBINARY, or a long, data source-specific data type) than was specified in the length buffer corresponding to a column in a row of data that was added or updated with **SQLSetPos**.

IM001	Driver does not support this function	(DM) The driver that corresponds the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application. **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. **SQLCancel** was called before data was sent for all data-at-execution parameters or columns.
S1010	Function sequence error	(DM) The previous function call was not a call to **SQLExecDirect**, **SQLExecute**, or **SQLSetPos** where the return code was SQL_NEED_DATA. The previous function call was a call to **SQLParamData**. (DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called.
S1T00	Timeout expired	The timeout period expired before the data source completed processing the parameter value. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

If **SQLParamData** is called while sending data for a parameter in an SQL statement, it can return any SQLSTATE that can be returned by the function called to execute the statement (**SQLExecute** or **SQLExecDirect**). If it is called while sending data for a column being updated or added with **SQLSetPos**, it can return any SQLSTATE that can be returned by **SQLSetPos**.

Comments

For an explanation of how data-at-execution parameter data is passed at statement execution time, see "Passing Parameter Values" in **SQLBindParameter**. For an explanation of how data-at-execution column data is updated or added, see "Using SQLSetPos" in **SQLSetPos**.

Code Example

See **SQLPutData**.

Related Functions

SQLCancel
SQLDescribeParam (extension)
SQLExecDirect
SQLExecute
SQLPutData (extension)
SQLBindParameter

SQLParamOptions (Extension Level 2, ODBC 1.0)

Code Example Related Functions

SQLParamOptions allows an application to specify multiple values for the set of parameters assigned by **SQLBindParameter**. 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	Number of values for each parameter. If *crow* is greater than 1, the *rgbValue* argument in **SQLBindParameter** points to an array of parameter values and *pcbValue* points to an array of lengths.
UDWORD FAR *	*pirow*	Input	Pointer to storage for the current row number. As each row of parameter values is processed, *pirow* is set to the number of that row. No row number will be returned if *pirow* is set to a null pointer.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLParamOptions** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLParamOptions** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1010	Function sequence error	(DM) An asynchronously executing function was called for the *hstmt* and was still executing when this function

		was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1107	Row value out of range	(DM) The value specified for the argument *crow* was equal to 0.

Comments

As a statement executes, the driver sets *pirow* to the number of the current row of parameter values; the first row is row number 1. The contents of *pirow* can be used as follows:

- When **SQLParamData** returns SQL_NEED_DATA for data-at-execution parameters, the application can access the value in *pirow* to determine which row of parameters is being executed.
- When **SQLExecute** or **SQLExecDirect** returns an error, the application can access the value in *pirow* to find out which row of parameters failed.
- When **SQLExecute**, **SQLExecDirect**, **SQLParamData**, or **SQLPutData** succeed, the value in *pirow* is set to *crow* – the total number of rows of parameters processed.

▪

Code Example

In the following example, an application specifies an array of parameter values with **SQLBindParameter** and **SQLParamOptions**. It then inserts those values into a table with a single **INSERT** statement and checks for any errors. If the first row fails, the application rolls back all changes. If any other row fails, the application commits the transaction, skips the failed row, rebinds the remaining parameters, and continues processing. (Note that **irow** is 1-based and **szData[]** is 0-based, so the **irow** entry of **szData[]** is skipped by rebinding at **szData[irow]**.)

```
#define CITY_LEN 256
SDWORD cbValue[ ] = {SQL_NTS, SQL_NTS, SQL_NTS, SQL_NTS, SQL_NTS};
UCHAR  szData[ ][CITY_LEN] = {"Boston","New York","Keokuk","Seattle",
"Eugene"};
UDWORD irow;
SQLSetConnectOption(hdbc, SQL_AUTOCOMMIT, 0);
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_DEFAULT, SQL_CHAR,
CITY_LEN, 0, szData, 0, cbValue);
SQLPrepare(hstmt, "INSERT INTO CITIES VALUES (?)", SQL_NTS);
SQLParamOptions(hstmt, 5, &irow);

while (TRUE) {

  retcode = SQLExecute(hstmt);

  /* Done if execution was successful */

  if (retcode != SQL_ERROR) {
    break;
  }

  /* On an error, print the error.  If the error is in row 1, roll */
  /* back the transaction and quit.  If the error is in another    */
  /* row, commit the transaction and, unless the error is in the   */
  /* last row, rebind to the next row and continue processing.     */

  show_error();
  if (irow == 1) {
    SQLTransact(henv, hstmt, SQL_ROLLBACK);
    break;
  } else {
    SQLTransact(henv, hstmt, SQL_COMMIT);
    if (irow == 5) {
      break;
    } else {
      SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_DEFAULT, SQL_CHAR,
CITY_LEN, 0, szData[irow], 0, cbValue[irow]);
      SQLParamOptions(hstmt, 5-irow, &irow);
```

```
        }
    }
}
```

Related Functions

SQLDescribeParam (extension)
SQLBindParameter

SQLPrepare (Core, ODBC 1.0)

Code Example Related Functions

SQLPrepare prepares an SQL string for execution.

Syntax

RETCODE **SQLPrepare**(*hstmt*, *szSqlStr*, *cbSqlStr*)

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_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLPrepare** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLPrepare** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
21S01	Insert value list does not match column list	The argument *szSqlStr* contained an **INSERT** statement and the number of values to be inserted did not match the degree of the derived table.
21S02	Degree of derived table does not match column list	The argument *szSqlStr* contained a **CREATE VIEW** statement and the number of names specified is not the same degree as the derived table defined by the query specification.
22005	Error in assignment	The argument *szSqlStr* contained an SQL statement that contained a literal or parameter and the value was incompatible with the data type of the associated table column.
24000	Invalid cursor state	(DM) A cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch** had been called. A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called.
34000	Invalid cursor name	The argument *szSqlStr* contained a positioned **DELETE** or a positioned **UPDATE** and the cursor referenced by the statement being prepared was not open.

37000	Syntax error or access violation	The argument *szSqlStr* contained an SQL statement that was not preparable or contained a syntax error.
42000	Syntax error or access violation	The argument *szSqlStr* contained a statement for which the user did not have the required privileges.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S0001	Base table or view already exists	The argument *szSqlStr* contained a **CREATE TABLE** or **CREATE VIEW** statement and the table name or view name specified already exists.
S0002	Base table not found	The argument *szSqlStr* contained a **DROP TABLE** or a **DROP VIEW** statement and the specified table name or view name did not exist.
		The argument *szSqlStr* contained an **ALTER TABLE** statement and the specified table name did not exist.
		The argument *szSqlStr* contained a **CREATE VIEW** statement and a table name or view name defined by the query specification did not exist.
		The argument *szSqlStr* contained a **CREATE INDEX** statement and the specified table name did not exist.
		The argument *szSqlStr* contained a **GRANT** or **REVOKE** statement and the specified table name or view name did not exist.
		The argument *szSqlStr* contained a **SELECT** statement and a specified table name or view name did not exist.
		The argument *szSqlStr* contained a **DELETE**, **INSERT**, or **UPDATE** statement and the specified table name did not exist.
		The argument *szSqlStr* contained a **CREATE TABLE** statement and a table specified in a constraint (referencing a table other than the one being created) did not exist.
S0011	Index already exists	The argument *szSqlStr* contained a **CREATE INDEX** statement and the specified index name already existed.
S0012	Index not found	The argument *szSqlStr* contained a **DROP INDEX** statement and the specified index name did not exist.
S0021	Column already exists	The argument *szSqlStr* contained an **ALTER TABLE** statement and the column specified in the **ADD** clause is not unique or identifies an existing column in the base table.
S0022	Column not found	The argument *szSqlStr* contained a **CREATE INDEX** statement and one or more of the column names specified in

		the column list did not exist. The argument *szSqlStr* contained a **GRANT** or **REVOKE** statement and a specified column name did not exist. The argument *szSqlStr* contained a **SELECT**, **DELETE**, **INSERT**, or **UPDATE** statement and a specified column name did not exist. The argument *szSqlStr* contained a **CREATE TABLE** statement and a column specified in a constraint (referencing a table other than the one being created) did not exist.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1009	Invalid argument value	(DM) The argument *szSqlStr* was a null pointer.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The argument *cbSqlStr* was less than or equal to 0, but not equal to SQL_NTS.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

The application calls **SQLPrepare** to send an SQL statement to the data source for preparation. 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.

Note If an application uses **SQLPrepare** to prepare and **SQLExecute** to submit a **COMMIT** or **ROLLBACK** statement, it will not be interoperable between DBMS products. To commit or roll back a transaction, call **SQLTransact**.

The driver modifies the statement to use the form of SQL used by the data source, then submits it to the data source for preparation. In particular, the driver modifies the escape clauses used to define ODBC-specific SQL. For the driver, an *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.

Once a statement is prepared, the application uses *hstmt* to refer to the statement in later function calls. The prepared statement associated with the *hstmt* may be reexecuted by calling **SQLExecute** until the application frees the *hstmt* with a call to **SQLFreeStmt** with the SQL_DROP option or until the *hstmt* is used in a call to **SQLPrepare**, **SQLExecDirect**, or one of the catalog functions (**SQLColumns**, **SQLTables**, and so on). Once the application prepares a statement, it can request information about the format of the result set.

Some drivers cannot 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**.

Depending on the capabilities of the driver and data source and on whether the application has called **SQLBindParameter**, parameter information (such as data types) might be checked when the statement is prepared or when it is executed. For maximum interoperability, an application should unbind all parameters that applied to an old SQL statement before preparing a new SQL statement on the same *hstmt*. This prevents errors that are due to old parameter information being applied to the new statement.

Important Committing or rolling back a transaction, either by calling **SQLTransact** or by using the SQL_AUTOCOMMIT connection option, can cause the data source to delete the access plans for all *hstmts* on an *hdbc*. For more information, see the SQL_CURSOR_COMMIT_BEHAVIOR and SQL_CURSOR_ROLLBACK_BEHAVIOR information types in **SQLGetInfo**.

Code Example

See **SQLBindParameter**, **SQLParamOptions**, **SQLPutData**, and **SQLSetPos**.

Related Functions

SQLAllocStmt
SQLBindCol
SQLCancel
SQLExecDirect
SQLExecute
SQLRowCount
SQLSetCursorName
SQLBindParameter
SQLTransact

SQLPrimaryKeys (Extension Level 2, ODBC 1.0)

Code Example Related Functions

SQLPrimaryKeys returns the column names that comprise the primary key for a table. The driver returns the information as a result set. This function does not support returning primary keys from multiple tables in a single call.

Syntax

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

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

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.
UCHAR FAR *	*szTableQualifier*	Input	Qualifier name. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	*cbTableQualifier*	Input	Length of *szTableQualifier*.
UCHAR FAR *	*szTableOwner*	Input	Table owner. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.
SWORD	*cbTableOwner*	Input	Length of *szTableOwner.*
UCHAR FAR *	*szTableName*	Input	Table name.
SWORD	*cbTableName*	Input	Length of *szTableName*.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLPrimaryKeys** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLPrimaryKeys** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
24000	Invalid cursor state	(DM) A cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch** had been called. A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had

		not been called.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the name length arguments was less than 0, but not equal to SQL_NTS. The value of one of the name length arguments exceeded the maximum length value for the corresponding qualifier or name.
S1C00	Driver not capable	A table qualifier was specified and the driver or data source does not support qualifiers. A table owner was specified and the driver or data source does not support owners. The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Timeout expired	The timeout period expired before the data source returned the requested result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

SQLPrimaryKeys returns the results as a standard result set, ordered by TABLE_QUALIFIER, TABLE_OWNER, TABLE_NAME, and KEY_SEQ. The following table lists the columns in the result set.

Note **SQLPrimaryKeys** might not return all primary keys. For example, a Paradox driver might only return primary keys for files (tables) in the current directory.

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the TABLE_QUALIFIER, TABLE_OWNER, TABLE_NAME, and COLUMN_NAME columns, call **SQLGetInfo** with the SQL_MAX_QUALIFIER_NAME_LEN, SQL_MAX_OWNER_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN options.

Column Name	**Data Type**	**Comments**
TABLE_QUALIFIER	Varchar(128)	Primary key table qualifier identifier; NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have qualifiers.
TABLE_OWNER	Varchar(128)	Primary key table owner identifier; NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.
TABLE_NAME	Varchar(128) not NULL	Primary key table identifier.
COLUMN_NAME	Varchar(128) not NULL	Primary key column identifier.
KEY_SEQ	Smallint not NULL	Column sequence number in key (starting with 1).
PK_NAME	Varchar(128)	Primary key identifier. NULL if not applicable to the data source.

Note The PK_NAME column was added in ODBC 2.0. ODBC 1.0 drivers may return a different, driver-specific column with the same column number.

Code Example

See **SQLForeignKeys**.

Related Functions

SQLBindCol
SQLCancel
SQLExtendedFetch (extension)
SQLFetch
SQLForeignKeys (extension)
SQLStatistics (extension)

SQLProcedureColumns (Extension Level 2, ODBC 1.0)

Code Example Related Functions

SQLProcedureColumns returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures. The driver returns the information as a result set on the specified *hstmt*.

Syntax

RETCODE **SQLProcedureColumns**(*hstmt*, *szProcQualifier*, *cbProcQualifier*, *szProcOwner*, *cbProcOwner*, *szProcName*, *cbProcName*, *szColumnName*, *cbColumnName*)

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

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.
UCHAR FAR *	*szProcQualifier*	Input	Procedure qualifier name. If a driver supports qualifiers for some procedures but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those procedures that do not have qualifiers.
SWORD	*cbProcQualifier*	Input	Length of *szProcQualifier*.
UCHAR FAR *	*szProcOwner*	Input	String search pattern for procedure owner names. If a driver supports owners for some procedures but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those procedures that do not have owners.
SWORD	*cbProcOwner*	Input	Length of *szProcOwner*.
UCHAR FAR *	*szProcName*	Input	String search pattern for procedure names.
SWORD	*cbProcName*	Input	Length of *szProcName*.
UCHAR FAR *	*szColumnName*	Input	String search pattern for column names.
SWORD	*cbColumnName*	Input	Length of *szColumnName*.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLProcedureColumns** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLProcedureColumns** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link	The communication link between the driver and the data source to which the

	failure	driver was connected failed before the function completed processing.
24000	Invalid cursor state	(DM) A cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch** had been called. A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the name length arguments was less than 0, but not equal to SQL_NTS. The value of one of the name length arguments exceeded the maximum length value for the corresponding qualifier or name.
S1C00	Driver not capable	A procedure qualifier was specified and the driver or data source does not support qualifiers. A procedure owner was specified and the driver or data source does not support owners. A string search pattern was specified for the procedure owner, procedure name, or column name and the data source does not support search patterns for

		one or more of those arguments. The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

This function is typically used before statement execution to retrieve information about procedure parameters and columns from the data source's catalog

Note **SQLProcedureColumns** might not return all columns used by a procedure. For example, a driver might only return information about the parameters used by a procedure and not the columns in a result set it generates.

The *szProcOwner*, *szProcName*, and *szColumnName* arguments accept search patterns.

SQLProcedureColumns returns the results as a standard result set, ordered by PROCEDURE_QUALIFIER, PROCEDURE_OWNER, PROCEDURE_NAME, and COLUMN_TYPE. The following table lists the columns in the result set. Additional columns beyond column 13 (REMARKS) can be defined by the driver.

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the PROCEDURE_QUALIFIER, PROCEDURE_OWNER, PROCEDURE_NAME, and COLUMN_NAME columns, an application can call **SQLGetInfo** with the SQL_MAX_QUALIFIER_NAME_LEN, SQL_MAX_OWNER_NAME_LEN, SQL_MAX_PROCEDURE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN options.

Column Name	**Data Type**	**Comments**
PROCEDURE_QUALIFIER	Varchar(128)	Procedure qualifier identifier; NULL if not applicable to the data source. If a driver supports qualifiers for some procedures but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those procedures that do not have qualifiers.
PROCEDURE_OWNER	Varchar(128)	Procedure owner identifier; NULL if not applicable to the data source. If a driver supports owners for some procedures but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those procedures that do not have owners.
PROCEDURE_NAME	Varchar(128) not NULL	Procedure identifier.
COLUMN_NAME	Varchar(128) not NULL	Procedure column identifier.
COLUMN_TYPE	Smallint not NULL	Defines the procedure column as parameter or a result set column: SQL_PARAM_TYPE_UNKNOWN: The procedure column is a parameter whose type is unknown. (ODBC 1.0) SQL_PARAM_INPUT: The procedure

		column is an input parameter. (ODBC 1.0) SQL_PARAM_INPUT_OUTPUT: the procedure column is an input/output parameter. (ODBC 1.0) SQL_PARAM_OUTPUT: The procedure column is an output parameter. (ODBC 1.0) SQL_RETURN_VALUE: The procedure column is the return value of the procedure. (ODBC 2.0) SQL_RESULT_COL: The procedure column is a result set column. (ODBC 1.0)
DATA_TYPE	Smallint not NULL	SQL data type. This can be an ODBC SQL data type or a driver-specific SQL data type. For a list of valid ODBC SQL data types, see SQL Data Types. For information about driver-specific SQL data types, see the driver's documentation.
TYPE_NAME	Varchar(128) not NULL	Data source-dependent data type name; for example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINARY", or "CHAR ( ) FOR BIT DATA".
PRECISION	Integer	The precision of the procedure column on the data source. NULL is returned for data types where precision is not applicable. For more information concerning precision, see Precision, Scale, Length, and Display Size.
LENGTH	Integer	The length in bytes of data transferred on an **SQLGetData** or **SQLFetch** operation if SQL_C_DEFAULT is specified. For numeric data, this size may be different than the size of the data stored on the data source. For more information, see Precision, Scale, Length, and Display Size.
SCALE	Smallint	The scale of the procedure column on the data source. NULL is returned for data types where scale is not applicable. For more information concerning scale, see Precision, Scale, Length, and Display Size.
RADIX	Smallint	For numeric data types, either 10 or 2. If it is 10, the values in PRECISION and SCALE give the number of decimal digits allowed for the column. For example, a DECIMAL(12,5) column would return a RADIX of 10, a PRECISION of 12, and a SCALE of 5; a FLOAT column could return a RADIX of 10, a PRECISION of 15 and a SCALE of NULL.

		If it is 2, the values in PRECISION and SCALE give the number of bits allowed in the column. For example, a FLOAT column could return a RADIX of 2, a PRECISION of 53, and a SCALE of NULL. NULL is returned for data types where radix is not applicable.
NULLABLE	Smallint not NULL	Whether the procedure column accepts a NULL value: SQL_NO_NULLS: The procedure column does not accept NULL values. SQL_NULLABLE: The procedure column accepts NULL values. SQL_NULLABLE_UNKNOWN: It is not known if the procedure column accepts NULL values.
REMARKS	Varchar(254)	A description of the procedure column.

Code Example

See **SQLProcedures**.

Related Functions

SQLBindCol
SQLCancel
SQLExtendedFetch (extension)
SQLFetch
SQLProcedures (extension)

SQLProcedures (Extension Level 2, ODBC 1.0)

Code Example Related Functions

SQLProcedures returns the list of procedure names stored in a specific data source. *Procedure* is a generic term used to describe an *executable object*, or a named entity that can be invoked using input and output parameters, and which can return result sets similar to the results returned by SQL **SELECT** expressions.

Syntax

RETCODE **SQLProcedures**(*hstmt*, *szProcQualifier*, *cbProcQualifier*, *szProcOwner*, *cbProcOwner*, *szProcName*, *cbProcName*)

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

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.
UCHAR FAR *	*szProcQualifier*	Input	Procedure qualifier. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	*cbProcQualifier*	Input	Length of *szProcQualifier*.
UCHAR FAR *	*szProcOwner*	Input	String search pattern for procedure owner names. If a driver supports owners for some procedures but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those procedures that do not have owners.
SWORD	*cbProcOwner*	Input	Length of *szProcOwner*.
UCHAR FAR *	*szProcName*	Input	String search pattern for procedure names.
SWORD	*cbProcName*	Input	Length of *szProcName*.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLProcedures** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLProcedures** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
24000	Invalid cursor state	(DM) A cursor was open on the *hstmt*

		and **SQLFetch** or **SQLExtendedFetch** had been called. A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support this function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the name length arguments was less than 0, but not equal to SQL_NTS. The value of one of the name length argu-ments exceeded the maximum length value for the corresponding qualifier or name.
S1C00	Driver not capable	A procedure qualifier was specified and the driver or data source does not support qualifiers. A procedure owner was specified and the driver or data source does not support owners. A string search pattern was specified for the procedure owner or procedure name and the data source does not support search patterns for one or more of those arguments. The combination of the current settings of the SQL_CONCURRENCY and

		SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Timeout expired	The timeout period expired before the data source returned the requested result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

SQLProcedures lists all procedures in the requested range. A user may or may not have permission to execute any of these procedures. To check accessibility, an application can call **SQLGetInfo** and check the SQL_ACCESSIBLE_PROCEDURES information value. Otherwise, the application must be able to handle a situation where the user selects a procedure which it cannot execute.

Note **SQLProcedures** might not return all procedures. Applications can use any valid procedure, regardless of whether it is returned by **SQLProcedures**.

SQLProcedures returns the results as a standard result set, ordered by PROCEDURE_QUALIFIER, PROCEDURE_OWNER, and PROCEDURE_NAME. The following table lists the columns in the result set.

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the PROCEDURE_QUALIFIER, PROCEDURE_OWNER, and PROCEDURE_NAME columns, an application can call **SQLGetInfo** with the SQL_MAX_QUALIFIER_NAME_LEN, SQL_MAX_OWNER_NAME_LEN, and SQL_MAX_PROCEDURE_NAME_LEN options.

Column Name	**Data Type**	**Comments**
PROCEDURE_QUAL-IFIER	Varchar(128)	Procedure qualifier identifier; NULL if not applicable to the data source. If a driver supports qualifiers for some procedures but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those procedures that do not have qualifiers.
PROCEDURE_OWNER	Varchar(128)	Procedure owner identifier; NULL if not applicable to the data source. If a driver supports owners for some procedures but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those procedures that do not have owners.
PROCEDURE_NAME	Varchar(128) not NULL	Procedure identifier.
NUM_INPUT_PARAMS	N/A	Reserved for future use. Applications should not rely on the data returned in these result columns.
NUM_OUTPUT _PARAMS	N/A	Reserved for future use. Applications should not rely on the data returned in these result columns.
NUM_RESULT_SETS	N/A	Reserved for future use. Applications should not rely on the data returned in these result columns.
REMARKS	Varchar(254)	A description of the procedure.
PROCEDURE_TYPE	Smallint	Defines the procedure type: SQL_PT_UNKNOWN: It cannot be

		determined whether the procedure returns a value. SQL_PT_PROCEDURE: The returned object is a procedure; that is, it does not have a return value. SQL_PT_FUNCTION: The returned object is a function; that is, it has a return value.

Note The PROCEDURE_TYPE column was added in ODBC 2.0. ODBC 1.0 drivers might return a different, driver-specific column with the same column number.

The *szProcOwner* and *szProcName* arguments accept search patterns.

▪

Code Example

In this example, an application uses the procedure **AddEmployee** to insert data into the EMPLOYEE table. The procedure contains input parameters for NAME, AGE, and BIRTHDAY columns. It also contains one output parameter that returns a remark about the new employee. The example also shows the use of a return value from a stored procedure. For the return value and each parameter in the procedure, the application calls **SQLBindParameter** to specify the ODBC C data type and the SQL data type of the parameter and to specify the storage location and length of the parameter. The application assigns data values to the storage locations for each parameter and calls **SQLExecDirect** to execute the procedure. If **SQLExecDirect** returns SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the return value and the value of each output or input/output parameter is automatically put into the storage location defined for the parameter in **SQLBindParameter**.

```
#define NAME_LEN 30
#define REM_LEN 128

UCHAR       szName[NAME_LEN], szRemark[REM_LEN];
SWORD       sAge, sEmpId;
SDWORD      cbEmpId, cbName, cbAge = 0, cbBirthday = 0, cbRemark;
DATE_STRUCT dsBirthday;

/* Define parameter for return value (Employee ID) from procedure. */

SQLBindParameter(hstmt, 1, SQL_PARAM_OUTPUT, SQL_C_SLONG, SQL_INTEGER, 0, 0,
&sEmpId, 0, &cbEmpId);

/* Define data types and storage locations for Name, Age, Birthday */
/* input parameter data.                                           */

SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, NAME_LEN,
0, szName, 0, &cbName);
SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_SSHORT, SQL_SMALLINT, 0, 0,
&sAge, 0, &cbAge);
SQLBindParameter(hstmt, 4, SQL_PARAM_INPUT, SQL_C_DATE, SQL_DATE, 0, 0,
&dsBirthday, 0, &cbBirthday);

/* Define data types and storage location for Remark output parameter */

SQLBindParameter(hstmt, 5, SQL_PARAM_OUTPUT, SQL_C_CHAR, SQL_CHAR, REM_LEN,
0, szRemark, REM_LEN, &cbRemark);

strcpy(szName, "Smith, John D.");    /* Specify first row of */
sAge = 40;                           /* parameter data.      */
dsBirthday.year = 1952;
dsBirthday.month = 2;
dsBirthday.day = 29;
cbName = SQL_NTS;
```

```
/* Execute procedure with first row of data. After the procedure */
/* is executed, sEmpId and szRemark will have the values         */
/* returned by AddEmployee.                                      */

retcode = SQLExecDirect(hstmt, "{?=call AddEmployee(?,?,?,?)}",SQL_NTS);

strcpy(szName, "Jones, Bob K.");     /* Specify second row of */
sAge = 52;                           /* parameter data        */
dsBirthday.year = 1940;
dsBirthday.month = 3;
dsBirthday.day = 31;

/* Execute procedure with second row of data. After the procedure */
/* is executed, sEmpId and szRemark will have the new values      */
/* returned by AddEmployee.                                       */

retcode = SQLExecDirect(hstmt, "{?=call AddEmployee(?,?,?,?)}", SQL_NTS);
```

Related Functions

SQLBindCol
SQLCancel
SQLExtendedFetch (extension)
SQLFetch
SQLGetInfo (extension)
SQLProcedureColumns (extension)

SQLPutData (Extension Level 1, ODBC 1.0)

Code Example Related Functions

SQLPutData allows an application to send data for a parameter or column to the driver at statement execution time. This function can be used to send character or binary data values in parts to a column with a character, binary, or data source-specific data type (for example, parameters of the SQL_LONGVARBINARY or SQL_LONGVARCHAR types).

Syntax

RETCODE **SQLPutData**(*hstmt*, *rgbValue*, *cbValue*)

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

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.
PTR	*rgbValue*	Input	Pointer to storage for the actual data for the parameter or column. The data must use the C data type specified in the *fCType* argument of **SQLBindParameter** (for parameter data) or **SQLBindCol** (for column data).
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 or column. *cbValue* is ignored unless it is SQL_NTS, SQL_NULL_DATA, or SQL_DEFAULT_PARAM; the C data type specified in **SQLBindParameter** or **SQLBindCol** is SQL_C_CHAR or SQL_C_BINARY; or the C data type is SQL_C_DEFAULT and the default C data type for the specified SQL data type is SQL_C_CHAR or SQL_C_BINARY. For all other types of C data, if *cbValue* is not SQL_NULL_DATA or SQL_DEFAULT_PARAM, the driver assumes that the size of *rgbValue* is the size of the C data type specified with *fCType* and sends the entire data value. For more information, see Converting Data from C to SQL Data Types.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLPutData** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLPutData** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	**Error**	**Description**
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The data sent for a character or binary parameter or column in one or more calls to **SQLPutData** exceeded the maximum length of the associated character or binary column. The fractional part of the data sent for a numeric or bit parameter or column was truncated. Timestamp data sent for a date or time parameter or column was truncated.
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
22001	String data right truncation	The SQL_NEED_LONG_DATA_LEN information type in **SQLGetInfo** was "Y" and more data was sent for a long parameter (the data type was SQL_LONGVARCHAR, SQL_LONGVARBINARY, or a long, data source-specific data type) than was specified with the *pcbValue* argument in **SQLBindParameter**. The SQL_NEED_LONG_DATA_LEN information type in **SQLGetInfo** was "Y" and more data was sent for a long column (the data type was SQL_LONGVARCHAR, SQL_LONGVARBINARY, or a long, data source-specific data type) than was specified in the length buffer corresponding to a column in a row of data that was added or updated with **SQLSetPos**.
22003	Numeric value out of range	**SQLPutData** was called more than once for a parameter or column and it was not being used to send character C data to a column with a character, binary, or data source-specific data type or to send binary C data to a column with a character, binary, or data source-specific data type. The data sent for a numeric parameter or column caused the whole (as opposed to fractional) part of the number to be truncated when assigned to the associated table column.

22005	Error in assignment	The data sent for a parameter or column was incompatible with the data type of the associated table column.
22008	Datetime field overflow	The data sent for a date, time, or timestamp parameter or column was, respectively, an invalid date, time, or timestamp.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application. **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. **SQLCancel** was called before data was sent for all data-at-execution parameters or columns.
S1009	Invalid argument value	(DM) The argument *rgbValue* was a null pointer and the argument *cbValue* was not 0, SQL_DEFAULT_PARAM, or SQL_NULL_DATA.
S1010	Function sequence error	(DM) The previous function call was not a call to **SQLPutData**. The previous function call was a call to **SQLExecDirect**, **SQLExecute**, or **SQLSetPos** where the return code was SQL_NEED_DATA. (DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called.
S1090	Invalid string or buffer length	The argument *rgbValue* was not a null pointer and the argument *cbValue* was less than 0, but not equal to SQL_NTS or SQL_NULL_DATA.
S1T00	Timeout expired	The timeout period expired before the data source completed processing the parameter value. The timeout period is set through **SQLSetStmtOption**,

SQL_QUERY_TIMEOUT.

Comments

For an explanation of how data-at-execution parameter data is passed at statement execution time, see "Passing Parameter Values" in **SQLBindParameter**. For an explanation of how data-at-execution column data is updated or added, see "Using SQLSetPos" in **SQLSetPos**.

Note An application can use **SQLPutData** to send data in parts only when sending character C data to a column with a character, binary, or data source-specific data type or when sending binary C data to a column with a character, binary, or data source-specific data type. If **SQLPutData** is called more than once under any other conditions, it returns SQL_ERROR and SQLSTATE 22003 (Numeric value out of range).

▪

Code Example

In the following example, an application prepares an SQL statement to insert data into the EMPLOYEE table. The statement contains parameters for the NAME, ID, and PHOTO columns. For each parameter, the application calls **SQLBindParameter** to specify the C and SQL data types of the parameter. It also specifies that the data for the first and third parameters will be passed at execution time, and passes the values 1 and 3 for later retrieval by **SQLParamData**. These values will identify which parameter is being processed.

The application calls **GetNextID** to get the next available employee ID number. It then calls **SQLExecute** to execute the statement. **SQLExecute** returns SQL_NEED_DATA when it needs data for the first and third parameters. The application calls **SQLParamData** to retrieve the value it stored with **SQLBindParameter**; it uses this value to determine which parameter to send data for. For each parameter, the application calls **InitUserData** to initialize the data routine. It repeatedly calls **GetUserData** and **SQLPutData** to get and send the parameter data. Finally, it calls **SQLParamData** to indicate it has sent all the data for the parameter and to retrieve the value for the next parameter. After data has been sent for both parameters, **SQLParamData** returns SQL_SUCCESS.

For the first parameter, **InitUserData** does not do anything and **GetUserData** calls a routine to prompt the user for the employee name. For the third parameter, **InitUserData** calls a routine to prompt the user for the name of a file containing a bitmap photo of the employee and opens the file. **GetUserData** retrieves the next MAX_DATA_LEN bytes of photo data from the file. After it has retrieved all the photo data, it closes the photo file.

Note that some application routines are omitted for clarity.

```
#define NAME_LEN 30
#define MAX_DATA_LEN 1024
SDWORD  cbNameParam, cbID = 0; cbPhotoParam, cbData;
SWORD   sID;
PTR     pToken, InitValue;
UCHAR   Data[MAX_DATA_LEN];

retcode = SQLPrepare(hstmt, "INSERT INTO EMPLOYEE (NAME, ID, PHOTO) VALUES
(?, ?, ?)", SQL_NTS);
if (retcode == SQL_SUCCESS) {

  /* Bind the parameters. For parameters 1 and 3, pass the       */
  /* parameter number in rgbValue instead of a buffer address.   */

  SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, NAME_LEN,
0, 1, 0, &cbNameParam);
  SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_SSHORT, SQL_SMALLINT, 0,
0, &sID, 0, &cbID);
  SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
SQL_LONGVARBINARY, 0, 0, 3, 0, &cbPhotoParam);

  /* Set values so data for parameters 1 and 3 will be passed  */
  /* at execution. Note that the length parameter in the macro */
  /* SQL_LEN_DATA_AT_EXEC is 0. This assumes that the driver   */
  /* returns "N" for the SQL_NEED_LONG_DATA_LEN information    */
  /* type in SQLGetInfo.                                       */
  cbNameParam = cbPhotoParam = SQL_LEN_DATA_AT_EXEC(0);

  sID = GetNextID();  /* Get next available employee ID number. */

  retcode = SQLExecute(hstmt);

  /* For data-at-execution parameters, call SQLParamData to get the */
  /* parameter number set by SQLBindParameter. Call InitUserData.   */
  /* Call GetUserData and SQLPutData repeatedly to get and put all  */
  /* data for the parameter. Call SQLParamData to finish processing */
  /* this parameter and start processing the next parameter.        */

  while (retcode == SQL_NEED_DATA) {
    retcode = SQLParamData(hstmt, &pToken);
    if (retcode == SQL_NEED_DATA) {
      InitUserData((SWORD)pToken, InitValue);
      while (GetUserData(InitValue, (SWORD)pToken, Data, &cbData))
        SQLPutData(hstmt, Data, cbData);
    }
  }
}

VOID InitUserData(sParam, InitValue)
SWORD sParam;
PTR   InitValue;
{
UCHAR  szPhotoFile[MAX_FILE_NAME_LEN];
switch sParam {
  case 3:

    /* Prompt user for bitmap file containing employee photo.    */
    /* OpenPhotoFile opens the file and returns the file handle. */

    PromptPhotoFileName(szPhotoFile);
    OpenPhotoFile(szPhotoFile, (FILE *)InitValue);
    break;
}
}

BOOL GetUserData(InitValue, sParam, Data, cbData)
PTR    InitValue;
SWORD  sParam;
UCHAR  *Data;
SDWORD *cbData;
```

```
{

switch sParam {
  case 1:
    /* Prompt user for employee name. */

    PromptEmployeeName(Data);
    *cbData = SQL_NTS;
    return (TRUE);

  case 3:
    /* GetNextPhotoData returns the next piece of photo data and  */
    /* the number of bytes of data returned (up to MAX_DATA_LEN). */

    Done = GetNextPhotoData((FILE *)InitValue, Data, MAX_DATA_LEN, &cbData);
    if (Done) {
      ClosePhotoFile((FILE *)InitValue);
      return (TRUE);
    }
    return (FALSE);
}
return (FALSE);
}
```

Related Functions

- **SQLCancel**
- **SQLExecDirect**
- **SQLExecute**
- **SQLParamData** (extension)
- **SQLBindParameter**

SQLRowCount (Core, ODBC 1.0)

Related Functions

SQLRowCount returns the number of rows affected by an **UPDATE**, **INSERT**, or **DELETE** statement or by a SQL_UPDATE, SQL_ADD, or SQL_DELETE operation in **SQLSetPos**.

Syntax

RETCODE **SQLRowCount**(*hstmt*, *pcrow*)

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

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.
SDWORD FAR *	*pcrow*	Output	For **UPDATE**, **INSERT**, and **DELETE** statements and for the SQL_UPDATE, SQL_ADD, and SQL_DELETE operations in **SQLSetPos**, *pcrow* is the number of rows affected by the request or -1 if the number of affected rows is not available. For other statements and functions, the driver may define the value of *pcrow*. For example, some data sources may be able to return the number of rows returned by a **SELECT** statement or a catalog function before fetching the rows. **Note** Many data sources cannot return the number of rows in a result set before fetching them; for maximum interoperability, applications should not rely on this behavior.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLRowCount** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLRowCount** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was

		defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1010	Function sequence error	(DM) The function was called prior to calling **SQLExecute**, **SQLExecDirect**, **SQLSetPos** for the *hstmt*. (DM) An asynchronously executing function was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.

Comments

If the last executed statement associated with *hstmt* was not an **UPDATE**, **INSERT**, or **DELETE** statement, or if the *fOption* argument in the previous call to **SQLSetPos** was not SQL_UPDATE, SQL_ADD, or SQL_DELETE, the value of *pcrow* is driver-defined.

Related Functions

<u>SQLExecDirect</u>

<u>SQLExecute</u>

SQLSetConnectOption (Extension Level 1, ODBC 1.0)

Code Example Related Function

SQLSetConnectOption sets options that govern aspects of connections.

Syntax

RETCODE **SQLSetConnectOption**(*hdbc*, *fOption*, *vParam*)

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

Type	**Argument**	**Use**	**Description**
HDBC	*hdbc*	Input	Connection handle.
UWORD	*fOption*	Input	Option to set, listed in "Comments."
UDWORD	*vParam*	Input	Value associated with *fOption*. Depending on the value of *fOption*, *vParam* will be a 32-bit integer value or point to a null-terminated character string.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLSetConnectOption** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLSetConnectOption** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

The driver can return SQL_SUCCESS_WITH_INFO to provide information about the result of setting an option. For example, setting SQL_ACCESS_MODE to read-only 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.

SQLSTATE	**Error**	**Description**
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01S02	Option value changed	The driver did not support the specified value of the *vParam* argument and substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
08002	Connection in use	The argument *fOption* was SQL_ODBC_CURSORS and the driver was already connected to the data source.
08003	Connection not open	An *fOption* value was specified that required an open connection, but the *hdbc* was not in a connected state.
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
IM001	Driver does not support this function	(DM) The driver associated with the *hdbc* does not support the function.
IM009	Unable to load translation DLL	The driver was unable to load the translation DLL that was specified for the connection. This error can only be returned when *fOption* is

		SQL_TRANSLATE_DLL.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1009	Invalid argument value	Given the specified *fOption* value, an invalid value was specified for the argument *vParam*. (The Driver Manager returns this SQLSTATE only for connection and statement options that accept a discrete set of values, such as SQL_ACCESS_MODE or SQL_ASYNC_ENABLE. For all other connection and statement options, the driver must verify the value of the argument *vParam*.)
S1010	Function sequence error	(DM) An asynchronously executing function was called for an *hstmt* associated with the *hdbc* and was still executing when **SQLSetConnectOption** was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for an *hstmt* associated with the *hdbc* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns. (DM) **SQLBrowseConnect** was called for the *hdbc* and returned SQL_NEED_DATA. This function was called before **SQLBrowseConnect** returned SQL_SUCCESS_WITH_INFO or SQL_SUCCESS.
S1011	Operation invalid at this time	The argument *fOption* was SQL_TXN_ISOLATION and a transaction was open.
S1092	Option type out of range	(DM) The value specified for the argument *fOption* was in the block of numbers reserved for ODBC connection and statement options, but was not valid for the version of ODBC supported by the driver.
S1C00	Driver not capable	The value specified for the argument *fOption* was a valid ODBC connection or statement option for the version of ODBC supported by the driver, but was not supported by the driver. The value specified for the argument *fOption* was in the block of numbers reserved for driver-specific connection and statement options, but was not supported by the driver.

When *fOption* is a statement option, **SQLSetConnectOption** can return any SQLSTATEs returned by **SQLSetStmtOption**.

Comments

The currently defined options and the version of ODBC in which they were introduced are shown below; it is expected that more will be defined to take advantage of different data sources. Options from 0 to 999 are reserved by ODBC; driver developers must reserve values greater than or equal to SQL_CONNECT_OPT_DRVR_START for driver-specific use.

An application can call **SQLSetConnectOption** and include a statement option. The driver sets the statement option for any *hstmts* associated with the specified *hdbc* and establishes the statement option as a default for any *hstmts* later allocated for that *hdbc*. For a list of statement options, see **SQLSetStmtOption**.

All connection and statement options successfully set by the application for the *hdbc* persist until **SQLFreeConnect** is called on the *hdbc*. For example, if an application calls **SQLSetConnectOption** before connecting to a data source, the option persists even if **SQLSetConnectOption** fails in the driver when the application connects to the data source; if an application sets a driver-specific option, the option persists even if the application connects to a different driver on the *hdbc*.

Some connection and statement options support substitution of a similar value if the data source does not support the specified value of *vParam*. In such cases, the driver returns SQL_SUCCESS_WITH_INFO and SQLSTATE 01S02 (Option value changed). For example, if *fOption* is SQL_PACKET_SIZE and *vParam* exceeds the maximum packet size, the driver substitutes the maximum size. To determine the substituted value, an application calls **SQLGetConnectOption** (for connection options) or **SQLGetStmtOption** (for statement options).

The format of information set through *vParam* depends on the specified *fOption*. **SQLSetConnectOption** will accept option information in one of two different formats: a null-terminated character string or a 32-bit integer value. The format of each is noted in the option's description. Character strings pointed to by the *vParam* argument of **SQLSetConnectOption** have a maximum length of SQL_MAX_OPTION_STRING_LENGTH bytes (excluding the null termination byte).

fOption	***vParam* Contents**
SQL_ACCESS_MODE (ODBC 1.0)	A 32-bit integer value. SQL_MODE_READ_ONLY is used by the driver or data source as an indicator that the connection is not required to support SQL statements that cause updates to occur. This mode can be used to optimize locking strategies, transaction management, or other areas as appropriate to the driver or data source. The driver is not required to prevent such statements from being submitted to the data source. The behavior of the driver and data source when asked to process SQL statements that are not read-only during a read-only connection is implementation defined. SQL_MODE_READ_WRITE is the default.
SQL_AUTOCOMMIT (ODBC 1.0)	A 32-bit integer value that specifies whether to use auto-commit or manual-commit mode: SQL_AUTOCOMMIT_OFF = The driver uses manual-commit mode, and the application must explicitly commit or roll back transactions with **SQLTransact**. SQL_AUTOCOMMIT_ON = The driver uses auto-commit mode. Each statement is committed immediately after it is executed. This is the default. Note that changing from manual-commit mode to auto-commit mode commits any open transactions on the connection. **Important** Some data sources delete the access plans and close the cursors for all *hstmts* on an *hdbc* each time a statement is committed; autocommit mode can cause this to happen after each statement is executed. For more information, see the SQL_CURSOR_COMMIT_BEHAVIOR and SQL_CURSOR_ROLLBACK_BEHAVIOR information

	types in **SQLGetInfo**.
SQL_CURRENT_QUALIFIER (ODBC 2.0)	A null-terminated character string containing the name of the qualifier to be used by the data source. For example, in SQL Server, the qualifier is a database, so the driver sends a **USE** *database* statement to the data source, where *database* is the database specified in *vParam*. For a single-tier driver, the qualifier might be a directory, so the driver changes its current directory to the directory specified in *vParam*.
SQL_LOGIN_TIMEOUT (ODBC 1.0)	A 32-bit integer value corresponding to the number of seconds to wait for a login request to complete before returning to the application. The default is driver-dependent and must be nonzero. If *vParam* is 0, the timeout is disabled and a connection attempt will wait indefinitely. If the specified timeout exceeds the maximum login timeout in the data source, the driver substitutes that value and returns SQLSTATE 01S02 (Option value changed).
SQL_ODBC_CURSORS (ODBC 2.0)	A 32-bit option specifying how the Driver Manager uses the ODBC cursor library: SQL_CUR_USE_IF_NEEDED = The Driver Manager uses the ODBC cursor library only if it is needed. If the driver supports the SQL_FETCH_PRIOR option in **SQLExtendedFetch**, the Driver Manager uses the scrolling capabilities of the driver. Otherwise, it uses the ODBC cursor library. SQL_CUR_USE_ODBC = The Driver Manager uses the ODBC cursor library. SQL_CUR_USE_DRIVER = The Driver Manager uses the scrolling capabilities of the driver. This is the default setting.
SQL_OPT_TRACE (ODBC 1.0)	A 32-bit integer value telling the Driver Manager whether to perform tracing: SQL_OPT_TRACE_OFF = Tracing off (the default) SQL_OPT_TRACE_ON = Tracing on When tracing is on, the Driver Manager writes each ODBC function call to the trace file. On Windows and WOW, the Driver Manager writes to the trace file each time any application calls a function. On Windows NT, the Driver Manager writes to the trace file only for the application that turned tracing on. **Note** When tracing is on, the Driver Manager can return SQLSTATE IM013 (Trace file error) from any function. An application specifies a trace file with the SQL_OPT_TRACEFILE option. If the file already exists, the Driver Manager appends to the file. Otherwise, it creates the file. If tracing is on and no trace file has been specified, the Driver Manager writes to the file \SQL.LOG. On Windows NT, tracing should only be used for a single application or each application should specify a different trace file. Otherwise, two or more applications will attempt to open the same trace file at the same time, causing an error. If the **Trace** keyword in the [ODBC] section of the ODBC.INI file (or registry) is set to 1 when an application calls **SQLAllocEnv**, tracing is enabled. On Windows and WOW, it is enabled for all applications; on Windows NT it is enabled

only for the application that called **SQLAllocEnv**.

SQL_OPT_TRACEFILE
(ODBC 1.0)

A null-terminated character string containing the name of the trace file.

The default value of the SQL_OPT_TRACEFILE option is specified with the TraceFile keyname in the [ODBC] section of the ODBC.INI file (or registry).

SQL_PACKET_SIZE
(ODBC 2.0)

A 32-bit integer value specifying the network packet size in bytes.

Note Many data sources either do not support this option or can only return the network packet size.

If the specified size exceeds the maximum packet size or is smaller than the minimum packet size, the driver substitutes that value and returns SQLSTATE 01S02 (Option value changed).

SQL_QUIET_MODE
(ODBC 2.0)

A 32-bit window handle (*hwnd*).

If the window handle is a null pointer, the driver does not display any dialog boxes.

If the window handle is not a null pointer, it should be the parent window handle of the application. The driver uses this handle to display dialog boxes. This is the default.

If the application has not specified a parent window handle for this option, the driver uses a null parent window handle to display dialog boxes or return in **SQLGetConnectOption**.

Note The SQL_QUIET_MODE connection option does not apply to dialog boxes displayed by **SQLDriverConnect**.

SQL_TRANSLATE_DLL
(ODBC 1.0)

A null-terminated character string containing the name of a DLL containing the functions **SQLDriverToDataSource** and **SQLDataSourceToDriver** that the driver loads and uses to perform tasks such as character set translation. This option may only be specified if the driver has connected to the data source.

SQL_TRANSLATE_OPTION
(ODBC 1.0)

A 32-bit flag value that is passed to the translatation DLL. This option may only be specified if the driver has connected to the data source.

SQL_TXN_ISOLATION
(ODBC 1.0)

A 32-bit bitmask that sets the transaction isolation level for the current *hdbc*. An application must call **SQLTransact** to commit or roll back all open transactions on an *hdbc*, before calling **SQLSetConnectOption** with this option.

The valid values for *vParam* can be determined by calling **SQLGetInfo** with *fInfoType* equal to SQL_TXN_ISOLATION_OPTIONS. The following terms are used to define transaction isolation levels:

Dirty Read Transaction 1 changes a row. Transaction 2 reads the changed row before transaction 1 commits the change. If transaction 1 rolls back the change, transaction 2 will have read a row that is considered to have never existed.

Nonrepeatable Read Transaction 1 reads a row. Transaction 2 updates or deletes that row and commits this change. If transaction 1 attempts to reread the row, it will receive different row values or discover that the row has been deleted.

Phantom Transaction 1 reads a set of rows that satisfy some search criteria. Transaction 2 inserts a row that matches the search criteria. If transaction 1 reexecutes the statement that read the rows, it receives a different set of rows.

vParam must be one of the following values:

SQL_TXN_READ_UNCOMMITTED = Dirty reads, nonrepeatable reads, and phantoms are possible.

SQL_TXN_READ_COMMITTED = Dirty reads are not possible. Nonrepeatable reads and phantoms are possible.

SQL_TXN_REPEATABLE_READ = Dirty reads and nonrepeatable reads are not possible. Phantoms are possible.

SQL_TXN_SERIALIZABLE = Transactions are serializable. Dirty reads, nonrepeatable reads, and phantoms are not possible.

SQL_TXN_VERSIONING = Transactions are serializable, but higher concurrency is possible than with SQL_TXN_SERIALIZABLE. Dirty reads are not possible. Typically, SQL_TXN_SERIALIZABLE is implemented by using locking protocols that reduce concurrency and SQL_TXN_VERSIONING is implemented by using a non-locking protocol such as record versioning. Oracle's Read Consistency isolation level is an example of SQL_TXN_VERSIONING.

Data Translation

Data translation will be performed for all data flowing between the driver and the data source.

The translation option (set with the SQL_TRANSLATE_OPTION option) can be any 32-bit value. Its meaning depends on the translation DLL being used. A new option can be set at any time. The new option will be applied to the next exchange of data following the call to **SQLSetConnectOption**. A default translation DLL may be specified for the data source in its data source specification in the ODBC.INI file or registry. The default translation DLL is loaded by the driver at connection time. A translation option (SQL_TRANSLATE_OPTION) may be specified in the data source specification as well.

To change the translation DLL for a connection, an application calls **SQLSetConnectOption** with the SQL_TRANSLATE_DLL option after it has connected to the data source. The driver will attempt to load the specified DLL and, if the attempt fails, return SQL_ERROR with the SQLSTATE IM009 (Unable to load translation DLL).

If no translation DLL has been specified in the ODBC initialization file or by calling **SQLSetConnectOption**, the driver will not attempt to translate data. Any value set for the translation option will be ignored.

Code Example

See **SQLConnect** and **SQLParamOptions**.

Related Functions

SQLGetConnectOption (extension)
SQLGetStmtOption (extension)
SQLSetStmtOption (extension)

SQLSetCursorName (Core, ODBC 1.0)

Code Example Related Functions

SQLSetCursorName associates a cursor name with an active *hstmt*. If an application does not call **SQLSetCursorName**, the driver generates cursor names as needed for SQL statement processing.

Syntax

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

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_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLSetCursorName** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLSetCursorName** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
24000	Invalid cursor state	The statement corresponding to *hstmt* was already in an executed or cursor-positioned state.
34000	Invalid cursor name	The cursor name specified by the argument *szCursor* was invalid. For example, the cursor name exceeded the maximum length as defined by the driver.
3C000	Duplicate cursor name	The cursor name specified by the argument *szCursor* already exists.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1009	Invalid argument value	(DM) The argument *szCursor* was a null pointer.
S1010	Function sequence error	(DM) An asynchronously executing function was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt*

		and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The argument *cbCursor* was less than 0, but not equal to SQL_NTS.

Comments

The only ODBC SQL statements that use a cursor name are a positioned update and delete (for example, **UPDATE** *table-name* ...**WHERE CURRENT OF** *cursor-name*). If the application does not call **SQLSetCursorName** to define a cursor name, on execution of a **SELECT** statement the driver generates a name that begins with the letters SQL_CUR and does not exceed 18 characters in length.

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 no more than 18 characters.

A cursor name that is set either explicitly or implicitly remains set until the *hstmt* with which it is associated is dropped, using **SQLFreeStmt** with the SQL_DROP option.

▪

Code Example

In the following example, an application uses **SQLSetCursorName** to set a cursor name for an *hstmt*. It then uses that *hstmt* to retrieve results from the EMPLOYEE table. Finally, it performs a positioned update to change the name of 25-year-old John Smith to John D. Smith. Note that the application uses different *hstmts* for the **SELECT** and **UPDATE** statements.

For more code examples, see **SQLSetPos**.

```
#define NAME_LEN 30

HSTMT     hstmtSelect,
HSTMT     hstmtUpdate;
UCHAR     szName[NAME_LEN];
SWORD     sAge;
SDWORD    cbName;
SDWORD    cbAge;

/* Allocate the statements and set the cursor name */

SQLAllocStmt(hdbc, &hstmtSelect);
SQLAllocStmt(hdbc, &hstmtUpdate);
SQLSetCursorName(hstmtSelect, "C1", SQL_NTS);

/* SELECT the result set and bind its columns to local storage */

SQLExecDirect(hstmtSelect, "SELECT NAME, AGE FROM EMPLOYEE FOR UPDATE",
SQL_NTS);
SQLBindCol(hstmtSelect, 1, SQL_C_CHAR, szName, NAME_LEN, &cbName);
SQLBindCol(hstmtSelect, 2, SQL_C_SSHORT, &sAge, 0, &cbAge);

/* Read through the result set until the cursor is      */
/* positioned on the row for the 25-year-old John Smith */

do
   retcode = SQLFetch(hstmtSelect);
while ((retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) &&
(strcmp(szName, "Smith, John") != 0 || sAge != 25));

/* Perform a positioned update of John Smith's name */

if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
  SQLExecDirect(hstmtUpdate, "UPDATE EMPLOYEE SET NAME=\"Smith, John D.\"
WHERE CURRENT OF C1", SQL_NTS);
}
```

Related Functions

SQLExecDirect
SQLExecute
SQLGetCursorName
SQLSetScrollOptions (extension)

SQLSetParam (Deprecated, ODBC 1.0)

In ODBC 2.0, the ODBC 1.0 function **SQLSetParam** has been replaced by **SQLBindParameter**. For more information, see **SQLBindParameter**.

SQLSetPos (Extension Level 2, ODBC 1.0)

Code Example Related Functions

SQLSetPos sets the cursor position in a rowset and allows an application to refresh, update, delete, or add data to the rowset.

Syntax

RETCODE **SQLSetPos**(*hstmt*, *irow*, *fOption*, *fLock*)

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

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.
UWORD	*irow*	Input	Position of the row in the rowset on which to perform the operation specified with the *fOption* argument. If *irow* is 0, the operation applies to every row in the rowset. For additional information, see "Comments."
UWORD	*fOption*	Input	Operation to perform: SQL_POSITION SQL_REFRESH SQL_UPDATE SQL_DELETE SQL_ADD For more information, see "Comments."
UWORD	*fLock*	Input	Specifies how to lock the row after performing the operation specified in the *fOption* argument. SQL_LOCK_NO_CHANGE SQL_LOCK_EXCLUSIVE SQL_LOCK_UNLOCK For more information, see "Comments."

Returns

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

Diagnostics

When **SQLSetPos** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLSetPos** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated	The argument *fOption* was SQL_ADD or SQL_UPDATE and the value specified for a character or binary column exceeded the maximum length of the associated table column. (Function

		returns SQL_SUCCESS_WITH_INFO.) The argument *fOption* was SQL_ADD or SQL_UPDATE and the fractional part of the value specified for a numeric column was truncated. (Function returns SQL_SUCCESS_WITH_INFO.) The argument *fOption* was SQL_ADD or SQL_UPDATE and a timestamp value specified for a date or time column was truncated. (Function returns SQL_SUCCESS_WITH_INFO.)
01S01	Error in row	The *irow* argument was 0 and an error occurred in one or more rows while performing the operation specified with the *fOption* argument. (Function returns SQL_SUCCESS_WITH_INFO.)
01S03	No rows updated or deleted	The argument *fOption* was SQL_UPDATE or SQL_DELETE and no rows were updated or deleted. (Function returns SQL_SUCCESS_WITH_INFO.)
01S04	More than one row updated or deleted	The argument *fOption* was SQL_UPDATE or SQL_DELETE and more than one row was updated or deleted. (Function returns SQL_SUCCESS_WITH_INFO.)
21S02	Degree of derived table does not match column list	The argument *fOption* was SQL_ADD or SQL_UPDATE and no columns were bound with **SQLBindCol**.
22003	Numeric value out of range	The argument *fOption* was SQL_ADD or SQL_UPDATE and the whole part of a numeric value was truncated.
22005	Error in assignment	The argument *fOption* was SQL_ADD or SQL_UPDATE and a value was incompatible with the data type of the associated column.
22008	Datetime field overflow	The argument *fOption* was SQL_ADD or SQL_UPDATE and a date, time, or timestamp value was, respectively, an invalid date, time, or timestamp.
23000	Integrity constraint violation	The argument *fOption* was SQL_ADD or SQL_UPDATE and a value was NULL for a column defined as NOT NULL in the associated column or some other integrity constraint was violated. The argument *fOption* was SQL_ADD and a column that was not bound with **SQLBindCol** is defined as NOT NULL or has no default.
24000	Invalid cursor state	(DM) The *hstmt* was in an executed state but no result set was associated with the *hstmt*. (DM) A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called. A cursor was open on the *hstmt* and **SQLExtendedFetch** had been called,

		but the cursor was positioned before the start of the result set or after the end of the result set. The argument *fOption* was SQL_DELETE, SQL_REFRESH, or SQL_UPDATE and the cursor was positioned before the start of the result set or after the end of the result set.
42000	Syntax error or access violation	The driver was unable to lock the row as needed to perform the operation requested in the argument *fOption*. The driver was unable to lock the row as requested in the argument *fLock*.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S0023	No default for column	The *fOption* argument was SQL_ADD and a column that was not bound did not have a default value and could not be set to NULL. The *fOption* argument was SQL_ADD, the length specified in the *pcbValue* buffer bound by **SQLBindCol** was SQL_IGNORE, and the column did not have a default value.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1009	Invalid argument value	(DM) The value specified for the argument *fOption* was invalid. (DM) The value specified for the argument *fLock* was invalid. The argument *irow* was greater than the number of rows in the rowset and the *fOption* argument was not SQL_ADD. The value specified for the argument *fOption* was SQL_ADD, SQL_UPDATE, or SQL_DELETE, the value specified for the argument *fLock* was SQL_LOCK_NO_CHANGE, and the SQL_CONCURRENCY statement

		option was SQL_CONCUR_READ_ONLY.
S1010	Function sequence error	(DM) The specified *hstmt* was not in an executed state. The function was called without first calling **SQLExecDirect**, **SQLExecute**, or a catalog function. (DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	The *fOption* argument was SQL_ADD or SQL_UPDATE, a data value was a null pointer, and the column length value was not 0, SQL_DATA_AT_EXEC, SQL_IGNORE, SQL_NULL_DATA, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET. The *fOption* argument was SQL_ADD or SQL_UPDATE, a data value was not a null pointer, and the column length value was less than 0, but not equal to SQL_DATA_AT_EXEC, SQL_IGNORE, SQL_NTS, or SQL_NULL_DATA, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.
S1107	Row value out of range	The value specified for the argument *irow* was greater than the number of rows in the rowset and the *fOption* argument was not SQL_ADD.
S1109	Invalid cursor position	The cursor associated with the *hstmt* was defined as forward only, so the cursor could not be positioned within the rowset. See the description for the SQL_CURSOR_TYPE option in **SQLSetStmtOption**. The *fOption* argument was SQL_REFRESH, SQL_UPDATE, or SQL_DELETE and the value in the *rgfRowStatus* array for the row specified by the *irow* argument was SQL_ROW_DELETED or SQL_ROW_ERROR.
S1C00	Driver not capable	The driver or data source does not support the operation requested in the *fOption* argument or the *fLock* argument.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

irow Argument

The *irow* argument specifies the number of the row in the rowset on which to perform the operation specified by the *fOption* argument. If *irow* is 0, the operation applies to every row in the rowset. Except for the SQL_ADD operation, *irow* must be a value from 0 to the number of rows in the rowset. For the SQL_ADD operation, *irow* can be any value; generally it is either 0 (to add as many rows as there are in the rowset) or the number of rows in the rowset plus 1 (to add the data from an extra row of buffers allocated for this purpose).

Note In the C language, arrays are 0-based, while the *irow* argument is 1-based. For example, to update the fifth row of the rowset, an application modifies the rowset buffers at array index 4, but specifies an *irow* of 5.

All operations except for SQL_ADD position the cursor on the row specified by *irow*; the SQL_ADD operation does not change the cursor position. The following operations require a cursor position:

- Positioned update and delete statements.
- Calls to **SQLGetData**.
- Calls to **SQLSetPos** with the SQL_DELETE, SQL_REFRESH, and SQL_UPDATE options.

For example, if the cursor is positioned on the second row of the rowset, a positioned delete statement deletes that row; if it is positioned on the entire rowset (*irow* is 0), a positioned delete statement deletes every row in the rowset.

An application can specify a cursor position when it calls **SQLSetPos**. Generally, it calls **SQLSetPos** with the SQL_POSITION or SQL_REFRESH operation to position the cursor before executing a positioned update or delete statement or calling **SQLGetData**.

fOption Argument

The *fOption* argument supports the following operations. To determine which options are supported by a data source, an application calls **SQLGetInfo** with the SQL_POS_OPERATIONS information type.

***fOption* Argument**	**Operation**
SQL_POSITION	The driver positions the cursor on the row specified by *irow*. This is the same as the FALSE value of this argument in ODBC 1.0.
SQL_REFRESH	The driver positions the cursor on the row specified by *irow* and refreshes data in the rowset buffers for that row. For more information about how the driver returns data in the rowset buffers, see the descriptions of row-wise and column-wise binding in **SQLExtendedFetch**. This is the same as the TRUE value of this argument in ODBC 1.0.
SQL_UPDATE	The driver positions the cursor on the row specified by *irow* and updates the underlying row of data with the values in the rowset buffers (the *rgbValue* argument in **SQLBindCol**). It retrieves the lengths of the data from the number-of-bytes buffers (the *pcbValue* argument in **SQLBindCol**). If the length of any column is SQL_IGNORE, the column is not updated. After updating the row, the driver changes the *rgfRowStatus* array specified in **SQLExtendedFetch** to SQL_ROW_UPDATED.
SQL_DELETE	The driver positions the cursor on the row specified by *irow* and deletes the underlying row of data. It changes the *rgfRowStatus* array specified in **SQLExtendedFetch** to SQL_ROW_DELETED. After the row has been deleted, positioned update and delete statements, calls to

	SQLGetData and calls to **SQLSetPos** with *fOption* set to anything except SQL_POSITION are not valid for the row. Whether the row remains visible depends on the cursor type. For example, deleted rows are visible to static and keyset-driven cursors but invisible to dynamic cursors.
SQL_ADD	The driver adds a new row of data to the data source. Where the row is added to the data source and whether it is visible in the result set is driver-defined. The driver retrieves the data from the rowset buffers (the *rgbValue* argument in **SQLBindCol**) according to the value of the *irow* argument. It retrieves the lengths of the data from the number-of-bytes buffers (the *pcbValue* argument in **SQLBindCol**). Generally, the application allocates an extra row of buffers for this purpose. For columns not bound to the rowset buffers, the driver uses default values (if they are available) or NULL values (if default values are not available). For columns with a length of SQL_IGNORE, the driver uses default values. If *irow* is less than or equal to the rowset size, the driver changes the *rgfRowStatus* array specified in **SQLExtendedFetch** to SQL_ROW_ADDED after adding the row. At this point, the rowset buffers do not match the cursor for the row. To restore the rowset buffers to match the data in the cursor, an application calls **SQLSetPos** with the SQL_REFRESH option. This operation does not affect the cursor position.

fLock Argument

The *fLock* argument provides a way for applications to control concurrency and simulate transactions on data sources that do not support them. Generally, data sources that support concurrency levels and transactions will only support the SQL_LOCK_NO_CHANGE value of the *fLock* argument.

The *fLock* argument specifies the lock state of the row after **SQLSetPos** has been executed. To simulate a transaction, an application uses the SQL_LOCK_RECORD macro to lock each of the rows in the transaction. It then uses the SQL_UPDATE_RECORD or SQL_DELETE_RECORD macro to update or delete each row; the driver may temporarily change the lock state of the row while performing the operation specified by the *fOption* argument. Finally, it uses the SQL_LOCK_RECORD macro to unlock each row. For an example of how an application might do this, see the second code example. Note that if the driver is unable to lock the row either to perform the requested operation or to satisfy the *fLock* argument, it returns SQL_ERROR and SQLSTATE 42000 (Syntax error or access violation).

Although the *fLock* argument is specified for an *hstmt*, the lock accords the same privileges to all *hstmts* on the connection. In particular, a lock that is acquired by one *hstmt* on a connection can be unlocked by a different *hstmt* on the same connection.

A row locked through **SQLSetPos** remains locked until the application calls **SQLSetPos** for the row with *fLock* set to SQL_LOCK_UNLOCK or the application calls **SQLFreeStmt** with the SQL_CLOSE or SQL_DROP option.

The *fLock* argument supports the following types of locks. To determine which locks are supported by a data source, an application calls **SQLGetInfo** with the SQL_LOCK_TYPES information type.

***fLock* Argument**	**Lock Type**
SQL_LOCK_NO _CHANGE	The driver or data source ensures that the row is in the same locked or unlocked state as it was before **SQLSetPos** was called. This value of *fLock* allows data sources that do not support explicit row-level locking to use whatever locking is required by the current concurrency and transaction isolation levels. This is the same as the FALSE value of the *fLock*

	argument in ODBC 1.0.
SQL_LOCK_EXCLUSIVE	The driver or data source locks the row exclusively. An *hstmt* on a different *hdbc* or in a different application cannot be used to acquire any locks on the row. This is the same as the TRUE value of the *fLock* argument in ODBC 1.0.
SQL_LOCK_UNLOCK	The driver or data source unlocks the row.

For the add, update, and delete operations in **SQLSetPos**, the application uses the *fLock* argument as follows:

- To guarantee that a row does not change after it is retrieved, an application calls **SQLSetPos** with *fOption* set to SQL_REFRESH and *fLock* set to SQL_LOCK_EXCLUSIVE.
- If the application sets *fLock* to SQL_LOCK_NO_CHANGE, the driver guarantees an update, or delete operation will succeed only if the application specified SQL_CONCUR_LOCK for the SQL_CONCURRENCY statement option.
- If the application specifies SQL_CONCUR_ROWVER or SQL_CONCUR_VALUES for the SQL_CONCURRENCY statement option, the driver compares row versions or values and rejects the operation if the row has changed since the application fetched the row.
- If the application specifies SQL_CONCUR_READ_ONLY for the SQL_CONCURRENCY statement option, the driver rejects any update or delete operation.

For more information about the SQL_CONCURRENCY statement option, see **SQLSetStmtOption**.

Using SQLSetPos

Before an application calls **SQLSetPos**, it must:

1. If the application will call **SQLSetPos** with *fOption* set to SQL_ADD or SQL_UPDATE, call **SQLBindCol** for each column to specify its data type and associate storage for the column's data and length.
2. Call **SQLExecDirect**, **SQLExecute**, or a catalog function to create a result set.
3. Call **SQLExtendedFetch** to retrieve the data.

To delete data with **SQLSetPos**, an application:

- Calls **SQLSetPos** with *irow* set to the number of the row to delete.

An application can pass the value for a column either in the *rgbValue* buffer or with one or more calls to **SQLPutData**. Columns whose data is passed with **SQLPutData** are known as *data-at-execution* columns. These are commonly used to send data for SQL_LONGVARBINARY and SQL_LONGVARCHAR columns and can be mixed with other columns.

To update or add data with **SQLSetPos**, an application:

1. Places values in the *rgbValue* and *pcbValue* buffers bound with **SQLBindCol**:

- For normal columns, the application places the new column value in the *rgbValue* buffer and the length of that value in the *pcbValue* buffer. If the row is being updated and the column is not to be changed, the application places SQL_IGNORE in the *pcbValue* buffer.
- For data-at-execution columns, the application places an application-defined value, such as the column number, in the *rgbValue* buffer. The value can be used later to identify the column.

    It places the result of the SQL_LEN_DATA_AT_EXEC(*length*) macro in the *pcbValue* buffer. If the SQL data type of the column is SQL_LONGVARBINARY, SQL_LONGVARCHAR, or a long, data source-specific data type and the driver returns "Y" for the SQL_NEED_LONG_DATA_LEN information type in **SQLGetInfo**, *length* is the number of bytes of data to be sent for the parameter; otherwise, it must be a nonnegative value and is ignored.

2. Calls **SQLSetPos** or uses an **SQLSetPos** macro to update or add the row of data.

- If there are no data-at-execution columns, the process is complete.
- If there are any data-at-execution columns, the function returns SQL_NEED_DATA.

3. Calls **SQLParamData** to retrieve the address of the *rgbValue* buffer for the first data-at-execution column to be processed. The application retrieves the application-defined value from the *rgbValue* buffer.

Note Although data-at-execution parameters are similar to data-at-execution columns, the value returned by **SQLParamData** is different for each.

Data-at-execution parameters are parameters in an SQL statement for which data will be sent with **SQLPutData** when the statement is executed with **SQLExecDirect** or **SQLExecute**. They are bound with **SQLBindParameter**. The value returned by **SQLParamData** is a 32-bit value passed to **SQLBindParameter** in the *rgbValue* argument.

Data-at-execution columns are columns in a rowset for which data will be sent with **SQLPutData** when a row is updated or added with **SQLSetPos**. They are bound with **SQLBindCol**. The value returned by **SQLParamData** is the address of the row in the *rgbValue* buffer that is being processed.

4. Calls **SQLPutData** one or more times to send data for the column. More than one call is needed if the data value is larger than the *rgbValue* buffer specified in **SQLPutData**; note that multiple calls to **SQLPutData** for the same column are allowed only when sending character C data to a column with a character, binary, or data source-specific data type or when sending binary C data to a column with a character, binary, or data source-specific data type.
5. Calls **SQLParamData** again to signal that all data has been sent for the column.

- If there are more data-at-execution columns, **SQLParamData** returns SQL_NEED_DATA and the address of the *rgbValue* buffer for the next data-at-execution column to be processed. The application repeats steps 4 and 5.
- If there are no more data-at-execution columns, the process is complete. If the statement was executed successfully, **SQLParamData** returns SQL_SUCCESS or SQL_SUCCESS_WITH_INFO; if the execution failed, it returns SQL_ERROR. at this point, **SQLParamData** can return any SQLSTATE that can be returned by **SQLSetPos**.

After **SQLSetPos** returns SQL_NEED_DATA, and before data is sent for all data-at-execution columns, the operation is canceled, or an error occurs in **SQLParamData** or **SQLPutData**, the application can only call **SQLCancel**, **SQLGetFunctions**, **SQLParamData**, or **SQLPutData** with the *hstmt* or the *hdbc* associated with the *hstmt*. If it calls any other function with the *hstmt* or the *hdbc* associated with the *hstmt*, the function returns SQL_ERROR and SQLSTATE S1010 (Function sequence error).

If the application calls **SQLCancel** while the driver still needs data for data-at-execution columns, the driver cancels the operation; the application can then call **SQLSetPos** again; canceling does not affect the cursor state or the current cursor position. If the application calls **SQLParamData** or **SQLPutData** after canceling the operation, the function returns SQL_ERROR and SQLSTATE S1008 (Operation canceled).

Performing Bulk Operations

If the *irow* argument is 0, the driver performs the operation specified in the *fOption* argument for every row in the rowset. If an error occurs that pertains to the entire rowset, such as SQLSTATE S1T00 (Timeout expired), the driver returns SQL_ERROR and the appropriate SQLSTATE. The contents of the rowset buffers are undefined and the cursor position is unchanged.

If an error occurs that pertains to a single row, the driver:

- Sets the element in the *rgfRowStatus* array for the row to SQL_ROW_ERROR.
- Posts SQLSTATE 01S01 (Error in row) in the error queue.
- Posts one or more additional SQLSTATEs for the error after SQLSTATE 01S01 (Error in row) in the error queue.

After it has processed the error or warning, the driver continues the operation for the remaining rows in the rowset and returns SQL_SUCCESS_WITH_INFO. Thus, for each row that returned an error, the error queue contains SQLSTATE 01S01 (Error in row) followed by zero or more additional SQLSTATEs.

If the driver returns any warnings, such as SQLSTATE 01004 (Data truncated), it returns warnings that apply to the entire rowset or to unknown rows in the rowset before it returns the error information that applies to specific rows. It returns warnings for specific rows along with any other error information about those rows.

SQLSetPos Macros

As an aid to programming, the following macros for calling **SQLSetPos** are defined in the SQLEXT.H file.

Macro name	**Function call**

SQL_POSITION_TO(*hstmt*, *irow*)	**SQLSetPos**(*hstmt*, *irow*, SQL_POSITION, SQL_LOCK_NO_CHANGE)
SQL_LOCK_RECORD(*hstmt*, *irow*, *fLock*)	**SQLSetPos**(*hstmt*, *irow*, SQL_POSITION, *fLock*)
SQL_REFRESH_RECORD(*hstmt*, *irow*, *fLock*)	**SQLSetPos**(*hstmt*, *irow*, SQL_REFRESH, *fLock*)
SQL_UPDATE_RECORD(*hstmt*, *irow*)	**SQLSetPos**(*hstmt*, *irow*, SQL_UPDATE, SQL_LOCK_NO_CHANGE)
SQL_DELETE_RECORD(*hstmt*, *irow*)	**SQLSetPos**(*hstmt*, *irow*, SQL_DELETE, SQL_LOCK_NO_CHANGE)
SQL_ADD_RECORD(*hstmt*, *irow*)	**SQLSetPos**(*hstmt*, *irow*, SQL_ADD, SQL_LOCK_NO_CHANGE)

▪

Code Example

In the following example, an application allows a user to browse the EMPLOYEE table and update employee birthdays. The cursor is keyset-driven with a rowset size of 20 and uses optimistic concurrency control comparing row versions. After each rowset is fetched, the application prints them and allows the user to select and update an employee's birthday. The application uses **SQLSetPos** to position the cursor on the selected row and performs a positioned update of the row. (Error handling is omitted for clarity.)

```
#define ROWS 20
#define NAME_LEN 30
#define BDAY_LEN 11

UCHAR  szName[ROWS][NAME_LEN], szBirthday[ROWS][BDAY_LEN], szReply[3];
SDWORD cbName[ROWS], cbBirthday[ROWS];
UWORD  rgfRowStatus[ROWS];
UDWORD crow, irow;
HSTMT  hstmtS, hstmtU;

SQLSetStmtOption(hstmtS, SQL_CONCURRENCY, SQL_CONCUR_ROWVER);
SQLSetStmtOption(hstmtS, SQL_CURSOR_TYPE, SQL_CURSOR_KEYSET_DRIVEN);
SQLSetStmtOption(hstmtS, SQL_ROWSET_SIZE, ROWS);
SQLSetCursorName(hstmtS, "C1", SQL_NTS);
SQLExecDirect(hstmtS, "SELECT NAME, BIRTHDAY FROM EMPLOYEE FOR UPDATE OF
BIRTHDAY", SQL_NTS);

SQLBindCol(hstmtS, 1, SQL_C_CHAR, szName, NAME_LEN, cbName);
SQLBindCol(hstmtS, 1, SQL_C_CHAR, szBirthday, BDAY_LEN, cbBirthday);

while (SQLExtendedFetch(hstmtS, FETCH_NEXT, 0, &crow, rgfRowStatus) !=
SQL_ERROR) {
  for (irow = 0; irow < crow; irow++) {
    if (rgfRowStatus[irow] != SQL_ROW_DELETED)
      printf("%d %-*s %*s\n", irow, NAME_LEN-1, szName[irow], BDAY_LEN-1,
szBirthday[irow]);
  }
  while (TRUE) {
    printf("\nRow number to update?");
    gets(szReply);
    irow = atoi(szReply);
    if (irow > 0 && irow <= crow) {
      printf("\nNew birthday?");
      gets(szBirthday[irow-1]);
      SQLSetPos(hstmtS, irow, SQL_POSITION, SQL_LOCK_NO_CHANGE);
      SQLPrepare(hstmtU, "UPDATE EMPLOYEE SET BIRTHDAY=? WHERE CURRENT OF
C1", SQL_NTS);
```

```
      SQLBindParameter(hstmtU, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_DATE,
BDAY_LEN, 0, szBirthday, 0, NULL);
      SQLExecute(hstmtU);
    } else if (irow == 0) {
      break;
    }
  }
}
```

In the following code fragment, an application simulates a transaction for rows 1 and 2. It locks the rows, updates them, then unlocks them. The code uses the **SQLSetPos** macros.

```
/* Lock rows 1 and 2                                      */

SQL_LOCK_RECORD(hstmt, 1, SQL_LOCK_EXCLUSIVE);
SQL_LOCK_RECORD(hstmt, 2, SQL_LOCK_EXCLUSIVE);


/* Modify the rowset buffers for rows 1 and 2 (not shown).*/
/* Update rows 1 and 2.                                   */

SQL_UPDATE_RECORD(hstmt, 1);
SQL_UPDATE_RECORD(hstmt, 2);


/* Unlock rows 1 and 2                                    */

SQL_LOCK_RECORD(hstmt, 1, SQL_LOCK_UNLOCK);
SQL_LOCK_RECORD(hstmt, 2, SQL_LOCK_UNLOCK);
```

Related Functions

SQLBindCol
SQLCancel
SQLExtendedFetch (extension)
SQLSetStmtOption (extension)

SQLSetScrollOptions (Extension Level 2, ODBC 1.0)

Related Functions

SQLSetScrollOptions sets options that control the behavior of cursors associated with an *hstmt*. **SQLSetScrollOptions** allows the application to specify the type of cursor behavior desired in three areas: concurrency control, sensitivity to changes made by other transactions, and rowset size.

Note In ODBC 2.0, **SQLSetScrollOptions** has been superceded by the SQL_CURSOR_TYPE, SQL_CONCURRENCY, SQL_KEYSET_SIZE, and SQL_ROWSET_SIZE statement options. ODBC 2.0 drivers must support this function for backwards compatibility; ODBC 2.0 applications should only call this function in ODBC 1.0 drivers.

If an application calls **SQLSetScrollOptions**, a driver must be able to return the values of the aforementioned statement options with **SQLGetStmtOption**. For more information, see **SQLSetStmtOption**.

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 following values: SQL_CONCUR_READ_ONLY: Cursor is read-only. No updates are allowed. SQL_CONCUR_LOCK: Cursor uses the lowest level of locking sufficient to ensure that the row can be updated. SQL_CONCUR_ROWVER: Cursor uses optimistic concurrency control, comparing row versions, such as SQLBase ROWID or Sybase TIMESTAMP. SQL_CONCUR_VALUES: Cursor uses optimistic concurrency control, comparing values.
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 values: SQL_SCROLL_FORWARD_ONLY: The cursor only scrolls forward. SQL_SCROLL_STATIC: The data in the result set is static. SQL_SCROLL_KEYSET_DRIVEN: The driver saves and uses the keys for every row in the result set. 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_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLSetScrollOptions** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLSetScrollOptions** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1010	Function sequence error	(DM) The specified *hstmt* was in a prepared or executed state. The function must be called before calling **SQLPrepare** or **SQLExecDirect**. (DM) An asynchronously executing function was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1107	Row value out of range	(DM) The value specified for the argument *crowKeyset* was less than 1, but was not equal to SQL_SCROLL_FORWARD_ONLY, SQL_SCROLL_STATIC, SQL_SCROLL_KEYSET_DRIVEN, or SQL_SCROLL_DYNAMIC. (DM) The value specified for the argument *crowKeyset* is greater than 0,

		but less than *crowRowset*. (DM) The value specified for the argument *crowRowset* was 0.
S1108	Concurrency option out of range	(DM) The value specified for the argument *fConcurrency* was not equal to SQL_CONCUR_READ_ONLY, SQL_CONCUR_LOCK, SQL_CONCUR_ROWVER, or SQL_CONCUR_VALUES.
S1C00	Driver not capable	The driver or data source does not support the concurrency control option specified in the argument *fConcurrency*. The driver does not support the cursor model specified in the argument *crowKeyset*.

Comments

If an application calls **SQLSetScrollOptions** for an *hstmt*, it must do so before it calls **SQLPrepare** or **SQLExecDirect** or creating a result set with a catalog function.

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 SQL_SCROLL_FORWARD_ONLY, and *fConcurrency* equals SQL_CONCUR_READ_ONLY.

Related Functions

SQLBindCol
SQLExtendedFetch (extension)
SQLSetPos (extension)
SQLSetStmtOption

SQLSetStmtOption (Extension Level 1, ODBC 1.0)

Code Example Related Functions

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*)

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

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.
UWORD	*fOption*	Input	Option to set, listed in "Comments."
UDWORD	*vParam*	Input	Value associated with *fOption*. Depending on the value of *fOption*, *vParam* will be a 32-bit integer value or point to a null-terminated character string.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLSetStmtOption** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLSetStmtOption** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01S02	Option value changed	The driver did not support the specified value of the *vParam* argument and substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
24000	Invalid cursor state	The *fOption* was SQL_CONCURRENCY, SQL_CURSOR_TYPE, SQL_SIMULATE_CURSOR, or SQL_USE_BOOKMARKS and the cursor was open.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution

		or completion of the function.
S1009	Invalid argument value	Given the specified *fOption* value, an invalid value was specified for the argument *vParam*. (The Driver Manager returns this SQLSTATE only for statement options that accept a discrete set of values, such as SQL_ASYNC_ENABLE. For all other statement options, the driver must verify the value of the argument *vParam*.)
S1010	Function sequence error	(DM) An asynchronously executing function was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1011	Operation invalid at this time	The *fOption* was SQL_CONCURRENCY, SQL_CURSOR_TYPE, SQL_SIMULATE_CURSOR, or SQL_USE_BOOKMARKS and the statement was prepared.
S1092	Option type out of range	(DM) The value specified for the argument *fOption* was in the block of numbers reserved for ODBC connection and statement options, but was not valid for the version of ODBC supported by the driver.
S1C00	Driver not capable	The value specified for the argument *fOption* was a valid ODBC statement option for the version of ODBC supported by the driver, but was not supported by the driver. The value specified for the argument *fOption* was in the block of numbers reserved for driver-specific connection and statement options, but was not supported by the driver.

Comments

Statement options for an *hstmt* remain in effect until they are changed by another call to **SQLSetStmtOption** or the *hstmt* is dropped by calling **SQLFreeStmt** with the SQL_DROP option. Calling **SQLFreeStmt** with the SQL_CLOSE, SQL_UNBIND, or SQL_RESET_PARAMS options does not reset statement options.

Some statement options support substitution of a similar value if the data source does not support the specified value of *vParam*. In such cases, the driver returns SQL_SUCCESS_WITH_INFO and SQLSTATE 01S02 (Option value changed). For example, if *fOption* is SQL_CONCURRENCY, *vParam* is SQL_CONCUR_ROWVER, and the data source does not support this, the driver substitutes SQL_CONCUR_VALUES. To determine the substituted value, an application calls **SQLGetStmtOption**.

The currently defined options and the version of ODBC in which they were introduced are shown below; it is expected that more will be defined to take advantage of different data sources. Options from 0 to 999 are reserved by ODBC; driver developers must reserve values greater than or equal to SQL_CONNECT_OPT_DRVR_START for driver-specific use.

The format of information set with *vParam* depends on the specified *fOption*. **SQLSetStmtOption** accepts option information in one of two different formats: a null-terminated character string or a 32-bit integer value. The format of each is noted in the option's description. This format applies to the information returned for each option in **SQLGetStmtOption**. Character strings pointed to by the *vParam* argument of **SQLSetStmtOption** have a maximum length of SQL_MAX_OPTION_STRING_LENGTH bytes (excluding the null termination byte).

fOption — ***vParam* Contents**

SQL_ASYNC_ENABLE
(ODBC 1.0)

A 32-bit integer value that specifies whether a function called with the specified *hstmt* is executed asynchronously:

SQL_ASYNC_ENABLE_OFF = Off (the default)
SQL_ASYNC_ENABLE_ON = On

Once a function has been called asynchronously, no other functions can be called on the *hstmt* or the *hdbc* associated with the *hstmt* except for the original function, **SQLAllocStmt**, **SQLCancel**, or **SQLGetFunctions**, until the original function returns a code other than SQL_STILL_EXECUTING. Any other function called on the *hstmt* returns SQL_ERROR with an SQLSTATE of S1010 (Function sequence error). Functions can be called on other *hstmts*.

The following functions can be executed asynchronously:

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

SQL_BIND_TYPE
(ODBC 1.0)

A 32-bit integer value that sets the binding orientation to be used when **SQLExtendedFetch** is called on the associated *hstmt*. Column-wise binding is selected by supplying the defined constant SQL_BIND_BY_COLUMN for the argument *vParam*. Row-wise binding is selected by supplying a value for *vParam* specifying the length of a structure or an instance of a buffer into which result columns will be bound.

The length specified in *vParam* must include space for all of the bound columns and any padding of the structure or buffer to ensure that when the address of a bound column is incremented with the specified length, the result will point to the beginning of the same column in the next row. When using the **sizeof** operator with structures or unions in ANSI C, this behavior is guaranteed.

Column-wise binding is the default binding orientation for **SQLExtendedFetch**.

SQL_CONCURRENCY
(ODBC 2.0)

A 32-bit integer value that specifies the cursor concurrency:

SQL_CONCUR_READ_ONLY = Cursor is read-only. No updates are allowed.

SQL_CONCUR_LOCK = Cursor uses the lowest level of locking sufficient to ensure that the row can be updated.

SQL_CONCUR_ROWVER = Cursor uses optimistic

	concurrency control, comparing row versions, such as SQLBase ROWID or Sybase TIMESTAMP. SQL_CONCUR_VALUES = Cursor uses optimistic concurrency control, comparing values. The default value is SQL_CONCUR_READ_ONLY. This option cannot be specified for an open cursor and can also be set through the *fConcurrency* argument in **SQLSetScrollOptions**. If the specified concurrency is not supported by the data source, the driver substitutes a different concurrency and returns SQLSTATE 01S02 (Option value changed). For SQL_CONCUR_VALUES, the driver substitutes SQL_CONCUR_ROWVER, and vice versa. For SQL_CONCUR_LOCK, the driver substitutes, in order, SQL_CONCUR_ROWVER or SQL_CONCUR_VALUES.
SQL_CURSOR_TYPE (ODBC 2.0)	A 32-bit integer value that specifies the cursor type: SQL_CURSOR_FORWARD_ONLY = The cursor only scrolls forward. SQL_CURSOR_STATIC = The data in the result set is static. SQL_CURSOR_KEYSET_DRIVEN = The driver saves and uses the keys for the number of rows specified in the SQL_KEYSET_SIZE statement option. SQL_CURSOR_DYNAMIC = The driver only saves and uses the keys for the rows in the rowset. The default value is SQL_CURSOR_FORWARD_ONLY. This option cannot be specified for an open cursor and can also be set through the *crowKeyset* argument in **SQLSetScrollOptions**. If the specified cursor type is not supported by the data source, the driver substitutes a different cursor type and returns SQLSTATE 01S02 (Option value changed). For a mixed or dynamic cursor, the driver substitutes, in order, a keyset-driven or static cursor. For a keyset-driven cursor, the driver substitutes a static cursor.
SQL_KEYSET_SIZE (ODBC 2.0)	A 32-bit integer value that specifies the number of rows in the keyset for a keyset-driven cursor. If the keyset size is 0 (the default), the cursor is fully keyset-driven. If the keyset size is greater than 0, the cursor is mixed (keyset-driven within the keyset and dynamic outside of the keyset). The default keyset size is 0. If the specified size exceeds the maximum keyset size, the driver substitutes that size and returns SQLSTATE 01S02 (Option value changed). **SQLExtendedFetch** returns an error if the keyset size is greater than 0 and less than the rowset size.
SQL_MAX_LENGTH (ODBC 1.0)	A 32-bit integer value that specifies the maximum amount of data that the driver returns from a character or binary column. If *vParam* is less than the length of the available data, **SQLFetch** or **SQLGetData** truncates the data and returns SQL_SUCCESS. If *vParam* is 0 (the default), the driver attempts to return all available data. If the specified length is less than the minimum amount of data that the data source can return (the minimum is 254 bytes on many data sources), or greater than the maximum amount of data that the data source can return, the driver substitutes that value and returns SQLSTATE 01S02 (Option

	value changed). This option is intended to reduce network traffic and should only be supported when the data source (as opposed to the driver) in a multiple-tier driver can implement it. To truncate data, an application should specify the maximum buffer length in the *cbValueMax* argument in **SQLBindCol** or **SQLGetData**. **Note** In ODBC 1.0, this statement option only applied to SQL_LONGVARCHAR and SQL_LONGVARBINARY columns.
SQL_MAX_ROWS (ODBC 1.0)	A 32-bit integer value corresponding to the 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. This option is intended to reduce network traffic. Conceptually, it is applied when the result set is created and limits the result set to the first *vParam* rows. If the specified number of rows exceeds the number of rows that can be returned by the data source, the driver substitutes that value and returns SQLSTATE 01S02 (Option value changed).
SQL_NOSCAN (ODBC 1.0)	A 32-bit integer value that specifies whether the driver does not scan SQL strings for escape clauses: SQL_NOSCAN_OFF = The driver scans SQL strings for escape clauses (the default). SQL_NOSCAN_ON = The driver does not scan SQL strings for escape clauses. Instead, the driver sends the statement directly to the data source.
SQL_QUERY_TIMEOUT (ODBC 1.0)	A 32-bit integer value corresponding to the 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. If the specified timeout exceeds the maximum timeout in the data source or is smaller than the minimum timeout, the driver substitutes that value and returns SQLSTATE 01S02 (Option value changed). Note that the application need not call **SQLFreeStmt** with the SQL_CLOSE option to reuse the *hstmt* if a **SELECT** statement timed out.
SQL_RETRIEVE_DATA (ODBC 2.0)	A 32-bit integer value: SQL_RD_ON = **SQLExtendedFetch** retrieves data after it positions the cursor to the specified location. This is the default. SQL_RD_OFF = **SQLExtendedFetch** does not retrieve data after it positions the cursor. By setting SQL_RETRIEVE_DATA to SQL_RD_OFF, an application can verify if a row exists or retrieve a bookmark for the row without incurring the overhead of retrieving rows.
SQL_ROWSET_SIZE (ODBC 2.0)	A 32-bit integer value that specifies the number of rows in the rowset. This is the number of rows returned by each call to **SQLExtendedFetch**. The default value is 1. If the specified rowset size exceeds the maximum rowset size supported by the data source, the driver substitutes that value and returns SQLSTATE 01S02 (Option value

	changed). This option can be specified for an open cursor and can also be set through the *crowRowset* argument in **SQLSetScrollOptions**.
SQL_SIMULATE_CURSOR (ODBC 2.0)	A 32-bit integer value that specifies whether drivers that simulate positioned update and delete statements guarantee that such statements affect only one single row. To simulate positioned update and delete statements, most drivers construct a searched **UPDATE** or **DELETE** statement containing a **WHERE** clause that specifies the value of each column in the current row. Unless these columns comprise a unique key, such a statement may affect more than one row. To guarantee that such statements affect only one row, the driver determines the columns in a unique key and adds these columns to the result set. If an application guarantees that the columns in the result set comprise a unique key, the driver is not required to do so. This may reduce execution time. SQL_SC_NON_UNIQUE = The driver does not guarantee that simulated positioned update or delete statements will affect only one row; it is the application's responsibility to do so. If a statement affects more than one row, **SQLExecute** or **SQLExecDirect** returns SQLSTATE 01000 (General warning). SQL_SC_TRY_UNIQUE = The driver attempts to guarantee that simulated positioned update or delete statements affect only one row. The driver always executes such statements, even if they might affect more than one row, such as when there is no unique key. If a statement affects more than one row, **SQLExecute** or **SQLExecDirect** returns SQLSTATE 01000 (General warning). SQL_SC_UNIQUE = The driver guarantees that simulated positioned update or delete statements affect only one row. If the driver cannot guarantee this for a given statement, **SQLExecDirect** or **SQLPrepare** returns an error. If the specified cursor simulation type is not supported by the data source, the driver substitutes a different simulation type and returns SQLSTATE 01S02 (Option value changed). For SQL_SC_UNIQUE, the driver substitutes, in order, SQL_SC_TRY_UNIQUE or SQL_SC_NON_UNIQUE. For SQL_SC_TRY_UNIQUE, the driver substitutes SQL_SC_NON_UNIQUE. If a driver does not simulate positioned update and delete statements, it returns SQLSTATE S1C00 (Driver not capable).

SQL_USE_BOOKMARKS (ODBC 2.0)	A 32-bit integer value that specifies whether an application will use bookmarks with a cursor:
	SQL_UB_OFF = Off (the default)
	SQL_UB_ON = On
	To use bookmarks with a cursor, the application must specify this option with the SQL_UB_ON value before opening the cursor.

Code Example

See **SQLExtendedFetch**.

Related Functions

SQLCancel

SQLGetConnectOption (extension)

SQLGetStmtOption (extension)

SQLSetConnectOption (extension)

SQLSpecialColumns (Extension Level 1, ODBC 1.0)

Code Example Related Functions

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*)

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

Type	**Argument**	**Use**	**Description**
HSTMT	*hstmt*	Input	Statement handle.
UWORD	*fColType*	Input	Type of column to return. Must be one of the following values: 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).
UCHAR FAR *	*szTableQualifier*	Input	Qualifier name for the table. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	*cbTableQualifier*	Input	Length of *szTableQualifier*.
UCHAR FAR *	*szTableOwner*	Input	Owner name for the table. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.
SWORD	*cbTableOwner*	Input	Length of *szTableOwner*.
UCHAR FAR *	*szTableName*	Input	Table name.
SWORD	*cbTableName*	Input	Length of *szTableName*.
UWORD	*fScope*	Input	Minimum required scope of the

			rowid. The returned rowid may be of greater scope. Must be one of the following: 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).
UWORD	*fNullable*	Input	Determines whether to return special columns that can have a NULL value. Must be one of the following: SQL_NO_NULLS: Exclude special columns that can have NULL values. SQL_NULLABLE: Return special columns even if they can have NULL values.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLSpecialColumns** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLSpecialColumns** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
24000	Invalid cursor state	(DM) A cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch** had been called. A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was

		no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the length arguments was less than 0, but not equal to SQL_NTS. The value of one of the length arguments exceeded the maximum length value for the corresponding qualifier or name. The maximum length of each qualifier or name may be obtained by calling **SQLGetInfo** with the *fInfoType* values: SQL_MAX_QUALIFIER_NAME_LEN, SQL_MAX_OWNER_NAME_LEN, or SQL_MAX_TABLE_NAME_LEN.
S1097	Column type out of range	(DM) An invalid *fColType* value was specified.
S1098	Scope type out of range	(DM) An invalid *fScope* value was specified.
S1099	Nullable type out of range	(DM) An invalid *fNullable* value was specified.
S1C00	Driver not capable	A table qualifier was specified and the driver or data source does not support qualifiers. A table owner was specified and the driver or data source does not support owners. The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement

		options was not supported by the driver or data source.
S1T00	Timeout expired	The timeout period expired before the data source returned the requested result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

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

When the *fColType* argument is SQL_BEST_ROWID, **SQLSpecialColumns** returns the column or columns that uniquely identify each row in the table. These columns can always be used in a *select-list* or **WHERE** clause. However, **SQLColumns** does not necessarily return these columns. For example, **SQLColumns** might not return the Oracle ROWID pseudo-column ROWID. If there are no columns that uniquely identify each row in the table, **SQLSpecialColumns** returns a rowset with no rows; a subsequent call to **SQLFetch** or **SQLExtendedFetch** on the *hstmt* returns SQL_NO_DATA_FOUND.

If the *fColType*, *fScope*, or *fNullable* arguments specify characteristics that are not supported by the data source, **SQLSpecialColumns** returns a result set with no rows (as opposed to the function returning SQL_ERROR with SQLSTATE S1C00 (Driver not capable)). A subsequent call to **SQLFetch** or **SQLExtendedFetch** on the *hstmt* will return SQL_NO_DATA_FOUND.

SQLSpecialColumns returns the results as a standard result set, ordered by SCOPE. The following table lists the columns in the result set.

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual length of the COLUMN_NAME column, an application can call **SQLGetInfo** with the SQL_MAX_COLUMN_NAME_LEN option.

Column Name	**Data Type**	**Comments**
SCOPE	Smallint	Actual scope of the rowid. Contains one of the following values: SQL_SCOPE_CURROW SQL_SCOPE_TRANSACTION SQL_SCOPE_SESSION NULL is returned when *fColType* is SQL_ROWVER. For a description of each value, see the description of *fScope* in the "Syntax" section above.
COLUMN_NAME	Varchar(128) not NULL	Column identifier.
DATA_TYPE	Smallint not NULL	SQL data type. This can be an ODBC SQL data type or a driver-specific SQL data type. For a list of valid ODBC SQL data types, see SQL Data Types. For information about driver-specific SQL data types, see the driver's documentation.
TYPE_NAME	Varchar(128) not NULL	Data source-dependent data type name; for example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINARY", or "CHAR ( ) FOR BIT DATA".
PRECISION	Integer	The precision of the column on the data source. NULL is returned for data types where precision is not applicable. For more information concerning precision, see Precision, Scale, Length, and Display Size.

LENGTH	Integer	The length in bytes of data transferred on an **SQLGetData** or **SQLFetch** operation if SQL_C_DEFAULT is specified. For numeric data, this size may be different than the size of the data stored on the data source. This value is the same as the PRECISION column for character or binary data. For more information, see Precision, Scale, Length, and Display Size.
SCALE	Smallint	The scale of the column on the data source. NULL is returned for data types where scale is not applicable. For more information concerning scale, see Precision, Scale, Length, and Display Size.
PSEUDO_COLUMN	Smallint	Indicates whether the column is a pseudo-column, such as Oracle ROWID: SQL_PC_UNKNOWN SQL_PC_PSEUDO SQL_PC_NOT_PSEUDO **Note** For maximum interoperability, pseudo-columns should not be quoted with the identifier quote character returned by **SQLGetInfo**.

Note The PSEUDO_COLUMN column was added in ODBC 2.0. ODBC 1.0 drivers might return a different, driver-specific column with the same column number.

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 the rowid columns were modified. The opposite is not true: even if the rowid has not changed, the other columns in the row may have changed.

Columns returned for column 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 column 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 values in the SQL_ROWVER columns to the ones just fetched. If the value in a SQL_ROWVER column differs from the previous value, the application can alert the user that data on the display has changed.

Code Example

For a code example of a similar function, see **SQLColumns**.

Related Functions

SQLBindCol
SQLCancel
SQLColumns (extension)
SQLExtendedFetch (extension)
SQLFetch
SQLPrimaryKeys (extension)

SQLStatistics (Extension Level 1, ODBC 1.0)

Code Example Related Functions

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.
UCHAR FAR *	*szTableQualifier*	Input	Qualifier name. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	*cbTableQualifier*	Input	Length of *szTableQualifier*.
UCHAR FAR *	*szTableOwner*	Input	Owner name. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.
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_INDEX_UNIQUE or SQL_INDEX_ALL.
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_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLStatistics** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLStatistics** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
24000	Invalid cursor state	(DM) A cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch** had been called. A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the name length arguments was less than 0, but not equal to SQL_NTS. The value of one of the name length arguments exceeded the maximum length value for the corresponding qualifier or name.
S1100	Uniqueness option type out of range	(DM) An invalid *fUnique* value was specified.
S1101	Accuracy option	(DM) An invalid *fAccuracy* value was

	type out of range	specified.
S1C00	Driver not capable	A table qualifier was specified and the driver or data source does not support qualifiers. A table owner was specified and the driver or data source does not support owners. The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Timeout expired	The timeout period expired before the data source returned the requested result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

SQLStatistics returns information about a single table as a standard result set, ordered by NON_UNIQUE, TYPE, INDEX_QUALIFIER, INDEX_NAME, and SEQ_IN_INDEX. The result set combines statistics information for the table with information about each index. The following table lists the columns in the result set.

Note **SQLStatistics** might not return all indexes. For example, an Xbase driver might only return indexes in files in the current directory. Applications can use any valid index, regardless of whether it is returned by **SQLStatistics**.

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the TABLE_QUALIFIER, TABLE_OWNER, TABLE_NAME, and COLUMN_NAME columns, an application can call **SQLGetInfo** with the SQL_MAX_QUALIFIER_NAME_LEN, SQL_MAX_OWNER_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN options.

Column Name	Data Type	Comments
TABLE_QUALIFIER	Varchar(128)	Table qualifier identifier of the table to which the statistic or index applies; NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have qualifiers.
TABLE_OWNER	Varchar(128)	Table owner identifier of the table to which the statistic or index applies; NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.
TABLE_NAME	Varchar(128) not NULL	Table identifier of the table to which the statistic or index applies.
NON_UNIQUE	Smallint	Indicates whether the index prohibits duplicate values: TRUE if the index values can be

		nonunique. FALSE if the index values must be unique. NULL is returned if TYPE is SQL_TABLE_STAT.
INDEX_QUALIFIER	Varchar(128)	The identifier that is used to qualify the index name doing a **DROP INDEX**; NULL is returned if an index qualifier is not supported by the data source or if TYPE is SQL_TABLE_STAT. If a non-null value is returned in this column, it must be used to qualify the index name on a **DROP INDEX** statement; otherwise the TABLE_OWNER name should be used to qualify the index name.
INDEX_NAME	Varchar(128)	Index identifier; NULL is returned if TYPE is SQL_TABLE_STAT.
TYPE	Smallint not NULL	Type of information being returned: SQL_TABLE_STAT indicates a statistic for the table. SQL_INDEX_CLUSTERED indicates a clustered index. SQL_INDEX_HASHED indicates a hashed index. SQL_INDEX_OTHER indicates another type of index.
SEQ_IN_INDEX	Smallint	Column sequence number in index (starting with 1); NULL is returned if TYPE is SQL_TABLE_STAT.
COLUMN_NAME	Varchar(128)	Column identifier. If the column is based on an expression, such as SALARY + BENEFITS, the expression is returned; if the expression cannot be determined, an empty string is returned. If the index is a filtered index, each column in the filter condition is returned; this may require more than one row. NULL is returned if TYPE is SQL_TABLE_STAT.
COLLATION	Char(1)	Sort sequence for the column; "A" for ascending; "D" for descending; NULL is returned if column sort sequence is not supported by the data source or if TYPE is SQL_TABLE_STAT.
CARDINALITY	Integer	Cardinality of table or index; number of rows in table if TYPE is SQL_TABLE_STAT; number of unique values in the index if TYPE is not SQL_TABLE_STAT; NULL is returned if the value is not available from the data source.
PAGES	Integer	Number of pages used to store the index or table; number of pages for the table if TYPE is SQL_TABLE_STAT; number of pages

		for the index if TYPE is not SQL_TABLE_STAT; NULL is returned if the value is not available from the data source, or if not applicable to the data source.
FILTER_CONDITION	Varchar(128)	If the index is a filtered index, this is the filter condition, such as SALARY > 30000; if the filter condition cannot be determined, this is an empty string. NULL if the index is not a filtered index, it cannot be determined whether the index is a filtered index, or TYPE is SQL_TABLE_STAT.

Note The FILTER_CONDITION column was added in ODBC 2.0. ODBC 1.0 drivers might return a different, driver-specific column with the same column number.

If the row in the result set corresponds to a table, the driver sets TYPE to SQL_TABLE_STAT 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.

Code Example

For a code example of a similar function, see **SQLColumns**.

Related Functions

SQLBindCol
SQLCancel
SQLExtendedFetch (extension)
SQLFetch
SQLForeignKeys (extension)
SQLPrimaryKeys (extension)

SQLTablePrivileges (Extension Level 2, ODBC 1.0)

Code Example Related Functions

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*)

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

Type	Argument	Use	Description
HSTMT	*hstmt*	Input	Statement handle.
UCHAR FAR *	*szTableQualifier*	Input	Table qualifier. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
SWORD	*cbTableQualifier*	Input	Length of *szTableQualifier*.
UCHAR FAR *	*szTableOwner*	Input	String search pattern for owner names. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.
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_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLTablePrivileges** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLTablePrivileges** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.
24000	Invalid cursor state	(DM) A cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch** had been called.

		A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the name length arguments was less than 0, but not equal to SQL_NTS. The value of one of the name length arguments exceeded the maximum length value for the corresponding qualifier or name.
S1C00	Driver not capable	A table qualifier was specified and the driver or data source does not support qualifiers. A table owner was specified and the driver or data source does not support owners. A string search pattern was specified for the table owner, table name, or column name and the data source does not support search patterns for one or more of those arguments. The combination of the current settings of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver

		or data source.
S1T00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

Comments

The *szTableOwner* and *szTableName* arguments accept search patterns.

SQLTablePrivileges returns the results as a standard result set, ordered by TABLE_QUALIFIER, TABLE_OWNER, TABLE_NAME, and PRIVILEGE. The following table lists the columns in the result set.

Note **SQLTablePrivileges** might not return privileges for all tables. For example, an Xbase driver might only return privileges for files (tables) in the current directory. Applications can use any valid table, regardless of whether it is returned by **SQLTablePrivileges**.

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the TABLE_QUALIFIER, TABLE_OWNER, and TABLE_NAME columns, an application can call **SQLGetInfo** with the SQL_MAX_QUALIFIER_NAME_LEN, SQL_MAX_OWNER_NAME_LEN, and SQL_MAX_TABLE_NAME_LEN options.

Column Name	Data Type	Comments
TABLE_QUALIFIER	Varchar(128)	Table qualifier identifier; NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have qualifiers.
TABLE_OWNER	Varchar(128)	Table owner identifier; NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.
TABLE_NAME	Varchar(128) not NULL	Table identifier.
GRANTOR	Varchar(128)	Identifier of the user who granted the privilege; NULL if not applicable to the data source.
GRANTEE	Varchar(128) not NULL	Identifier of the user to whom the privilege was granted.
PRIVILEGE	Varchar(128) not NULL	Identifies the table privilege. May be one of the following or a data source-specific privilege. SELECT: The grantee is permitted to retrieve data for one or more columns of the table. INSERT: The grantee is permitted to insert new rows containing data for one or more columns into to the table. UPDATE: The grantee is permitted to update the data in one or more

		columns of the table.
		DELETE: The grantee is permitted to delete rows of data from the table.
		REFERENCES: The grantee is permitted to refer to one or more columns of the table within a constraint (for example, a unique, referential, or table check constraint).
		The scope of action permitted the grantee by a given table privilege is data source-dependent. For example, the UPDATE privilege might permit the grantee to update all columns in a table on one data source and only those columns for which the grantor has the UPDATE privilege on another data source.
IS_GRANTABLE	Varchar(3)	Indicates whether the grantee is permitted to grant the privilege to other users; "YES", "NO", or NULL if unknown or not applicable to the data source.

Code Example

For a code example of a similar function, see **SQLColumns**.

Related Functions

<u>**SQLBindCol**</u>
<u>**SQLCancel**</u>
<u>**SQLColumnPrivileges**</u> (extension)
<u>**SQLColumns**</u> (extension)
<u>**SQLExtendedFetch**</u> (extension)
<u>**SQLFetch**</u>
<u>**SQLStatistics**</u> (extension)
<u>**SQLTables**</u> (extension)

SQLTables (Extension Level 1, ODBC 1.0)

Code Example Related Functions

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*)

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

Type	**Argument**	**Use**	**Description**
HSTMT	*hstmt*	Input	Statement handle for retrieved results.
UCHAR FAR *	*szTableQualifier*	Input	Qualifier name. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have qualifiers.
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. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those tables that do not have owners.
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_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR or SQL_INVALID_HANDLE.

Diagnostics

When **SQLTables** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLTables** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	**Error**	**Description**
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.

24000	Invalid cursor state	(DM) A cursor was open on the *hstmt* and **SQLFetch** or **SQLExtendedFetch** had been called. A cursor was open on the *hstmt* but **SQLFetch** or **SQLExtendedFetch** had not been called.
IM001	Driver does not support this function	(DM) The driver associated with the *hstmt* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1008	Operation canceled	Asynchronous processing was enabled for the *hstmt*. The function was called and before it completed execution, **SQLCancel** was called on the *hstmt*. Then the function was called again on the *hstmt*. The function was called and, before it completed execution, **SQLCancel** was called on the *hstmt* from a different thread in a multithreaded application.
S1010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the *hstmt* and was still executing when this function was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for the *hstmt* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1090	Invalid string or buffer length	(DM) The value of one of the name length arguments was less than 0, but not equal to SQL_NTS. The value of one of the name length arguments exceeded the maximum length value for the corresponding qualifier or name.
S1C00	Driver not capable	A table qualifier was specified and the driver or data source does not support qualifiers. A table owner was specified and the driver or data source does not support owners. A string search pattern was specified for the table owner or table name and the data source does not support search patterns for one or more of those arguments. The combination of the current settings

		of the SQL_CONCURRENCY and SQL_CURSOR_TYPE statement options was not supported by the driver or data source.
S1T00	Timeout expired	The timeout period expired before the data source returned the requested result set. The timeout period is set through **SQLSetStmtOption**, SQL_QUERY_TIMEOUT.

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 info value.
- 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 accept search patterns.

To support enumeration of qualifiers, owners, and table types, **SQLTables** defines the following special semantics for the *szTableQualifier*, *szTableOwner*, *szTableName*, and *szTableType* arguments:

- If *szTableQualifier* is a single percent character (%) and *szTableOwner* and *szTableName* are empty strings, then the result set contains a list of valid qualifiers for the data source. (All columns except the TABLE_QUALIFIER column contain NULLs.)
- If *szTableOwner* is a single percent character (%) and *szTableQualifier* and *szTableName* are empty strings, then the result set contains a list of valid owners for the data source. (All columns except the TABLE_OWNER column contain NULLs.)
- If *szTableType* is a single percent character (%) and *szTableQualifier*, *szTableOwner*, and *szTableName* are empty strings, then the result set contains a list of valid table types for the data source. (All columns except the TABLE_TYPE column contain NULLs.)

If *szTableType* is not an empty string, it must contain a list of comma-separated, values for the types of interest; each value may be enclosed in single quotes (') or unquoted. For example, "'TABLE','VIEW'" or "TABLE, VIEW". If the data source does not support a specified table type, **SQLTables** does not return any results for that type.

SQLTables returns the results as a standard result set, ordered by TABLE_TYPE, TABLE_QUALIFIER, TABLE_OWNER, and TABLE_NAME. The following table lists the columns in the result set.

Note **SQLTables** might not return all qualifiers, owners, or tables. For example, an Xbase driver, for which a qualifier is a directory, might only return the current directory instead of all directories on the system. It might also only return files (tables) in the current directory. Applications can use any valid qualifier, owner, or table, regardless of whether it is returned by **SQLTables**.

The lengths of VARCHAR columns shown in the table are maximums; the actual lengths depend on the data source. To determine the actual lengths of the TABLE_QUALIFIER, TABLE_OWNER, and TABLE_NAME columns, an application can call **SQLGetInfo** with the SQL_MAX_QUALIFIER_NAME_LEN, SQL_MAX_OWNER_NAME_LEN, and SQL_MAX_TABLE_NAME_LEN options.

Column Name	**Data Type**	**Comments**
TABLE_QUALIFIER	Varchar(128)	Table qualifier identifier; NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have qualifiers.
TABLE_OWNER	Varchar(128)	Table owner identifier; NULL if not

		applicable to the data source. If a driver supports owners for some tables but not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.
TABLE_NAME	Varchar(128)	Table identifier.
TABLE_TYPE	Varchar(128)	Table type identifier; one of the following: "TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL TEMPORARY", "LOCAL TEMPORARY", "ALIAS", "SYNONYM" or a data source - specific type identifier.
REMARKS	Varchar(254)	A description of the table.

Code Example

For a code example of a similar function, see **SQLColumns**.

Related Functions

SQLBindCol
SQLCancel
SQLColumnPrivileges (extension)
SQLColumns (extension)
SQLExtendedFetch (extension)
SQLFetch
SQLStatistics (extension)
SQLTablePrivileges (extension)

SQLTransact (Core, ODBC 1.0)

Code Example Related Functions

SQLTransact requests a commit or rollback operation for all active operations on all *hstmts* associated with a connection. **SQLTransact** can also request that a commit or rollback operation be performed for all connections associated with the *henv*.

Syntax

RETCODE **SQLTransact**(*henv*, *hdbc*, *fType*)

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

Type	Argument	Use	Description
HENV	*henv*	Input	Environment handle.
HDBC	*hdbc*	Input	Connection handle.
UWORD	*fType*	Input	One of the following two values: SQL_COMMIT SQL_ROLLBACK

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLTransact** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLError**. The following table lists the SQLSTATE values commonly returned by **SQLTransact** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of SQLSTATEs returned by the Driver Manager. The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection not open	(DM) The *hdbc* was not in a connected state.
08007	Connection failure during transaction	The connection associated with the *hdbc* failed during the execution of the function and it cannot be determined whether the requested **COMMIT** or **ROLLBACK** occurred before the failure.
IM001	Driver does not support this function	(DM) The driver associated with the *hdbc* does not support the function.
S1000	General error	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by **SQLError** in the argument *szErrorMsg* describes the error and its cause.
S1001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
S1010	Function sequence error	(DM) An asynchronously executing function was called for an *hstmt* associated with the *hdbc* and was still executing when **SQLTransact** was called. (DM) **SQLExecute**, **SQLExecDirect**, or **SQLSetPos** was called for an *hstmt*

		associated with the *hdbc* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
S1012	Invalid transaction operation code	(DM) The value specified for the argument *fType* was neither SQL_COMMIT nor SQL_ROLLBACK.
S1C00	Driver not capable	The driver or data source does not support the **ROLLBACK** operation.

Comments

If *hdbc* is SQL_NULL_HDBC and *henv* is a valid environment handle, then the Driver Manager will attempt to commit or roll back transactions on all *hdbcs* that are in a connected state. The Driver Manager calls **SQLTransact** in the driver associated with each *hdbc*. The Driver Manager will return SQL_SUCCESS only if it receives SQL_SUCCESS for each *hdbc*. If the Driver Manager receives SQL_ERROR on one or more *hdbcs*, it will return SQL_ERROR to the application. To determine which connection(s) failed during the commit or rollback operation, the application can call **SQLError** for each *hdbc*.

Note The Driver Manager does not simulate a global transaction across all *hdbcs* and therefore does not use two-phase commit protocols.

If *hdbc* is a valid connection handle, *henv* is ignored and the Driver Manager calls **SQLTransact** in the driver for the *hdbc*.

If *hdbc* is SQL_NULL_HDBC and *henv* is SQL_NULL_HENV, **SQLTransact** returns SQL_INVALID_HANDLE.

If *fType* is SQL_COMMIT, **SQLTransact** issues a commit request for all active operations on any *hstmt* associated with an affected *hdbc*. If *fType* is SQL_ROLLBACK, **SQLTransact** issues a rollback request for all active operations on any *hstmt* associated with an affected *hdbc*. If no transactions are active, **SQLTransact** returns SQL_SUCCESS with no effect on any data sources.

If the driver is in manual-commit mode (by calling **SQLSetConnectOption** with the SQL_AUTOCOMMIT option set to zero), a new transaction is implicitly started when an SQL statement that can be contained within a transaction is executed against the current data source.

To determine how transaction operations affect cursors, an application calls **SQLGetInfo** with the SQL_CURSOR_ROLLBACK_BEHAVIOR and SQL_CURSOR_COMMIT_BEHAVIOR options.

If the SQL_CURSOR_ROLLBACK_BEHAVIOR or SQL_CURSOR_COMMIT_BEHAVIOR value equals SQL_CB_DELETE, **SQLTransact** closes and deletes all open cursors on all *hstmts* associated with the *hdbc* and discards all pending results. **SQLTransact** leaves any *hstmt* present in an allocated (unprepared) state; the application can reuse them for subsequent SQL requests or can call **SQLFreeStmt** to deallocate them.

If the SQL_CURSOR_ROLLBACK_BEHAVIOR or SQL_CURSOR_COMMIT_BEHAVIOR value equals SQL_CB_CLOSE, **SQLTransact** closes all open cursors on all *hstmts* associated with the *hdbc*. **SQLTransact** leaves any *hstmt* present in a prepared state; the application can call **SQLExecute** for an *hstmt* associated with the *hdbc* without first calling **SQLPrepare**.

If the SQL_CURSOR_ROLLBACK_BEHAVIOR or SQL_CURSOR_COMMIT_BEHAVIOR value equals SQL_CB_PRESERVE, **SQLTransact** does not affect open cursors associated with the *hdbc*. Cursors remain at the row they pointed to prior to the call to **SQLTransact**.

For drivers and data sources that support transactions, calling **SQLTransact** with either SQL_COMMIT or SQL_ROLLBACK when no transaction is active will return SQL_SUCCESS (indicating that there is no work to be committed or rolled back) and have no effect on the data source.

Drivers or data sources that do not support transactions (**SQLGetInfo** *fOption* SQL_TXN_CAPABLE is 0) are effectively always in autocommit mode. Therefore, calling **SQLTransact** with SQL_COMMIT will return SQL_SUCCESS. However, calling **SQLTransact** with SQL_ROLLBACK will result in SQLSTATE S1C00 (Driver not capable), indicating that a rollback can never be performed.

Code Example

See **SQLParamOptions**.

Related Functions

SQLGetInfo (extension)

SQLFreeStmt

Data Types

Overview

Data stored on a data source has an SQL data type, which may be specific to that data source. A driver maps data source-specific SQL data types to ODBC SQL data types and driver-specific SQL data types. (A driver returns these mappings through **SQLGetTypeInfo**. It also returns the SQL data types when describing the data types of columns and parameters in **SQLColAttributes**, **SQLColumns**, **SQLDescribeCol**, **SQLDescribeParam**, **SQLProcedureColumns**, and **SQLSpecialColumns**.)

Each SQL data type corresponds to an ODBC C data type. By default, the driver assumes that the C data type of a storage location corresponds to the SQL data type of the column or parameter to which the location is bound. If the C data type of a storage location is not the *default* C data type, the application can specify the correct C data type with the *fCType* argument in **SQLBindCol**, **SQLGetData**, or **SQLBindParameter**. Before returning data from the data source, the driver converts it to the specified C data type. Before sending data to the data source, the driver converts it from the specified C data type.

For information about driver-specific SQL data types, see the driver's documentation.

SQL Data Types

The ODBC SQL grammar defines three sets of SQL data types, each of which is a superset of the previous set.

- **Minimum** SQL data types provide a basic level of ODBC conformance.
- **Core** SQL data types are the data types in the X/Open and SQL Access Group SQL CAE specification (1992) and are supported by most SQL data sources.
- **Extended** SQL data types are additional data types supported by some SQL data sources.

A given driver and data source do not necessarily support all of the SQL data types defined in the ODBC grammar. Furthermore, they may support additional, driver-specific SQL data types. To determine which data types a driver supports, an application calls **SQLGetTypeInfo**. For information about driver-specific SQL data types, see the driver's documentation.

Minimum SQL Data Types

The following table lists valid values of *fSqlType* for the minimum SQL data types. These values are defined in SQL.H. The table also lists the name and description of the corresponding data type from the X/Open and SQL Access Group SQL CAE specification (1992).

Note The minimum SQL grammar requires that a data source support at least one character SQL data type. This table is only a guideline and shows commonly used names and limits of these data types. For a given data source, the characteristics of these data types may differ from those listed below. For information about the data types in a specific data source, see the documentation for that data source.

To determine which data types are supported by a data source and the characteristics of those data types, an application calls **SQLGetTypeInfo**.

fSqlType	SQL Data Type	Description
SQL_CHAR	CHAR(*n*)	Character string of fixed string length *n* ($1 \le n \le 254$).
SQL_VARCHAR	VARCHAR(*n*)	Variable-length character string with a maximum string length *n* ($1 \le n \le 254$).
SQL_LONGVARCHAR	LONG VARCHAR	Variable length character data. Maximum length is data source-dependent.

Core SQL Data Types

The following table lists valid values of *fSqlType* for the core SQL data types. These values are defined in SQL.H. The table also lists the name and description of the corresponding data type from the X/Open and SQL Access Group SQL CAE specification (1992). In the table, precision refers to the total number of digits and scale refers to the number of digits to the right of the decimal point.

Note This table is only a guideline and shows commonly used names, ranges, and limits of core SQL data types. A given data source may support only some of the listed data types and the characteristics of the supported data types may differ from those listed below. For example, some data sources support unsigned numeric data types. For information about the data types in a specific data source, see the documentation for that data source.

To determine which data types are supported by a data source and the characteristics of those data types, an application calls **SQLGetTypeInfo**.

fSqlType	SQL Data Type	Description
SQL_DECIMAL	DECIMAL(*p*,*s*)	Signed, exact, numeric value with a precision *p* and scale *s* ($1 \le p \le 15$; $0 \le s \le p$).
SQL_NUMERIC	NUMERIC(*p*,*s*)	Signed, exact, numeric value with a precision *p* and scale *s* ($1 \le p \le 15$; $0 \le s \le p$).
SQL_SMALLINT	SMALLINT	Exact numeric value with precision 5 and scale 0 (signed: $-32,768 \le n \le 32,767$, unsigned: $0 \le n \le 65,535$) [a].
SQL_INTEGER	INTEGER	Exact numeric value with precision 10 and scale 0 (signed: $-2^{31} \le n \le 2^{31} - 1$, unsigned: $0 \le n \le 2^{32} - 1$) [a].
SQL_REAL	REAL	Signed, approximate, numeric value with a mantissa precision 7 (zero or absolute value 10^{-38} to 10^{38}).
SQL_FLOAT	FLOAT	Signed, approximate, numeric value with a mantissa precision 15 (zero or absolute value 10^{-308} to 10^{308}).
SQL_DOUBLE	DOUBLE PRECISION	Signed, approximate, numeric value with a mantissa precision 15 (zero or absolute value 10^{-308} to 10^{308}).

[a] An application uses **SQLGetTypeInfo** or **SQLColAttributes** to determine if a particular data type or a particular column in a result set is unsigned.

Extended SQL Data Types

The following table lists valid values of *fSqlType* for the extended SQL data types. These values are defined in SQLEXT.H. The table also lists the name and description of the corresponding data type. In the table, precision refers to the total number of digits and scale refers to the number of digits to the right of the decimal point.

Note This table is only a guideline and shows commonly used names, ranges, and limits of extended SQL data types. A given data source may support only some of the listed data types and the characteristics of the supported data types may differ from those listed below. For example, some data sources support unsigned numeric data types. For information about the data types in a specific data source, see the documentation for that data source.

To determine which data types are supported by a data source and the characteristics of those data types, an application calls **SQLGetTypeInfo**.

---

fSqlType	Typical SQL Data Type	Description
SQL_BIT	BIT	Single bit binary data.
SQL_TINYINT	TINYINT	Exact numeric value with precision 3 and scale 0 (signed: $-128 \leq n \leq 127$, unsigned: $0 \leq n \leq 255$) [a].
SQL_BIGINT	BIGINT	Exact numeric value with precision 19 (if signed) or 20 (if unsigned) and scale 0 (signed: $-2^{63} \leq n \leq 2^{63} - 1$, unsigned: $0 \leq n \leq 2^{64} - 1$) [a].
SQL_BINARY	BINARY(*n*)	Binary data of fixed length *n* ($1 \leq n \leq 255$).
SQL_VARBINARY	VARBINARY(*n*)	Variable length binary data of maximum length *n* ($1 \leq n \leq 255$).
SQL_LONGVARBINARY	LONG VARBINARY	Variable length binary data. Maximum length is data source-dependent.
SQL_DATE	DATE	Date data.
SQL_TIME	TIME	Time data.
SQL_TIMESTAMP	TIMESTAMP	Date/time data.

[a] An application uses **SQLGetTypeInfo** or **SQLColAttributes** to determine if a particular data type or a particular column in a result set is unsigned.

C Data Types

Data is stored in the application in ODBC C data types. The Core C Data Types are those that support the minimum and core SQL data types. They also support some extended SQL data types. The extended C data types are those that only support extended SQL data types. The bookmark C data type is used only to retrieve bookmark values and should not be converted to other data types.

Note Unsigned C data types for integers were added to ODBC 2.0. Drivers must support the integer C data types specified in both ODBC 1.0 and ODBC 2.0; ODBC 2.0 or later applications must use the ODBC 1.0 integer C data types with ODBC 1.0 drivers and the ODBC 2.0 integer C data types with ODBC 2.0 drivers.

The C data type is specified in the **SQLBindCol**, **SQLGetData**, and **SQLBindParameter** functions with the *fCType* argument.

Core C Data Types

The following table lists valid values of *fCType* for the core C data types. These values are defined in SQL.H. The table also lists the ODBC C data type that implements each value of *fCType* and the definition of this data type from SQL.H.

fCType	**ODBC C Typedef**	**C Type**
SQL_C_CHAR	UCHAR FAR *	unsigned char FAR *
SQL_C_SSHORT	SWORD	short int
SQL_C_USHORT	UWORD	unsigned short int
SQL_C_SLONG	SDWORD	long int
SQL_C_ULONG	UDWORD	unsigned long int
SQL_C_FLOAT	SFLOAT	float
SQL_C_DOUBLE	SDOUBLE	double

Note   Because objects of the CString class in Microsoft C++ are signed and string arguments in ODBC functions are unsigned, applications that pass CString objects to ODBC functions without casting them will receive compiler warnings.

Extended C Data Types

The following table lists valid values of *fCType* for the extended C data types. These values are defined in SQLEXT.H. The table also lists the ODBC C data type that implements each value of *fCType* and the definition of this data type from SQLEXT.H or SQL.H.

fCType	**ODBC C Typedef**	**C Type**
SQL_C_BIT	UCHAR	unsigned char
SQL_C_STINYINT	SCHAR	signed char
SQL_C_UTINYINT	UCHAR	unsigned char
SQL_C_BINARY	UCHAR FAR *	unsigned char FAR *

SQL_C_DATE	DATE_STRUCT	struct tagDATE_STRUCT {
		SWORD year; [a]
		UWORD month; [b]
		UWORD day; [c]
		}

SQL_C_TIME	TIME_STRUCT	struct tagTIME_STRUCT {
		UWORD hour; [d]
		UWORD minute; [e]
		UWORD second; [f]
		}

SQL_C_TIMESTAMP	TIMESTAMP_STRUCT	struct tagTIMESTAMP_STRUCT {
		SWORD year; [a]
		UWORD month; [b]
		UWORD day; [c]
		UWORD hour; [d]
		UWORD minute; [e]
		UWORD second; [f]
		UDWORD fraction; [g]
		}

a The value of the year field must be in the range from 0 to 9,999. Years are measured from 0 A.D. Some data sources do not support the entire range of years.

b The value of the month field must be in the range from 1 to 12.

c The value of day field must be in the range from 1 to the number of days in the month. The number of days in the month is determined from the values of the year and month fields and is 28, 29, 30, or 31.

d The value of the hour field must be in the range from 0 to 23.

e The value of the minute field must be in the range from 0 to 59.

f The value of the second field must be in the range from 0 to 59.

g The value of the fraction field is the number of billionths of a second and ranges from 0 to 999,999,999 (1 less than 1 billion). For example, the value of the fraction field for a half-second is 500,000,000, for a thousandth of a second (one millisecond) is 1,000,000, for a millionth of a second (one microsecond) is 1,000, and for a billionth of a second (one nanosecond) is 1.

Bookmark C Data Type

Bookmarks are 32-bit values used by an application to return to a specific row; an application retrieves a bookmark either from column 0 of the result set with **SQLExtendedFetch** or **SQLGetData** or by calling **SQLGetStmtOption**.

The following table lists the value of *fCType* for the bookmark C data type, the ODBC C data type that implements the bookmark C data type, and the definition of this data type from SQL.H.

fCType	ODBC C Typedef	C Type
SQL_C_BOOKMARK	BOOKMARK	unsigned long int

ODBC 1.0 C Data Types

In ODBC 1.0, all integer C data types were signed. The following table lists values of *fCType* for the integer C data types that were valid in ODBC 1.0. To remain compatible with applications that use ODBC 1.0, all drivers must support these values of *fCType*. To remain compatible with drivers that use ODBC 1.0, ODBC 2.0 or later applications must pass these values of *fCType* to ODBC 1.0 drivers. However, ODBC 2.0 or later applications must not pass these values to ODBC 2.0 or later drivers.

fCType	ODBC C Typedef	C Type
SQL_C_TINYINT	SCHAR	signed char
SQL_C_SHORT	SWORD	short int
SQL_C_LONG	SDWORD	long int

Because the ODBC 1.0 integer C data types (SQL_C_TINYINT, SQL_C_SHORT, and SQL_C_LONG) are signed, and because the ODBC integer SQL data types can be signed or unsigned, ODBC 1.0 applications and drivers had to interpret signed integer C data as signed or unsigned.

ODBC 2.0 applications and drivers treat the ODBC 1.0 integer C data types as unsigned only when:

- The column from which data will be retrieved is unsigned, and
- The C data type of the storage location in which the data will be placed is the default C data type for that column. (For a list of default C data types, see Default C Data Types.

In all other cases, these applications and drivers treat the ODBC 1.0 integer C data types as signed.

In other words, for any conversion except the default conversion, ODBC 2.0 drivers check the validity of the conversion based on the numeric data value. For the default conversion, the drivers simply pass the data value without attempting to validate it numerically and applications interpret the data value according to whether the column is signed. (Applications call **SQLGetTypeInfo** to determine whether a column is signed or unsigned.)

For example, the following table shows how an ODBC 2.0 driver interprets ODBC 1.0 integer C data sent to both signed and unsigned SQL_SMALLINT columns.

From C Data Type	To SQL Data Type	C Data Values	SQL Data Values
SQL_C_TINYINT	SQL_SMALLINT (signed)	-128 to 127	-128 to 127
	SQL_SMALLINT (unsigned)	< 0 0 to 127	--- [a] 0 to 127
SQL_C_SHORT (default conversion)	SQL_SMALLINT (signed)	-32,768 to 32,767	-32,768 to 32,767
	SQL_SMALLINT (unsigned)	-32,768 to -1 0 to 32,767	32,768 to 65,535 0 to 32,767
SQL_C_LONG	SQL_SMALLINT (signed)	< -32,768 -32,768 to 32,767 > 32,767	--- [a] -32,768 to 32,767 --- [a]
	SQL_SMALLINT (unsigned)	< 0 0 to 32,767 > 32,767	--- [a] 0 to 32,767 --- [a]

(a) The driver returns SQLSTATE 22003 (Numeric value out of range).

Default C Data Types

If an application specifies SQL_C_DEFAULT for the *fCType* argument in **SQLBindCol**, **SQLGetData**, or **SQLBindParameter**, the driver assumes that the C data type of the output or input buffer corresponds to the SQL data type of the column or parameter to which the buffer is bound. For each ODBC SQL data type, the following table shows the corresponding, or *default*, C data type. For information about driver-specific SQL data types, see the driver's documentation.

Note For maximum interoperability, applications should specify a C data type other than SQL_C_DEFAULT. This allows drivers that promote SQL data types (and therefore cannot always determine default C data types) to return data. It also allows drivers that cannot determine whether an integer column is signed or unsigned to correctly return data.

Note ODBC 2.0 drivers use the ODBC 2.0 default C data types for both ODBC 1.0 and ODBC 2.0 integer C data.

SQL Data Type	Default C Data Type
SQL_CHAR	SQL_C_CHAR
SQL_VARCHAR	SQL_C_CHAR
SQL_LONGVARCHAR	SQL_C_CHAR
SQL_DECIMAL	SQL_C_CHAR
SQL_NUMERIC	SQL_C_CHAR
SQL_BIT	SQL_C_BIT
SQL_TINYINT	SQL_C_STINYINT or SQL_C_UTINYINT [a]
SQL_SMALLINT	SQL_C_SSHORT or SQL_C_USHORT [a]
SQL_INTEGER	SQL_C_SLONG or SQL_C_ULONG [a]
SQL_BIGINT	SQL_C_CHAR
SQL_REAL	SQL_C_FLOAT
SQL_FLOAT	SQL_C_DOUBLE
SQL_DOUBLE	SQL_C_DOUBLE
SQL_BINARY	SQL_C_BINARY
SQL_VARBINARY	SQL_C_BINARY
SQL_LONGVARBINARY	SQL_C_BINARY
SQL_DATE	SQL_C_DATE
SQL_TIME	SQL_C_TIME
SQL_TIMESTAMP	SQL_C_TIMESTAMP

a If the driver can determine whether the column is signed or unsigned, such as when the driver is fetching data from the data source or when the data source supports only a signed type or only an unsigned type, but not both, the driver uses the corresponding signed or unsigned C data type. If the driver cannot determine whether the column is signed or unsigned, it passes the data value without attempting to validate it numerically.

Transferring Data in its Binary Form

Among data sources that use the same DBMS, an application can safely transfer data in the internal form used by that DBMS. For a given piece of data, the SQL data types must be the same in the source and target data sources. The C data type is SQL_C_BINARY.

When the application calls **SQLFetch**, **SQLExtendedFetch**, or **SQLGetData** to retrieve the data from the source data source, the driver retrieves the data from the data source and transfers it, without conversion, to a storage location of type SQL_C_BINARY. When the application calls **SQLExecute**, **SQLExecDirect**, or **SQLPutData** to send the data to the target data source, the driver retrieves the data from the storage location and transfers it, without conversion, to the target data source.

Note Applications that transfer any data (except binary data) in this manner are not interoperable among DBMS's.

Precision, Scale, Length, and Display Size

SQLColAttributes, **SQLColumns**, and **SQLDescribeCol** return the precision, scale, length, and display size of a column in a table. **SQLProcedureColumns** returns the precision, scale, and length of a column in a procedure. **SQLDescribeParam** returns the precision or scale of a parameter in an SQL statement; **SQLBindParameter** sets the precision or scale of a parameter in an SQL statement. **SQLGetTypeInfo** returns the maximum precision and the minimum and maximum scales of an SQL data type on a data source.

Due to limitations in the size of the arguments these functions use, precision, length, and display size are limited to the size of an SDWORD, or 2,147,483,647.

Precision

The precision of a numeric column or parameter refers to the maximum number of digits used by the data type of the column or parameter. The precision of a nonnumeric column or parameter generally refers to either the maximum length or the defined length of the column or parameter. To determine the maximum precision allowed for a data type, an application calls **SQLGetTypeInfo**. The following table defines the precision for each ODBC SQL data type.

fSqlType	Precision
SQL_CHAR SQL_VARCHAR	The defined length of the column or parameter. For example, the precision of a column defined as CHAR(10) is 10.
SQL_LONGVARCHAR [a, b]	The maximum length of the column or parameter.
SQL_DECIMAL SQL_NUMERIC	The defined number of digits. For example, the precision of a column defined as NUMERIC(10,3) is 10.
SQL_BIT [c]	1
SQL_TINYINT [c]	3
SQL_SMALLINT [c]	5
SQL_INTEGER [c]	10
SQL_BIGINT [c]	19 (if signed) or 20 (if unsigned)
SQL_REAL [c]	7
SQL_FLOAT [c]	15
SQL_DOUBLE [c]	15
SQL_BINARY SQL_VARBINARY	The defined length of the column or parameter. For example, the precision of a column defined as BINARY(10) is 10.
SQL_LONGVARBINARY [a, b]	The maximum length of the column or parameter.
SQL_DATE [c]	10 (the number of characters in the yyyy-mm-dd format).
SQL_TIME [c]	8 (the number of characters in the hh:mm:ss format).
SQL_TIMESTAMP	The number of characters in the "yyyy-mm-dd hh:mm:ss[.f...]" format used by the TIMESTAMP data type. For example, if a timestamp does not use seconds or fractional seconds, the precision is 16 (the number of characters in the "yyyy-mm-dd hh:mm" format). If a timestamp uses thousandths of a second, the precision is 23 (the number of characters in the "yyyy-mm-dd hh:mm:ss.fff" format).

(a) For an ODBC 1.0 application calling **SQLSetParam** in an ODBC 2.0 driver, and for an ODBC 2.0 application calling **SQLBindParameter** in an ODBC 1.0 driver, when *pcbValue* is SQL_DATA_AT_EXEC, *cbColDef* must be set to the total length of the data to be sent, not the precision as defined in this table.

(b) If the driver cannot determine the column or parameter length, it returns SQL_NO_TOTAL.

(c) The *cbColDef* argument of **SQLBindParameter** is ignored for this data type.

Scale

The scale of a numeric column or parameter refers to the maximum number of digits to the right of the decimal point. For approximate floating point number columns or parameters, the scale is undefined, since the number of digits to the right of the decimal point is not fixed. (For the SQL_DECIMAL and SQL_NUMERIC data types, the maximum scale is generally the same as the maximum precision. However, some data sources impose a separate limit on the maximum scale. To determine the minimum and maximum scales allowed for a data type, an application calls **SQLGetTypeInfo**.) The following table defines the scale for each ODBC SQL data type.

fSqlType	**Scale**
SQL_CHAR [a] SQL_VARCHAR [a] SQL_LONGVARCHAR [a]	Not applicable.
SQL_DECIMAL SQL_NUMERIC	The defined number of digits to the right of the decimal point. For example, the scale of a column defined as NUMERIC(10,3) is 3.
SQL_BIT [a] SQL_TINYINT [a] SQL_SMALLINT [a] SQL_INTEGER [a] SQL_BIGINT [a]	0
SQL_REAL [a] SQL_FLOAT [a] SQL_DOUBLE [a]	Not applicable.
SQL_BINARY [a] SQL_VARBINARY [a] SQL_LONGVARBINARY [a]	Not applicable.
SQL_DATE [a] SQL_TIME [a]	Not applicable.
SQL_TIMESTAMP	The number of digits to the right of the decimal point in the "yyyy-mm-dd hh:mm:ss[.f...]" format. For example, if the TIMESTAMP data type uses the "yyyy-mm-dd hh:mm:ss.fff" format, the scale is 3.

[a] The *ibScale* argument of **SQLBindParameter** is ignored for this data type.

Length

The length of a column is the maximum number of bytes returned to the application when data is transferred to its default C data type. For character data, the length does not include the null termination byte. Note that the length of a column may be different than the number of bytes required to store the data on the data source. For a list of default C data types, see the Default C Data Types.

The following table defines the length for each ODBC SQL data type.

fSqlType	**Length**
SQL_CHAR SQL_VARCHAR	The defined length of the column. For example, the length of a column defined as CHAR(10) is 10.
SQL_LONGVARCHAR [a]	The maximum length of the column.
SQL_DECIMAL SQL_NUMERIC	The maximum number of digits plus 2. Since these data types are returned as character strings, characters are needed for the digits, a sign, and a decimal point. For example, the length of a column defined as NUMERIC(10,3) is 12.
SQL_BIT SQL_TINYINT	1 (one byte).
SQL_SMALLINT	2 (two bytes).
SQL_INTEGER	4 (four bytes).
SQL_BIGINT	20 (since this data type is returned as a character string, characters are needed for 19 digits and a sign, if signed, or 20 digits, if unsigned).
SQL_REAL	4 (four bytes).
SQL_FLOAT	8 (eight bytes).
SQL_DOUBLE	8 (eight bytes).
SQL_BINARY SQL_VARBINARY	The defined length of the column. For example, the length of a column defined as BINARY(10) is 10.
SQL_LONGVARBINARY [a]	The maximum length of the column.
SQL_DATE SQL_TIME	6 (the size of the DATE_STRUCT or TIME_STRUCT structure).
SQL_TIMESTAMP	16 (the size of the TIMESTAMP_STRUCT structure).

[a] If the driver cannot determine the column or parameter length, it returns SQL_NO_TOTAL.

Display Size

The display size of a column is the maximum number of bytes needed to display data in character form. The following table defines the display size for each ODBC SQL data type.

fSqlType	**Display Size**
SQL_CHAR SQL_VARCHAR	The defined length of the column. For example, the display size of a column defined as CHAR(10) is 10.
SQL_LONGVARCHAR [a]	The maximum length of the column.
SQL_DECIMAL SQL_NUMERIC	The precision of the column plus 2 (a sign, *precision* digits, and a decimal point). For example, the display size of a column defined as NUMERIC(10,3) is 12.
SQL_BIT	1 (1 digit).
SQL_TINYINT	4 if signed (a sign and 3 digits) or 3 if unsigned (3 digits).
SQL_SMALLINT	6 if signed (a sign and 5 digits) or 5 if unsigned (5 digits).
SQL_INTEGER	11 if signed (a sign and 10 digits) or 10 if unsigned (10 digits).
SQL_BIGINT	20 (a sign and 19 digits if signed or 20 digits if unsigned).
SQL_REAL	13 (a sign, 7 digits, a decimal point, the letter E, a sign, and 2 digits).
SQL_FLOAT SQL_DOUBLE	22 (a sign, 15 digits, a decimal point, the letter E, a sign, and 3 digits).
SQL_BINARY SQL_VARBINARY	The defined length of the column times 2 (each binary byte is represented by a 2 digit hexadecimal number). For example, the display size of a column defined as BINARY(10) is 20.
SQL_LONGVARBINARY [a]	The maximum length of the column times 2.
SQL_DATE	10 (a date in the format yyyy-mm-dd).
SQL_TIME	8 (a time in the format hh:mm:ss).
SQL_TIMESTAMP	19 (if the scale of the timestamp is 0) or 20 plus the scale of the timestamp (if the scale is greater than 0). This is the number of characters in the "yyyy-mm-dd hh:mm:ss[.f...]" format. For example, the display size of a column storing thousandths of a second is 23 (the number of characters in "yyyy-mm-dd hh:mm:ss.fff").

[a] If the driver cannot determine the column or parameter length, it returns SQL_NO_TOTAL.

Converting Data from SQL to C Data Types

Overview: Converting Data from SQL to C Data Types

When an application calls **SQLExtendedFetch**, **SQLFetch**, or **SQLGetData**, the driver retrieves the data from the data source. If necessary, it converts the data from the data type in which the driver retrieved it to the data type specified by the *fCType* argument in **SQLBindCol** or **SQLGetData**. Finally, it stores the data in the location pointed to by the *rgbValue* argument in **SQLBindCol** or **SQLGetData**.

Note The word *convert* is used in this section in a broad sense, and includes the transfer of data, without a conversion in data type, from one storage location to another.

The tables in the following sections describe how the driver or data source converts data retrieved from the data source; drivers are required to support conversions to all ODBC C data types from the ODBC SQL data types that they support. For a given ODBC SQL data type, the first column of the table lists the legal input values of the *fCType* argument in **SQLBindCol** and **SQLGetData**. The second column lists the outcomes of a test, often using the *cbValueMax* argument specified in **SQLBindCol** or **SQLGetData**, which the driver performs to determine if it can convert the data. For each outcome, the third and fourth columns list the values of the *rgbValue* and *pcbValue* arguments specified in **SQLBindCol** or **SQLGetData** after the driver has attempted to convert the data. The last column lists the SQLSTATE returned for each outcome by **SQLExtendedFetch**, **SQLFetch**, or **SQLGetData**.

If the *fCType* argument in **SQLBindCol** or **SQLGetData** contains a value for an ODBC C data type not shown in the table for a given ODBC SQL data type, **SQLExtendedFetch**, **SQLFetch**, or **SQLGetData** returns SQLSTATE 07006 (Restricted data type attribute violation). If the *fCType* argument contains a value that specifies a conversion from a driver-specific SQL data type to an ODBC C data type and this conversion is not supported by the driver, **SQLExtendedFetch**, **SQLFetch**, or **SQLGetData** returns SQLSTATE S1C00 (Driver not capable).

Though it is not shown in the tables, the *pcbValue* argument contains SQL_NULL_DATA when the SQL data value is NULL. For an explanation of the use of *pcbValue* when multiple calls are made to retrieve data, see **SQLGetData**. When SQL data is converted to character C data, the character count returned in *pcbValue* does not include the null termination byte. If *rgbValue* is a null pointer, **SQLBindCol** or **SQLGetData** returns SQLSTATE S1009 (Invalid argument value).

The following terms and conventions are used in the tables:

- **Length of data** is the number of bytes of C data available to return in *rgbValue*, regardless of whether the data will be truncated before it is returned to the application. For string data, this does not include the null termination byte.
- **Display size** is the total number of bytes needed to display the data in character format.
- Words in *italics* represent function arguments or elements of the ODBC SQL grammar.

SQL to C: Character

The character ODBC SQL data types are:

SQL_CHAR
SQL_VARCHAR
SQL_LONGVARCHAR

The following table shows the ODBC C data types to which character SQL data may be converted.

fCType	Test	rgbValue	pcbValue	SQL-STATE
SQL_C_CHAR	Length of data < *cbValueMax*	Data	Length of data	N/A
	Length of data ≥ *cbValueMax*	Truncated data	Length of data	01004
SQL_C_STINYINT SQL_C_UTINYINT SQL_C_TINYINT [a] SQL_C_SSHORT SQL_C_USHORT SQL_C_SHORT [a] SQL_C_SLONG SQL_C_ULONG SQL_C_LONG [a]	Data converted without truncation [b]	Data	Size of the C data type	N/A
	Data converted with truncation of fractionaldigits [b]	Truncated data	Size of the C data type	01004
	Conversion of data would result in loss of whole (as opposed to fractional)digits [b]	Untouched	Untouched	22003
	Data is not a *numeric-literal* [b]	Untouched	Untouched	22005
SQL_C_FLOAT SQL_C_DOUBLE	Data is within the range of the data type to which the number is being converted [b]	Data	Size of the C data type	N/A
	Data is outside the range of the data type to which the number is being converted [b]	Untouched	Untouched	22003
	Data is not a *numeric-literal* [b]	Untouched	Untouched	22005
SQL_C_BIT	Data is 0 or 1 [a]	Data	1 [c]	N/A
	Data is greater than 0, less than 2, and not equal to 1 [a]	Truncated data	1 [c]	01004
	Data is less than 0 or greater than or equal to 2 [a]	Untouched	Untouched	22003
	Data is not a *numeric-literal* [a]	Untouched	Untouched	22005
SQL_C_BINARY	Length of data ≤ *cbValueMax*	Data	Length of data	N/A
	Length of data > *cbValueMax*	Truncated data	Length of data	01004
SQL_C_DATE	Data value is a valid *date-value* [b]	Data	6 [c]	N/A
	Data value is a valid *timestamp-value*; time portion is zero [b]	Data	6 [c]	N/A
	Data value is a valid *timestamp-value*; time portion is non-zero [b, d]	Truncated data	6 [c]	01004
	Data value is not a valid *date-value* or *timestamp-value* [b]	Untouched	Untouched	22008
SQL_C_TIME	Data value is a valid *time-value* [b]	Data	6 [c]	N/A
	Data value is a valid *timestamp-value*; fractional seconds portion is zero [b, e]	Data	6 [c]	N/A
	Data value is a valid *timestamp-value*; fractional seconds portion is non-zero [b, e, f]	Truncated data	6 [c]	01004
	Data value is not a valid *time-value* or	Untouched	Untouched	22008

	timestamp-value [b]			
SQL_C_TIMESTAMP	Data value is a valid *timestamp-value*; fractional seconds portion not truncated [b]	Data	16 [c]	N/A
	Data value is a valid *timestamp-value*; fractional seconds portion truncated [b]	Truncated data	16 [c]	N/A
	Data value is a valid *date-value* [b]	Data [g]	16 [c]	N/A
	Data value is a valid *time-value* [b]	Data [h]	16 [c]	N/A
	Data value is not a valid *date-value*, *time-value*, or *timestamp-value* [b]	Untouched	Untouched	22008

[a] For more information, see ODBC 1.0 C Data Types, earlier in this appendix.

[b] The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

[c] This is the size of the corresponding C data type.

[d] The time portion of the *timestamp-value* is truncated.

[e] The date portion of the *timestamp-value* is ignored.

[f] The fractional seconds portion of the timestamp is truncated.

[g] The time fields of the timestamp structure are set to zero.

[h] The date fields of the timestamp structure are set to the current date.

When character SQL data is converted to numeric, date, time, or timestamp C data, leading and trailing spaces are ignored.

All drivers that support date, time, and timestamp data can convert character SQL data to date, time, or timestamp C data as specified in the previous table. Drivers may be able to convert character SQL data from other, driver-specific formats to date, time, or timestamp C data. Such conversions are not interoperable among data sources.

SQL to C: Numeric

The numeric ODBC SQL data types are:

SQL_DECIMAL	SQL_BIGINT
SQL_NUMERIC	SQL_REAL
SQL_TINYINT	SQL_FLOAT
SQL_SMALLINT	SQL_DOUBLE
SQL_INTEGER	

The following table shows the ODBC C data types to which numeric SQL data may be converted.

fCType	Test	rgbValue	pcbValue	SQL-STATE
SQL_C_CHAR	Display size < *cbValueMax*	Data	Length of data	N/A
	Number of whole (as opposed to fractional) digits < *cbValueMax*	Truncated data	Length of data	01004
	Number of whole (as opposed to fractional) digits ≥ *cbValueMax*	Untouched	Untouched	22003
SQL_C_STINYINT SQL_C_UTINYINT SQL_C_TINYINT [a] SQL_C_SSHORT SQL_C_USHORT SQL_C_SHORT [a] SQL_C_SLONG SQL_C_ULONG SQL_C_LONG [a]	Data converted without truncation [b]	Data	Size of the C data type	N/A
	Data converted with truncation of fractional digits [b]	Truncated data	Size of the C data type	01004
	Conversion of data would result in loss of whole (as opposed to fractional) digits [b]	Untouched	Untouched	22003
SQL_C_FLOAT SQL_C_DOUBLE	Data is within the range of the data type to which the number is being converted [b]	Data	Size of the C data type	N/A
	Data is outside the range of the data type to which the number is being converted [b]	Untouched	Untouched	22003
SQL_C_BIT	Data is 0 or 1 [b]	Data	1 [c]	N/A
	Data is greater than 0, less than 2, and not equal to 1 [b]	Truncated data	1 [c]	01004
	Data is less than 0 or greater than or equal to 2 [b]	Untouched	Untouched	22003
SQL_C_BINARY	Length of data ≤ *cbValueMax*	Data	Length of data	N/A
	Length of data > *cbValueMax*	Untouched	Untouched	22003

The bit ODBC SQL data type is:

SQL_BIT

The following table shows the ODBC C data types to which bit SQL data may be converted.

fCType	Test	rgbValue	pcbValue	SQL-STATE
SQL_C_CHAR	*cbValueMax* > 1	Data	1	N/A
	cbValueMax ≤ 1	Untouched	Untouched	22003
SQL_C_STINYINT SQL_C_UTINYINT SQL_C_TINYINT [a] SQL_C_SSHORT SQL_C_USHORT SQL_C_SHORT [a] SQL_C_SLONG	None [b]	Data	Size of the C data type	N/A

SQL_C_ULONG SQL_C_LONG [a] SQL_C_FLOAT SQL_C_DOUBLE				
SQL_C_BIT	None [b]	Data	1 [c]	N/A
SQL_C_BINARY	*cbValueMax* $\geq$ 1	Data	1	N/A
	cbValueMax < 1	Untouched	Untouched	22003

[a] For more information, see ODBC 1.0 C Data Types, earlier in this appendix.

[b] The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

[c] This is the size of the corresponding C data type.

When bit SQL data is converted to character C data, the possible values are "0" and "1".

SQL to C: Binary

The binary ODBC SQL data types are:

SQL_BINARY
SQL_VARBINARY
SQL_LONGVARBINARY

The following table shows the ODBC C data types to which binary SQL data may be converted.

fCType	**Test**	**rgbValue**	**pcbValue**	**SQL-STATE**
SQL_C_CHAR	(Length of data) * 2 < *cbValueMax*	Data	Length of data	N/A
	(Length of data) * 2 ≥ *cbValueMax*	Truncated data	Length of data	01004
SQL_C_BINARY	Length of data ≤ *cbValueMax*	Data	Length of data	N/A
	Length of data > *cbValueMax*	Truncated data	Length of data	01004

When binary SQL data is converted to character C data, each byte (8 bits) of source data is represented as two ASCII characters. These characters are the ASCII character representation of the number in its hexadecimal form. For example, a binary 00000001 is converted to "01" and a binary 11111111 is converted to "FF".

The driver always converts individual bytes to pairs of hexadecimal digits and terminates the character string with a null byte. Because of this, if *cbValueMax* is even and is less than the length of the converted data, the last byte of the *rgbValue* buffer is not used. (The converted data requires an even number of bytes, the next-to-last byte is a null byte, and the last byte cannot be used.)

SQL to C: Date

The date ODBC SQL data type is:

SQL_DATE

The following table shows the ODBC C data types to which date SQL data may be converted.

fCType	**Test**	**rgbValue**	**pcbValue**	**SQL-STATE**
SQL_C_CHAR	*cbValueMax* ≥ 11	Data	10	N/A
	cbValueMax < 11	Untouched	Untouched	22003
SQL_C_BINARY	Length of data ≤ *cbValueMax*	Data	Length of data	N/A
	Length of data > *cbValueMax*	Untouched	Untouched	22003
SQL_C_DATE	None [a]	Data	6 [c]	N/A
SQL_C_TIMESTAMP	None [a]	Data [b]	16 [c]	N/A

[a] The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

[b] The time fields of the timestamp structure are set to zero.

[c] This is the size of the corresponding C data type.

When date SQL data is converted to character C data, the resulting string is in the "yyyy-mm-dd" format.

SQL to C: Time

The time ODBC SQL data type is:

SQL_TIME

The following table shows the ODBC C data types to which time SQL data may be converted.

fCType	**Test**	**rgbValue**	**pcbValue**	**SQL-STATE**
SQL_C_CHAR	*cbValueMax* ≥ 9	Data	8	N/A
	cbValueMax < 9	Untouched	Untouched	22003
SQL_C_BINARY	Length of data ≤ *cbValueMax*	Data	Length of data	N/A
	Length of data > *cbValueMax*	Untouched	Untouched	22003
SQL_C_TIME	None [a]	Data	6 [c]	N/A
SQL_C_TIMESTAMP	None [a]	Data [b]	16 [c]	N/A

a The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

b The date fields of the timestamp structure are set to the current date and the fractional seconds field of the timestamp structure is set to zero.

c This is the size of the corresponding C data type.

When time SQL data is converted to character C data, the resulting string is in the "hh:mm:ss" format.

SQL to C: Timestamp

The timestamp ODBC SQL data type is:

SQL_TIMESTAMP

The following table shows the ODBC C data types to which timestamp SQL data may be converted.

fCType	Test	rgbValue	pcbValue	SQL-STATE
SQL_C_CHAR	*cbValueMax* > Display size	Data	Length of data	N/A
	20 ≤ *cbValueMax* ≤ Display size	Truncated data [b]	Length of data	01004
	cbValueMax < 20	Untouched	Untouched	22003
SQL_C_BINARY	Length of data ≤ *cbValueMax*	Data	Length of data	N/A
	Length of data > *cbValueMax*	Untouched	Untouched	22003
SQL_C_DATE	Time portion of timestamp is zero [a]	Data	6 [f]	N/A
	Time portion of timestamp is non-zero [a]	Truncated data [c]	6 [f]	01004
SQL_C_TIME	Fractional seconds portion of timestamp is zero [a]	Data [d]	6 [f]	N/A
	Fractional seconds portion of timestamp is non-zero [a]	Truncated data [d, e]	6 [f]	01004
SQL_C_TIMESTAMP	Fractional seconds portion of timestamp is not truncated [a]	Data [e]	16 [f]	N/A
	Fractional seconds portion of timestamp is truncated [a]	Truncated data [e]		

a The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

b The fractional seconds of the timestamp are truncated.

c The time portion of the timestamp is truncated.

d The date portion of the timestamp is ignored.

e The fractional seconds portion of the timestamp is truncated.

f This is the size of the corresponding C data type.

When timestamp SQL data is converted to character C data, the resulting string is in the "yyyy-mm-dd hh:mm:ss[.f...]" format, where up to nine digits may be used for fractional seconds. (Except for the decimal point and fractional seconds, the entire format must be used, regardless of the precision of the timestamp SQL data type.)

SQL to C Data Conversion Examples

The following examples illustrate how the driver converts SQL data to C data:

SQL Data Type	SQL Data Value	C Data Type	cbValueMax	rgbValue	SQL-STATE
SQL_CHAR	abcdef	SQL_C_CHAR	7	abcdef\0 [a]	N/A
SQL_CHAR	abcdef	SQL_C_CHAR	6	abcde\0 [a]	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	8	1234.56\0 [a]	N/A
SQL_DECIMAL	1234.56	SQL_C_CHAR	5	1234\0 [a]	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	4	----	22003
SQL_DECIMAL	1234.56	SQL_C_FLOAT	ignored	1234.56	N/A
SQL_DECIMAL	1234.56	SQL_C_SSHORT	ignored	1234	01004
SQL_DECIMAL	1234.56	SQL_C_STINYINT	ignored	----	22003
SQL_DOUBLE	1.2345678	SQL_C_DOUBLE	ignored	1.2345678	N/A
SQL_DOUBLE	1.2345678	SQL_C_FLOAT	ignored	1.234567	N/A
SQL_DOUBLE	1.2345678	SQL_C_STINYINT	ignored	1	N/A
SQL_DATE	1992-12-31	SQL_C_CHAR	11	1992-12-31\0 [a]	N/A
SQL_DATE	1992-12-31	SQL_C_CHAR	10	-----	22003
SQL_DATE	1992-12-31	SQL_C_TIMESTAMP	ignored	1992,12,31, 0,0,0,0 [b]	N/A
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	23	1992-12-31 23:45:55.12\0 [a]	N/A
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	22	1992-12-31 23:45:55.1\0 [a]	01004
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	18	----	22003

[a] "\0" represents a null-termination byte. The driver always null-terminates SQL_C_CHAR data.

[b] The numbers in this list are the numbers stored in the fields of the TIMESTAMP_STRUCT structure.

Converting Data from C to SQL Data Types

Overview: Converting Data from C to SQL Data Types

When an application calls **SQLExecute** or **SQLExecDirect**, the driver retrieves the data for any parameters bound with **SQLBindParameter** from storage locations in the application. For data-at-execution parameters, the application sends the parameter data with **SQLPutData**. If necessary, the driver converts the data from the data type specified by the *fCType* argument in **SQLBindParameter** to the data type specified by the *fSqlType* argument in **SQLBindParameter**. Finally, the driver sends the data to the data source.

Note The word *convert* is used in this section in a broad sense, and includes the transfer of data, without a conversion in data type, from one storage location to another.

The tables in the following sections describe how the driver or data source converts data sent to the data source; drivers are required to support conversions from all ODBC C data types to the ODBC SQL data types that they support. For a given ODBC C data type, the first column of the table lists the legal input values of the *fSqlType* argument in **SQLBindParameter**. The second column lists the outcomes of a test that the driver performs to determine if it can convert the data. The third column lists the SQLSTATE returned for each outcome by **SQLExecDirect**, **SQLExecute**, or **SQLPutData**. Data is sent to the data source only if SQL_SUCCESS is returned.

If the *fSqlType* argument in **SQLBindParameter** contains a value for an ODBC SQL data type that is not shown in the table for a given C data type, **SQLBindParameter** returns SQLSTATE 07006 (Restricted data type attribute violation). If the *fSqlType* argument contains a driver-specific value and the driver does not support the conversion from the specific ODBC C data type to that driver-specific SQL data type, **SQLBindParameter** returns SQLSTATE S1C00 (Driver not capable).

If the *rgbValue* and *pcbValue* arguments specified in **SQLBindParameter** are both null pointers, that function returns SQLSTATE S1009 (Invalid argument value). Though it is not shown in the tables, an application sets the value pointed to by the *pcbValue* argument of **SQLBindParameter** or the value of the *cbValue* argument to SQL_NULL_DATA to specify a NULL SQL data value. The application sets these values to SQL_NTS to specify that the value in *rgbValue* is a null-terminated string.

The following terms are used in the tables:

- **Length of data** is the number of bytes of SQL data available to send to the data source, regardless of whether the data will be truncated before it is sent to the data source. For string data, this does not include the null termination byte.
- **Column length** and **display size** are defined for each SQL data type in the section Precision, Scale, Length, and Display Size.
- **Number of digits** is the number of characters used to represent a number, including the minus sign, decimal point, and exponent (if needed).
- Words in *italics* represent elements of the ODBC SQL grammar.

C to SQL: Character

The character ODBC C data type is:

SQL_C_CHAR

The following table shows the ODBC SQL data types to which C character data may be converted.

fSqlType	Test	SQL-STATE
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR	Length of data ≤ Column length	N/A
	Length of data > Column length	01004
SQL_DECIMAL SQL_NUMERIC SQL_TINYINT SQL_SMALLINT SQL_INTEGER SQL_BIGINT	Data converted without truncation	N/A
	Data converted with truncation of fractional digits	01004
	Conversion of data would result in loss of whole (as opposed to fractional) digits	22003
	Data value is not a *numeric-literal*	22005
SQL_REAL SQL_FLOAT SQL_DOUBLE	Data is within the range of the data type to which the number is being converted	N/A
	Data is outside the range of the data type to which the number is being converted	22003
	Data value is not a *numeric-literal*	22005
SQL_BIT	Data is 0 or 1	N/A
	Data is greater than 0, less than 2, and not equal to 1	01004
	Data is less than 0 or greater than or equal to 2	22003
	Data is not a *numeric-literal*	22005
SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY	(Length of data) / 2 ≤ Column length	N/A
	(Length of data) / 2 > Column length	01004
	Data value is not a hexadecimal value	22005
SQL_DATE	Data value is a valid *ODBC-date-literal*	N/A
	Data value is a valid *ODBC-timestamp-literal*; time portion is zero	N/A
	Data value is a valid *ODBC-timestamp-literal*; time portion is non-zero [a]	01004
	Data value is not a valid *ODBC-date-literal* or *ODBC-timestamp-literal*	22008
SQL_TIME	Data value is a valid *ODBC-time-literal*	N/A
	Data value is a valid *ODBC-timestamp-literal*; fractional seconds portion is zero [b]	N/A
	Data value is a valid *ODBC-timestamp-literal*; fractional seconds portion is non-zero [b, c]	01004
	Data value is not a valid *ODBC-time-literal* or *ODBC-timestamp-literal*	22008
SQL_TIMESTAMP	Data value is a valid *ODBC-timestamp-literal*; fractional seconds portion not truncated	N/A
	Data value is a valid *ODBC-timestamp-literal*; fractional seconds portion	01004

truncated	
Data value is a valid *ODBC-date-literal* [d]	N/A
Data value is a valid *ODBC-time-literal* [e]	N/A
Data value is not a valid *ODBC-date-literal*, *ODBC-time-literal*,or *ODBC-timestamp-literal*	22008

a The time portion of the timestamp is truncated.

b The date portion of the timestamp is ignored.

c The fractional seconds portion of the timestamp is truncated.

d The time portion of the timestamp is set to zero.

e The date portion of the timestamp is set to the current date.

When character C data is converted to numeric, date, time, or timestamp SQL data, leading and trailing blanks are ignored.

When character C data is converted to binary SQL data, each two bytes of character data are converted to a single byte (8 bits) of binary data. Each two bytes of character data represent a number in hexadecimal form. For example, "01" is converted to a binary 00000001 and "FF" is converted to a binary 11111111.

The driver always converts pairs of hexadecimal digits to individual bytes and ignores the null termination byte. Because of this, if the length of the character string is odd, the last byte of the string (excluding the null termination byte, if any) is not converted.

All drivers that support date, time, and timestamp data can convert character C data to date, time, or timestamp SQL data as specified in the previous table. Drivers may be able to convert character C data from other, driver-specific formats to date, time, or timestamp SQL data. Such conversions are not interoperable among data sources.

C to SQL: Numeric

The numeric ODBC C data types are:

SQL_C_STINYINT	SQL_C_SLONG
SQL_C_UTINYINT	SQL_C_ULONG
SQL_C_TINYINT	SQL_C_LONG
SQL_C_SSHORT	SQL_C_FLOAT
SQL_C_USHORT	SQL_C_DOUBLE
SQL_C_SHORT	

For more information about the SQL_C_TINYINT, SQL_C_SHORT, and SQL_C_LONG data types, see ODBC 1.0 C Data Types, earlier in this appendix. The following table shows the ODBC SQL data types to which numeric C data may be converted.

fSqlType	**Test**	**SQL-STATE**
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR	Number of digits ≤ Column length	N/A
	Number of whole (as opposed to fractional) digits ≤ Column length	01004
	Number of whole (as opposed to fractional) digits > Column length	22003
SQL_DECIMAL SQL_NUMERIC SQL_TINYINT SQL_SMALLINT SQL_INTEGER SQL_BIGINT	Data converted without truncation	N/A
	Data converted with truncation of fractional digits	01004
	Conversion of data would result in loss of whole (as opposed to fractional) digits	22003
SQL_REAL SQL_FLOAT SQL_DOUBLE	Data is within the range of the data type to which the number is being converted	N/A
	Data is outside the range of the data type to which the number is being converted	22003
SQL_BIT	Data is 0 or 1	N/A
	Data is greater than 0, less than 2, and not equal to 1	01004
	Data is less than 0 or greater than or equal to 2	22003

The value pointed to by the *pcbValue* argument of **SQLBindParameter** and the value of the *cbValue* argument of **SQLPutData** are ignored when data is converted from the numeric C data types. The driver assumes that the size of *rgbValue* is the size of the numeric C data type.

C to SQL: Bit

The bit ODBC C data type is:

SQL_C_BIT

The following table shows the ODBC SQL data types to which bit C data may be converted.

fSqlType	Test	SQL-STATE
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR	None	N/A
SQL_DECIMAL SQL_NUMERIC SQL_TINYINT SQL_SMALLINT SQL_INTEGER SQL_BIGINT SQL_REAL SQL_FLOAT SQL_DOUBLE	None	N/A
SQL_BIT	None	N/A

The value pointed to by the *pcbValue* argument of **SQLBindParameter** and the value of the *cbValue* argument of **SQLPutData** are ignored when data is converted from the bit C data type. The driver assumes that the size of *rgbValue* is the size of the bit C data type.

C to SQL: Binary

The binary ODBC C data type is:

SQL_C_BINARY

The following table shows the ODBC SQL data types to which binary C data may be converted.

fSqlType	Test	SQL-STATE
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR	Length of data ≤ Column length Length of data > Column length	N/A 01004
SQL_DECIMAL SQL_NUMERIC SQL_TINYINT SQL_SMALLINT SQL_INTEGER SQL_BIGINT SQL_REAL SQL_FLOAT SQL_DOUBLE	Length of data = SQL data length [a] Length of data ≠ SQL data length [a]	N/A 22003
SQL_BIT	Length of data = SQL data length [a] Length of data ≠ SQL data length [a]	N/A 22003
SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY	Length of data ≤ Column length Length of data > Column length	N/A 01004
SQL_DATE SQL_TIME SQL_TIMESTAMP	Length of data = SQL data length [a] Length of data ≠ SQL data length [a]	N/A 22003

[a] The SQL data length is the number of bytes needed to store the data on the data source. (This may be different than the column length, as defined earlier in this appendix.)

C to SQL: Date

The date ODBC C data type is:

SQL_C_DATE

The following table shows the ODBC SQL data types to which date C data may be converted.

fSqlType	Test	SQL-STATE
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR	Column length ≥ 10	N/A
	Column length < 10	22003
	Data value is not a valid date	22008
SQL_DATE	Data value is a valid date	N/A
	Data value is not a valid date	22008
SQL_TIMESTAMP	Data value is a valid date [a]	N/A
	Data value is not a valid date	22008

[a] The time portion of the timestamp is set to zero.

For information about what values are valid in a SQL_C_DATE structure, see Extended C Data Types, earlier in this appendix.

When date C data is converted to character SQL data, the resulting character data is in the "yyyy-mm-dd" format.

The value pointed to by the *pcbValue* argument of **SQLBindParameter** and the value of the *cbValue* argument of **SQLPutData** are ignored when data is converted from the date C data type. The driver assumes that the size of *rgbValue* is the size of the date C data type.

C to SQL: Time

The time ODBC C data type is:

SQL_C_TIME

The following table shows the ODBC SQL data types to which time C data may be converted.

fSqlType	Test	SQL-STATE
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR	Column length ≥ 8	N/A
	Column length < 8	22003
	Data value is not a valid time	22008
SQL_TIME	Data value is a valid time	N/A
	Data value is not a valid time	22008
SQL_TIMESTAMP	Data value is a valid time [a]	N/A
	Data value is not a valid time	22008

[a] The date portion of the timestamp is set to the current date and the fractional seconds portion of the timestamp is set to zero.

For information about what values are valid in a SQL_C_TIME structure, see Extended C Data Types, earlier in this appendix.

When time C data is converted to character SQL data, the resulting character data is in the "hh:mm:ss" format.

The value pointed to by the *pcbValue* argument of **SQLBindParameter** and the value of the *cbValue* argument of **SQLPutData** are ignored when data is converted from the time C data type. The driver assumes that the size of *rgbValue* is the size of the time C data type.

C to SQL: Timestamp

The timestamp ODBC C data type is:

SQL_C_TIMESTAMP

The following table shows the ODBC SQL data types to which timestamp C data may be converted.

fSqlType	**Test**	**SQL-STATE**
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR	Column length ≥ Display size	N/A
	19 ≤ Column length < Display size [a]	01004
	Column length < 19	22003
	Data value is not a valid date	22008
SQL_DATE	Time fields are zero	N/A
	Time fields are non-zero [b]	01004
	Data value does not contain a valid date	22008
SQL_TIME	Fractional seconds fields are zero [c]	N/A
	Fractional seconds fields are non-zero [c, d]	01004
	Data value does not contain a valid time	22008
SQL_TIMESTAMP	Fractional seconds fields are not truncated	N/A
	Fractional seconds fields are truncated [d]	01004
	Data value is not a valid timestamp	22008

[a] The fractional seconds of the timestamp are truncated.

[b] The time fields of the timestamp structure are truncated.

[c] The date fields of the timestamp structure are ignored.

[d] The fractional seconds fields of the timestamp structure are truncated.

For information about what values are valid in a SQL_C_TIMESTAMP structure, see Extended C Data Types, earlier in this appendix.

When timestamp C data is converted to character SQL data, the resulting character data is in the "yyyy-mm-dd hh:mm:ss[.f...]" format.

The value pointed to by the *pcbValue* argument of **SQLBindParameter** and the value of the *cbValue* argument of **SQLPutData** are ignored when data is converted from the timestamp C data type. The driver assumes that the size of *rgbValue* is the size of the timestamp C data type.

C to SQL Data Conversion Examples

The following examples illustrate how the driver converts C data to SQL data:

C DataType	C Data Value	SQL Data Type	Column length	SQL Data Value	SQL-STATE
SQL_C_CHAR	abcdef\0 [a]	SQL_CHAR	6	abcdef	N/A
SQL_C_CHAR	abcdef\0 [a]	SQL_CHAR	5	abcde	01004
SQL_C_CHAR	1234.56\0 [a]	SQL_DECIMAL	8 [b]	1234.56	N/A
SQL_C_CHAR	1234.56\0 [a]	SQL_DECIMAL	7 [b]	1234.5	01004
SQL_C_CHAR	1234.56\0 [a]	SQL_DECIMAL	4	----	22003
SQL_C_FLOAT	1234.56	SQL_FLOAT	not applicable	1234.56	N/A
SQL_C_FLOAT	1234.56	SQL_INTEGER	not applicable	1234	01004
SQL_C_FLOAT	1234.56	SQL_TINYINT	not applicable	----	22003
SQL_C_DATE	1992,12,31 [c]	SQL_CHAR	10	1992-12-31	N/A
SQL_C_DATE	1992,12,31 [c]	SQL_CHAR	9	----	22003
SQL_C_DATE	1992,12,31 [c]	SQL_TIMESTAMP	not applicable	1992-12-31 00:00:00.0	N/A
SQL_C_TIMESTAMP	1992,12,31, 23,45,55, 120000000 [d]	SQL_CHAR	22	1992-12-31 23:45:55.12	N/A
SQL_C_TIMESTAMP	1992,12,31, 23,45,55, 120000000 [d]	SQL_CHAR	21	1992-12-31 23:45:55.1	01004
SQL_C_TIMESTAMP	1992,12,31, 23,45,55, 120000000 [d]	SQL_CHAR	18	----	22003

[a] "\0" represents a null-termination byte. The null-termination byte is required only if the length of the data is SQL_NTS.

[b] In addition to bytes for numbers, one byte is required for a sign and another byte is required for the decimal point.

[c] The numbers in this list are the numbers stored in the fields of the DATE_STRUCT structure.

[d] The numbers in this list are the numbers stored in the fields of the TIMESTAMP_STRUCT structure.