



Microsoft Data Access Components 2.5 SDK

PART II – Microsoft ODBC API Specification

ODBC API Functions

This PDF file contains contents from the Microsoft® Data Access SDK online *ODBC Programmer's Reference* at the following web site:

<http://msdn.microsoft.com/isapi/msdnlib2.idc?theURL=/library/psdk/dasdk/odin8w4s.htm>

For your convenience, the main sections of the Microsoft documentation are provided in three PDF files that are available on the SOLID Web site:

- The Part I Microsoft ODBC API Specification PDF file contains introductory information on ODBC and information on developing applications and drivers.
- The Part II Microsoft ODBC API Specification PDF file contains the ODBC API Reference, descriptions of each ODBC function.
- The Part III Microsoft ODBC API Specification PDF file contains information on installing and configuring ODBC software, and setup, installer, and translation DLL functions.

NOTE: Refer to Chapter 2, “Using SOLID ODBC API,” in the **SOLID Programmer Guide** for SOLID-specific usage. This PDF file also refers you to SOLID manuals for information on SOLID usage and to the Microsoft Web site (noted above) for those portions of the documentation that are not included in the PDF file.

Copyright © 2000 Microsoft Corporation. All rights reserved.

Information in this document is subject to change without notice and does not represent a commitment on the part of Solid Information Technology.

Contents

Chapter 20: Function Summary	4
ODBC Function Summary	4
ODBC API Reference	8
SQLAllocConnect	9
SQLAllocEnv	9
SQLAllocHandle	9
SQLAllocStmt	16
SQLBindCol	17
SQLBindParameter	28
SQLBrowseConnect	47
SQLBulkOperations	55
SQLCancel	70
SQLCloseCursor	73
SQLColAttribute	75
SQLColumnPrivileges	86
SQLColumns	92
SQLConnect	102
SQLCopyDesc	110
SQLDataSources	115
SQLDescribeParam	119
SQLDescribeCol	124
SQLDisconnect	128
SQLDriverConnect	130
SQLDrivers	141
SQLEndTran	144
SQLError	149
SQLExecDirect	150
SQLExecute	160
SQLExtendedFetch	168
SQLFetch	174
SQLFetchScroll	184
SQLForeignKeys	198
SQLFreeConnect	209
SQLFreeEnv	210
SQLFreeHandle	211
SQLFreeStmt	214
SQLGetConnectAttr	217
SQLGetConnectOption	220
SQLGetCursorName	221
SQLGetData	223
SQLGetDescField	234
SQLGetDescRec	238
SQLGetDiagField	242
SQLGetDiagRec	252
SQLGetEnvAttr	255
SQLGetFunctions	258
SQLGetInfo	262
SQLGetStmtAttr	319

SQLGetStmtOption	323
SQLGetTypeInfo	323
SQLMoreResults	332
SQLNativeSql	336
SQLNumParams	338
SQLNumResultCols	341
SQLParamData	343
SQLParamOptions	347
SQLPrepare	348
SQLPrimaryKeys	354
SQLProcedureColumns	359
SQLProcedures	368
SQLPutData	374
SQLRowCount	381
SQLSetConnectAttr	383
SQLSetConnectOption	395
SQLSetCursorName	395
SQLSetDescField	398
SQLSetDescRec	421
SQLSetEnvAttr	426
SQLSetParam	430
SQLSetPos	430
SQLSetScrollOptions	445
SQLSetStmtAttr	446
SQLSetStmtOption	466
SQLSpecialColumns	466
SQLStatistics	474
SQLTablePrivileges	481
SQLTables	487
SQLTransact	493

Chapter 20: Function Summary

This chapter summarizes the functions used by ODBC-enabled applications and related software.

ODBC 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. For more information about conformance designations, see the Part I PDF file, “ODBC and the Standard CLI” (Chapter 1, “Introduction to ODBC”). For more information about the syntax and semantics for each function, “ODBC API Reference.”

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**.

Task	Function name	Conformance	Purpose
Connecting to a data source			
	SQLAllocHandle	ISO 92 ODBC	Returns the list of available data sources. Returns the list of installed drivers and their attributes.
	SQLGetInfo	ISO 92	Returns information about a specific driver and data source.
	SQLGetFunctions	ISO 92	Returns supported driver functions.
	SQLGetTypeInfo	ISO 92	Returns information about supported data types.
Setting and retrieving driver attributes			
	SQLSetConnectAttr	ISO 92	Sets a connection attribute.
	SQLGetConnectAttr	ISO 92	Returns the value of a connection attribute.
	SQLSetEnvAttr	ISO 92	Sets an environment attribute.
	SQLGetEnvAttr	ISO 92	Returns the value of an environment attribute.
	SQLSetStmtAttr	ISO 92	Sets a statement attribute.
	SQLGetStmtAttr	ISO 92	Returns the value of a statement attribute.
Setting and retrieving descriptor fields			
	SQLGetDescField	ISO 92	Returns the value of a single descriptor field.

	SQLGetDescRec	ISO 92	Returns the values of multiple descriptor fields.
	SQLSetDescField	ISO 92	Sets a single descriptor field.
	SQLSetDescRec	ISO 92	Sets multiple descriptor fields.
Preparing SQL requests			
	SQLPrepare	ISO 92	Prepares an SQL statement for later execution.
	SQLBindParameter	ODBC	Assigns storage for a parameter in an SQL statement.
	SQLGetCursorName	ISO 92	Returns the cursor name associated with a statement handle.
	SQLSetCursorName	ISO 92	Specifies a cursor name.
	SQLSetScrollOptions	ODBC	Sets options that control cursor behavior.
Submitting requests			
	SQLExecute	ISO 92	Executes a prepared statement.
	SQLExecDirect	ISO 92	Executes a statement.
	SQLNativeSql	ODBC	Returns the text of an SQL statement as translated by the driver.
	SQLDescribeParam	ODBC	Returns the description for a specific parameter in a statement.
	SQLNumParams	ISO 92	Returns the number of parameters in a statement.
	SQLParamData	ISO 92	Used in conjunction with SQLPutData to supply parameter data at execution time. (Useful for long data values.)
	SQLPutData	ISO 92	Sends part or all of a data value for a parameter. (Useful for long data values.)
Retrieving results and information about results			
	SQLRowCount	ISO 92	Returns the number of rows affected by an insert, update, or delete request.
	SQLNumResultCols	ISO 92	Returns the number of columns in the result set.
	SQLDescribeCol	ISO 92	Describes a column in the result set.

	SQLColAttribute	ISO 92	Describes attributes of a column in the result set.
	SQLBindCol	ISO 92	Assigns storage for a result column and specifies the data type.
	SQLFetch	ISO 92	Returns multiple result rows.
	SQLFetchScroll	ISO 92	Returns scrollable result rows.
	SQLGetData	ISO 92	Returns part or all of one column of one row of a result set. (Useful for long data values.)
	SQLSetPos	ODBC	Positions a cursor within a fetched block of data and allows an application to refresh data in the rowset or to update or delete data in the result set.
	SQLBulkOperations	ODBC	Performs bulk insertions and bulk bookmark operations, including update, delete, and fetch by bookmark.
	SQLMoreResults	ODBC	Determines whether there are more result sets available and, if so, initializes processing for the next result set.
	SQLGetDiagField	ISO 92	Returns additional diagnostic information (a single field of the diagnostic data structure).
	SQLGetDiagRec	ISO 92	Returns additional diagnostic information (multiple fields of the diagnostic data structure).
Obtaining information about the data source's system tables (catalog functions)			
	SQLColumnPrivileges	ODBC	Returns a list of columns and associated privileges for one or more tables.
	SQLColumns	X/Open	Returns the list of column names in specified tables.
	SQLForeignKeys	ODBC	Returns a list of column names that make up foreign keys, if they exist for a specified table.
	SQLPrimaryKeys	ODBC	Returns the list of column names that make up the primary key for a table.
	SQLProcedureColumns	ODBC	Returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures.

	SQLProcedures	ODBC	Returns the list of procedure names stored in a specific data source.
	SQLSpecialColumns	X/Open	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	ISO 92	Returns statistics about a single table and the list of indexes associated with the table.
	SQLTablePrivileges	ODBC	Returns a list of tables and the privileges associated with each table.
	SQLTables	X/Open	Returns the list of table names stored in a specific data source.
Terminating a statement			
	SQLFreeStmt	ISO 92	Ends statement processing, discards pending results, and, optionally, frees all resources associated with the statement handle.
	SQLCloseCursor	ISO 92	Closes a cursor that has been opened on a statement handle.
	SQLCancel	ISO 92	Cancels an SQL statement.
	SQLEndTran	ISO 92	Commits or rolls back a transaction.
Terminating a connection			
	SQLDisconnect	ISO 92	Closes the connection.
	SQLFreeHandle	ISO 92	Releases an environment, connection, statement, or descriptor handle.

ODBC API Reference

The following topics describe each ODBC function in alphabetical order. Each function is defined as a C programming language function. Descriptions include the following:

- Purpose
- ODBC version
- Standard CLI conformance level
- Syntax
- Arguments
- Return values
- Diagnostics
- Comments about usage and implementation
- Code example
- References to related functions

The standard CLI conformance level can be one of the following: ISO 92, X/Open, ODBC, or Deprecated. A function tagged as ISO 92-conformant also appears in X/Open version 1, because X/Open is a pure superset of ISO 92. A function tagged as X/Open-compliant also appears in ODBC 3.x, because ODBC 3.x is a pure superset of X/Open version 1. A function tagged as ODBC-compliant appears in neither standard. A function tagged as deprecated has been deprecated in ODBC 3.x.

Handling of diagnostic information is described in the [SQLGetDiagField](#) function description. The text associated with SQLSTATE values is included to provide a description of the condition but is not intended to prescribe specific text.

SQLAllocConnect

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: Deprecated

Summary

In ODBC 3.x, the ODBC 2.x function **SQLAllocConnect** has been replaced by **SQLAllocHandle**. For more information, see **SQLAllocHandle**.

Note For more information about what the Driver Manager maps this function to when an ODBC 2.x application is working with an ODBC 3.x driver, see the "Mapping Deprecated Functions" (Appendix G, "Driver Guidelines for Backward Compatibility") contained on the Microsoft Web site (ODBC Programmers Reference).

SQLAllocEnv

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: Deprecated

Summary

In ODBC 3.x, the ODBC 2.x function **SQLAllocEnv** has been replaced by **SQLAllocHandle**. For more information, see **SQLAllocHandle** in the following section.

Note For more information about what the Driver Manager maps this function to when an ODBC 2.x application is working with an ODBC 3.x driver, see "Mapping Deprecated Functions" (Appendix G, "Driver Guidelines for Backward Compatibility") contained on the Microsoft Web site (ODBC Programmer's Reference).

SQLAllocHandle

Conformance

Version Introduced: ODBC 3.0
Standards Compliance: ISO 92

Summary

SQLAllocHandle allocates an environment, connection, statement, or descriptor handle.

Note This function is a generic function for allocating handles that replaces the ODBC 2.0 functions **SQLAllocConnect**, **SQLAllocEnv**, and **SQLAllocStmt**. To allow applications calling **SQLAllocHandle** to work with ODBC 2.x drivers, a call to **SQLAllocHandle** is mapped in the Driver Manager to **SQLAllocConnect**, **SQLAllocEnv**, or **SQLAllocStmt**, as appropriate. For more information, see "Comments." For more information about what the Driver Manager maps this function to when an ODBC 3.x application is working with an ODBC 2.x driver, see the Part I PDF file "Mapping Replacement Functions for Backward Compatibility" in Chapter 17, "Programming Considerations."

Syntax

SQLRETURN SQLAllocHandle(
SQLSMALLINT	HandleType,
SQLHANDLE	InputHandle,
SQLHANDLE *	OutputHandlePtr);

Arguments

HandleType

[Input]

The type of handle to be allocated by **SQLAllocHandle**. Must be one of the following values:

SQL_HANDLE_ENV
SQL_HANDLE_DBC
SQL_HANDLE_STMT
SQL_HANDLE_DESC

InputHandle

[Input]

The input handle in whose context the new handle is to be allocated. If *HandleType* is SQL_HANDLE_ENV, this is SQL_NULL_HANDLE. If *HandleType* is SQL_HANDLE_DBC, this must be an environment handle, and if it is SQL_HANDLE_STMT or SQL_HANDLE_DESC, it must be a connection handle.

OutputHandlePtr

[Output]

Pointer to a buffer in which to return the handle to the newly allocated data structure.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, or SQL_ERROR.

When allocating a handle other than an environment handle, if **SQLAllocHandle** returns SQL_ERROR, it sets *OutputHandlePtr* to SQL_NULL_HDBC, SQL_NULL_HSTMT, or SQL_NULL_HDESC, depending on the value of *HandleType*, unless the output argument is a null pointer. The application can then obtain additional information from the diagnostic data structure associated with the handle in the *InputHandle* argument.

Environment Handle Allocation Errors

Environment allocation occurs both within the Driver Manager and within each driver. The error returned by **SQLAllocHandle** with a *HandleType* of SQL_HANDLE_ENV depends on the level in which the error occurred.

If the Driver Manager cannot allocate memory for **OutputHandlePtr* when **SQLAllocHandle** with a *HandleType* of SQL_HANDLE_ENV is called, or the application provides a null pointer for *OutputHandlePtr*, **SQLAllocHandle** returns SQL_ERROR. The Driver Manager sets **OutputHandlePtr* to

SQL_NULL_HENV (unless the application provided a null pointer, which returns SQL_ERROR). There is no handle with which to associate additional diagnostic information. (If the driver has additional diagnostic information, it will put the information on a skeletal handle that it allocates; the Driver Manager will read the information from the diagnostic structure associated with this handle.)

The Driver Manager does not call the driver-level environment handle allocation function until the application calls **SQLConnect**, **SQLBrowseConnect**, or **SQLDriverConnect**. If an error occurs in the driver-level **SQLAllocHandle** function, then the Driver Manager-level **SQLConnect**, **SQLBrowseConnect**, or **SQLDriverConnect** function returns SQL_ERROR. The diagnostic data structure contains SQLSTATE IM004 (Driver's **SQLAllocHandle** failed), followed by a driver-specific SQLSTATE value from the driver. For example, SQLSTATE HY001 (Memory allocation error) indicates that the Driver Manager's call to the driver-level **SQLAllocHandle** returned SQL_ERROR. The error is returned on a connection handle.

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

Diagnostics

When **SQLAllocHandle** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLGetDiagRec** with the appropriate *HandleType* and *Handle* set to the value of *InputHandle*. SQL_SUCCESS_WITH_INFO (but not SQL_ERROR) can be returned for the *OutputHandle* argument. The following table lists the SQLSTATE values commonly returned by **SQLAllocHandle** 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 does not exist	(DM) The <i>HandleType</i> argument was SQL_HANDLE_STMT or SQL_HANDLE_DESC, but the connection specified by the <i>InputHandle</i> argument was not open. The connection process must be completed successfully (and the connection must be open) for the driver to allocate a statement or descriptor handle.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	(DM) The Driver Manager was unable to allocate memory for the specified handle. The driver was unable to allocate memory for the specified handle.
HY009	Invalid use of null pointer	(DM) The <i>OutputHandlePtr</i> argument was a null pointer.
HY010	Function sequence error	(DM) The <i>HandleType</i> argument was SQL_HANDLE_DBC, and SQLSetEnvAttr has not

		been called to set the SQL_ODBC_VERSION environment attribute.
HY013	Memory management error	The <i>HandleType</i> argument was SQL_HANDLE_DBC, SQL_HANDLE_STMT, or SQL_HANDLE_DESC; and the function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY014	Limit on the number of handles exceeded	The driver-defined limit for the number of handles that can be allocated for the type of handle indicated by the <i>HandleType</i> argument has been reached.
HY092	Invalid attribute/option identifier	(DM) The <i>HandleType</i> argument was not: SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_STMT, or SQL_HANDLE_DESC.
HYC00	Optional feature not implemented	The <i>HandleType</i> argument was SQL_HANDLE_DESC and the driver was an ODBC 2.x driver.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The <i>HandleType</i> argument was SQL_HANDLE_STMT, and the driver was not a valid ODBC driver. (DM) The <i>HandleType</i> argument was SQL_HANDLE_DESC, and the driver does not support allocating a descriptor handle.

Comments

SQLAllocHandle is used to allocate handles for environments, connections, statements, and descriptors, as described in the following sections. For general information about handles, see the Part I PDF file, "Handles" in Chapter 4, "ODBC Fundamentals."

More than one environment, connection, or statement handle can be allocated by an application at a time if multiple allocations are supported by the driver. In ODBC, no limit is defined on the number of environment, connection, statement, or descriptor handles that can be allocated at any one time. Drivers may impose a limit on the number of a certain type of handle that can be allocated at a time; for more information, see the driver documentation.

If the application calls **SQLAllocHandle** with **OutputHandlePtr* set to an environment, connection, statement, or descriptor handle that already exists, the driver overwrites the information associated with the *handle*, unless the application is using connection pooling (see "Allocating an Environment Attribute for Connection Pooling" later in this section). The Driver Manager does not check to see whether the *handle* entered in **OutputHandlePtr* is already in use, nor does it check the previous contents of a handle before overwriting them.

Note It is incorrect ODBC application programming to call **SQLAllocHandle** twice with the same application variable defined for **OutputHandlePtr* without calling **SQLFreeHandle** to free the handle

before reallocating it. Overwriting ODBC handles in such a manner can lead to inconsistent behavior or errors on the part of ODBC drivers.

On operating systems that support multiple threads, applications can use the same environment, connection, statement, or descriptor handle on different threads. Drivers must therefore support safe, multithread access to this information; one way of achieving this, for example, is through the use of a critical section or a semaphore. For more information about threading, see the Part I PDF file, "Multithreading" in Chapter 17, "Programming Considerations."

SQLAllocHandle does not set the `SQL_ATTR_ODBC_VERSION` environment attribute when it is called to allocate an environment handle; the environment attribute must be set by the application, or `SQLSTATE HY010` (Function sequence error) will be returned when **SQLAllocHandle** is called to allocate a connection handle.

For standards-compliant applications, **SQLAllocHandle** is mapped to **SQLAllocHandleStd** at compile time. The difference between these two functions is that **SQLAllocHandleStd** sets the `SQL_ATTR_ODBC_VERSION` environment attribute to `SQL_OV_ODBC3` when it is called with the *HandleType* argument set to `SQL_HANDLE_ENV`. This is done because standards-compliant applications are always ODBC 3.x applications. Moreover, the standards do not require the application version to be registered. This is the only difference between these two functions; otherwise, they are identical.

Allocating an Environment Handle

An environment handle provides access to global information such as valid connection handles and active connection handles. For general information about environment handles, see Part I PDF file, "Environment Handles," in Chapter 4, "ODBC Fundamentals."

To request an environment handle, an application calls **SQLAllocHandle** with a *HandleType* of `SQL_HANDLE_ENV` and an *InputHandle* of `SQL_NULL_HANDLE`. The driver allocates memory for the environment information and passes the value of the associated handle back in the **OutputHandlePtr* argument. The application passes the **OutputHandle* value in all subsequent calls that require an environment handle argument. For more information, see Part I PDF file, "Allocating the Environment Handle" in Chapter 6, "Connecting to a Data Source or Driver."

Under a Driver Manager's environment handle, if there already exists a driver's environment handle, then **SQLAllocHandle** with a *HandleType* of `SQL_HANDLE_ENV` is not called in that driver when a connection is made, only **SQLAllocHandle** with a *HandleType* of `SQL_HANDLE_DBC`. If a driver's environment handle does not exist under the Driver Manager's environment handle, then both **SQLAllocHandle** with a *HandleType* of `SQL_HANDLE_ENV` and **SQLAllocHandle** with a *HandleType* of `SQL_HANDLE_DBC` are called in the driver when the first connection handle of the environment is connected to the driver.

When the Driver Manager processes the **SQLAllocHandle** function with a *HandleType* of `SQL_HANDLE_ENV`, it checks the **Trace** keyword in the [ODBC] section of the system information. If it is set to 1, the Driver Manager enables tracing for the current application on a computer running Microsoft® Windows® 95/98, Microsoft Windows NT® Server/Windows 2000 Server, or Microsoft Windows NT Workstation/Windows 2000 Professional. If the trace flag is set, tracing starts when the first environment handle is allocated and ends when the last environment handle is freed. For more information, see the Part III PDF file, Chapter 19, Configuring Data Sources."

After allocating an environment handle, an application must call **SQLSetEnvAttr** on the environment handle to set the `SQL_ATTR_ODBC_VERSION` environment attribute. If this attribute is not set before **SQLAllocHandle** is called to allocate a connection handle on the environment, the call to allocate the

connection will return SQLSTATE HY010 (Function sequence error). For more information, see the Part I PDF file, “Declaring the Application’s ODBC Version” in Chapter 6, “Connecting to a Data Source or Driver.”

Allocating Shared Environments for Connection Pooling

Environments can be shared among multiple components on a single process. A shared environment can be used by more than one component simultaneously. When a component uses a shared environment, it can use pooled connections, which allow it to allocate and use an existing connection without re-creating that connection.

Before allocating a shared environment to be used for connection pooling, an application must call **SQLSetEnvAttr** to set the SQL_ATTR_CONNECTION_POOLING environment attribute to SQL_CP_ONE_PER_DRIVER or SQL_CP_ONE_PER_HENV. **SQLSetEnvAttr** in this case is called with *EnvironmentHandle* set to null, which makes the attribute a process-level attribute.

After connection pooling has been enabled, an application calls **SQLAllocHandle** with the *HandleType* argument set to SQL_HANDLE_ENV. The environment allocated by this call will be an implicit shared environment because connection pooling has been enabled. (For more information about connection pooling, see **SQLConnect**.)

When a shared environment is allocated, the environment to be used is not determined until **SQLAllocHandle** with a *HandleType* of SQL_HANDLE_DBC is called. At that point, the Driver Manager attempts to find an existing environment that matches the environment attributes requested by the application. If no such environment exists, one is created as a shared environment. The Driver Manager maintains a reference count for each shared environment; the count is set to 1 when the environment is first created. If a matching environment is found, the handle of that environment is returned to the application and the reference count is incremented. An environment handle allocated this way can be used in any ODBC function that accepts an environment handle as an input argument.

Allocating a Connection Handle

A connection handle provides access to information such as the valid statement and descriptor handles on the connection and whether a transaction is currently open. For general information about connection handles, see the Part I PDF file, “Connection Handles,” in Chapter 4, “ODBC Fundamentals.”

To request a connection handle, an application calls **SQLAllocHandle** with a *HandleType* of SQL_HANDLE_DBC. The *InputHandle* argument is set to the environment handle that was returned by the call to **SQLAllocHandle** that allocated that handle. The driver allocates memory for the connection information and passes the value of the associated handle back in **OutputHandlePtr*. The application passes the **OutputHandlePtr* value in all subsequent calls that require a connection handle. For more information, see the Part I PDF file, “Allocating the Connection Handle” in Chapter 6, “Connecting to a Data Source or Driver.”

The Driver Manager processes the **SQLAllocHandle** function and calls the driver’s **SQLAllocHandle** function when the application calls **SQLConnect**, **SQLBrowseConnect**, or **SQLDriverConnect**. (For more information, see **SQLConnect**.)

If the SQL_ATTR_ODBC_VERSION environment attribute is not set before **SQLAllocHandle** is called to allocate a connection handle on the environment, the call to allocate the connection will return SQLSTATE HY010 (Function sequence error).

When an application calls **SQLAllocHandle** with the *InputHandle* argument set to **SQL_HANDLE_DBC** and also set to a shared environment handle, the Driver Manager attempts to find an existing shared environment that matches the environment attributes set by the application. If no such environment exists, one is created, with a reference count (maintained by the Driver Manager) of 1. If a matching shared environment is found, that handle is returned to the application and its reference count is incremented.

The actual connection to be used is not determined by the Driver Manager until **SQLConnect** or **SQLDriverConnect** is called. The Driver Manager uses the connection options in the call to **SQLConnect** (or the connection keywords in the call to **SQLDriverConnect**) and the connection attributes set after connection allocation to determine which connection in the pool should be used. For more information, see **SQLConnect**.

Allocating a Statement Handle

A statement handle provides access to statement information, such as error messages, the cursor name, and status information for SQL statement processing. For general information about statement handles, see the Part I PDF file, "Statement Handle" in Chapter 4, "ODBC Fundamentals."

To request a statement handle, an application connects to a data source and then calls **SQLAllocHandle** prior to submitting SQL statements. In this call, *HandleType* should be set to **SQL_HANDLE_STMT** and *InputHandle* should be set to the connection handle that was returned by the call to **SQLAllocHandle** that allocated that handle. The driver allocates memory for the statement information, associates the statement handle with the specified connection, and passes the value of the associated handle back in **OutputHandlePtr*. The application passes the **OutputHandlePtr* value in all subsequent calls that require a statement handle. For more information, see the Part I PDF file "Allocating a Statement Handle" in Chapter 9, "Executing Statements."

When the statement handle is allocated, the driver automatically allocates a set of four descriptors and assigns the handles for these descriptors to the **SQL_ATTR_APP_ROW_DESC**, **SQL_ATTR_APP_PARAM_DESC**, **SQL_ATTR_IMP_ROW_DESC**, and **SQL_ATTR_IMP_PARAM_DESC** statement attributes. These are called *implicitly* allocated descriptors. To allocate an application descriptor explicitly, see the following section, "Allocating a Descriptor Handle."

Allocating a Descriptor Handle

When an application calls **SQLAllocHandle** with a *HandleType* of **SQL_HANDLE_DESC**, the driver allocates an application descriptor. These are called *explicitly* allocated descriptors. The application directs a driver to use an explicitly allocated application descriptor in place of an automatically allocated one for a given statement handle by calling the **SQLSetStmtAttr** function with the **SQL_ATTR_APP_ROW_DESC** or **SQL_ATTR_APP_PARAM_DESC** attribute. An implementation descriptor cannot be allocated explicitly, nor can an implementation descriptor be specified in an **SQLSetStmtAttr** function call.

Explicitly allocated descriptors are associated with a connection handle rather than a statement handle (as automatically allocated descriptors are). Descriptors remain allocated only when an application is actually connected to the database. Because explicitly allocated descriptors are associated with a connection handle, an application can associate an explicitly allocated descriptor with more than one statement within a connection. An implicitly allocated application descriptor, on the other hand, cannot be associated with more than one statement handle. (It cannot be associated with any statement handle other than the one that it was allocated for.) Explicitly allocated descriptor handles can be freed explicitly either by the application or by calling **SQLFreeHandle** with a *HandleType* of **SQL_HANDLE_DESC**, or implicitly when the connection is closed.

When the explicitly allocated descriptor is freed, the implicitly allocated descriptor is once again associated with the statement. (The SQL_ATTR_APP_ROW_DESC or SQL_ATTR_APP_PARAM_DESC attribute for that statement is once again set to the implicitly allocated descriptor handle.) This is true for all statements that were associated with the explicitly allocated descriptor on the connection.

For more information about descriptors, see the Part I PDF file, Chapter 13, “Overview of Descriptors.”

Code Example

See [SQLBrowseConnect](#), [SQLConnect](#), and [SQLSetCursorName](#).

Related Functions

For information about	See
Executing an SQL statement	SQLExecDirect
Executing a prepared SQL statement	SQLExecute
Freeing an environment, connection, statement, or descriptor handle	SQLFreeHandle
Preparing a statement for execution	SQLPrepare
Setting a connection attribute	SQLSetConnectAttr
Setting a descriptor field	SQLSetDescField
Setting an environment attribute	SQLSetEnvAttr
Setting a statement attribute	SQLSetStmtAttr

SQLAllocStmt

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: Deprecated

Summary

In ODBC 3.x, the ODBC 2.x function **SQLAllocStmt** has been replaced by **SQLAllocHandle**. For more information, see [SQLAllocHandle](#).

Note For more information about what the Driver Manager maps this function to when an ODBC 2.x application is working with an ODBC 3.x driver, see “Mapping Deprecated Functions” (Appendix G, “Driver Guidelines for Backward Compatibility”) contained on the Microsoft Web site (ODBC Programmer” Reference.

SQLBindCol

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

SQLBindCol binds application data buffers to columns in the result set.

Syntax

SQLRETURN SQLBindCol (
SQLHSTMT	StatementHandle,
SQLUSMALLINT	ColumnNumber,
SQLSMALLINT	TargetType,
SQLPOINTER	TargetValuePtr,
SQLINTEGER	BufferLength,
SQLINTEGER *	StrLen_or_IndPtr);

Arguments

StatementHandle

[Input]

Statement handle.

ColumnNumber

[Input]

Number of the result set column to bind. Columns are numbered in increasing column order starting at 0, where column 0 is the bookmark column. If bookmarks are not used—that is, the SQL_ATTR_USE_BOOKMARKS statement attribute is set to SQL_UB_OFF—then column numbers start at 1.

TargetType

[Input]

The identifier of the C data type of the **TargetValuePtr* buffer. When retrieving data from the data source with **SQLFetch**, **SQLFetchScroll**, **SQLBulkOperations**, or **SQLSetPos**, the driver converts the data to

this type; when sending data to the data source with **SQLBulkOperations** or **SQLSetPos**, the driver converts the data from this type.

For a list of valid C data types and type identifiers, see the "C Data Types" section in Appendix D of the **SOLID Programmer Guide**.

If the *TargetType* argument is an interval data type, the default interval leading precision (2) and the default interval seconds precision (6), as set in the **SQL_DESC_DATETIME_INTERVAL_PRECISION** and **SQL_DESC_PRECISION** fields of the ARD, respectively, are used for the data. If the *TargetType* argument is **SQL_C_NUMERIC**, the default precision (driver-defined) and default scale (0), as set in the **SQL_DESC_PRECISION** and **SQL_DESC_SCALE** fields of the ARD, are used for the data. If any default precision or scale is not appropriate, the application should explicitly set the appropriate descriptor field by a call to **SQLSetDescField** or **SQLSetDescRec**.

TargetValuePtr

[Deferred Input/Output]

Pointer to the data buffer to bind to the column. **SQLFetch** and **SQLFetchScroll** return data in this buffer. **SQLBulkOperations** returns data in this buffer when *Operation* is **SQL_FETCH_BY_BOOKMARK**; it retrieves data from this buffer when *Operation* is **SQL_ADD** or **SQL_UPDATE_BY_BOOKMARK**. **SQLSetPos** returns data in this buffer when *Operation* is **SQL_REFRESH**; it retrieves data from this buffer when *Operation* is **SQL_UPDATE**.

If *TargetValuePtr* is a null pointer, the driver unbinds the data buffer for the column. An application can unbind all columns by calling **SQLFreeStmt** with the **SQL_UNBIND** option. An application can unbind the data buffer for a column but still have a length/indicator buffer bound for the column, if the *TargetValuePtr* argument in the call to **SQLBindCol** is a null pointer but the *StrLen_or_IndPtr* argument is a valid value.

BufferLength

[Input]

Length of the **TargetValuePtr* buffer in bytes.

The driver uses *BufferLength* to avoid writing past the end of the **TargetValuePtr* buffer when returning variable-length data, such as character or binary data. Note that the driver counts the null-termination character when returning character data to **TargetValuePtr*. **TargetValuePtr* must therefore contain space for the null-termination character or the driver will truncate the data.

When the driver returns fixed-length data, such as an integer or a date structure, the driver ignores *BufferLength* and assumes the buffer is large enough to hold the data. It is therefore important for the application to allocate a large enough buffer for fixed-length data or the driver will write past the end of the buffer.

SQLBindCol returns SQLSTATE HY090 (Invalid string or buffer length) when *BufferLength* is less than 0 but not when *BufferLength* is 0. However, if *TargetType* specifies a character type, an application should not set *BufferLength* to 0, because ISO CLI-compliant drivers return SQLSTATE HY090 (Invalid string or buffer length) in that case.

StrLen_or_IndPtr

[Deferred Input/Output]

Pointer to the length/indicator buffer to bind to the column. **SQLFetch** and **SQLFetchScroll** return a value in this buffer. **SQLBulkOperations** retrieves a value from this buffer when *Operation* is **SQL_ADD**,

SQL_UPDATE_BY_BOOKMARK, or SQL_DELETE_BY_BOOKMARK. **SQLBulkOperations** returns a value in this buffer when *Operation* is SQL_FETCH_BY_BOOKMARK. **SQLSetPos** returns a value in this buffer when *Operation* is SQL_REFRESH; it retrieves a value from this buffer when *Operation* is SQL_UPDATE.

SQLFetch, **SQLFetchScroll**, **SQLBulkOperations**, and **SQLSetPos** can return the following values in the length/indicator buffer:

The length of the data available to return

SQL_NO_TOTAL

SQL_NULL_DATA

The application can place the following values in the length/indicator buffer for use with **SQLBulkOperations** or **SQLSetPos**:

The length of the data being sent

SQL_NTS

SQL_NULL_DATA

SQL_DATA_AT_EXEC

The result of the SQL_LEN_DATA_AT_EXEC macro

SQL_COLUMN_IGNORE

If the indicator buffer and the length buffer are separate buffers, the indicator buffer can return only SQL_NULL_DATA, while the length buffer can return all other values.

For more information, see [SQLBulkOperations](#), [SQLFetch](#), [SQLSetPos](#), and the Part I PDF file, "Using Length/Indicator Values" section in Chapter 4, "ODBC Fundamentals."

If *StrLen_or_IndPtr* is a null pointer, no length or indicator value is used. This is an error when fetching data and the data is NULL.

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 **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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.)
07006	Restricted data type attribute violation	(DM) The <i>ColumnNumber</i> argument was 0, and the <i>TargetType</i> argument was not SQL_C_BOOKMARK or SQL_C_VARBOOKMARK.
07009	Invalid descriptor index	The value specified for the argument <i>ColumnNumber</i> exceeded the maximum number of columns in the result set.
HY000	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 SQLGetDiagRec in the * <i>MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY003	Invalid application buffer type	The argument <i>TargetType</i> was neither a valid data type nor SQL_C_DEFAULT.
HY010	Function sequence error	(DM) An asynchronously executing function was called for the <i>StatementHandle</i> and was still executing when this function was called. (DM) SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	(DM) The value specified for the argument <i>BufferLength</i> was less than 0. (DM) The driver was an ODBC 2.x driver, the <i>ColumnNumber</i> argument was set to 0, and the value specified for the argument <i>BufferLength</i> was not equal to 4.
HYC00	Optional feature not implemented	The driver or data source does not support the conversion specified by the combination of the <i>TargetType</i> argument and the driver-specific SQL data type of the corresponding column. The argument <i>ColumnNumber</i> was 0 and the driver does not support bookmarks. The driver supports only ODBC 2.x and the argument <i>TargetType</i> was one of the following: SQL_C_GUID SQL_C_NUMERIC SQL_C_SBIGINT SQL_C_UBIGINT and any of the interval C data types listed in "C Data Types" in Appendix D of the SOLID Programmer Guide .
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr ,

		SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

SQLBindCol is used to associate, or *bind*, columns in the result set to data buffers and length/indicator buffers in the application. When the application calls **SQLFetch**, **SQLFetchScroll**, or **SQLSetPos** to fetch data, the driver returns the data for the bound columns in the specified buffers; for more information, see [SQLFetch](#). When the application calls **SQLBulkOperations** to update or insert a row or **SQLSetPos** to update a row, the driver retrieves the data for the bound columns from the specified buffers; for more information, see [SQLBulkOperations](#) or [SQLSetPos](#). For more information about binding, see the Part I PDF file, Chapter 10, "Overview of Retrieving Results (Basic)."

Note that columns do not have to be bound to retrieve data from them. An application can also call **SQLGetData** to retrieve data from columns. Although it is possible to bind some columns in a row and call **SQLGetData** for others, this is subject to some restrictions. For more information, see [SQLGetData](#).

Binding, Unbinding, and Rebinding Columns

A column can be bound, unbound, or rebound at any time, even after data has been fetched from the result set. The new binding takes effect the next time a function that uses bindings is called. For example, suppose an application binds the columns in a result set and calls **SQLFetch**. The driver returns the data in the bound buffers. Now suppose the application binds the columns to a different set of buffers. The driver does not place the data for the just-fetched row in the newly bound buffers. Instead, it waits until **SQLFetch** is called again and then places the data for the next row in the newly bound buffers.

Note The statement attribute SQL_ATTR_USE_BOOKMARKS should always be set before binding a column to column 0. This is not required but is strongly recommended.

Binding Columns

To bind a column, an application calls **SQLBindCol** and passes the column number, type, address, and length of a data buffer, and the address of a length/indicator buffer. For information about how these addresses are used, see "Buffer Addresses," later in this section. For more information about binding columns, see the Part I PDF file, "Using SQLBindCol" in Chapter 10, "Retrieving Results (Basic)."

The use of these buffers is deferred; that is, the application binds them in **SQLBindCol** but the driver accesses them from other functions—namely **SQLBulkOperations**, **SQLFetch**, **SQLFetchScroll**, or **SQLSetPos**. It is the application's responsibility to make sure that the pointers specified in **SQLBindCol** remain valid as long as the binding remains in effect. If the application allows these pointers to become invalid—for example, it frees a buffer—and then calls a function that expects them to be valid, the consequences are undefined. For more information, see the Part I PDF file, "Deferred Buffers" in Chapter 4, "ODBC Fundamentals."

The binding remains in effect until it is replaced by a new binding, the column is unbound, or the statement is freed.

Unbinding Columns

To unbind a single column, an application calls **SQLBindCol** with *ColumnNumber* set to the number of that column and *TargetValuePtr* set to a null pointer. If *ColumnNumber* refers to an unbound column, **SQLBindCol** still returns `SQL_SUCCESS`.

To unbind all columns, an application calls **SQLFreeStmt** with *fOption* set to `SQL_UNBIND`. This also can be accomplished by setting the `SQL_DESC_COUNT` field of the ARD to zero.

Rebinding Columns

An application can perform either of two operations to change a binding:

- Call **SQLBindCol** to specify a new binding for a column that is already bound. The driver overwrites the old binding with the new one.
- Specify an offset to be added to the buffer address that was specified by the binding call to **SQLBindCol**. For more information, see the next section, "Binding Offsets."

Binding Offsets

A binding offset is a value that is added to the addresses of the data and length/indicator buffers (as specified in the *TargetValuePtr* and *StrLen_or_IndPtr* argument) before they are dereferenced. When offsets are used, the bindings are a "template" of how the application's buffers are laid out, and the application can move this "template" to different areas of memory by changing the offset. Because the same offset is added to each address in each binding, the relative offsets between buffers for different columns must be the same within each set of buffers. This is always true when row-wise binding is used; the application must carefully lay out its buffers for this to be true when column-wise binding is used.

Using a binding offset has much the same effect as rebinding a column by calling **SQLBindCol**. The difference is that a new call to **SQLBindCol** specifies new addresses for the data buffer and length/indicator buffer, while use of a binding offset does not change the addresses but merely adds an offset to them. The application can specify a new offset whenever it wants, and this offset is always added to the originally bound addresses. In particular, if the offset is set to 0 or if the statement attribute is set to a null pointer, the driver uses the originally bound addresses.

To specify a binding offset, the application sets the `SQL_ATTR_ROW_BIND_OFFSET_PTR` statement attribute to the address of an `SQLINTEGER` buffer. Before the application calls a function that uses bindings, it places an offset in bytes in this buffer. To determine the address of the buffer to use, the driver adds the offset to the address in the binding. The sum of the address and the offset must be a valid address, but the address to which the offset is added need not be valid. For more information about how binding offsets are used, see "Buffer Addresses," later in this section.

Binding Arrays

If the rowset size (the value of the `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute) is greater than 1, the application binds arrays of buffers rather than single buffers. For more information, see the Part I PDF file, "Block Cursors" in Chapter 11, "Retrieving Results (Advanced)."

The application can bind arrays in two ways:

- Bind an array to each column. This is called *column-wise binding* because each data structure (array) contains data for a single column.

- Define a structure to hold the data for an entire row and bind an array of these structures. This is called *row-wise binding* because each data structure contains the data for a single row.

Each array of buffers must have at least as many elements as the size of the rowset.

Note An application must verify that alignment is valid. For more information about alignment considerations, see the Part I PDF file, “Alignment” in Chapter 17, “Programming Considerations.”

Column-Wise Binding

In column-wise binding, the application binds separate data and length/indicator arrays to each column.

To use column-wise binding, the application first sets the `SQL_ATTR_ROW_BIND_TYPE` statement attribute to `SQL_BIND_BY_COLUMN`. (This is the default.) For each column to be bound, the application performs the following steps:

- Allocates a data buffer array.
- Allocates an array of length/indicator buffers.

Note If the application writes directly to descriptors when column-wise binding is used, separate arrays can be used for length and indicator data.

Calls `SQLBindCol` with the following arguments:

- *TargetType* is the type of a single element in the data buffer array.
- *TargetValuePtr* is the address of the data buffer array.
- *BufferLength* is the size of a single element in the data buffer array. The *BufferLength* argument is ignored when the data is fixed-length data.
- *StrLen_or_IndPtr* is the address of the length/indicator array.

For more information about how this information is used, see “Buffer Addresses,” later in this section. For more information about column-wise binding, see the Part I PDF file, “Column-Wise Binding” in Chapter 11, “Retrieving Results (Advanced).”

Row-Wise Binding

In row-wise binding, the application defines a structure containing data and length/indicator buffers for each column to be bound.

To use row-wise binding, the application performs the following steps:

- Defines a structure to hold a single row of data (including both data and length/indicator buffers) and allocates an array of these structures.

Note If the application writes directly to descriptors when row-wise binding is used, separate fields can be used for length and indicator data.

- Sets the `SQL_ATTR_ROW_BIND_TYPE` statement attribute to the size of the structure containing a single row of data or to the size of an instance of a buffer into which the results columns will be bound. The length must include space for all of the bound columns, and any padding of the structure or buffer, to make sure 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 in ANSI C, this behavior is guaranteed.

- Calls **SQLBindCol** with the following arguments for each column to be bound:
- *TargetType* is the type of the data buffer member to be bound to the column.
- *TargetValuePtr* is the address of the data buffer member in the first array element.
- *BufferLength* is the size of the data buffer member.
- *StrLen_or_IndPtr* is the address of the length/indicator member to be bound.

For more information about how this information is used, see "Buffer Addresses," later in this section. For more information about column-wise binding, see the Part I PDF file, "Row-Wise Binding" in Chapter 11, "Retrieving Results (Advanced)."

Buffer Addresses

The *buffer address* is the actual address of the data or length/indicator buffer. The driver calculates the buffer address just prior to writing to the buffers (such as during fetch time). It is calculated from the following formula, which uses the addresses specified in the *TargetValuePtr* and *StrLen_or_IndPtr* arguments, the binding offset, and the row number:

Bound Address + Binding Offset + ((Row Number – 1) x Element Size)

where the formula's variables are defined as described in the following table.

Variable	Description
Bound Address	<p>For data buffers, the address specified with the <i>TargetValuePtr</i> argument in SQLBindCol.</p> <p>For length/indicator buffers, the address specified with the <i>StrLen_or_IndPtr</i> argument in SQLBindCol. For more information, see "Additional Comments" in the "Descriptors and SQLBindCol" section.</p> <p>If the bound address is 0, no data value is returned, even if the address as calculated by the previous formula is nonzero.</p>
Binding Offset	<p>If row-wise binding is used, the value stored at the address specified with the SQL_ATTR_ROW_BIND_OFFSET_PTR statement attribute.</p> <p>If column-wise binding is used or if the value of the SQL_ATTR_ROW_BIND_OFFSET_PTR statement attribute is a null pointer, <i>Binding Offset</i> is 0.</p>
Row Number	The 1-based number of the row in the rowset. For single-row fetches, which are the default, this is 1.
Element Size	<p>The size of an element in the bound array.</p> <p>If column-wise binding is used, this is sizeof(SQLINTEGER) for length/indicator buffers. For data buffers, it is the value of the <i>BufferLength</i> argument in SQLBindCol if the data type is variable length, and the size of the data type if the data type is fixed length.</p>

	If row-wise binding is used, this is the value of the <code>SQL_ATTR_ROW_BIND_TYPE</code> statement attribute for both data and length/indicator buffers.
--	-----------------------------------------------------------------------------------------------------------------------------------------------------------

Descriptors and SQLBindCol

The following sections describe how **SQLBindCol** interacts with descriptors.

Caution Calling **SQLBindCol** for one statement can affect other statements. This occurs when the ARD associated with the statement is explicitly allocated and is also associated with other statements. Because **SQLBindCol** modifies the descriptor, the modifications apply to all statements with which this descriptor is associated. If this is not the required behavior, the application should dissociate this descriptor from the other statements before calling **SQLBindCol**.

Argument Mappings

Conceptually, **SQLBindCol** performs the following steps in sequence:

- Calls **SQLGetStmtAttr** to obtain the ARD handle.
- Calls **SQLGetDescField** to get this descriptor's `SQL_DESC_COUNT` field, and if the value in the *ColumnNumber* argument exceeds the value of `SQL_DESC_COUNT`, calls **SQLSetDescField** to increase the value of `SQL_DESC_COUNT` to *ColumnNumber*.
- Calls **SQLSetDescField** multiple times to assign values to the following fields of the ARD:
- Sets `SQL_DESC_TYPE` and `SQL_DESC_CONCISE_TYPE` to the value of *TargetType*, except that if *TargetType* is one of the concise identifiers of a datetime or interval subtype, it sets `SQL_DESC_TYPE` to `SQL_DATETIME` or `SQL_INTERVAL`, respectively; sets `SQL_DESC_CONCISE_TYPE` to the concise identifier; and sets `SQL_DESC_DATETIME_INTERVAL_CODE` to the corresponding datetime or interval subcode.
- Sets one or more of `SQL_DESC_LENGTH`, `SQL_DESC_PRECISION`, `SQL_DESC_SCALE`, and `SQL_DESC_DATETIME_INTERVAL_PRECISION`, as appropriate for *TargetType*.
- Sets the `SQL_DESC_OCTET_LENGTH` field to the value of *BufferLength*.
- Sets the `SQL_DESC_DATA_PTR` field to the value of *TargetValue*.
- Sets the `SQL_DESC_INDICATOR_PTR` field to the value of *StrLen_or_Ind*. (See the following paragraph.)
- Sets the `SQL_DESC_OCTET_LENGTH_PTR` field to the value of *StrLen_or_Ind*. (See the following paragraph.)

The variable that the *StrLen_or_Ind* argument refers to is used for both indicator and length information. If a fetch encounters a null value for the column, it stores `SQL_NULL_DATA` in this variable; otherwise, it stores the data length in this variable. Passing a null pointer as *StrLen_or_Ind* keeps the fetch operation from returning the data length but makes the fetch fail if it encounters a null value and has no way to return `SQL_NULL_DATA`.

If the call to **SQLBindCol** fails, the content of the descriptor fields it would have set in the ARD are undefined and the value of the SQL_DESC_COUNT field of the ARD is unchanged.

Implicit Resetting of COUNT Field

SQLBindCol sets SQL_DESC_COUNT to the value of the *ColumnNumber* argument only when this would increase the value of SQL_DESC_COUNT. If the value in the *TargetValuePtr* argument is a null pointer and the value in the *ColumnNumber* argument is equal to SQL_DESC_COUNT (that is, when unbinding the highest bound column), then SQL_DESC_COUNT is set to the number of the highest remaining bound column.

Cautions Regarding SQL_DEFAULT

To retrieve column data successfully, the application must determine correctly the length and starting point of the data in the application buffer. When the application specifies an explicit *TargetType*, application misconceptions are readily detected. However, when the application specifies a *TargetType* of SQL_DEFAULT, **SQLBindCol** can be applied to a column of a different data type from the one intended by the application, either from changes to the metadata or by applying the code to a different column. In this case, the application may fail to determine the start or length of the fetched column data. This can lead to unreported data errors or memory violations.

Code Example

In the following example, an application executes a **SELECT** statement on the Customers table to return a result set of the customer IDs, names, and phone numbers, sorted by name. It then calls **SQLBindCol** to bind the columns of data to local buffers. Finally, the application fetches each row of data with **SQLFetch** and prints each customer's name, ID, and phone number.

For more code examples, see [SQLBulkOperations](#), [SQLColumns](#), [SQLFetchScroll](#), and [SQLSetPos](#).

```
#define NAME_LEN 50
#define PHONE_LEN 10

SQLCHAR szName[NAME_LEN], szPhone[PHONE_LEN];
SQLINTEGER sCustID, cbName, cbCustID, cbPhone;
SQLHSTMT hstmt;
SQLRETURN retcode;

retcode = SQLExecDirect(hstmt,
    "SELECT CUSTID, NAME, PHONE FROM CUSTOMERS ORDER BY 2, 1, 3",
    SQL_NTS);
if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
    /* Bind columns 1, 2, and 3 */
    SQLBindCol(hstmt, 1, SQL_C_ULONG, &sCustID, 0, &cbCustID);
    SQLBindCol(hstmt, 2, SQL_C_CHAR, szName, NAME_LEN, &cbName);
    SQLBindCol(hstmt, 3, SQL_C_CHAR, szPhone, PHONE_LEN, &cbPhone);
    /* 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 %-5d %s", NAME_LEN-1, szName,
                sCustID, PHONE_LEN-1, szPhone);
        }
    }
}
```

```

} else {
break;
}
}
}

```

Related Functions

For information about	See
Returning information about a column in a result set	SQLDescribeParam
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Fetching multiple rows of data	SQLFetch
Releasing column buffers on the statement	SQLFreeStmt
Fetching part or all of a column of data	SQLGetData
Returning the number of result set columns	SQLNumResultCols

SQLBindParameter

Conformance

Version Introduced: ODBC 2.0

Standards Compliance: ODBC

Summary

SQLBindParameter binds a buffer to a parameter marker in an SQL statement. **SQLBindParameter** supports binding to a Unicode C data type, even if the underlying driver does not support Unicode data.

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

Syntax

SQLRETURN SQLBindParameter (
SQLHSTMT	<i>StatementHandle</i> ,
SQLUSMALLINT	<i>ParameterNumber</i> ,
SQLSMALLINT	<i>InputOutputType</i> ,
SQLSMALLINT	<i>ValueType</i> ,
SQLSMALLINT	<i>ParameterType</i> ,
SQLUIINTEGER	<i>ColumnSize</i> ,
SQLSMALLINT	<i>DecimalDigits</i> ,
SQLPOINTER	<i>ParameterValuePtr</i> ,
SQLINTEGER	<i>BufferLength</i> ,
SQLINTEGER *	<i>StrLen_or_IndPtr</i>);

Arguments

StatementHandle

[Input]
Statement handle.

ParameterNumber

[Input]
Parameter number, ordered sequentially in increasing parameter order, starting at 1.

InputOutputType

[Input]
The type of the parameter. For more information, see "*InputOutputType* Argument" in "Comments."

ValueType

[Input]
The C data type of the parameter. For more information, see "*ValueType* Argument" in "Comments."

ParameterType

[Input]
The SQL data type of the parameter. For more information, see "*ParameterType* Argument" in "Comments."

ColumnSize

[Input]
The size of the column or expression of the corresponding parameter marker. For more information, see "*ColumnSize* Argument" in "Comments."

DecimalDigits

[Input]
The decimal digits of the column or expression of the corresponding parameter marker. For further information concerning column size, see "Decimal Digits," in Appendix D, "Data Types" of the **SOLID Programmer Guide**.

ParameterValuePtr

[Deferred Input]
A pointer to a buffer for the parameter's data. For more information, see "*ParameterValuePtr* Argument" in "Comments."

BufferLength

[Input/Output]
Length of the *ParameterValuePtr* buffer in bytes. For more information, see "*BufferLength* Argument" in "Comments."

StrLen_or_IndPtr

[Deferred Input]
A pointer to a buffer for the parameter's length. For more information, see "*StrLen_or_IndPtr* 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 **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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 type identified by the <i>ValueType</i> argument cannot be converted to the data type identified by the <i>ParameterType</i> argument. Note that this error may be returned by SQLExecDirect , SQLExecute , or SQLPutData at execution time, instead of by SQLBindParameter .
07009	Invalid descriptor index	(DM) The value specified for the argument <i>ParameterNumber</i> was less than 1.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY003	Invalid application buffer type	The value specified by the argument <i>ValueType</i> was not a valid C data type or SQL_C_DEFAULT.
HY004	Invalid SQL data type	The value specified for the argument <i>ParameterType</i> was neither a valid ODBC SQL data type identifier nor a driver-specific SQL data type identifier supported by the driver.
HY009	Invalid use of null pointer	(DM) The argument <i>ParameterValuePtr</i> was a null pointer, the argument <i>StrLen_or_IndPtr</i> was a null pointer, and the argument <i>InputOutputType</i> was not SQL_PARAM_OUTPUT.
HY010	Function sequence error	(DM) An asynchronously executing function was called for the <i>StatementHandle</i> and was still executing when this function was called. (DM) SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY021	Inconsistent descriptor information	The descriptor information checked during a consistency check was not consistent. (See the "Consistency Checks" section in SQLSetDescField .) The value specified for the argument <i>DecimalDigits</i> was outside the range of values supported by the data source for a column of the SQL data type specified by the <i>ParameterType</i> argument.
HY090	Invalid string or buffer	(DM) The value in <i>BufferLength</i> was less than 0. (See

	length	the description of the SQL_DESC_DATA_PTR field in SQLSetDescField .)
HY104	Invalid precision or scale value	The value specified for the argument <i>ColumnSize</i> or <i>DecimalDigits</i> was outside the range of values supported by the data source for a column of the SQL data type specified by the <i>ParameterType</i> argument.
HY105	Invalid parameter type	(DM) The value specified for the argument <i>InputOutputType</i> was invalid. (See "Comments.")
HYC00	Optional feature not implemented	<p>The driver or data source does not support the conversion specified by the combination of the value specified for the argument <i>ValueType</i> and the driver-specific value specified for the argument <i>ParameterType</i>.</p> <p>The value specified for the argument <i>ParameterType</i> was a valid ODBC SQL data type identifier for the version of ODBC supported by the driver but was not supported by the driver or data source.</p> <p>The driver supports only ODBC 2.x and the argument <i>ValueType</i> was one of the following:</p> <p>SQL_C_GUID SQL_C_NUMERIC SQL_C_SBIGINT SQL_C_UBIGINT</p> <p>and all of the interval C data types listed in "C Data Types" in Appendix D, "Data Types" of the SOLID Programmer Guide.</p>
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

An application calls **SQLBindParameter** to bind each parameter marker in an SQL statement. Bindings remain in effect until the application calls **SQLBindParameter** again, calls **SQLFreeStmt** with the SQL_RESET_PARAMS option, or calls **SQLSetDescField** to set the SQL_DESC_COUNT header field of the APD to 0.

For more information about parameters, see the Part 1 PDF file, Chapter 9, "Executing Statements." For more information concerning parameter data types and parameter markers, see "Parameter Data Types" and "Parameter Markers" in Appendix C, "SQL Minimum Grammar" of the **SOLID Programmer Guide**.

ParameterNumber Argument

If *ParameterNumber* in the call to **SQLBindParameter** is greater than the value of `SQL_DESC_COUNT`, **SQLSetDescField** is called to increase the value of `SQL_DESC_COUNT` to *ParameterNumber*.

InputOutputType Argument

The *InputOutputType* argument specifies the type of the parameter. This argument sets the `SQL_DESC_PARAMETER_TYPE` field of the IPD. 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 whose type cannot be determined are assumed to be input parameters.)

The *InputOutputType* 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. For example, the parameters in **INSERT INTO Employee VALUES (?, ?, ?)** are input parameters, while the parameters in **{call AddEmp(?, ?, ?)}** can be, but are not necessarily, input parameters.

When the statement is executed, the driver sends data for the parameter to the data source; the **ParameterValuePtr* buffer must contain a valid input value, or the **StrLen_or_IndPtr* 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 *InputOutputType* 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 **ParameterValuePtr* buffer must contain a valid input value, or the **StrLen_or_IndPtr* 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 **StrLen_or_IndPtr* 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 *InputOutputType* 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; in either case, these are 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 *ParameterValuePtr* and *StrLen_or_IndPtr* 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 **StrLen_or_IndPtr* buffer to `SQL_NULL_DATA`.

ValueType Argument

The *ValueType* argument specifies the C data type of the parameter. This argument sets the SQL_DESC_TYPE, SQL_DESC_CONCISE_TYPE, and SQL_DESC_DATETIME_INTERVAL_CODE fields of the APD. This must be one of the values in the “C Data Types” section of Appendix D, “Data Types” in the **SOLID Programmer Guide**.

If the *ValueType* argument is one of the interval data types, the SQL_DESC_TYPE field of the *ParameterNumber* record of the APD is set to SQL_INTERVAL, the SQL_DESC_CONCISE_TYPE field of the APD is set to the concise interval data type, and the SQL_DESC_DATETIME_INTERVAL_CODE field of the *ParameterNumber* record is set to a subcode for the specific interval data type. (See Appendix D, “Data Types” in the **SOLID Programmer Guide**.) The default interval leading precision (2) and default interval seconds precision (6), as set in the SQL_DESC_DATETIME_INTERVAL_PRECISION and SQL_DESC_PRECISION fields of the APD, respectively, are used for the data. If either default precision is not appropriate, the application should explicitly set the descriptor field by a call to **SQLSetDescField** or **SQLSetDescRec**.

If the *ValueType* argument is one of the datetime data types, the SQL_DESC_TYPE field of the *ParameterNumber* record of the APD is set to SQL_DATETIME, the SQL_DESC_CONCISE_TYPE field of the *ParameterNumber* record of the APD is set to the concise datetime C data type, and the SQL_DESC_DATETIME_INTERVAL_CODE field of the *ParameterNumber* record is set to a subcode for the specific datetime data type. (See Appendix D, “Data Types” in the **SOLID Programmer Guide**.)

If the *ValueType* argument is an SQL_C_NUMERIC data type, the default precision (which is driver-defined) and the default scale (0), as set in the SQL_DESC_PRECISION and SQL_DESC_SCALE fields of the APD, are used for the data. If the default precision or scale is not appropriate, the application should explicitly set the descriptor field by a call to **SQLSetDescField** or **SQLSetDescRec**.

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

For more information, see “Default C Data Types,” “Converting Data from C to SQL to SQL Data Types,” and “Converting Data from SQL to C Data Types” in Appendix D, “Data Types” of the **SOLID Programmer Guide**.

ParameterType Argument

This must be one of the values listed in the “SQL Data Types” section of Appendix D, “Data Types,” of the **SOLID Programmer Guide** or it must be a driver-specific value. This argument sets the SQL_DESC_TYPE, SQL_DESC_CONCISE_TYPE, and SQL_DESC_DATETIME_INTERVAL_CODE fields of the IPD.

If the *ParameterType* argument is one of the datetime identifiers, the SQL_DESC_TYPE field of the IPD is set to SQL_DATETIME, the SQL_DESC_CONCISE_TYPE field of the IPD is set to the concise datetime SQL data type, and the SQL_DESC_DATETIME_INTERVAL_CODE field is set to the appropriate datetime subcode value.

If *ParameterType* is one of the interval identifiers, the SQL_DESC_TYPE field of the IPD is set to SQL_INTERVAL, the SQL_DESC_CONCISE_TYPE field of the IPD is set to the concise SQL interval data type, and the SQL_DESC_DATETIME_INTERVAL_CODE field of the IPD is set to the appropriate interval subcode. The SQL_DESC_DATETIME_INTERVAL_PRECISION field of the IPD is set to the interval leading precision, and the SQL_DESC_PRECISION field is set to the interval seconds precision, if applicable. If the default value of SQL_DESC_DATETIME_INTERVAL_PRECISION or

SQL_DESC_PRECISION is not appropriate, the application should explicitly set it by calling **SQLSetDescField**. For more information about any of these fields, see [SQLSetDescField](#).

If the *ValueType* argument is a SQL_NUMERIC data type, the default precision (which is driver-defined) and the default scale (0), as set in the SQL_DESC_PRECISION and SQL_DESC_SCALE fields of the IPD, are used for the data. If the default precision or scale is not appropriate, the application should explicitly set the descriptor field by a call to **SQLSetDescField** or **SQLSetDescRec**.

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” in Appendix D, "Data Types" of the **SOLID Programmer Guide**.

ColumnSize Argument

The *ColumnSize* argument specifies the size of the column or expression corresponding to the parameter marker, the length of that data, or both. This argument sets different fields of the IPD, depending on the SQL data type (the *ParameterType* argument). The following rules apply to this mapping:

- If *ParameterType* is SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR, SQL_BINARY, SQL_VARBINARY, SQL_LONGVARBINARY, or one of the concise SQL datetime or interval data types, the SQL_DESC_LENGTH field of the IPD is set to the value of *ColumnSize*. (For more information, see the Appendix D, "Data Types" of the **SOLID Programmer Guide**.)
- If *ParameterType* is SQL_DECIMAL, SQL_NUMERIC, SQL_FLOAT, SQL_REAL, or SQL_DOUBLE, the SQL_DESC_PRECISION field of the IPD is set to the value of *ColumnSize*.
- For other data types, the *ColumnSize* argument is ignored.

For more information, see "Passing Parameter Values" and SQL_DATA_AT_EXEC in "*StrLen_or_IndPtr* Argument."

DecimalDigits Argument

If *ParameterType* is SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP, SQL_INTERVAL_SECOND, SQL_INTERVAL_DAY_TO_SECOND, SQL_INTERVAL_HOUR_TO_SECOND, or SQL_INTERVAL_MINUTE_TO_SECOND, the SQL_DESC_PRECISION field of the IPD is set to *DecimalDigits*. If *ParameterType* is SQL_NUMERIC or SQL_DECIMAL, the SQL_DESC_SCALE field of the IPD is set to *DecimalDigits*. For all other data types, the *DecimalDigits* argument is ignored.

ParameterValuePtr Argument

The *ParameterValuePtr* 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 *ValueType* argument. This argument sets the SQL_DESC_DATA_PTR field of the APD. An application can set the *ParameterValuePtr* argument to a null pointer, as long as **StrLen_or_IndPtr* is SQL_NULL_DATA or SQL_DATA_AT_EXEC. (This applies only to input or input/output parameters.)

If **StrLen_or_IndPtr* is the result of the SQL_LEN_DATA_AT_EXEC(*length*) macro or SQL_DATA_AT_EXEC, then *ParameterValuePtr* is an application-defined, 32-bit value that is associated with the parameter. It is returned to the application through **SQLParamData**. For example, *ParameterValuePtr* 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, *ParameterValuePtr* must be a pointer to a buffer where the output value will be stored. If the value in the `SQL_ATTR_PARAMSET_SIZE` statement attribute is greater than 1, the application can use the value pointed to by the `SQL_ATTR_PARAMS_PROCESSED_PTR` statement attribute in conjunction with the *ParameterValuePtr* argument. For example, *ParameterValuePtr* might point to an array of values and the application might use the value pointed to by `SQL_ATTR_PARAMS_PROCESSED_PTR` to retrieve the correct value from the array. For more information, see "Passing Parameter Values" later in this section.

If the *InputOutputType* argument is `SQL_PARAM_INPUT_OUTPUT` or `SQL_PARAM_OUTPUT`, *ParameterValuePtr* points to a buffer in which the driver returns the output value. If the procedure returns one or more result sets, the **ParameterValuePtr* buffer is not guaranteed to be set until all result sets/row counts have been processed. If the buffer is not set until processing is complete, the output parameters and return values are unavailable until **SQLMoreResults** returns `SQL_NO_DATA`. Calling **SQLCloseCursor** or **SQLFreeStmt** with an Option of `SQL_CLOSE` will cause these values to be discarded.

If the value in the `SQL_ATTR_PARAMSET_SIZE` statement attribute is greater than 1, *ParameterValuePtr* 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.

BufferLength Argument

For character and binary C data, the *BufferLength* argument specifies the length of the **ParameterValuePtr* buffer (if it is a single element) or the length of an element in the **ParameterValuePtr* array (if the value in the `SQL_ATTR_PARAMSET_SIZE` statement attribute is greater than 1). This argument sets the `SQL_DESC_OCTET_LENGTH` record field of the APD. If the application specifies multiple values, *BufferLength* is used to determine the location of values in the **ParameterValuePtr* 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 *BufferLength*, the data in **ParameterValuePtr* is truncated to *BufferLength* less the length of a null-termination character and is null-terminated by the driver.
- For binary C data, if the number of bytes available to return is greater than *BufferLength*, the data in **ParameterValuePtr* is truncated to *BufferLength* bytes.

For all other types of C data, the *BufferLength* argument is ignored. The length of the **ParameterValuePtr* buffer (if it is a single element) or the length of an element in the **ParameterValuePtr* array (if the application calls **SQLSetStmtAttr** with an *Attribute* argument of `SQL_ATTR_PARAMSET_SIZE` 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 3.x driver, the Driver Manager converts this to a call to **SQLBindParameter** in which the *BufferLength* argument is always `SQL_SETPARAM_VALUE_MAX`. Because the Driver Manager returns an error if an ODBC 3.x application sets *BufferLength* to `SQL_SETPARAM_VALUE_MAX`, an ODBC 3.x driver can use this to determine when it is called by an ODBC 1.0 application.

In **SQLSetParam**, the way in which an application specifies the length of the **ParameterValuePtr* 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.

StrLen_or_IndPtr Argument

The *StrLen_or_IndPtr* argument points to a buffer that, when **SQLExecute** or **SQLExecDirect** is called, contains one of the following. (This argument sets the **SQL_DESC_OCTET_LENGTH_PTR** and **SQL_DESC_INDICATOR_PTR** record fields of the application parameter pointers.)

- The length of the parameter value stored in **ParameterValuePtr*. 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 called in ODBC canonical syntax, and then only if the *InputOutputType* argument is **SQL_PARAM_INPUT** or **SQL_PARAM_INPUT_OUTPUT**. When **StrLen_or_IndPtr* is **SQL_DEFAULT_PARAM**, the *ValueType*, *ParameterType*, *ColumnSize*, *DecimalDigits*, *BufferLength*, and *ParameterValuePtr* arguments are ignored for input parameters and are used only to define the output parameter value for input/output parameters.
- The result of the **SQL_LEN_DATA_AT_EXEC(*length*)** macro. The data for the parameter will be sent with **SQLPutData**. If the *ParameterType* argument is **SQL_LONGVARIABLE**, **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," later in this section.

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

- **SQL_DATA_AT_EXEC**. The data for the parameter will be sent with **SQLPutData**. This value is used by ODBC 1.0 applications when calling ODBC 3.x drivers. For more information, see "Passing Parameter Values," later in this section.

If *StrLen_or_IndPtr* 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 *InputOutputType* is **SQL_PARAM_OUTPUT** and *ParameterValuePtr* and *StrLen_or_IndPtr* are both null pointers, the driver discards the output value.

Note Application developers are strongly discouraged from specifying a null pointer for *StrLen_or_IndPtr* when the data type of the parameter is **SQL_C_BINARY**. To ensure that a driver does not unexpectedly truncate **SQL_C_BINARY** data, *StrLen_or_IndPtr* should contain a pointer to a valid length value.

If the *InputOutputType* argument is **SQL_PARAM_INPUT_OUTPUT** or **SQL_PARAM_OUTPUT**, *StrLen_or_IndPtr* points to a buffer in which the driver returns **SQL_NULL_DATA**, the number of bytes available to return in **ParameterValuePtr* (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 **StrLen_or_IndPtr* buffer is not guaranteed to be set until all results have been fetched.

If the value in the `SQL_ATTR_PARAMSET_SIZE` statement attribute is greater than 1, *StrLen_or_IndPtr* points to an array of `SQLINTEGER` 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 **ParameterValuePtr* 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 performs the following sequence of steps:

- Calls **SQLBindParameter** for each parameter to bind buffers for the parameter's value (*ParameterValuePtr* argument) and length/indicator (*StrLen_or_IndPtr* argument). For data-at-execution parameters, *ParameterValuePtr* 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.
- Places values for input and input/output parameters in the **ParameterValuePtr* and **StrLen_or_IndPtr* buffers:
 - For normal parameters, the application places the parameter value in the **ParameterValuePtr* buffer and the length of that value in the **StrLen_or_IndPtr* buffer. For more information, see the Part I PDF file, "Setting Parameter Values" in Chapter 9, "Executing Statements."
 - 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) in the **StrLen_or_IndPtr* buffer.
- 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`.
- Calls **SQLParamData** to retrieve the application-defined value specified in the *ParameterValuePtr* argument of **SQLBindParameter** for the first data-at-execution parameter to be processed. **SQLParamData** returns `SQL_NEED_DATA`.

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 *ParameterValuePtr* 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 **SQLBulkOperations** or updated with **SQLSetPos**. They are

bound with **SQLBindCol**. The value returned by **SQLParamData** is the address of the row in the **TargetValuePtr* buffer (set by a call to **SQLBindCol**) that is being processed.

- 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 **ParameterValuePtr* buffer specified in **SQLPutData**; 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.
- 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 4 and 5.
 - 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 are available in the **ParameterValuePtr* and **StrLen_or_IndPtr* buffers after the application retrieves all result sets generated by the statement.

Calling **SQLExecute** or **SQLExecDirect** puts the statement in an **SQL_NEED_DATA** state. At this point, the application can call only **SQLCancel**, **SQLGetDiagField**, **SQLGetDiagRec**, **SQLGetFunctions**, **SQLParamData**, or **SQLPutData** with the statement or the *connection handle* associated with the statement. If it calls any other function with the statement or the connection associated with the statement, the function returns **SQLSTATE HY010** (Function sequence error). The statement leaves the **SQL_NEED_DATA** state when **SQLParamData** or **SQLPutData** returns an error, **SQLParamData** returns **SQL_SUCCESS** or **SQL_SUCCESS_WITH_INFO**, or the statement is canceled.

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.

Using Arrays of Parameters

When an application prepares a statement with parameter markers and passes in an array of parameters, there are two different ways this can be executed. One way is for the driver to rely on the array-processing capabilities of the back end, in which case the entire statement with the array of parameters is treated as one atomic unit. Oracle is an example of a data source that supports array processing capabilities. Another way to implement this feature is for the driver to generate a batch of SQL statements, one SQL statement for each set of parameters in the parameter array, and execute the batch. Arrays of parameters cannot be used with an **UPDATE WHERE CURRENT OF** statement.

When an array of parameters is processed, individual result sets/row counts (one for each parameter set) can be available or result sets/rows counts can be rolled up into one. The **SQL_PARAM_ARRAY_ROW_COUNTS** option in **SQLGetInfo** indicates whether row counts are available for each set of parameters (**SQL_PARC_BATCH**) or only one row count is available (**SQL_PARC_NO_BATCH**).

The `SQL_PARAM_ARRAY_SELECTS` option in **SQLGetInfo** indicates whether a result set is available for each set of parameters (`SQL_PAS_BATCH`) or only one result set is available (`SQL_PAS_NO_BATCH`). If the driver does not allow a result set-generating statement to be executed with an array of parameters, `SQL_PARAM_ARRAY_SELECTS` returns `SQL_PAS_NO_SELECT`.

For more information, see [SQLGetInfo](#).

To support arrays of parameters, the `SQL_ATTR_PARAMSET_SIZE` statement attribute is set to specify the number of values for each parameter. If the field is greater than 1, the `SQL_DESC_DATA_PTR`, `SQL_DESC_INDICATOR_PTR`, and `SQL_DESC_OCTET_LENGTH_PTR` fields of the APD must point to arrays. The cardinality of each array is equal to the value of `SQL_ATTR_PARAMSET_SIZE`.

The `SQL_DESC_ROWS_PROCESSED_PTR` field of the APD points to a buffer that contains the number of sets of parameters that have been processed, including error sets. As each set of parameters is processed, the driver stores a new value in the buffer. No number will be returned if this is a null pointer. When arrays of parameters are used, the value pointed to by the `SQL_DESC_ROWS_PROCESSED_PTR` field of the APD is populated even if `SQL_ERROR` is returned by the setting function. If `SQL_NEED_DATA` is returned, the value pointed to by the `SQL_DESC_ROWS_PROCESSED_PTR` field of the APD is set to the set of parameters that is being processed.

What occurs when an array of parameters is bound and an **UPDATE WHERE CURRENT OF** statement is executed is driver-defined.

Column-Wise Parameter Binding

In column-wise binding, the application binds separate parameter and length/indicator arrays to each parameter.

To use column-wise binding, the application first sets the `SQL_ATTR_PARAM_BIND_TYPE` statement attribute to `SQL_PARAM_BIND_BY_COLUMN`. (This is the default.) For each column to be bound, the application performs the following steps:

- Allocates a parameter buffer array.
- Allocates an array of length/indicator buffers.

Note If the application writes directly to descriptors when column-wise binding is used, separate arrays can be used for length and indicator data.

- Calls **SQLBindParameter** with the following arguments:
 - *ValueType* is the C type of a single element in the parameter buffer array.
 - *ParameterType* is the SQL type of the parameter.
 - *ParameterValuePtr* is the address of the parameter buffer array.
 - *BufferLength* is the size of a single element in the parameter buffer array. The *BufferLength* argument is ignored when the data is fixed-length data.
 - *StrLen_or_IndPtr* is the address of the length/indicator array.

For more information about how this information is used, see "ParameterValuePtr Argument" in "Comments," later in this section. For more information about column-wise binding of parameters, see the Part I PDF file, "Binding Arrays of Parameters" section in Chapter 9, "Executing Statements."

Row-Wise Parameter Binding

In row-wise binding, the application defines a structure containing parameter and length/indicator buffers for each parameter to be bound.

To use row-wise binding, the application performs the following steps:

- Defines a structure to hold a single set of parameters (including both parameter and length/indicator buffers) and allocates an array of these structures.

Note If the application writes directly to descriptors when row-wise binding is used, separate fields can be used for length and indicator data.
- Sets the `SQL_ATTR_PARAM_BIND_TYPE` statement attribute to the size of the structure containing a single set of parameters or to the size of an instance of a buffer into which the parameters will be bound. The length must include space for all of the bound parameters, and any padding of the structure or buffer, to make sure that when the address of a bound parameter is incremented with the specified length, the result will point to the beginning of the same parameter in the next row. When using the `sizeof` operator in ANSI C, this behavior is guaranteed.
- Calls **SQLBindParameter** with the following arguments for each parameter to be bound:
 - *ValueType* is the type of the parameter buffer member to be bound to the column.
 - *ParameterType* is the SQL type of the parameter.
 - *ParameterValuePtr* is the address of the parameter buffer member in the first array element.
 - *BufferLength* is the size of the parameter buffer member.
 - *StrLen_or_IndPtr* is the address of the length/indicator member to be bound.

For more information about how this information is used, see "*ParameterValuePtr* Argument," later in this section. For more information about row-wise binding of parameters, see the Part I PDF file, "Binding Arrays of Parameters" section in Chapter 9, "Executing Statements."

Error Information

If a driver does not implement parameter arrays as batches (the `SQL_PARAM_ARRAY_ROW_COUNTS` option is equal to `SQL_PARC_NO_BATCH`), error situations are handled as if one statement were executed. If the driver does implement parameter arrays as batches, an application can use the `SQL_DESC_ARRAY_STATUS_PTR` header field of the IPD to determine which parameter of an SQL statement or which parameter in an array of parameters caused **SQLExecDirect** or **SQLExecute** to return an error. This field contains status information for each row of parameter values. If the field indicates that an error has occurred, fields in the diagnostic data structure will indicate the row and parameter number of the parameter that failed. The number of elements in the array will be defined by the

SQL_DESC_ARRAY_SIZE header field in the APD, which can be set by the SQL_ATTR_PARAMSET_SIZE statement attribute.

Note The SQL_DESC_ARRAY_STATUS_PTR header field in the APD is used to ignore parameters. For more information about ignoring parameters, see the next section, "Ignoring a Set of Parameters."

When **SQLExecute** or **SQLExecDirect** returns SQL_ERROR, the elements in the array pointed to by the SQL_DESC_ARRAY_STATUS_PTR field in the IPD will contain SQL_PARAM_ERROR, SQL_PARAM_SUCCESS, SQL_PARAM_SUCCESS_WITH_INFO, SQL_PARAM_UNUSED, or SQL_PARAM_DIAG_UNAVAILABLE.

For each element in this array, the diagnostic data structure contains one or more status records. The SQL_DIAG_ROW_NUMBER field of the structure indicates the row number of the parameter values that caused the error. If it is possible to determine the particular parameter in a row of parameters that caused the error, the parameter number will be entered in the SQL_DIAG_COLUMN_NUMBER field.

SQL_PARAM_UNUSED is entered when a parameter has not been used because an error occurred in an earlier parameter that forced **SQLExecute** or **SQLExecDirect** to abort. For example, if there are 50 parameters and an error occurred while executing the fortieth set of parameters that caused **SQLExecute** or **SQLExecDirect** to abort, then SQL_PARAM_UNUSED is entered in the status array for parameters 41 through 50.

SQL_PARAM_DIAG_UNAVAILABLE is entered when the driver treats arrays of parameters as a monolithic unit, so it does not generate this individual parameter level of error information.

Some errors in the processing of a single set of parameters cause processing of the subsequent sets of parameters in the array to stop. Other errors do not affect the processing of subsequent parameters. Which errors will stop processing is driver-defined. If processing is not stopped, all parameters in the array are processed, SQL_SUCCESS_WITH_INFO is returned as a result of the error, and the buffer defined by SQL_ATTR_PARAMS_PROCESSED_PTR is set to the total number of sets of parameters processed (as defined by the SQL_ATTR_PARAMSET_SIZE statement attribute), which includes error sets.

Caution ODBC behavior when an error occurs in the processing of an array of parameters is different in ODBC 3.x than it was in ODBC 2.x. In ODBC 2.x, the function returned SQL_ERROR and processing ceased. The buffer pointed to by the *pirow* argument of **SQLParamOptions** contained the number of the error row. In ODBC 3.x, the function returns SQL_SUCCESS_WITH_INFO and processing may either cease or continue. If it continues, the buffer specified by SQL_ATTR_PARAMS_PROCESSED_PTR will be set to the value of all parameters processed, including those that resulted in an error. This change in behavior may cause problems for existing applications.

When **SQLExecute** or **SQLExecDirect** returns before completing the processing of all parameter sets in a parameter array, such as when SQL_ERROR or SQL_NEED_DATA is returned, the status array contains statuses for those parameters that have already been processed. The location pointed to by the SQL_DESC_ROWS_PROCESSED_PTR field in the IPD contains the row number in the parameter array that caused the SQL_ERROR or SQL_NEED_DATA error code. When an array of parameters is sent to a SELECT statement, the availability of status array values is driver-defined; they may be available after the statement has been executed or as result sets are fetched.

Ignoring a Set of Parameters

The SQL_DESC_ARRAY_STATUS_PTR field of the APD (as set by the SQL_ATTR_PARAM_STATUS_PTR statement attribute) can be used to indicate that a set of bound

parameters in an SQL statement should be ignored. To direct the driver to ignore one or more sets of parameters during execution, an application should perform the following steps:

- Call **SQLSetDescField** to set the `SQL_DESC_ARRAY_STATUS_PTR` header field of the APD to point to an array of `SQLUSMALLINT` values to contain status information. This field can also be set by calling **SQLSetStmtAttr** with an *Attribute* of `SQL_ATTR_PARAM_OPERATION_PTR`, which allows an application to set the field without obtaining a descriptor handle.
- Set each element of the array defined by the `SQL_DESC_ARRAY_STATUS_PTR` field of the APD to one of two values:
 - `SQL_PARAM_IGNORE`, to indicate that the row is excluded from statement execution.
 - `SQL_PARAM_PROCEED`, to indicate that the row is included in statement execution.
- Call **SQLExecDirect** or **SQLExecute** to execute the prepared statement.

The following rules apply to the array defined by the `SQL_DESC_ARRAY_STATUS_PTR` field of the APD:

- The pointer is set to null by default.
- If the pointer is null, then all sets of parameters are used, as if all elements were set to `SQL_ROW_PROCEED`.
- Setting an element to `SQL_PARAM_PROCEED` does not guarantee that the operation will use that particular set of parameters.
- `SQL_PARAM_PROCEED` is defined as 0 in the header file.

An application can set the `SQL_DESC_ARRAY_STATUS_PTR` field in the APD to point to the same array as that pointed to by the `SQL_DESC_ARRAY_STATUS_PTR` field in the IRD. This is useful when binding parameters to row data. Parameters then can be ignored according to the status of the row data. In addition to `SQL_PARAM_IGNORE`, the following codes cause a parameter in an SQL statement to be ignored: `SQL_ROW_DELETED`, `SQL_ROW_UPDATED`, and `SQL_ROW_ERROR`. In addition to `SQL_PARAM_PROCEED`, the following codes cause an SQL statement to proceed: `SQL_ROW_SUCCESS`, `SQL_ROW_SUCCESS_WITH_INFO`, and `SQL_ROW_ADDED`.

Rebinding Parameters

An application can perform either of two operations to change a binding:

- Call **SQLBindParameter** to specify a new binding for a column that is already bound. The driver overwrites the old binding with the new one.
- Specify an offset to be added to the buffer address that was specified by the binding call to **SQLBindParameter**. For more information, see the next section, "Rebinding with Offsets."

Rebinding with Offsets

Rebinding of parameters is especially useful when an application has a buffer area setup that is capable of containing many parameters but a call to **SQLExecDirect** or **SQLExecute** uses only a few of the parameters. The remaining space in the buffer area can be used for the next set of parameters by modifying the existing binding by an offset.

The SQL_DESC_BIND_OFFSET_PTR header field in the APD points to the binding offset. If the field is non-null, the driver dereferences the pointer and, if none of the values in the SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR fields is a null pointer, adds the dereferenced value to those fields in the descriptor records at execution time. The new pointer values are used when the SQL statements are executed. The offset remains valid after rebinding. Because SQL_DESC_BIND_OFFSET_PTR is a pointer to the offset rather than the offset itself, an application can change the offset directly, without having to call [SQLSetDescField](#) or [SQLSetDescRec](#) to change the descriptor field. The pointer is set to null by default. The SQL_DESC_BIND_OFFSET_PTR field of the ARD can be set by a call to [SQLSetDescField](#) or by a call to [SQLSetStmtAttr](#) with an *fAttribute* of SQL_ATTR_PARAM_BIND_OFFSET_PTR.

The binding offset is always added directly to the values in the SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR fields. If the offset is changed to a different value, the new value is still added directly to the value in each descriptor field. The new offset is not added to the sum of the field value and any earlier offsets.

Descriptors

How a parameter is bound is determined by fields of the APDs and IPDs. The arguments in **SQLBindParameter** are used to set those descriptor fields. The fields also can be set by the **SQLSetDescField** functions, although **SQLBindParameter** is more efficient to use because the application does not have to obtain a descriptor handle to call **SQLBindParameter**.

Caution Calling **SQLBindParameter** for one statement can affect other statements. This occurs when the ARD associated with the statement is explicitly allocated and is also associated with other statements. Because **SQLBindParameter** modifies the fields of the APD, the modifications apply to all statements with which this descriptor is associated. If this is not the required behavior, the application should dissociate this descriptor from the other statements before calling **SQLBindParameter**.

Conceptually, **SQLBindParameter** performs the following steps in sequence:

- Calls [SQLSetStmtAttr](#) to obtain the APD handle.
- Calls [SQLGetDescField](#) to get the APD's SQL_DESC_COUNT field, and if the value of the *ColumnNumber* argument exceeds the value of SQL_DESC_COUNT, calls **SQLSetDescField** to increase the value of SQL_DESC_COUNT to *ColumnNumber*.
- Calls [SQLSetDescField](#) multiple times to assign values to the following fields of the APD:
 - Sets SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE to the value of *ValueType*, except that if *ValueType* is one of the concise identifiers of a datetime or interval subtype, it sets SQL_DESC_TYPE to SQL_DATETIME or SQL_INTERVAL, respectively, sets SQL_DESC_CONCISE_TYPE to the concise identifier, and sets SQL_DESC_DATETIME_INTERVAL_CODE to the corresponding datetime or interval subcode.
 - Sets the SQL_DESC_OCTET_LENGTH field to the value of *BufferLength*.

- Sets the SQL_DESC_DATA_PTR field to the value of *ParameterValue*.
- Sets the SQL_DESC_OCTET_LENGTH_PTR field to the value of *StrLen_or_Ind*.
- Sets the SQL_DESC_INDICATOR_PTR field also to the value of *StrLen_or_Ind*.

The *StrLen_or_Ind* parameter specifies both the indicator information and the length for the parameter value.

- Calls [SQLSetStmtAttr](#) to obtain the IPD handle.
- Calls [SQLGetDescField](#) to get the IPD's SQL_DESC_COUNT field, and if the value of the *ColumnNumber* argument exceeds the value of SQL_DESC_COUNT, calls **SQLSetDescField** to increase the value of SQL_DESC_COUNT to *ColumnNumber*.
- Calls [SQLSetDescField](#) multiple times to assign values to the following fields of the IPD:
 - Sets SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE to the value of *ParameterType*, except that if *ParameterType* is one of the concise identifiers of a datetime or interval subtype, it sets SQL_DESC_TYPE to SQL_DATETIME or SQL_INTERVAL, respectively, sets SQL_DESC_CONCISE_TYPE to the concise identifier, and sets SQL_DESC_DATETIME_INTERVAL_CODE to the corresponding datetime or interval subcode.
 - Sets one or more of SQL_DESC_LENGTH, SQL_DESC_PRECISION, and SQL_DESC_DATETIME_INTERVAL_PRECISION, as appropriate for *ParameterType*.
 - Sets SQL_DESC_SCALE to the value of *DecimalDigits*.

If the call to **SQLBindParameter** fails, the content of the descriptor fields that it would have set in the APD are undefined, and the SQL_DESC_COUNT field of the APD is unchanged. In addition, the SQL_DESC_LENGTH, SQL_DESC_PRECISION, SQL_DESC_SCALE, and SQL_DESC_TYPE fields of the appropriate record in the IPD are undefined and the SQL_DESC_COUNT field of the IPD is unchanged.

Conversion of Calls to and from SQLSetParam

When an ODBC 1.0 application calls **SQLSetParam** in an ODBC 3.x driver, the ODBC 3.x Driver Manager maps the call as shown in the following table.

Call by ODBC 1.0 application	Call to ODBC 3.x driver
SQLSetParam(StatementHandle, ParameterNumber, ValueType, ParameterType, LengthPrecision, ParameterScale, ParameterValuePtr, StrLen_or_IndPtr);	SQLBindParameter(StatementHandle, ParameterNumber, SQL_PARAM_INPUT_OUTPUT, ValueType, ParameterType, ColumnSize, DecimalDigits, ParameterValuePtr,

	SQL_SETPARAM_VALUE_MAX, StrLen_or_IndPtr);
--	-----------------------------------------------

Code Example

In the following example, an application prepares an SQL statement to insert data into the ORDERS table. The SQL statement contains parameters for the ORDERID, CUSTID, OPENDATE, SALESPERSON, and STATUS 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 [SQLBulkOperations](#), [SQLProcedures](#), [SQLPutData](#), and [SQLSetPos](#).

```
#define SALES_PERSON_LEN 10
#define STATUS_LEN 6

SQLSMALLINT      sOrderID;
SQLSMALLINT      sCustID;
DATE_STRUCT      dsOpenDate;
SQLCHAR          szSalesPerson[SALES_PERSON_LEN];
SQLCHAR          szStatus[STATUS_LEN];
SQLINTEGER       cbOrderID = 0, cbCustID = 0, cbOpenDate = 0, cbSalesPerson =
SQL_NTS,
cbStatus = SQL_NTS;
SQLRETURN retcode;
SQLHSTMT hstmt;

/* Prepare the SQL statement with parameter markers. */
retcode = SQLPrepare(hstmt,
"INSERT INTO ORDERS (ORDERID, CUSTID, OPENDATE, SALESPERSON,
STATUS) VALUES (?, ?, ?, ?, ?)", SQL_NTS);
if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
/* Specify data types and buffers for OrderID, CustID, OpenDate, SalesPerson,
*/
/* Status parameter data. */
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_SSHORT,
SQL_INTEGER, 0, 0, &sOrderID, 0, &cbOrderID);
SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_SSHORT,
SQL_INTEGER, 0, 0, &sCustID, 0, &cbCustID);
SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_TYPE_DATE,
SQL_TYPE_DATE, 0, 0, &dsOpenDate, 0, &cbOpenDate);
SQLBindParameter(hstmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR, SALES_PERSON_LEN, 0, szSalesPerson, 0, &cbSalesPerson);
SQLBindParameter(hstmt, 5, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR, STATUS_LEN, 0, szStatus, 0, &cbStatus);
/* Specify first row of parameter data. */
sOrderID = 1001;
sCustID = 298;
dsOpenDate.year = 1996;
dsOpenDate.month = 3;
dsOpenDate.day = 8;
strcpy(szSalesPerson, "Johnson");
strcpy(szStatus, "Closed");

/* Execute statement with first row. */
retcode = SQLExecute(hstmt);
```

```

/* Specify second row of parameter data. */
sOrderID = 1002;
sCustID = 501;
dsOpenDate.year = 1996;
dsOpenDate.month = 3;
dsOpenDate.day = 9;
strcpy(szSalesPerson, "Bailey");
strcpy(szStatus, "Open");

/* Execute statement with second row. */
retcode = SQLEExecute(hstmt);
}

```

Code Example

In the following example, an application executes an SQL Server stored procedure using a named parameter. Note that the named parameter (@quote) is bound with a *ParameterNumber* of 1, while it is the second parameter in the procedure definition. Because the first parameter (@title_id) has a default value of 1, the named parameter is the only dynamic parameter.

```

/* Define the stored procedure "test" */
CREATE PROCEDURE test @title_id int = 1, @quote char(30)
AS <blah>
/* Prepare the procedure invocation statement */
SQLPrepare(hstmt, "{call test(?)}", SQL_NTS);
/* Populate record 1 of IPD */
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
30, 0, szQuote, 0, &cbValue);
/* Get IPD handle and set the NAMED and UNNAMED fields for record # 1 */
SQLGetStmtAttr(hstmt, SQL_ATTR_IMP_PARAM_DESC, &hIpd, 0, 0);
SQLSetDescField(hIpd, 1, SQL_DESC_NAME, "@quote", SQL_NTS);
/* Assuming that szQuote has been appropriately initialized, execute the
statement */
SQLEExecute(hstmt);

```

Related Functions

For information about	See
Returning information about a parameter in a statement	SQLDescribeParam
Executing an SQL statement	SQLExecDirect
Executing a prepared SQL statement	SQLEExecute
Releasing parameter buffers on the statement	SQLFreeStmt
Returning the number of statement parameters	SQLNumParams
Returning the next parameter to send data for	SQLParamData
Specifying multiple parameter values	SQLParamOptions
Sending parameter data at execution time	SQLPutData

SQLBrowseConnect

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ODBC

Summary

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

SQLRETURN SQLBrowseConnect(
SQLHDBC	ConnectionHandle,
SQLCHAR *	InConnectionString,
SQLSMALLINT	StringLength1,
SQLCHAR *	OutConnectionString,
SQLSMALLINT	BufferLength,
SQLSMALLINT *	StringLength2Ptr);

Arguments

ConnectionHandle

[Input]

Connection handle.

InConnectionString

[Input]

Browse request connection string (see "*InConnectionString* Argument" in the following comments section).

StringLength1

[Input]

Length of **InConnectionString*.

OutConnectionString

[Output]

Pointer to a buffer in which to return the browse result connection string (see "*OutConnectionString* Argument" in the following Comments section).

BufferLength

[Input]

Length of the **OutConnectionString* buffer.

StringLength2Ptr

[Output]

The total number of bytes (excluding the null-termination byte) available to return in

**OutConnectionString*. If the number of bytes available to return is greater than or equal to *BufferLength*, the connection string in **OutConnectionString* is truncated to *BufferLength* minus the length of a null-termination character.

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *ConnectionHandle*. 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	String data, right truncated	The buffer <i>*OutConnectionString</i> was not large enough to return the entire browse result connection string, so the string was truncated. The buffer <i>*StringLength2Ptr</i> 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 (<i>InConnectionString</i>). (Function returns SQL_NEED_DATA.) An attribute keyword was specified in the browse request connection string (<i>InConnectionString</i>) that does not apply to the current connection level. (Function returns SQL_NEED_DATA.)
01S02	Value changed	The driver did not support the specified value of the <i>ValuePtr</i> argument in SQLSetConnectAttr and substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
08001	Client unable to establish connection	The driver was unable to establish a connection with the data source.
08002	Connection name in use	(DM) The specified connection had already been used

		to establish a connection with a data source, and the connection was open.
08004	Server rejected the 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 (<i>InConnectionString</i>), violated restrictions defined by the data source.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	(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.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	(DM) The value specified for argument <i>StringLength1</i> was less than 0 and was not equal to SQL_NTS. (DM) The value specified for argument <i>BufferLength</i> was less than 0.
HYT00	Timeout expired	The login timeout period expired before the connection to the data source completed. The timeout period is set through SQLSetConnectAttr , SQL_ATTR_LOGIN_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
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 (<i>InConnectionString</i>) was not found in the system information, nor was there a default driver specification. (DM) ODBC data source and default driver information could not be found in the system information.

IM003	Specified driver could not be loaded	(DM) The driver listed in the data source specification in the system information or specified by the DRIVER keyword was not found or could not be loaded for some other reason.
IM004	Driver's SQLAllocHandle on SQL_HANDLE_ENV failed	(DM) During SQLBrowseConnect , the Driver Manager called the driver's SQLAllocHandle function with a <i>HandleType</i> of SQL_HANDLE_ENV and the driver returned an error.
IM005	Driver's SQLAllocHandle on SQL_HANDLE_DBC failed	(DM) During SQLBrowseConnect , the Driver Manager called the driver's SQLAllocHandle function with a <i>HandleType</i> of SQL_HANDLE_DBC and the driver returned an error.
IM006	Driver's SQLSetConnectAttr failed	(DM) During SQLBrowseConnect , the Driver Manager called the driver's SQLSetConnectAttr 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.

Comments

InConnectionString Argument

A browse request connection string has the following syntax:

```

connection-string ::= attribute[;] | attribute; connection-string
attribute ::= attribute-keyword=attribute-value | DRIVER=[{ }attribute-value{ }]
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 not case-sensitive; *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 grammar in the system information, keywords and data source names cannot contain the backslash (`\`) character. For an ODBC 2.x driver, braces are required around the attribute value for the DRIVER keyword.

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.

For information about how an application chooses a data source or driver, see the Part I PDF file, "Choosing a Data Source or Driver" in Chapter 6, "Connecting to a Data Source or Driver."

OutConnectionString 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 ::= identifier[:localized-identifier]
attribute-value ::= { attribute-value-list } | ?
(The braces are literal; they are returned by the driver.)
attribute-value-list ::= character-string [:localized-character string] | character-string [:localized-character
string], attribute-value-list
```

where *character-string* and *localized-character string* have zero or more characters; *identifier* and *localized-identifier* have one or more characters; *attribute-keyword* is not case-sensitive; 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 grammar in the system information, 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 can 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 connection. 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 about when this occurs, see the "Comments" section in **SQLConnect**. The driver may establish a connection with the data source during the browsing process. If **SQLBrowseConnect** returns **SQL_ERROR**, outstanding connections are terminated and the connection is returned to an unconnected state.

Note **SQLBrowseConnect** does not support connection pooling. If **SQLBrowseConnect** is called while connection pooling is enabled, **SQLSTATE HY000** (General error) will be returned.

When **SQLBrowseConnect** is called for the first time on a connection, 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 system information:

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 system information.

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 system information. "DEFAULT" is passed to the driver for the DSN.

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 system information. Because the **DRIVER** keyword does not use information from the system information, 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**. All mandatory attributes (those not preceded by an asterisk in the *OutConnectionString* argument) must be included in 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. The complete connection string cannot be used in another call to **SQLBrowseConnect**, however; if **SQLBrowseConnect** were called again, the entire sequence of calls would have to be repeated.

SQLBrowseConnect also returns **SQL_NEED_DATA** if there are recoverable, nonfatal errors during the browse process; for example, an invalid password or 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 can call **SQLGetDiagRec** to return the SQLSTATE for browse-time errors. This permits the application to correct the attribute and continue the browse.

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

For more information, see the Part I PDF file, "Connecting with SQLBrowseConnect" in Chapter 6, "Connecting to a Data Source or Driver."

If a driver supports **SQLBrowseConnect**, the driver keyword section in the system information 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 **OutConnectionString*. The application passes *OutConnectionString* 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 **InConnectionString*. 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 a more detailed example of connecting to a SQL Server driver by calling **SQLBrowseConnect**, see the Part I PDF file, "SQL Server Browsing Example" in Chapter 6, "Connecting to a Data Source or Driver."

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

```
"DSN=Sales"
```

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

```
"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 **InConnectionString*, 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, and then returns the following string in **OutConnectionString*:

```
"*DATABASE:Database={SalesEmployees,SalesGoals,SalesOrders}"
```

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 this string:

```
"DATABASE=SalesOrders"
```

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

```
"DSN=Sales;HOST=red;UID=Smith;PWD=Sesame;DATABASE=SalesOrders"
```

```

#define BRWS_LEN 100
SQLHENV  henv;
SQLHDBC  hdbc;
SQLHSTMT hstmt;
SQLRETURN retcode;
SQLCHAR  szConnStrIn[BRWS_LEN], szConnStrOut[BRWS_LEN];
SQLSMALLINT  cbConnStrOut;

/* Allocate the environment handle. */
retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
/* Set the version environment attribute. */
retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, SQL_OV_ODBC3, 0);
if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
/* Allocate the connection handle. */
retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
/* 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=Sales");
    do {
        retcode = SQLBrowseConnect(hdbc, 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){

/* Allocate the statement handle. */
retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
/* Process data after successful connection */
;
;
;
SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
}
SQLDisconnect(hdbc);
}
}
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
}
}
SQLFreeHandle(SQL_HANDLE_ENV, henv);

```

Related Functions

For information about	See
Allocating a connection handle	SQLAllocHandle
Connecting to a data source	SQLConnect

Disconnecting from a data source	SQLDisconnect
Connecting to a data source using a connection string or dialog box	SQLDriverConnect
Returning driver descriptions and attributes	SQLDrivers
Freeing a connection handle	SQLFreeHandle

SQLBulkOperations

Conformance

Version Introduced: ODBC 3.0

Standards Compliance: ODBC

Summary

SQLBulkOperations performs bulk insertions and bulk bookmark operations, including update, delete, and fetch by bookmark.

Syntax

SQLRETURN SQLBulkOperations(
SQLHSTMT	StatementHandle,
SQLUSMALLINT	Operation);

Arguments

StatementHandle

[Input]
Statement handle.

Operation

[Input]
Operation to perform:

SQL_ADD
SQL_UPDATE_BY_BOOKMARK
SQL_DELETE_BY_BOOKMARK
SQL_FETCH_BY_BOOKMARK

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 **SQLBulkOperations** returns `SQL_ERROR` or `SQL_SUCCESS_WITH_INFO`, an associated `SQLSTATE` value can be obtained by calling **SQLGetDiagRec** with a *HandleType* of `SQL_HANDLE_STMT` and a *Handle* of *StatementHandle*. The following table lists the `SQLSTATE` values commonly returned by **SQLBulkOperations** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of `SQLSTATE`s returned by the Driver Manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR`, unless noted otherwise.

For all those `SQLSTATE`s that can return `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR` (except 01xxx `SQLSTATE`s), `SQL_SUCCESS_WITH_INFO` is returned if an error occurs on one or more, but not all, rows of a multirow operation, and `SQL_ERROR` is returned if an error occurs on a single-row operation.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	String data right truncation	The <i>Operation</i> argument was <code>SQL_FETCH_BY_BOOKMARK</code> , and string or binary data returned for a column or columns with a data type of <code>SQL_C_CHAR</code> or <code>SQL_C_BINARY</code> resulted in the truncation of nonblank character or non-NULL binary data.
01S01	Error in row	The <i>Operation</i> argument was <code>SQL_ADD</code> , and an error occurred in one or more rows while performing the operation but at least one row was successfully added. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .) (This error is raised only when an application is working with an ODBC 2.x driver.)
01S07	Fractional truncation	The <i>Operation</i> argument was <code>SQL_FETCH_BY_BOOKMARK</code> , the data type of the application buffer was not <code>SQL_C_CHAR</code> or <code>SQL_C_BINARY</code> , and the data returned to application buffers for one or more columns was truncated. (For numeric C data types, the fractional part of the number was truncated. For time, timestamp, and interval C data types containing a time component, the fractional portion of the time was truncated.) (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
07006	Restricted data type attribute violation	The <i>Operation</i> argument was <code>SQL_FETCH_BY_BOOKMARK</code> , and the data value of a column in the result set could not be converted to the data type specified by the <i>TargetType</i> argument in the call to SQLBindCol . The <i>Operation</i> argument was <code>SQL_UPDATE_BY_BOOKMARK</code> or <code>SQL_ADD</code> , and the data value in the application buffers could not be converted to the data type of a column in the result set.
07009	Invalid descriptor index	The argument <i>Operation</i> was <code>SQL_ADD</code> , and a column was bound with a column number greater than

		the number of columns in the result set.
21S02	Degree of derived table does not match column list	The argument <i>Operation</i> was SQL_UPDATE_BY_BOOKMARK; and no columns were updatable because all columns were either unbound or read-only, or the value in the bound length/indicator buffer was SQL_COLUMN_IGNORE.
22001	String data right truncation	The assignment of a character or binary value to a column in the result set resulted in the truncation of nonblank (for characters) or non-null (for binary) characters or bytes.
22003	Numeric value out of range	<p>The <i>Operation</i> argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, and the assignment of a numeric value to a column in the result set caused the whole (as opposed to fractional) part of the number to be truncated.</p> <p>The argument <i>Operation</i> was SQL_FETCH_BY_BOOKMARK, and returning the numeric value for one or more bound columns would have caused a loss of significant digits.</p>
22007	Invalid datetime format	<p>The <i>Operation</i> argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, and the assignment of a date or timestamp value to a column in the result set caused the year, month, or day field to be out of range.</p> <p>The argument <i>Operation</i> was SQL_FETCH_BY_BOOKMARK, and returning the date or timestamp value for one or more bound columns would have caused the year, month, or day field to be out of range.</p>
22008	Date/time field overflow	<p>The <i>Operation</i> argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, and the performance of datetime arithmetic on data being sent to a column in the result set resulted in a datetime field (the year, month, day, hour, minute, or second field) of the result falling outside the permissible range of values for the field or being invalid based on the Gregorian calendar's natural rules for datetimes.</p> <p>The <i>Operation</i> argument was SQL_FETCH_BY_BOOKMARK, and the performance of datetime arithmetic on data being retrieved from the result set resulted in a datetime field (the year, month, day, hour, minute, or second field) of the result falling outside the permissible range of values for the field or being invalid based on the Gregorian calendar's natural rules for datetimes.</p>
22015	Interval field overflow	The <i>Operation</i> argument was SQL_ADD or

		<p>SQL_UPDATE_BY_BOOKMARK, and the assignment of an exact numeric or interval C type to an interval SQL data type caused a loss of significant digits.</p> <p>The <i>Operation</i> argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK; when assigning to an interval SQL type, there was no representation of the value of the C type in the interval SQL type.</p> <p>The <i>Operation</i> argument was SQL_FETCH_BY_BOOKMARK, and assigning from an exact numeric or interval SQL type to an interval C type caused a loss of significant digits in the leading field.</p> <p>The <i>Operation</i> argument was SQL_FETCH_BY_BOOKMARK; when assigning to an interval C type, there was no representation of the value of the SQL type in the interval C type.</p>
22018	Invalid character value for cast specification	<p>The <i>Operation</i> argument was SQL_FETCH_BY_BOOKMARK; the C type was an exact or approximate numeric, a datetime, or an interval data type; the SQL type of the column was a character data type; and the value in the column was not a valid literal of the bound C type.</p> <p>The argument <i>Operation</i> was SQL_ADD or SQL_UPDATE_BY_BOOKMARK; the SQL type was an exact or approximate numeric, a datetime, or an interval data type; the C type was SQL_C_CHAR; and the value in the column was not a valid literal of the bound SQL type.</p>
23000	Integrity constraint violation	<p>The <i>Operation</i> argument was SQL_ADD, SQL_DELETE_BY_BOOKMARK, or SQL_UPDATE_BY_BOOKMARK, and an integrity constraint was violated.</p> <p>The <i>Operation</i> argument was SQL_ADD, and a column that was not bound is defined as NOT NULL and has no default.</p> <p>The <i>Operation</i> argument was SQL_ADD, the length specified in the bound <i>StrLen_or_IndPtr</i> buffer was SQL_COLUMN_IGNORE, and the column did not have a default value.</p>
24000	Invalid cursor state	The <i>StatementHandle</i> was in an executed state, but no result set was associated with the <i>StatementHandle</i> .
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion	The associated connection failed during the execution

	unknown	of this function, and the state of the transaction cannot be determined.
42000	Syntax error or access violation	The driver was unable to lock the row as needed to perform the operation requested in the <i>Operation</i> argument.
44000	WITH CHECK OPTION violation	The <i>Operation</i> argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, and the insert or update was performed on a viewed table (or a table derived from the viewed table) that was created by specifying WITH CHECK OPTION , such that one or more rows affected by the insert or update will no longer be present in the viewed table.
HY000	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 SQLGetDiagRec in the * <i>MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>. Then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY010	Function sequence error	<p>(DM) The specified <i>StatementHandle</i> was not in an executed state. The function was called without first calling SQLExecDirect, SQLExecute, or a catalog function.</p> <p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p> <p>(DM) The driver was an ODBC 2.x driver, and SQLBulkOperations was called for a <i>StatementHandle</i> before SQLFetchScroll or SQLFetch was called.</p> <p>(DM) SQLBulkOperations was called after SQLExtendedFetch was called on the</p>

		<i>StatementHandle</i> .
HY011	Attribute cannot be set now	(DM) The driver was an ODBC 2.x driver, and the SQL_ATTR_ROW_STATUS_PTR statement attribute was set between calls to SQLFetch or SQLFetchScroll and SQLBulkOperations .
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	<p>The <i>Operation</i> argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, a data value was a null pointer, and the column length value was not 0, SQL_DATA_AT_EXEC, SQL_COLUMN_IGNORE, SQL_NULL_DATA, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.</p> <p>The <i>Operation</i> argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK; a data value was not a null pointer; the C data type was SQL_C_BINARY or SQL_C_CHAR; and the column length value was less than 0, but not equal to SQL_DATA_AT_EXEC, SQL_COLUMN_IGNORE, SQL_NTS, or SQL_NULL_DATA, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.</p> <p>The value in a length/indicator buffer was SQL_DATA_AT_EXEC; the SQL type was either SQL_LONGVARCHAR, SQL_LONGVARIABLE, or a long data source-specific data type; and the SQL_NEED_LONG_DATA_LEN information type in SQLGetInfo was "Y".</p> <p>The <i>Operation</i> argument was SQL_ADD, the SQL_ATTR_USE_BOOKMARK statement attribute was set to SQL_UB_VARIABLE, and column 0 was bound to a buffer whose length was not equal to the maximum length for the bookmark for this result set. (This length is available in the SQL_DESC_OCTET_LENGTH field of the IRD and can be obtained by calling SQLDescribeCol, SQLColAttribute, or SQLGetDescField.)</p>
HY092	Invalid attribute identifier	<p>(DM) The value specified for the <i>Operation</i> argument was invalid.</p> <p>The <i>Operation</i> argument was SQL_ADD, SQL_UPDATE_BY_BOOKMARK, or SQL_DELETE_BY_BOOKMARK, and the SQL_ATTR_CONCURRENCY statement attribute was set to SQL_CONCUR_READ_ONLY.</p> <p>The <i>Operation</i> argument was SQL_DELETE_BY_BOOKMARK,</p>

		SQL_FETCH_BY_BOOKMARK, or SQL_UPDATE_BY_BOOKMARK, and the bookmark column was not bound or the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF.
HYC00	Optional feature not implemented	The driver or data source does not support the operation requested in the <i>Operation</i> argument.
HYT00	Timeout expired	The query timeout period expired before the data source returned the result set. The timeout period is set through SQLSetStmtAttr with an <i>Attribute</i> argument of SQL_ATTR_QUERY_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

Caution For information about what statement states **SQLBulkOperations** can be called in and what it needs to do for compatibility with ODBC 2.x applications, see “Block Cursors, Scrollable Cursors, and Backward Compatibility” (Appendix G, “Driver Guidelines for Backward Compatibility”) contained on the Microsoft Web site (ODBC Programmer’s Reference).

An application uses **SQLBulkOperations** to perform the following operations on the base table or view that corresponds to the current query:

Add new rows.

Update a set of rows where each row is identified by a bookmark.

Delete a set of rows where each row is identified by a bookmark.

Fetch a set of rows where each row is identified by a bookmark.

After a call to **SQLBulkOperations**, the block cursor position is undefined. The application has to call **SQLFetchScroll** to set the cursor position. An application should call **SQLFetchScroll** only with a *FetchOrientation* argument of SQL_FETCH_FIRST, SQL_FETCH_LAST, SQL_FETCH_ABSOLUTE, or SQL_FETCH_BOOKMARK. The cursor position is undefined if the application calls **SQLFetch** or **SQLFetchScroll** with a *FetchOrientation* argument of SQL_FETCH_PRIOR, SQL_FETCH_NEXT, or SQL_FETCH_RELATIVE.

A column can be ignored in bulk operations performed by a call to **SQLBulkOperations** by setting the column length/indicator buffer specified in the call to **SQLBindCol**, to SQL_COLUMN_IGNORE.

It is not necessary for the application to set the SQL_ATTR_ROW_OPERATION_PTR statement attribute when calling **SQLBulkOperations** because rows cannot be ignored when performing bulk operations with this function.

The buffer pointed to by the SQL_ATTR_ROWS_FETCHED_PTR statement attribute contains the number of rows affected by a call to **SQLBulkOperations**.

When the *Operation* argument is `SQL_ADD` or `SQL_UPDATE_BY_BOOKMARK` and the select-list of the query specification associated with the cursor contains more than one reference to the same column, it is driver-defined whether an error is generated or the driver ignores the duplicated references and performs the requested operations.

For more information about using **SQLBulkOperations**, see the Part 1 PDF file, “Updating Data with SQL BulkOperations” in Chapter 12, “Updating Data.”

Performing Bulk Inserts

To insert data with **SQLBulkOperations**, an application performs the following sequence of steps:

- Executes a query that returns a result set.
- Sets the `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute to the number of rows that it wants to insert.
- Calls **SQLBindCol** to bind the data that it wants to insert. The data is bound to an array with a size equal to the value of `SQL_ATTR_ROW_ARRAY_SIZE`.

Note The size of the array pointed to by the `SQL_ATTR_ROW_STATUS_PTR` statement attribute should either be equal to `SQL_ATTR_ROW_ARRAY_SIZE` or `SQL_ATTR_ROW_STATUS_PTR` should be a null pointer.

- Calls **SQLBulkOperations**(*StatementHandle*, `SQL_ADD`) to perform the insertion.

If the application has set the `SQL_ATTR_ROW_STATUS_PTR` statement attribute, it can inspect this array to see the result of the operation.

If an application binds column 0 before calling **SQLBulkOperations** with an *Operation* argument of `SQL_ADD`, the driver will update the bound column 0 buffers with the bookmark values for the newly inserted row. For this to occur, the application must have set the `SQL_ATTR_USE_BOOKMARKS` statement attribute to `SQL_UB_VARIABLE` before executing the statement. (This does not work with an ODBC 2.x driver.)

Long data can be added in parts by **SQLBulkOperations**, using calls to **SQLParamData** and **SQLPutData**. For more information, see “Providing Long Data for Bulk Inserts and Updates” later in this function reference.

It is not necessary for the application to call **SQLFetch** or **SQLFetchScroll** before calling **SQLBulkOperations** (except when going against an ODBC 2.x driver; see the Part I PDF file, “Backward Compatibility and Standards Compliance” in Chapter 17, “Programming Considerations”).

The behavior is driver-defined if **SQLBulkOperations**, with an *Operation* argument of `SQL_ADD`, is called on a cursor that contains duplicate columns. The driver can return a driver-defined `SQLSTATE`, add the data to the first column that appears in the result set, or perform other driver-defined behavior.

Performing Bulk Updates Using Bookmarks

To perform bulk updates using bookmarks with **SQLBulkOperations**, an application performs the following steps in sequence:

- Sets the `SQL_ATTR_USE_BOOKMARKS` statement attribute to `SQL_UB_VARIABLE`.

- Executes a query that returns a result set.
- Sets the `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute to the number of rows that it wants to update.
- Calls **SQLBindCol** to bind the data that it wants to update. The data is bound to an array with a size equal to the value of `SQL_ATTR_ROW_ARRAY_SIZE`. It also calls **SQLBindCol** to bind column 0 (the bookmark column).
- Copies the bookmarks for rows that it is interested in updating into the array bound to column 0.
- Updates the data in the bound buffers.

Note The size of the array pointed to by the `SQL_ATTR_ROW_STATUS_PTR` statement attribute should either be equal to `SQL_ATTR_ROW_ARRAY_SIZE` or `SQL_ATTR_ROW_STATUS_PTR` should be a null pointer.

- Calls **SQLBulkOperations**(*StatementHandle*, `SQL_UPDATE_BY_BOOKMARK`).

Note If the application has set the `SQL_ATTR_ROW_STATUS_PTR` statement attribute, it can inspect this array to see the result of the operation.

Optionally calls **SQLBulkOperations**(*StatementHandle*, `SQL_FETCH_BY_BOOKMARK`) to fetch data into the bound application buffers to verify that the update has occurred.

If data has been updated, the driver changes the value in the row status array for the appropriate rows to `SQL_ROW_UPDATED`.

Bulk updates performed by **SQLBulkOperations** can include long data by using calls to **SQLParamData** and **SQLPutData**. For more information, see "Providing Long Data for Bulk Inserts and Updates" later in this function reference.

If bookmarks persist across cursors, the application does not need to call **SQLFetch** or **SQLFetchScroll** before updating by bookmarks. It can use bookmarks that it has stored from a previous cursor. If bookmarks do not persist across cursors, the application has to call **SQLFetch** or **SQLFetchScroll** to retrieve the bookmarks.

The behavior is driver-defined if **SQLBulkOperations**, with an *Operation* argument of `SQL_UPDATE_BY_BOOKMARK`, is called on a cursor that contains duplicate columns. The driver can return a driver-defined `SQLSTATE`, update the first column that appears in the result set, or perform other driver-defined behavior.

Performing Bulk Fetches Using Bookmarks

To perform bulk fetches using bookmarks with **SQLBulkOperations**, an application performs the following steps in sequence:

- Sets the `SQL_ATTR_USE_BOOKMARKS` statement attribute to `SQL_UB_VARIABLE`.
- Executes a query that returns a result set.
- Sets the `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute to the number of rows that it wants to fetch.

- Calls **SQLBindCol** to bind the data that it wants to fetch. The data is bound to an array with a size equal to the value of `SQL_ATTR_ROW_ARRAY_SIZE`. It also calls **SQLBindCol** to bind column 0 (the bookmark column).
- Copies the bookmarks for rows that it is interested in fetching into the array bound to column 0. (This assumes that the application has already obtained the bookmarks separately.)
Note The size of the array pointed to by the `SQL_ATTR_ROW_STATUS_PTR` statement attribute should either be equal to `SQL_ATTR_ROW_ARRAY_SIZE` or `SQL_ATTR_ROW_STATUS_PTR` should be a null pointer.
- Calls **SQLBulkOperations**(*StatementHandle*, `SQL_FETCH_BY_BOOKMARK`).
- If the application has set the `SQL_ATTR_ROW_STATUS_PTR` statement attribute, then it can inspect this array to see the result of the operation.

If bookmarks persist across cursors, the application does not need to call **SQLFetch** or **SQLFetchScroll** before fetching by bookmarks. It can use bookmarks that it has stored from a previous cursor. If bookmarks do not persist across cursors, the application has to call **SQLFetch** or **SQLFetchScroll** once to retrieve the bookmarks.

Performing Bulk Deletes Using Bookmarks

To perform bulk deletes using bookmarks with **SQLBulkOperations**, an application performs the following steps in sequence:

- Sets the `SQL_ATTR_USE_BOOKMARKS` statement attribute to `SQL_UB_VARIABLE`.
- Executes a query that returns a result set.
- Sets the `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute to the number of rows that it wants to delete.
- Calls **SQLBindCol** to bind column 0 (the bookmark column).
- Copies the bookmarks for rows that it is interested in deleting into the array bound to column 0.
Note The size of the array pointed to by the `SQL_ATTR_ROW_STATUS_PTR` statement attribute should either be equal to `SQL_ATTR_ROW_ARRAY_SIZE` or `SQL_ATTR_ROW_STATUS_PTR` should be a null pointer.
- Calls **SQLBulkOperations**(*StatementHandle*, `SQL_DELETE_BY_BOOKMARK`).
- If the application has set the `SQL_ATTR_ROW_STATUS_PTR` statement attribute, it can inspect this array to see the result of the operation.

If bookmarks persist across cursors, the application does not need to call **SQLFetch** or **SQLFetchScroll** before deleting by bookmarks. It can use bookmarks that it has stored from a previous cursor. If bookmarks do not persist across cursors, the application has to call **SQLFetch** or **SQLFetchScroll** once to retrieve the bookmarks.

Providing Long Data for Bulk Inserts and Updates

Long data can be provided for bulk inserts and updates performed by calls to **SQLBulkOperations**. To insert or update long data, an application performs the following steps in addition to the steps described in the "Performing Bulk Inserts" and "Performing Bulk Updates Using Bookmarks" sections earlier in this section.

When it binds the data using **SQLBindCol**, the application places an application-defined value, such as the column number, in the **TargetValuePtr* buffer for data-at-execution columns. The value can be used later to identify the column.

The application places the result of the `SQL_LEN_DATA_AT_EXEC(length)` macro in the **StrLen_or_IndPtr* buffer. If the SQL data type of the column is `SQL_LONGVARIABLE`, `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.

When **SQLBulkOperations** is called, if there are data-at-execution columns, the function returns `SQL_NEED_DATA` and proceeds to step 3 below. (If there are no data-at-execution columns, the process is complete.)

The application calls **SQLParamData** to retrieve the address of the **TargetValuePtr* buffer for the first data-at-execution column to be processed. **SQLParamData** returns `SQL_NEED_DATA`. The application retrieves the application-defined value from the **TargetValuePtr* 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 columns are columns in a rowset for which data will be sent with **SQLPutData** when a row is updated or inserted with **SQLBulkOperations**. They are bound with **SQLBindCol**. The value returned by **SQLParamData** is the address of the row in the **TargetValuePtr* buffer that is being processed.

The application calls **SQLPutData** one or more times to send data for the column. More than one call is needed if all the data value cannot be returned in the **TargetValuePtr* buffer specified in **SQLPutData**; 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.

The application 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 *TargetValuePtr* 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 **SQLBulkOperations**.

If the operation is canceled or an error occurs in **SQLParamData** or **SQLPutData** after **SQLBulkOperations** returns `SQL_NEED_DATA` and before data is sent for all data-at-execution columns, the application can call only **SQLCancel**, **SQLGetDiagField**, **SQLGetDiagRec**, **SQLGetFunctions**, **SQLParamData**, or **SQLPutData** for the statement or the connection associated with the statement. If it calls any other function for the statement or the connection associated with the statement, the function returns `SQL_ERROR` and `SQLSTATE HY010` (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 **SQLBulkOperations** again; canceling does not affect the cursor state or the current cursor position.

Row Status Array

The row status array contains status values for each row of data in the rowset after a call to **SQLBulkOperations**. The driver sets the status values in this array after a call to **SQLFetch**, **SQLFetchScroll**, **SQLSetPos**, or **SQLBulkOperations**. This array is initially populated by a call to **SQLBulkOperations** if **SQLFetch** or **SQLFetchScroll** has not been called prior to **SQLBulkOperations**. This array is pointed to by the `SQL_ATTR_ROW_STATUS_PTR` statement attribute. The number of elements in the row status arrays must equal the number of rows in the rowset (as defined by the `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute). For information about this row status array, see [SQLFetch](#).

Code Example

The following example fetches 10 rows of data at a time from the Customers table. It then prompts the user for an action to take. To reduce network traffic, the example buffer updates, deletes, and inserts locally in the bound arrays, but at offsets past the rowset data. When the user chooses to send updates, deletes, and inserts to the data source, the code sets the binding offset appropriately and calls **SQLBulkOperations**. For simplicity, the user cannot buffer more than 10 updates, deletes, or inserts.

```
#define UPDATE_ROW 100
#define DELETE_ROW 101
#define ADD_ROW 102
#define SEND_TO_DATA_SOURCE 103
#define UPDATE_OFFSET 10
#define INSERT_OFFSET 20
#define DELETE_OFFSET 30

// Define a structure to hold the customer data. Assume we know the maximum
bookmark
// size to be 10 bytes.
typedef tagCustStruct {
    SQLCHAR Bookmark[10];
    SQLINTEGER BookmarkLen;
    SQLUINTEGER CustID;
    SQLINTEGER CustIDInd;
    SQLCHAR Name[51];
    SQLINTEGER NameLenOrInd;
    SQLCHAR Address[51];
    SQLINTEGER AddressLenOrInd;
    SQLCHAR Phone[11];
    SQLINTEGER PhoneLenOrInd;
} CustStruct;

// Allocate 40 of these structures. Elements 0-9 are for the current rowset,
// elements 10-19 are for the buffered updates, elements 20-29 are for
// the buffered inserts, and elements 30-39 are for the buffered deletes.
CustStruct CustArray[40];
SQLUSMALLINT RowStatusArray[10], Action, RowNum, NumUpdates = 0, NumInserts =
0,
NumDeletes = 0;
SQLINTEGER BindOffset = 0;
SQLRETURN rc;
SQLHSTMT hstmt;
```

```

// Set the following statement attributes:
// SQL_ATTR_CURSOR_TYPE: Keyset-driven
// SQL_ATTR_ROW_BIND_TYPE: Row-wise
// SQL_ATTR_ROW_ARRAY_SIZE: 10
// SQL_ATTR_USE_BOOKMARKS: Use variable-length bookmarks
// SQL_ATTR_ROW_STATUS_PTR: Points to RowStatusArray
// SQL_ATTR_ROW_BIND_OFFSET_PTR: Points to BindOffset
SQLSetStmtAttr(hstmt, SQL_ATTR_CURSOR_TYPE, SQL_CURSOR_KEYSET_DRIVEN, 0);
SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_BIND_TYPE, sizeof(CustStruct), 0);
SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_ARRAY_SIZE, 10, 0);
SQLSetStmtAttr(hstmt, SQL_ATTR_USE_BOOKMARKS, SQL_UB_VARIABLE, 0);
SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_STATUS_PTR, RowStatusArray, 0);
SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_BIND_OFFSET_PTR, &BindOffset, 0);
// Bind arrays to the bookmark, CustID, Name, Address, and Phone columns.
SQLBindCol(hstmt, 0, SQL_C_VARBOOKMARK, CustArray[0].Bookmark,
sizeof(CustArray[0].Bookmark), &CustArray[0].BookmarkLen);
SQLBindCol(hstmt, 1, SQL_C_ULONG, &CustArray[0].CustID, 0,
&CustArray[0].CustIDInd);
SQLBindCol(hstmt, 2, SQL_C_CHAR, CustArray[0].Name, sizeof(CustArray[0].Name),
&CustArray[0].NameLenOrInd);
SQLBindCol(hstmt, 3, SQL_C_CHAR, CustArray[0].Address,
sizeof(CustArray[0].Address),
&CustArray[0].AddressLenOrInd);
SQLBindCol(hstmt, 4, SQL_C_CHAR, CustArray[0].Phone,
sizeof(CustArray[0].Phone),
&CustArray[0].PhoneLenOrInd);
// Execute a statement to retrieve rows from the Customers table.
SQLExecDirect(hstmt, "SELECT CustID, Name, Address, Phone FROM Customers",
SQL_NTS);
// Fetch and display the first 10 rows.
rc = SQLFetchScroll(hstmt, SQL_FETCH_NEXT, 0);
DisplayCustData(CustArray, 10);
// Call GetAction to get an action and a row number from the user.
while (GetAction(&Action, &RowNum)) {
switch (Action) {

case SQL_FETCH_NEXT:
case SQL_FETCH_PRIOR:
case SQL_FETCH_FIRST:
case SQL_FETCH_LAST:
case SQL_FETCH_ABSOLUTE:
case SQL_FETCH_RELATIVE:
// Fetch and display the requested data.
SQLFetchScroll(hstmt, Action, RowNum);
DisplayCustData(CustArray, 10);
break;

case UPDATE_ROW:
// Check if we have reached the maximum number of buffered updates.
if (NumUpdates < 10) {
// Get the new customer data and place it in the next available element of
// the buffered updates section of CustArray, copy the bookmark of the row
// being updated to the same element, and increment the update counter.
// Checking to see we have not already buffered an update for this
// row not shown.
GetNewCustData(CustArray, UPDATE_OFFSET + NumUpdates);
memcpy(CustArray[UPDATE_OFFSET + NumUpdates].Bookmark,
CustArray[RowNum - 1].Bookmark,

```

```

CustArray[RowNum - 1].BookmarkLen);
CustArray[UPDATE_OFFSET + NumUpdates].BookmarkLen =
CustArray[RowNum - 1].BookmarkLen;
NumUpdates++;
} else {
DisplayError("Buffers full. Send buffered changes to the data source.");
}
break;
case DELETE_ROW:
// Check if we have reached the maximum number of buffered deletes.
if (NumDeletes < 10) {
// Copy the bookmark of the row being deleted to the next available element
// of the buffered deletes section of CustArray and increment the delete
// counter. Checking to see we have not already buffered an update for
// this row not shown.
memcpy(CustArray[DELETE_OFFSET + NumDeletes].Bookmark,
CustArray[RowNum - 1].Bookmark,
CustArray[RowNum - 1].BookmarkLen);
CustArray[DELETE_OFFSET + NumDeletes].BookmarkLen =
CustArray[RowNum - 1].BookmarkLen;
NumDeletes++;
} else {
DisplayError("Buffers full. Send buffered changes to the data source.");
}
break;
case ADD_ROW:
// Check if we have reached the maximum number of buffered inserts.
if (NumInserts < 10) {
// Get the new customer data and place it in the next available element of
// the buffered inserts section of CustArray and increment the insert
// counter.
GetNewCustData(CustArray, INSERT_OFFSET + NumInserts);
NumInserts++;
} else {
DisplayError("Buffers full. Send buffered changes to the data source.");
}
break;
case SEND_TO_DATA_SOURCE:
// If there are any buffered updates, inserts, or deletes, set the array size
// to that number, set the binding offset to use the data in the buffered
// update, insert, or delete part of CustArray, and call SQLBulkOperations to
// do the updates, inserts, or deletes. Because we will never have more than
// 10 updates, inserts, or deletes, we can use the same row status array.
if (NumUpdates) {
SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_ARRAY_SIZE, NumUpdates, 0);
BindOffset = UPDATE_OFFSET * sizeof(CustStruct);
SQLBulkOperations(hstmt, SQL_UPDATE_BY_BOOKMARK);
NumUpdates = 0;
}

if (NumInserts) {
SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_ARRAY_SIZE, NumInserts, 0);
BindOffset = INSERT_OFFSET * sizeof(CustStruct);
SQLBulkOperations(hstmt, SQL_ADD);
NumInserts = 0;
}

if (NumDeletes) {
SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_ARRAY_SIZE, NumDeletes, 0);

```



```

BindOffset = DELETE_OFFSET * sizeof(CustStruct);
SQLBulkOperations(hstmt, SQL_DELETE_BY_BOOKMARK);
NumDeletes = 0;
}

// If there were any updates, inserts, or deletes, reset the binding offset
// and array size to their original values.
if (NumUpdates || NumInserts || NumDeletes) {
    SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_ARRAY_SIZE, 10, 0);
    BindOffset = 0;
}
break;
}
}

// Close the cursor.
SQLFreeStmt(hstmt, SQL_CLOSE);

```

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Getting a single field of a descriptor	SQLGetDescField
Getting multiple fields of a descriptor	SQLGetDescRec
Setting a single field of a descriptor	SQLSetDescField
Setting multiple fields of a descriptor	SQLSetDescRec
Positioning the cursor, refreshing data in the rowset, or updating or deleting data in the rowset	SQLSetPos
Setting a statement attribute	SQLSetStmtAttr

SQLCancel

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

SQLCancel cancels the processing on a statement.

Syntax

```
SQLRETURN SQLCancel(  
    SQLHSTMT StatementHandle);
```

Arguments

StatementHandle

[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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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.)
HY000	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 SQLGetDiagRec in the argument <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY018	Server declined cancel	The server declined the cancel request.

	request	
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

SQLCancel can cancel the following types of processing on a statement:

- A function running asynchronously on the statement.
- A function on a statement that needs data.
- A function running on the statement on another thread.

In ODBC 2.x, if an application calls **SQLCancel** when no processing is being done on the statement, **SQLCancel** has the same effect as **SQLFreeStmt** with the **SQL_CLOSE** option; this behavior is defined only for completeness and applications should call **SQLFreeStmt** or **SQLCloseCursor** to close cursors.

When **SQLCancel** is called, diagnostic records are returned for a function running asynchronously in a statement or for a function on a statement that needs data; diagnostic records are not returned, however, for a function running on a statement on another thread.

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-dependent 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** HY008 (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 HY008 (Operation canceled) if the function failed.

Note In ODBC 3.5, a call to **SQLCancel** when no processing is being done on the statement is not treated as **SQLFreeStmt** with the **SQL_CLOSE** option, but has no effect at all. To close a cursor, an application should call **SQLCloseCursor**, not **SQLCancel**.

For more information about asynchronous processing, see the Part I PDF file, "Asynchronous Execution" in Chapter 9, "Executing Statements."

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 **SQLBulkOperations** or **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 **SQLBulkOperations** or **SQLSetPos** again; canceling does not affect the cursor state or the current cursor position. For more information, see [SQLBulkOperations](#) or [SQLSetPos](#).

Canceling Functions in Multithread Applications

In a multithread application, the application can cancel a function that is running synchronously on a statement. To cancel the function, the application calls **SQLCancel** with the same statement handle as that used by the target function, but on a different thread. How the function is canceled depends on the driver and the operating system. As in canceling a function running asynchronously, the return code of the **SQLCancel** indicates only whether the driver processed the request successfully. Only `SQL_SUCCESS` or `SQL_ERROR` can be returned; no `SQLSTATE`s are returned. If the original function is canceled, it returns `SQL_ERROR` and `SQLSTATE HY008` (Operation canceled).

If an SQL statement is being executed when **SQLCancel** is called on another thread to cancel the statement execution, it is possible for the execution to succeed and return `SQL_SUCCESS` while the cancel is also successful. In this case, the Driver Manager assumes that the cursor opened by the statement execution is closed by the cancel, so the application will not be able to use the cursor.

For more information about threading, see the Part I PDF file, "Multithreading" in Chapter 17, "Programming Considerations."

Related Functions

For information about	See
Binding a buffer to a parameter	SQLBindParameter
Performing bulk insert or update operations	SQLBulkOperations
Executing an SQL statement	SQLExecDirect
Executing a prepared SQL statement	SQLExecute
Freeing a statement handle	SQLFreeStmt
Obtaining a field of a diagnostic record or a field of the diagnostic header	SQLGetDiagField
Obtaining multiple fields of a diagnostic data structure	SQLGetDiagRec
Returning the next parameter to send data for	SQLParamData
Sending parameter data at execution time	SQLPutData
Positioning the cursor in a rowset, refreshing data in the rowset, or updating or deleting data in the result set	SQLSetPos

SQLCloseCursor

Conformance

Version Introduced: ODBC 3.0

Standards Compliance: ISO 92

Summary

SQLCloseCursor closes a cursor that has been opened on a statement and discards pending results.

Syntax

```
SQLRETURN SQLCloseCursor(  
    SQLHSTMT StatementHandle);
```

Arguments

StatementHandle

[Input]

Statement handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLCloseCursor** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. The following table lists the SQLSTATE values commonly returned by **SQLCloseCursor** 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	No cursor was open on the <i>StatementHandle</i> . (This is returned only by an ODBC 3.x driver.)
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error	(DM) An asynchronously executing function was called for the <i>StatementHandle</i> and was still executing

		when this function was called. (DM) SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA . This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

SQLCloseCursor returns SQLSTATE 24000 (Invalid cursor state) if no cursor is open. Calling **SQLCloseCursor** is equivalent to calling **SQLFreeStmt** with the **SQL_CLOSE** option, with the exception that **SQLFreeStmt** with **SQL_CLOSE** has no effect on the application if no cursor is open on the statement, while **SQLCloseCursor** returns SQLSTATE 24000 (Invalid cursor state).

Note If an ODBC 3.x application working with an ODBC 2.x driver calls **SQLCloseCursor** when no cursor is open, SQLSTATE 24000 (Invalid cursor state) is not returned, because the Driver Manager maps **SQLCloseCursor** to **SQLFreeStmt** with **SQL_CLOSE**.

For more information, see the Part I PDF file, “Closing the Cursor” in Chapter 10, “Retrieving Results (Basic).”

Code Example

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

Related Functions

For information about	See
Canceling statement processing	SQLCancel
Freeing a handle	SQLFreeHandle
Processing multiple result sets	SQLMoreResults

SQLColAttribute

Conformance

Version Introduced: ODBC 3.0
Standards Compliance: ISO 92

Summary

SQLColAttribute returns descriptor information for a column in a result set. Descriptor information is returned as a character string, a 32-bit descriptor-dependent value, or an integer value.

Note For more information about what the Driver Manager maps this function to when an ODBC 3.x application is working with an ODBC 2.x driver, see the Part 1 PDF file, "Mapping Replacement Functions for Backward Compatibility of Applications" in Chapter 17, "Programming Considerations."

Syntax

SQLRETURN SQLColAttribute (
SQLHSTMT	StatementHandle,
SQLUSMALLINT	ColumnNumber,
SQLUSMALLINT	FieldIdentifier,
SQLPOINTER	CharacterAttributePtr,
SQLSMALLINT	BufferLength,
SQLSMALLINT *	StringLengthPtr,
SQLPOINTER	NumericAttributePtr);

Arguments

StatementHandle

[Input]
Statement handle.

ColumnNumber

[Input]
The number of the record in the IRD from which the field value is to be retrieved. This argument corresponds to the column number of result data, ordered sequentially in increasing column order, starting at 1. Columns can be described in any order.

Column 0 can be specified in this argument, but all values except SQL_DESC_TYPE and SQL_DESC_OCTET_LENGTH will return undefined values.

FieldIdentifier

[Input]
The field in row *ColumnNumber* of the IRD that is to be returned (see "Comments").

CharacterAttributePtr

[Output]

Pointer to a buffer in which to return the value in the *FieldIdentifier* field of the *ColumnNumber* row of the IRD, if the field is a character string. Otherwise, the field is unused.

BufferLength

[Input]

If *FieldIdentifier* is an ODBC-defined field and *CharacterAttributePtr* points to a character string or binary buffer, this argument should be the length of **CharacterAttributePtr*. If *FieldIdentifier* is an ODBC-defined field and **CharacterAttributePtr* is an integer, this field is ignored. If the **CharacterAttributePtr* is a Unicode string (when calling **SQLColAttributeW**), the *BufferLength* argument must be an even number. If *FieldIdentifier* is a driver-defined field, the application indicates the nature of the field to the Driver Manager by setting the *BufferLength* argument. *BufferLength* can have the following values:

If *CharacterAttributePtr* is a pointer to a pointer, *BufferLength* should have the value `SQL_IS_POINTER`. If *CharacterAttributePtr* is a pointer to a character string, the *BufferLength* is the length of the buffer. If *CharacterAttributePtr* is a pointer to a binary buffer, the application places the result of the `SQL_LEN_BINARY_ATTR(length)` macro in *BufferLength*. This places a negative value in *BufferLength*. If *CharacterAttributePtr* is a pointer to a fixed-length data type, *BufferLength* must be one of the following: `SQL_IS_INTEGER`, `SQL_IS_UNINTEGER`, `SQL_SMALLINT`, or `SQLUSMALLINT`.

StringLengthPtr

[Output]

Pointer to a buffer in which to return the total number of bytes (excluding the null-termination byte for character data) available to return in **CharacterAttributePtr*.

For character data, if the number of bytes available to return is greater than or equal to *BufferLength*, the descriptor information in **CharacterAttributePtr* is truncated to *BufferLength* minus the length of a null-termination character and is null-terminated by the driver.

For all other types of data, the value of *BufferLength* is ignored and the driver assumes the size of **CharacterAttributePtr* is 32 bits.

NumericAttributePtr

[Output]

Pointer to an integer buffer in which to return the value in the *FieldIdentifier* field of the *ColumnNumber* row of the IRD, if the field is a numeric descriptor type, such as `SQL_DESC_COLUMN_LENGTH`. Otherwise, the field is unused.

Returns

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

Diagnostics

When **SQLColAttribute** returns either `SQL_ERROR` or `SQL_SUCCESS_WITH_INFO`, an associated `SQLSTATE` value may be obtained by calling **SQLGetDiagRec** with a *HandleType* of `SQL_HANDLE_STMT` and a *Handle* of *StatementHandle*. The following table lists the `SQLSTATE` values commonly returned by **SQLColAttribute** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of `SQLSTATE`s 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 <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	String data, right truncated	The buffer <i>*CharacterAttributePtr</i> was not large enough to return the entire string value, so the string value was truncated. The length of the untruncated string value is returned in <i>*StringLengthPtr</i> . (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
07005	Prepared statement not a <i>cursor-specification</i>	The statement associated with the <i>StatementHandle</i> did not return a result set and <i>FieldIdentifier</i> was not <code>SQL_DESC_COUNT</code> . There were no columns to describe.
07009	Invalid descriptor index	(DM) The value specified for <i>ColumnNumber</i> was equal to 0, and the <code>SQL_ATTR_USE_BOOKMARKS</code> statement attribute was <code>SQL_UB_OFF</code> . The value specified for the argument <i>ColumnNumber</i> was greater than the number of columns in the result set.
HY000	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 SQLGetDiagField from the diagnostic data structure describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	Asynchronous processing was enabled for the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> . Then the function was called again on the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.
HY010	Function sequence error	(DM) The function was called prior to calling SQLPrepare , SQLExecDirect , or a catalog function for the <i>StatementHandle</i> . (DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called. (DM) SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos was called for the <i>StatementHandle</i> and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.

HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	(DM) <i>*CharacterAttributePtr</i> is a character string, and <i>BufferLength</i> was less than 0 but not equal to SQL_NTS.
HY091	Invalid descriptor field identifier	The value specified for the argument <i>FieldIdentifier</i> was not one of the defined values and was not an implementation-defined value.
HYC00	Driver not capable	The value specified for the argument <i>FieldIdentifier</i> was not supported by the driver.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

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

For performance reasons, an application should not call **SQLColAttribute** before executing a statement.

Comments

For information about how applications use the information returned by **SQLColAttribute**, see the Part I PDF file, "ResultSet Meta Data" in Chapter 10, "Retrieving Results (Basic)."

SQLColAttribute returns information either in **NumericAttributePtr* or in **CharacterAttributePtr*. Integer information is returned in **NumericAttributePtr* as a 32-bit, signed value; all other formats of information are returned in **CharacterAttributePtr*. When information is returned in **NumericAttributePtr*, the driver ignores *CharacterAttributePtr*, *BufferLength*, and *StringLengthPtr*. When information is returned in **CharacterAttributePtr*, the driver ignores *NumericAttributePtr*.

SQLColAttribute returns values from the descriptor fields of the IRD. The function is called with a statement handle rather than a descriptor handle. The values returned by **SQLColAttribute** for the *FieldIdentifier* values listed later in this section can also be retrieved by calling **SQLGetDescField** with the appropriate IRD handle.

The currently defined descriptor fields, the version of ODBC in which they were introduced, and the arguments in which information is returned for them are shown later in this section; more descriptor types may be defined by drivers to take advantage of different data sources.

An ODBC 3.x driver must return a value for each of the descriptor fields. If a descriptor field does not apply to a driver or data source and unless otherwise stated, the driver returns 0 in **StringLengthPtr* or an empty string in **CharacterAttributePtr*.

Backward Compatibility

The ODBC 3.x function **SQLColAttribute** replaces the deprecated ODBC 2.x function **SQLColAttributes**. When mapping **SQLColAttributes** to **SQLColAttribute** (when an ODBC 2.x application is working with an ODBC 3.x driver), or mapping **SQLColAttribute** to **SQLColAttributes** (when an ODBC 3.x application is working with an ODBC 2.x driver), the Driver Manager either passes the value of *FieldIdentifier* through, maps it to a new value, or returns an error, as follows:

Note The prefix used in *FieldIdentifier* values in ODBC 3.x has been changed from that used in ODBC 2.x. The new prefix is "SQL_DESC"; the old prefix was "SQL_COLUMN".

If the **#define** value of the ODBC 2.x *FieldIdentifier* is the same as the **#define** value of the ODBC 3.x *FieldIdentifier*, the value in the function call is just passed through.

The **#define** values of the ODBC 2.x *FieldIdentifiers* SQL_COLUMN_LENGTH, SQL_COLUMN_PRECISION, and SQL_COLUMN_SCALE are different from the **#define** values of the ODBC 3.x *FieldIdentifiers* SQL_DESC_PRECISION, SQL_DESC_SCALE, and SQL_DESC_LENGTH. An ODBC 2.x driver need only support the ODBC 2.x values. An ODBC 3.x driver must support both "SQL_COLUMN" and "SQL_DESC" values for these three *FieldIdentifiers*. These values are different because precision, scale, and length are defined differently in ODBC 3.x than they were in ODBC 2.x. For more information, see Appendix D, "Data Types" of the **SOLID Programmer Guide**.

If the **#define** value of the ODBC 2.x *FieldIdentifier* is different from the **#define** value of the ODBC 3.x *FieldIdentifier*, as occurs with the COUNT, NAME, and NULLABLE values, the value in the function call is mapped to the corresponding value. For example, SQL_COLUMN_COUNT is mapped to SQL_DESC_COUNT, and SQL_DESC_COUNT is mapped to SQL_COLUMN_COUNT, depending on the direction of the mapping.

If *FieldIdentifier* is a new value in ODBC 3.x, for which there was no corresponding value in ODBC 2.x, it will not be mapped when an ODBC 3.x application uses it in a call to **SQLColAttribute** in an ODBC 2.x driver, and the call will return SQLSTATE HY091 (Invalid descriptor field identifier).

The following table lists the descriptor types returned by **SQLColAttribute**.

FieldIdentifier	Information returned in	Description
SQL_DESC_AUTO_UNIQUE_VALUE (ODBC 1.0)	NumericAttributePtr	SQL_TRUE if the column is an autoincrementing column. SQL_FALSE if the column is not an autoincrementing column or is not numeric. This field is valid for numeric data type columns only. An application can insert values into a row containing an autoincrement column, but typically cannot update values in the column. When an insert is made into an autoincrement column, a unique value is inserted into the column at insert time. The increment is not defined, but is data source-specific. An application should not assume that an autoincrement column starts at any particular point or increments by any particular value.

SQL_DESC_BASE_COLUMN_NAME (ODBC 3.0)	CharacterAttributePtr	The base column name for the result set column. If a base column name does not exist (as in the case of columns that are expressions), then this variable contains an empty string. This information is returned from the SQL_DESC_BASE_COLUMN_NAME record field of the IRD, which is a read-only field.
SQL_DESC_BASE_TABLE_NAME (ODBC 3.0)	CharacterAttributePtr	The name of the base table that contains the column. If the base table name cannot be defined or is not applicable, then this variable contains an empty string. This information is returned from the SQL_DESC_BASE_TABLE_NAME record field of the IRD, which is a read-only field.
SQL_DESC_CASE_SENSITIVE (ODBC 1.0)	NumericAttributePtr	SQL_TRUE if the column is treated as case-sensitive for collations and comparisons. SQL_FALSE if the column is not treated as case-sensitive for collations and comparisons or is noncharacter.
SQL_DESC_CATALOG_NAME (ODBC 2.0)	CharacterAttributePtr	The catalog 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 catalogs or the catalog name cannot be determined, an empty string is returned. This VARCHAR record field is not limited to 128 characters.
SQL_DESC_CONCISE_TYPE (ODBC 1.0)	NumericAttributePtr	The concise data type. For the datetime and interval data types, this field returns the concise data type; for example, SQL_TYPE_TIME or SQL_INTERVAL_YEAR. (For more information, see “Data Type Identifiers and Descriptors” in Appendix D, of the SOLID Programmer Guide . This information is returned from the SQL_DESC_CONCISE_TYPE record field of the IRD.
SQL_DESC_COUNT (ODBC 1.0)	NumericAttributePtr	The number of columns available in the result set. This returns 0 if there are no columns in the result set. The value in the ColumnNumber argument is ignored. This information is returned from the SQL_DESC_COUNT header field of the IRD.

SQL_DESC_DISPLAY_SIZE (ODBC 1.0)	NumericAttributePtr	Maximum number of characters required to display data from the column. For more information about display size, see “Column Size, Decimal Digits, Transfer Octet Length, and Display Size” contained on the Microsoft Web site (ODBC Programming Reference).
SQL_DESC_FIXED_PREC_SCALE (ODBC 1.0)	NumericAttributePtr	SQL_TRUE if the column has a fixed precision and nonzero scale that are data source-specific. SQL_FALSE if the column does not have a fixed precision and nonzero scale that are data source-specific.
SQL_DESC_LABEL (ODBC 2.0)	CharacterAttributePtr	The column label or title. For example, a column named EmpName might be labeled Employee Name or might be labeled with an alias. 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_DESC_LENGTH (ODBC 3.0)	NumericAttributePtr	A numeric value that is either the maximum or actual character length of a character string or binary data type. It is the maximum character length for a fixed-length data type, or the actual character length for a variable-length data type. Its value always excludes the null-termination byte that ends the character string. This information is returned from the SQL_DESC_LENGTH record field of the IRD. For more information about length, Appendix D, “Data Types” of the SOLID Programmer Guide .
SQL_DESC_LITERAL_PREFIX (ODBC 3.0)	CharacterAttributePtr	This VARCHAR(128) record field contains the character or characters that the driver recognizes as a prefix for a literal of this data type. This field contains an empty string for a data type for which a literal prefix is not applicable. For more information, see the Part I PDF file, “Literal Prefixes and Suffixes” in Chapter 8, “SQL Statements.”
SQL_DESC_LITERAL_SUFFIX (ODBC 3.0)	CharacterAttributePtr	This VARCHAR(128) record field contains the character or characters that the driver recognizes as a suffix for a literal of this data type. This field contains an empty string for a data type for which a literal suffix is not applicable. For more

		information, see the Part I PDF file, "Literal Prefixes and Suffixes" in Chapter 8, "SQL Statements."
SQL_DESC_LOCAL_TYPE_NAME (ODBC 3.0)	CharacterAttributePtr	This VARCHAR(128) record field contains any localized (native language) name for the data type that may be different from the regular name of the data type. If there is no localized name, then an empty string is returned. This field is for display purposes only. The character set of the string is locale-dependent and is typically the default character set of the server.
SQL_DESC_NAME (ODBC 3.0)	CharacterAttributePtr	The column alias, if it applies. If the column alias does not apply, the column name is returned. In either case, SQL_DESC_UNNAMED is set to SQL_NAMED. If there is no column name or a column alias, an empty string is returned and SQL_DESC_UNNAMED is set to SQL_UNNAMED. This information is returned from the SQL_DESC_NAME record field of the IRD.
SQL_DESC_NULLABLE (ODBC 3.0)	NumericAttributePtr	SQL_NULLABLE if the column can have NULL values; SQL_NO_NULLS if the column does not have NULL values; or SQL_NULLABLE_UNKNOWN if it is not known whether the column accepts NULL values. This information is returned from the SQL_DESC_NULLABLE record field of the IRD.
SQL_DESC_NUM_PREX_RADIX (ODBC 3.0)	NumericAttributePtr	If the data type in the SQL_DESC_TYPE field is an approximate numeric data type, this SQLINTEGER field contains a value of 2 because the SQL_DESC_PRECISION field contains the number of bits. If the data type in the SQL_DESC_TYPE field is an exact numeric data type, this field contains a value of 10 because the SQL_DESC_PRECISION field contains the number of decimal digits. This field is set to 0 for all non-numeric data types.
SQL_DESC_OCTET_LENGTH (ODBC 3.0)	NumericAttributePtr	The length, in bytes, of a character string or binary data type. For fixed-length character or binary types, this is the actual length in bytes. For variable-length character or binary types, this is the maximum length in bytes. This value includes the null

		<p>terminator.</p> <p>This information is returned from the SQL_DESC_OCTET_LENGTH record field of the IRD.</p> <p>For more information about length, see Appendix D, “Data Types” in the SOLID Programmer Guide.</p>
SQL_DESC_PRECISION (ODBC 3.0)	NumericAttributePtr	<p>A numeric value that for a numeric data type denotes the applicable precision. For data types SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP, and all the interval data types that represent a time interval, its value is the applicable precision of the fractional seconds component.</p> <p>This information is returned from the SQL_DESC_PRECISION record field of the IRD.</p>
SQL_DESC_SCALE (ODBC 3.0)	NumericAttributePtr	<p>A numeric value that is the applicable scale for a numeric data type. For DECIMAL and NUMERIC data types, this is the defined scale. It is undefined for all other data types.</p> <p>This information is returned from the SCALE record field of the IRD.</p>
SQL_DESC_SCHEMA_NAME (ODBC 2.0)	CharacterAttributePtr	<p>The schema 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 schemas or the schema name cannot be determined, an empty string is returned. This VARCHAR record field is not limited to 128 characters.</p>
SQL_DESC_SEARCHABLE (ODBC 1.0)	NumericAttributePtr	<p>SQL_PRED_NONE if the column cannot be used in a WHERE clause. (This is the same as the SQL_UNSEARCHABLE value in ODBC 2.x.)</p> <p>SQL_PRED_CHAR if the column can be used in a WHERE clause but only with the LIKE predicate. (This is the same as the SQL_LIKE_ONLY value in ODBC 2.x.)</p> <p>SQL_PRED_BASIC if the column can be used in a WHERE clause with all the comparison operators except LIKE. (This is the same as the SQL_EXCEPT_LIKE value in ODBC 2.x.)</p> <p>SQL_PRED_SEARCHABLE if the column can be used in a WHERE clause with any comparison operator.</p>

		Columns of type SQL_LONGVARCHAR and SQL_LONGVARIABLE usually return SQL_PRED_CHAR.
SQL_DESC_TABLE_NAME (ODBC 2.0)	CharacterAttributePtr	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_DESC_TYPE (ODBC 3.0)	NumericAttributePtr	A numeric value that specifies the SQL data type. When ColumnNumber is equal to 0, SQL_BINARY is returned for variable-length bookmarks and SQL_INTEGER is returned for fixed-length bookmarks. For the datetime and interval data types, this field returns the verbose data type: SQL_DATETIME or SQL_INTERVAL. (For more information, see "Data Type Identifiers and Descriptors" in Appendix D, "Data Types" of the SOLID Programmer Guide . This information is returned from the SQL_DESC_TYPE record field of the IRD. Note To work against ODBC 2.x drivers, use SQL_DESC_CONCISE_TYPE instead.
SQL_DESC_TYPE_NAME (ODBC 1.0)	CharacterAttributePtr	Data source-dependent data 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_DESC_UNNAMED (ODBC 3.0)	NumericAttributePtr	SQL_NAMED or SQL_UNNAMED. If the SQL_DESC_NAME field of the IRD contains a column alias or a column name, SQL_NAMED is returned. If there is no column name or column alias, SQL_UNNAMED is returned. This information is returned from the SQL_DESC_UNNAMED record field of the IRD.
SQL_DESC_UNSIGNED (ODBC 1.0)	NumericAttributePtr	SQL_TRUE if the column is unsigned (or not numeric). SQL_FALSE if the column is signed.
SQL_DESC_UPDATABLE (ODBC 1.0)	NumericAttributePtr	Column is described by the values for the defined constants: SQL_ATTR_READONLY

		SQL_ATTR_WRITE SQL_ATTR_READWRITE_UNKNOWN SQL_DESC_UPDATABLE describes the updatability of the column in the result set, not the column in the base table. The updatability of the base column on which the result set column is based may be different from the value in this field. 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.
--	--	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

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

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Returning information about a column in a result set	SQLDescribeCol
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Fetching multiple rows of data	SQLFetch

SQLColumnPrivileges

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: ODBC

Summary

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 *StatementHandle*.

Syntax

SQLRETURN SQLColumnPrivileges(
SQLHSTMT	StatementHandle,
SQLCHAR *	CatalogName,
SQLSMALLINT	NameLength1,
SQLCHAR *	SchemaName,
SQLSMALLINT	NameLength2,
SQLCHAR *	TableName,
SQLSMALLINT	NameLength3,
SQLCHAR *	ColumnName,
SQLSMALLINT	NameLength4);

Arguments

StatementHandle

[Input]
Statement handle.

CatalogName

[Input]
Catalog name. If a driver supports names for some catalogs but not for others, such as when the driver retrieves data from different DBMSs, an empty string ("") denotes those catalogs that do not have names. *CatalogName* cannot contain a string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *CatalogName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *CatalogName* is an ordinary argument; it is treated literally, and its case is significant. For more information, see the Part I PDF file, "Arguments in Catalog Functions" in Chapter 7, "Catalog Functions."

NameLength1

[Input]
Length of **CatalogName*.

SchemaName

[Input]

Schema name. If a driver supports schemas 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 schemas. *SchemaName* cannot contain a string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *SchemaName* is treated as an identifier. If it is SQL_FALSE, *SchemaName* is an ordinary argument; it is treated literally, and its case is significant.

NameLength2

[Input]

Length of **SchemaName*.

TableName

[Input]

Table name. This argument cannot be a null pointer. *TableName* cannot contain a string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *TableName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *TableName* is an ordinary argument; it is treated literally, and its case is significant.

NameLength3

[Input]

Length of **TableName*.

ColumnName

[Input]

String search pattern for column names.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *ColumnName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *ColumnName* is a pattern value argument; it is treated literally, and its case is significant.

NameLength4

[Input]

Length of **ColumnName*.

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 **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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	A cursor was open on the <i>StatementHandle</i> , and SQLFetch or SQLFetchScroll had been called. This error is returned by the Driver Manager if SQLFetch or SQLFetchScroll has not returned SQL_NO_DATA, and is returned by the driver if SQLFetch or SQLFetchScroll has returned SQL_NO_DATA. A cursor was open on the <i>StatementHandle</i> , but SQLFetch or SQLFetchScroll had not been called.
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
HY000	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 SQLGetDiagRec in the * <i>MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	Asynchronous processing was enabled for the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> . Then the function was called again on the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.
HY009	Invalid use of null pointer	The <i>TableName</i> argument was a null pointer. (DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE, the <i>CatalogName</i> argument was a null pointer, and the SQL_CATALOG_NAME <i>InfoType</i> returns that catalog names are supported. (DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE, and the <i>SchemaName</i> or <i>ColumnName</i> argument was a null pointer.
HY010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still

		executing when this function was called. (DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	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 name. (See "Comments.")
HYC00	Optional feature not implemented	A catalog name was specified, and the driver or data source does not support catalogs. A schema name was specified, and the driver or data source does not support schemas. 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 attributes was not supported by the driver or data source. The SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_VARIABLE, and the SQL_ATTR_CURSOR_TYPE statement attribute was set to a cursor type for which the driver does not support bookmarks.
HYT00	Timeout expired	The query timeout period expired before the data source returned the result set. The timeout period is set through SQLSetStmtAttr, SQL_ATTR_QUERY_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr, SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

SQLColumnPrivileges returns the results as a standard result set, ordered by TABLE_CAT, TABLE_SCHEM, TABLE_NAME, COLUMN_NAME, and PRIVILEGE.

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 are not shown in the table; the actual lengths depend on the data source. To determine the actual lengths of the CATALOG_NAME, SCHEMA_NAME, TABLE_NAME, and COLUMN_NAME columns, an application can call **SQLGetInfo** with the SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN options.

Note For more information about the general use, arguments, and returned data of ODBC catalog functions, see the Part I PDF file, Chapter 7, "Catalog Functions."

The following columns have been renamed for ODBC 3.x. The column name changes do not affect backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM

The following table lists the columns in the result set. Additional columns beyond column 8 (IS_GRANTABLE) can be defined by the driver. An application should gain access to driver-specific columns by counting down from the end of the result set rather than specifying an explicit ordinal position. For more information, see the Part I PDF file, "Data Returned By Catalog Functions" in Chapter 7, "Catalog Functions."

Column name	Column number	Data type	Comments
TABLE_CAT (ODBC 1.0)	1	Varchar	Catalog identifier; NULL if not applicable to the data source. If a driver supports catalogs 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 catalogs.
TABLE_SCHEM (ODBC 1.0)	2	Varchar	Schema identifier; NULL if not applicable to the data source. If a driver supports schemas 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 schemas.
TABLE_NAME (ODBC 1.0)	3	Varchar not NULL	Table identifier.
COLUMN_NAME (ODBC 1.0)	4	Varchar not NULL	Column name. The driver returns an empty string for a column that does not have a name.
GRANTOR (ODBC 1.0)	5	Varchar	Name of the user who granted the privilege; NULL if not applicable to the data source. For all rows in which the value in the GRANTEE column is the owner of the object, the GRANTOR column will be "_SYSTEM".

GRANTEE (ODBC 1.0)	6	Varchar not NULL	Name of the user to whom the privilege was granted.
PRIVILEGE (ODBC 1.0)	7	Varchar 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 (ODBC 1.0)	8	Varchar	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. A privilege is either grantable or not grantable, but not both. The result set returned by SQLColumnPrivileges will never contain two rows for which all columns except the IS_GRANTABLE column contain the same value.

Code Example

For a code example of a similar function, see [SQLColumns](#).

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Returning the columns in a table or tables	SQLColumns
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Fetching multiple rows of data	SQLFetch
Returning privileges for a table or tables	SQLTablePrivileges
Returning a list of tables in a data source	SQLTables

SQLColumns

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: X/Open

Summary

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

Syntax

SQLRETURN SQLColumns(
SQLHSTMT	StatementHandle,
SQLCHAR *	CatalogName,
SQLSMALLINT	NameLength1,
SQLCHAR *	SchemaName,
SQLSMALLINT	NameLength2,
SQLCHAR *	TableName,
SQLSMALLINT	NameLength3,
SQLCHAR *	ColumnName,
SQLSMALLINT	NameLength4);

Arguments

StatementHandle

[Input]

Statement handle.

CatalogName

[Input]

Catalog name. If a driver supports catalogs 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 catalogs. *CatalogName* cannot contain a string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *CatalogName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *CatalogName* is an ordinary argument; it is treated literally, and its case is significant. For more information, see the Part 1 PDF file, Chapter 7, "Catalog Functions."

NameLength1

[Input]

Length of **CatalogName*.

SchemaName

[Input]

String search pattern for schema names. If a driver supports schemas 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 schemas.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *SchemaName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *SchemaName* is a pattern value argument; it is treated literally, and its case is significant.

NameLength2

[Input]

Length of **SchemaName*.

TableName

[Input]

String search pattern for table names.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *TableName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *TableName* is a pattern value argument; it is treated literally, and its case is significant.

NameLength3

[Input]

Length of **TableName*.

ColumnName

[Input]

String search pattern for column names.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *ColumnName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *ColumnName* is a pattern value argument; it is treated literally, and its case is significant.

NameLength4

[Input]

Length of **ColumnName*.

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 **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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	A cursor was open on the <i>StatementHandle</i> , and SQLFetch or SQLFetchScroll had been called. This error is returned by the Driver Manager if SQLFetch or SQLFetchScroll has not returned SQL_NO_DATA, and is returned by the driver if SQLFetch or SQLFetchScroll has returned SQL_NO_DATA. A cursor was open on the <i>StatementHandle</i> but SQLFetch or SQLFetchScroll had not been called.
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	Asynchronous processing was enabled for the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> . Then the function was called again on the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.
HY009	Invalid use of null pointer	(DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE, the <i>CatalogName</i> argument was a null pointer, and the SQL_CATALOG_NAME <i>InfoType</i> returns that catalog names are supported. (DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE, and the <i>SchemaName</i> , <i>TableName</i> , or <i>ColumnName</i> argument was a null pointer.
HY010	Function sequence error	(DM) An asynchronously executing function (not this

		one) was called for the <i>StatementHandle</i> and was still executing when this function was called. (DM) <i>SQLExecute</i> , <i>SQLExecDirect</i> , <i>SQLBulkOperations</i> , or <i>SQLSetPos</i> was called for the <i>StatementHandle</i> and returned <i>SQL_NEED_DATA</i> . This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	(DM) The value of one of the name length arguments was less than 0 but not equal to <i>SQL_NTS</i> .
		The value of one of the name length arguments exceeded the maximum length value for the corresponding catalog or name. The maximum length of each catalog or name may be obtained by calling <i>SQLGetInfo</i> with the <i>InfoType</i> values. (See "Comments.")
HYC00	Optional feature not implemented	A catalog name was specified, and the driver or data source does not support catalogs. A schema name was specified, and the driver or data source does not support schemas. A string search pattern was specified for the schema name, 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 <i>SQL_ATTR_CONCURRENCY</i> and <i>SQL_ATTR_CURSOR_TYPE</i> statement attributes was not supported by the driver or data source. The <i>SQL_ATTR_USE_BOOKMARKS</i> statement attribute was set to <i>SQL_UB_VARIABLE</i> , and the <i>SQL_ATTR_CURSOR_TYPE</i> statement attribute was set to a cursor type for which the driver does not support bookmarks.
HYT00	Timeout expired	The query timeout period expired before the data source returned the result set. The timeout period is set through <i>SQLSetStmtAttr</i> , <i>SQL_ATTR_QUERY_TIMEOUT</i> .
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through <i>SQLSetConnectAttr</i> , <i>SQL_ATTR_CONNECTION_TIMEOUT</i> .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

This function typically is used before statement execution to retrieve information about columns for a table or tables from the data source's catalog. **SQLColumns** can be used to retrieve data for all types of items returned by **SQLTables**. In addition to base tables, this may include (but is not limited to) views, synonyms, system tables, and so on. By contrast, the functions **SQLColAttribute** and **SQLDescribeCol** describe the columns in a result set and the function **SQLNumResultCols** returns the number of columns in a result set. For more information, see the Part I PDF file, "Uses of Catalog Data" in Chapter 7, "Catalog Functions."

Note For more information about the general use, arguments, and returned data of ODBC catalog functions, see the Part I PDF file, Chapter 7, "Catalog Functions."

SQLColumns returns the results as a standard result set, ordered by TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and ORDINAL_POSITION.

Note When an application works with an ODBC 2.x driver, no ORDINAL_POSITION column is returned in the result set. As a result, when working with ODBC 2.x drivers, the order of the columns in the column list returned by **SQLColumns** is not necessarily the same as the order of the columns returned when the application performs a SELECT statement on all columns in that table.

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, whether or not it is returned by **SQLColumns**.

Some columns that can be returned by **SQLStatistics** are not returned by **SQLColumns**. For example, **SQLColumns** does not return the columns in an index created over an expression or filter, such as SALARY + BENEFITS or DEPT = 0012.

The lengths of VARCHAR columns are not shown in the table; the actual lengths depend on the data source. To determine the actual lengths of the TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and COLUMN_NAME columns, an application can call **SQLGetInfo** with the SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN options.

The following columns have been renamed for ODBC 3.x. The column name changes do not affect backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM
PRECISION	COLUMN_SIZE
LENGTH	BUFFER_LENGTH
SCALE	DECIMAL_DIGITS
RADIX	NUM_PREC_RADIX

The following columns have been added to the result set returned by **SQLColumns** for ODBC 3.x:

CHAR_OCTET_LENGTH	ORDINAL_POSITION
COLUMN_DEF	SQL_DATA_TYPE
IS_NULLABLE	SQL_DATETIME_SUB

The following table lists the columns in the result set. Additional columns beyond column 18 (IS_NULLABLE) can be defined by the driver. An application should gain access to driver-specific columns by counting down from the end of the result set rather than specifying an explicit ordinal position. For more information, see the Part I PDF file, “Data Returned by Catalog Functions” in Chapter 7, “Catalog Functions.”

Column name	Column number	Data type	Comments
TABLE_CAT (ODBC 1.0)	1	Varchar	Catalog name; NULL if not applicable to the data source. If a driver supports catalogs 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 catalogs.
TABLE_SCHEM (ODBC 1.0)	2	Varchar	Schema name; NULL if not applicable to the data source. If a driver supports schemas 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 schemas.
TABLE_NAME (ODBC 1.0)	3	Varchar not NULL	Table name.
COLUMN_NAME (ODBC 1.0)	4	Varchar not NULL	Column name. The driver returns an empty string for a column that does not have a name.
DATA_TYPE (ODBC 1.0)	5	Smallint not NULL	SQL data type. This can be an ODBC SQL data type or a driver-specific SQL data type. For datetime and interval data types, this column returns the concise data type (such as SQL_TYPE_DATE or SQL_INTERVAL_YEAR_TO_MONTH, rather than the nonconcise data type such as SQL_DATETIME or SQL_INTERVAL). For a list of valid ODBC SQL data types, see “SQL Data Types” in Appendix D, “Data Types” of the SOLID Programmer Guide . For information about driver-specific SQL data types, see the driver's documentation. The data types returned for ODBC 3.x and ODBC 2.x applications may be different. For more information, see the Part I PDF file, “Backward Compatibility and Standards and Compliance” in Chapter 17, “Programming Considerations.”
TYPE_NAME (ODBC 1.0)	6	Varchar not NULL	Data source–dependent data type name; for example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINAR", or "CHAR () FOR BIT DATA".
COLUMN_SIZE (ODBC 1.0)	7	Integer	If DATA_TYPE is SQL_CHAR or SQL_VARCHAR, this column contains the

			maximum length in characters of the column. For datetime data types, this is the total number of characters required to display the value when converted to characters. For numeric data types, this is either the total number of digits or the total number of bits allowed in the column, according to the NUM_PREC_RADIX column. For interval data types, this is the number of characters in the character representation of the interval.
BUFFER_LENGTH (ODBC 1.0)	8	Integer	The length in bytes of data transferred on an SQLGetData, SQLFetch, or SQLFetchScroll 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 might be different than COLUMN_SIZE column for character data. For more information about length, see "Column Size, Decimal Digits, Transfer Octet Length, and Display Size" contained on the Microsoft Web site (ODBC Programming Reference).
DECIMAL_DIGITS (ODBC 1.0)	9	Smallint	The total number of significant digits to the right of the decimal point. For SQL_TYPE_TIME and SQL_TYPE_TIMESTAMP, this column contains the number of digits in the fractional seconds component. For the other data types, this is the decimal digits of the column on the data source. For interval data types that contain a time component, this column contains the number of digits to the right of the decimal point (fractional seconds). For interval data types that do not contain a time component, this column is 0. For more information about decimal digits, see Appendix D, "Data Types" of the SOLID Programmer Guide . NULL is returned for data types where DECIMAL_DIGITS is not applicable.
NUM_PREC_RADIX (ODBC 1.0)	10	Smallint	For numeric data types, either 10 or 2. If it is 10, the values in COLUMN_SIZE and DECIMAL_DIGITS give the number of decimal digits allowed for the column. For example, a DECIMAL(12,5) column would return a NUM_PREC_RADIX of 10, a COLUMN_SIZE of 12, and a DECIMAL_DIGITS of 5; a FLOAT column could return a NUM_PREC_RADIX of 10, a COLUMN_SIZE of 15, and a DECIMAL_DIGITS of NULL.

			<p>If it is 2, the values in COLUMN_SIZE and DECIMAL_DIGITS give the number of bits allowed in the column. For example, a FLOAT column could return a RADIX of 2, a COLUMN_SIZE of 53, and a DECIMAL_DIGITS of NULL.</p> <p>NULL is returned for data types where NUM_PREC_RADIX is not applicable.</p>
<p>NULLABLE (ODBC 1.0)</p>	11	Smallint not NULL	<p>SQL_NO_NULLS if the column could not include NULL values.</p> <p>SQL_NULLABLE if the column accepts NULL values.</p> <p>SQL_NULLABLE_UNKNOWN if it is not known whether the column accepts NULL values.</p> <p>The value returned for this column is different from the value returned for the IS_NULLABLE column. The NULLABLE column indicates with certainty that a column can accept NULLs, but cannot indicate with certainty that a column does not accept NULLs. The IS_NULLABLE column indicates with certainty that a column cannot accept NULLs, but cannot indicate with certainty that a column accepts NULLs.</p>
<p>REMARKS (ODBC 1.0)</p>	12	Varchar	A description of the column.
<p>COLUMN_DEF (ODBC 3.0)</p>	13	Varchar	<p>The default value of the column. The value in this column should be interpreted as a string if it is enclosed in quotation marks. If NULL was specified as the default value, then this column is the word NULL, not enclosed in quotation marks. If the default value cannot be represented without truncation, then this column contains TRUNCATED, with no enclosing single quotation marks. If no default value was specified, then this column is NULL.</p> <p>The value of COLUMN_DEF can be used in generating a new column definition, except when it contains the value TRUNCATED.</p>
<p>SQL_DATA_TYPE (ODBC 3.0)</p>	14	Smallint not NULL	<p>SQL data type, as it appears in the SQL_DESC_TYPE record field in the IRD. This can be an ODBC SQL data type or a driver-specific SQL data type. This column is the same as the DATA_TYPE column, with the exception of datetime and interval data types. This column returns the nonconcise data type (such as SQL_DATETIME or SQL_INTERVAL),</p>

			<p>rather than the concise data type (such as <code>SQL_TYPE_DATE</code> or <code>SQL_INTERVAL_YEAR_TO_MONTH</code>) for datetime and interval data types. If this column returns <code>SQL_DATETIME</code> or <code>SQL_INTERVAL</code>, the specific data type can be determined from the <code>SQL_DATETIME_SUB</code> column. For a list of valid ODBC SQL data types, see “SQL Data Types” in Appendix D, “Data Types” of the SOLID Programmer Guide. For information about driver-specific SQL data types, see the driver's documentation.</p> <p>The data types returned for ODBC 3.x and ODBC 2.x applications may be different. For more information, see the Part I PDF file, “Backward Compatibility and Standards Compliance” in Chapter 17, “Programming Considerations.”</p>
<code>SQL_DATETIME_SUB</code> (ODBC 3.0)	15	Smallint	<p>The subtype code for datetime and interval data types. For other data types, this column returns a NULL. For more information about datetime and interval subcodes, see “<code>SQL_DESC_DATETIME_INTERVAL_CODE</code>” in SQLSetDescField.</p>
<code>CHAR_OCTET_LENGTH</code> (ODBC 3.0)	16	Integer	<p>The maximum length in bytes of a character or binary data type column. For all other data types, this column returns a NULL.</p>
<code>ORDINAL_POSITION</code> (ODBC 3.0)	17	Integer not NULL	<p>The ordinal position of the column in the table. The first column in the table is number 1.</p>
<code>IS_NULLABLE</code> (ODBC 3.0)	18	Varchar	<p>"NO" if the column does not include NULLs. "YES" if the column could include NULLs. This column returns a zero-length string if nullability is unknown.</p> <p>ISO rules are followed to determine nullability. An ISO SQL-compliant DBMS cannot return an empty string.</p> <p>The value returned for this column is different from the value returned for the <code>NULLABLE</code> column. (See the description of the <code>NULLABLE</code> column.)</p>

Code Example

In the following example, an application declares buffers 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 buffers. Finally, the application fetches each row of data with **SQLFetch** and processes it.

```
#define STR_LEN 128+1
```



```

#define REM_LEN 254+1

/* Declare buffers for result set data */

SQLCHAR  szCatalog[STR_LEN], szSchema[STR_LEN];
SQLCHAR  szTableName[STR_LEN], szColumnName[STR_LEN];
SQLCHAR  szTypeName[STR_LEN], szRemarks[REM_LEN];
SQLCHAR  szColumnDefault[STR_LEN], szIsNullable[STR_LEN];
SQLINTEGER  ColumnSize, BufferLength, CharOctetLength, OrdinalPosition;
SQLSMALLINT  DataType, DecimalDigits, NumPrecRadix, Nullable;
SQLSMALLINT  SQLDataType, DatetimeSubtypeCode;
SQLRETURN retcode;
SQLHSTMT hstmt;

/* Declare buffers for bytes available to return */

SQLINTEGER cbCatalog, cbSchema, cbTableName, cbColumnName;
SQLINTEGER cbDataType, cbColumnNameSize, cbBufferLength;
SQLINTEGER cbDecimalDigits, cbNumPrecRadix, cbNullable, cbRemarks;
SQLINTEGER cbColumnDefault, cbSQLDataType, cbDatetimeSubtypeCode,
cbCharOctetLength;
SQLINTEGER cbOrdinalPosition, cbIsNullable;

retcode = SQLColumns(hstmt,
    NULL, 0, /* All catalogs */
    NULL, 0, /* All schemas */
    "CUSTOMERS", SQL_NTS, /* CUSTOMERS table */
    NULL, 0); /* All columns */

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

    /* Bind columns in result set to buffers */

    SQLBindCol(hstmt, 1, SQL_C_CHAR, szCatalog, STR_LEN, &cbCatalog);
    SQLBindCol(hstmt, 2, SQL_C_CHAR, szSchema, STR_LEN, &cbSchema);
    SQLBindCol(hstmt, 3, SQL_C_CHAR, szTableName, STR_LEN, &cbTableName);
    SQLBindCol(hstmt, 4, SQL_C_CHAR, szColumnName, STR_LEN, &cbColumnName);
    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, &ColumnSize, 0, &cbColumnSize);
    SQLBindCol(hstmt, 8, SQL_C_SLONG, &BufferLength, 0, &cbBufferLength);
    SQLBindCol(hstmt, 9, SQL_C_SSHORT, &DecimalDigits, 0, &cbDecimalDigits);
    SQLBindCol(hstmt, 10, SQL_C_SSHORT, &NumPrecRadix, 0, &cbNumPrecRadix);
    SQLBindCol(hstmt, 11, SQL_C_SSHORT, &Nullable, 0, &cbNullable);
    SQLBindCol(hstmt, 12, SQL_C_CHAR, szRemarks, REM_LEN, &cbRemarks);
    SQLBindCol(hstmt, 13, SQL_C_CHAR, szColumnDefault, STR_LEN,
    &cbColumnDefault);
    SQLBindCol(hstmt, 14, SQL_C_SSHORT, &SQLDataType, 0, &cbSQLDataType);
    SQLBindCol(hstmt, 15, SQL_C_SSHORT, &DatetimeSubtypeCode, 0,
    &cbDatetimeSubtypeCode);
    SQLBindCol(hstmt, 16, SQL_C_SLONG, &CharOctetLength, 0, &cbCharOctetLength);
    SQLBindCol(hstmt, 17, SQL_C_SLONG, &OrdinalPosition, 0, &cbOrdinalPosition);
    SQLBindCol(hstmt, 18, SQL_C_CHAR, szIsNullable, STR_LEN, &cbIsNullable);
    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

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Returning privileges for a column or columns	SQLColumnPrivileges
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Fetching multiple rows of data	SQLFetch
Returning columns that uniquely identify a row, or columns automatically updated by a transaction	SQLSpecialColumns
Returning table statistics and indexes	SQLStatistics
Returning a list of tables in a data source	SQLTables
Returning privileges for a table or tables	SQLTablePrivileges

SQLConnect

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

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

Syntax

SQLRETURN SQLConnect(
SQLHDBC	<i>ConnectionHandle,</i>
SQLCHAR *	<i>ServerName,</i>
SQLSMALLINT	<i>NameLength1,</i>
SQLCHAR *	<i>UserName,</i>
SQLSMALLINT	<i>NameLength2,</i>
SQLCHAR *	<i>Authentication,</i>
SQLSMALLINT	<i>NameLength3</i>);

Arguments

ConnectionHandle

[Input]

Connection handle.

ServerName

[Input]

Data source name. For information about how an application chooses a data source, see the Part I PDF file, "Choosing a Data Source or Driver" in Chapter 6, "Connecting to a Data Source or Driver."

NameLength1

[Input]

Length of **ServerName*.

UserName

[Input]

User identifier.

NameLength2

[Input]

Length of **UserName*.

Authentication

[Input]

Authentication string (typically the password).

NameLength3

[Input]

Length of **Authentication*.

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_DBC and a *Handle* of *ConnectionHandle*. 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.)

01S02	Option value changed	The driver did not support the specified value of the <i>ValuePtr</i> argument in SQLSetConnectAttr and substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
08001	Client unable to establish connection	The driver was unable to establish a connection with the data source.
08002	Connection name in use	(DM) The specified <i>ConnectionHandle</i> had already been used to establish a connection with a data source, and the connection was still open or the user was browsing for a connection.
08004	Server rejected the 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 <i>UserName</i> or the value specified for the argument <i>Authentication</i> violated restrictions defined by the data source.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	(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.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	(DM) The value specified for argument <i>NameLength1</i> , <i>NameLength2</i> , or <i>NameLength3</i> was less than 0 but not equal to SQL_NTS. (DM) The value specified for argument <i>NameLength1</i> exceeded the maximum length for a data source name.
HYT00	Timeout expired	The query timeout period expired before the connection to the data source completed. The timeout period is set through SQLSetConnectAttr , SQL_ATTR_LOGIN_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
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 found and no default driver specified	(DM) The data source name specified in the argument <i>ServerName</i> was not found in the system information, nor was there a default driver specification.
IM003	Specified driver could not be connected to	(DM) The driver listed in the data source specification in system information was not found or could not be connected to for some other reason.
IM004	Driver's SQLAllocHandle on SQL_HANDLE_ENV failed	(DM) During SQLConnect , the Driver Manager called the driver's SQLAllocHandle function with a <i>HandleType</i> of SQL_HANDLE_ENV and the driver returned an error.
IM005	Driver's SQLAllocHandle on SQL_HANDLE_DBC failed	(DM) During SQLConnect , the Driver Manager called the driver's SQLAllocHandle function with a <i>HandleType</i> of SQL_HANDLE_DBC and the driver returned an error.
IM006	Driver's SQLSetConnectAttr failed	During SQLConnect , the Driver Manager called the driver's SQLSetConnectAttr function and the driver returned an error. (Function returns SQL_SUCCESS_WITH_INFO.)
IM009	Unable to connect to translation DLL	The driver was unable to connect to the translation DLL that was specified for the data source.
IM010	Data source name too long	(DM) * <i>ServerName</i> was longer than SQL_MAX_DSN_LENGTH characters.

Comments

For information about why an application uses **SQLConnect**, see the Part I PDF file, "Connecting with SQLConnect" in Chapter 6, "Connecting to a Data Source or Driver."

The Driver Manager does not connect to 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 whether a driver is currently connected to for the specified *ConnectionHandle*:

- If a driver is not connected to, the Driver Manager connects to the driver and calls **SQLAllocHandle** with a *HandleType* of SQL_HANDLE_ENV, **SQLAllocHandle** with a *HandleType* of SQL_HANDLE_DBC, **SQLSetConnectAttr** (if the application specified any connection attributes), 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 **SQLSetConnectAttr**. For more information, the Part I PDF file, Chapter 6, "Overview of Connecting to a Data Source or Driver."
- If the specified driver is already connected to on the *ConnectionHandle*, the Driver Manager calls only the connection function in the driver. In this case, the driver must make sure that all connection attributes for the *ConnectionHandle* maintain their current settings.
- If a different driver is connected to, the Driver Manager calls **SQLFreeHandle** with a *HandleType* of SQL_HANDLE_DBC, and then, if no other driver is connected to in that environment, it calls

SQLFreeHandle with a *HandleType* of SQL_HANDLE_ENV in the connected driver and then disconnects that driver. It then performs the same operations as when a driver is not connected to.

The driver then allocates handles and initializes itself.

When the application calls **SQLDisconnect**, the Driver Manager calls **SQLDisconnect** in the driver. However, it does not disconnect the driver. This keeps the driver in memory for applications that repeatedly connect to and disconnect from a data source. When the application calls **SQLFreeHandle** with a *HandleType* of SQL_HANDLE_DBC, the Driver Manager calls **SQLFreeHandle** with a *HandleType* of SQL_HANDLE_DBC and then **SQLFreeHandle** with a *HandleType* of SQL_HANDLE_ENV in the driver, and then disconnects the driver.

An ODBC application can establish more than one connection.

Driver Manager Guidelines

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

- If **ServerName* contains a valid data source name, the Driver Manager locates the corresponding data source specification in the system information and connects to the associated driver. The Driver Manager passes each **SQLConnect** argument to the driver.
- If the data source name cannot be found or *ServerName* is a null pointer, the Driver Manager locates the default data source specification and connects to the associated driver. The Driver Manager passes to the driver the *UserName* and *Authentication* arguments unmodified, and "DEFAULT" for the *ServerName* argument.
- If the *ServerName* argument is "DEFAULT", the Driver Manager locates the default data source specification and connects to the associated driver. The Driver Manager passes each **SQLConnect** argument to the driver.
- If the data source name cannot be found or *ServerName* 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 connected to by the Driver Manager, a driver can locate its corresponding data source specification in the system information and use driver-specific information from the specification to complete its set of required connection information.

If a default translation library is specified in the system information for the data source, the driver connects to it. A different translation library can be connected to by calling **SQLSetConnectAttr** with the SQL_ATTR_TRANSLATE_LIB attribute. A translation option can be specified by calling **SQLSetConnectAttr** with the SQL_ATTR_TRANSLATE_OPTION attribute.

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

Connection Pooling

Connection pooling allows an application to reuse a connection that has already been created. When connection pooling is enabled and **SQLConnect** is called, the Driver Manager attempts to make the

connection using a connection that is part of a pool of connections in an environment that has been designated for connection pooling. This environment is a shared environment that is used by all applications that use the connections in the pool.

Connection pooling is enabled before the environment is allocated by calling **SQLSetEnvAttr** to set **SQL_ATTR_CONNECTION_POOLING** to **SQL_CP_ONE_PER_DRIVER** (which specifies a maximum of one pool per driver) or **SQL_CP_ONE_PER_HENV** (which specifies a maximum of one pool per environment). **SQLSetEnvAttr** in this case is called with *EnvironmentHandle* set to null, which makes the attribute a process-level attribute. If **SQL_ATTR_CONNECTION_POOLING** is set to **SQL_CP_OFF**, connection pooling is disabled.

Once connection pooling has been enabled, **SQLAllocHandle** with a *HandleType* of **SQL_HANDLE_ENV** is called to allocate an environment. The environment allocated by this call is a shared environment because connection pooling has been enabled. The environment to be used is not determined, however, until **SQLAllocHandle** with a *HandleType* of **SQL_HANDLE_DBC** is called.

SQLAllocHandle with a *HandleType* of **SQL_HANDLE_DBC** is called to allocate a connection. The Driver Manager attempts to find an existing shared environment that matches the environment attributes set by the application. If no such environment exists, one is created as an implicit *shared environment*. If a matching shared environment is found, the environment handle is returned to the application and its reference count is incremented.

The connection to be used is not determined, however, until **SQLConnect** is called. At that point, the Driver Manager attempts to find an existing connection in the connection pool that matches the criteria requested by the application. This criteria includes the connection options requested in the call to **SQLConnect** (the values of the *ServerName*, *UserName*, and *Authentication* keywords) and any connection attributes set since **SQLAllocHandle** with a *HandleType* of **SQL_HANDLE_DBC** was called. The Driver Manager checks this criteria against the corresponding connection keywords and attributes in connections in the pool. If a match is found, the connection in the pool is used. If no match is found, a new connection is created.

If the **SQL_ATTR_CP_MATCH** environment attribute is set to **SQL_CP_STRICT_MATCH**, the match must be exact for a connection in the pool to be used. If the **SQL_ATTR_CP_MATCH** environment attribute is set to **SQL_CP_RELAXED_MATCH**, the connection options in the call to **SQLConnect** must match but not all of the connection attributes must match.

The following rules are applied when a connection attribute, as set by the application before **SQLConnect** is called, does not match the connection attribute of the connection in the pool:

If the connection attribute must be set before the connection is made:

- If **SQL_ATTR_CP_MATCH** is **SQL_CP_STRICT_MATCH**, **SQL_ATTR_PACKET_SIZE** in the pooled connection must be identical to the attribute set by the application. If **SQL_CP_RELAXED_MATCH**, the values of **SQL_ATTR_PACKET_SIZE** can be different.
- The value of **SQL_ATTR_LOGIN_VALUE** does not affect the match.

If the connection attribute can be set either before or after the connection is made:

- If the connection attribute has not been set by the application but has been set on the connection in the pool, and there is a default, the connection attribute in the pooled connection is set back to the default and a match is declared. If there is no default, the pooled connection is not considered a match.

- If the connection attribute has been set by the application but has not been set on the connection in the pool, the connection attribute on the pool is changed to that set by the application and a match is declared.
- If the connection attribute has been set by the application, and has also been set on the connection in the pool but the values are different, the value of the application's connection attribute is used and a match is declared.

If the values of driver-specific connection attributes are not identical and `SQL_ATTR_CP_MATCH` is set to `SQL_CP_STRICT_MATCH`, the connection in the pool is not used.

When the application calls **SQLDisconnect** to disconnect, the connection is returned to the connection pool and is available for reuse.

When Connection Pooling is enabled and the database server has become unavailable, the Driver Manager attempts to reestablish a connection to the server repeatedly. **ODBCSetTryWaitValue** and **ODBCGetTryWaitValue** prevent this and, consequently, the large number of generated requests. **ODBCSetTryWaitValue** saves the information in the registry at the following location:

```
HKEY_LOCAL_MACHINE
SOFTWARE
  Odbc
    Odbcinst.ini
      ODBC Connection Pooling
        Retry Wait
```

If there is a bad connection in the pool, the ODBC Driver Manager will attempt to connect before the connection can be reused for the first time. If the connection still fails, the ODBC Driver Manager returns the error and marks the connection with the time. From that point until the **RetryWait** value expires, the ODBC Driver Manager returns a failure without testing the connection.

```
BOOL ODBCSetTryWaitValue ( DWORD dwValue );
```

Arguments

dwValue [Input]

The number of seconds to wait until the Driver Manager attempts to connect again.

Returns

The function returns TRUE if it is successful, FALSE if it fails.

```
DWORD ODBCGetTryWaitValue ( );
```

Arguments

None.

Returns

The function returns a **DWORD** value containing the number of seconds the Driver Manager will wait before attempting to connect again.

Optimizing Connection Pooling Performance

When distributed transactions are involved, it is possible to optimize connection pooling performance by using **SQL_DTC_TRANSITION_COST**, which is a **SQLINTEGER** bitmask. The transitions referred to are the transitions of the connection attribute **SQL_ATTR_ENLIST_IN_DTC** going from value 0 to nonzero, and vice versa. This is a connection going from not enlisted in a distributed transaction to enlisted in a distributed transaction, and vice versa. Depending on how the driver has implemented enlistment (setting connection attribute **SQL_ATTR_ENLIST_IN_DTC**), these transitions may be expensive and should therefore be avoided for best performance.

The value returned by the driver contains any combination of the following bits:

- **SQL_DTC_ENLIST_EXPENSIVE**, when set, implies the zero to nonzero transition is significantly more expensive than a transition from nonzero to another nonzero value (enlisting a previously enlisted connection in its next transaction).
- **SQL_DTC_UNENLIST_EXPENSIVE**, when set, implies the nonzero to zero transition is significantly more expensive than using a connection whose **SQL_ATTR_ENLIST_IN_DTC** attribute is already set to zero.

There is a performance vs. connection usage tradeoff. If a driver indicates that one or more of these transitions are expensive, the driver manager's connection pooler responds to this by keeping more connections in the pool. Some of the connections in the pool are preferred for nontransactional use, and some are preferred for transactional use. However, if the driver indicates that these transitions are not expensive, fewer connections can be used, perhaps alternating between nontransactional and transactional use.

Drivers that do not support **SQL_ATTR_ENLIST_IN_DTC** do not need to support **SQL_DTC_TRANSITION_COST**. For drivers that support **SQL_ATTR_ENLIST_IN_DTC** but not **SQL_DTC_TRANSITION_COST**, it is assumed that the transitions are not expensive, as though the driver returned 0 (no bits set) for this value.

Although **SQL_DTC_TRANSITION_COST** was introduced in ODBC 3.5, an ODBC 2.x driver can also support it because the driver manager will query this information regardless of the driver version.

Code Example

In the following example, an application allocates environment and connection handles. It then connects to the SalesOrders 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.

```
SQLHENV      henv;
SQLHDBC      hdbc;
SQLHSTMT     hstmt;
SQLRETURN    retcode;

    /*Allocate environment handle */
retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);

if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
    /* Set the ODBC version environment attribute */
    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3,
0);

    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
        /* Allocate connection handle */
```

```

retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
    /* Set login timeout to 5 seconds. */
    SQLSetConnectAttr(hdbc, (void*)SQL_LOGIN_TIMEOUT, 5, 0);

    /* Connect to data source */
    retcode = SQLConnect(hdbc, (SQLCHAR*) "Sales", SQL_NTS,
        (SQLCHAR*) "JohnS", SQL_NTS,
        (SQLCHAR*) "Sesame", SQL_NTS);

    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){
        /* Allocate statement handle */
        retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

        if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
            /* Process data */
            ;
            ;
            ;

            SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
        }
        SQLDisconnect(hdbc);
    }
    SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
}
SQLFreeHandle(SQL_HANDLE_ENV, henv);

```

Related Functions

For information about	See
Allocating a handle	SQLAllocHandle
Discovering and enumerating values required to connect to a data source	SQLBrowseConnect
Disconnecting from a data source	SQLDisconnect
Connecting to a data source using a connection string or dialog box	SQLDriverConnect
Returning the setting of a connection attribute	SQLSetConnectAttr
Setting a connection attribute	SQLGetConnectAttr

SQLCopyDesc

Conformance

Version Introduced: ODBC 3.0

Standards Compliance: ISO 92

Summary

SQLCopyDesc copies descriptor information from one descriptor handle to another.

Syntax

SQLRETURN SQLCopyDesc (
SQLHDESC	SourceDescHandle,
SQLHDESC	TargetDescHandle);

Arguments

SourceDescHandle

[Input]

Source descriptor handle.

TargetDescHandle

[Input]

Target descriptor handle. The *TargetDescHandle* argument can be a handle to an application descriptor or an IPD. *TargetDescHandle* cannot be set to a handle to an IRD, or **SQLCopyDesc** will return SQLSTATE HY016 (Cannot modify an implementation row descriptor).

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLCopyDesc** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_DESC and a *Handle* of *TargetDescHandle*. If an invalid *SourceDescHandle* was passed in the call, SQL_INVALID_HANDLE will be returned but no SQLSTATE will be returned. The following table lists the SQLSTATE values commonly returned by **SQLCopyDesc** 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.

When an error is returned, the call to **SQLCopyDesc** is immediately aborted, and the contents of the fields in the *TargetDescHandle* descriptor are undefined.

Because **SQLCopyDesc** may be implemented by calling **SQLGetDescField** and **SQLSetDescField**, **SQLCopyDesc** may return SQLSTATEs returned by **SQLGetDescField** or **SQLSetDescField**.

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.
HY000	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 SQLGetDiagRec in the *MessageText buffer describes the error and its cause.

HY001	Memory allocation error	The driver was unable to allocate the memory required to support execution or completion of the function.
HY007	Associated statement is not prepared	<i>SourceDescHandle</i> was associated with an IRD, and the associated statement handle was not in the prepared or executed state.
HY010	Function sequence error	(DM) The descriptor handle in <i>SourceDescHandle</i> or <i>TargetDescHandle</i> was associated with a <i>StatementHandle</i> for which an asynchronously executing function (not this one) was called and was still executing when this function was called. (DM) The descriptor handle in <i>SourceDescHandle</i> or <i>TargetDescHandle</i> was associated with a <i>StatementHandle</i> for which <code>SQLExecute</code> , <code>SQLExecDirect</code> , <code>SQLBulkOperations</code> , or <code>SQLSetPos</code> was called and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY016	Cannot modify an implementation row descriptor	<i>TargetDescHandle</i> was associated with an IRD.
HY021	Inconsistent descriptor information	The descriptor information checked during a consistency check was not consistent. For more information, see "Consistency Checks" in SQLSetDescField .
HY092	Invalid attribute/option identifier	The call to <code>SQLCopyDesc</code> prompted a call to <code>SQLSetDescField</code> , but <i>*ValuePtr</i> was not valid for the <i>FieldIdentifier</i> argument on <i>TargetDescHandle</i> .
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through <code>SQLSetConnectAttr</code> , <code>SQL_ATTR_CONNECTION_TIMEOUT</code> .
IM001	Driver does not support this function	(DM) The driver associated with the <i>SourceDescHandle</i> or <i>TargetDescHandle</i> does not support the function.

Comments

A call to **SQLCopyDesc** copies the fields of the source descriptor handle to the target descriptor handle. Fields can be copied only to an application descriptor or an IPD, but not to an IRD. Fields can be copied from either an application or an implementation descriptor.

Fields can be copied from an IRD only if the statement handle is in the prepared or executed state; otherwise, the function returns `SQLSTATE HY007` (Associated statement is not prepared).

Fields can be copied from an IPD whether or not a statement has been prepared. If an SQL statement with dynamic parameters has been prepared and automatic population of the IPD is supported and enabled, then

the IPD is populated by the driver. When **SQLCopyDesc** is called with the IPD as the *SourceDescHandle*, the populated fields are copied. If the IPD is not populated by the driver, the contents of the fields originally in the IPD are copied.

All fields of the descriptor, except `SQL_DESC_ALLOC_TYPE` (which specifies whether the descriptor handle was automatically or explicitly allocated), are copied, whether or not the field is defined for the destination descriptor. Copied fields overwrite the existing fields.

The driver copies all descriptor fields if the *SourceDescHandle* and *TargetDescHandle* arguments are associated with the same driver, even if the drivers are on two different connections or environments. If the *SourceDescHandle* and *TargetDescHandle* arguments are associated with different drivers, the Driver Manager copies ODBC-defined fields, but does not copy driver-defined fields or fields that are not defined by ODBC for the type of descriptor.

The call to **SQLCopyDesc** is aborted immediately if an error occurs.

When the `SQL_DESC_DATA_PTR` field is copied, a consistency check is performed on the target descriptor. If the consistency check fails, `SQLSTATE HY021` (Inconsistent descriptor information) is returned and the call to **SQLCopyDesc** is immediately aborted. For more information on consistency checks, see "Consistency Checks" in [SQLSetDescRec](#).

Descriptor handles can be copied across connections even if the connections are under different environments. If the Driver Manager detects that the source and the destination descriptor handles do not belong to the same connection and the two connections belong to separate drivers, it implements **SQLCopyDesc** by performing a field-by-field copy using **SQLGetDescField** and **SQLSetDescField**.

When **SQLCopyDesc** is called with a *SourceDescHandle* on one driver and a *TargetDescHandle* on another driver, the error queue of the *SourceDescHandle* is cleared. This occurs because **SQLCopyDesc** in this case is implemented by calls to **SQLGetDescField** and **SQLSetDescField**.

Note An application might be able to associate an explicitly allocated descriptor handle with a *StatementHandle*, rather than calling **SQLCopyDesc** to copy fields from one descriptor to another. An explicitly allocated descriptor can be associated with another *StatementHandle* on the same *ConnectionHandle* by setting the `SQL_ATTR_APP_ROW_DESC` or `SQL_ATTR_APP_PARAM_DESC` statement attribute to the handle of the explicitly allocated descriptor. When this is done, **SQLCopyDesc** does not have to be called to copy descriptor field values from one descriptor to another. A descriptor handle cannot be associated with a *StatementHandle* on another *ConnectionHandle*, however; to use the same descriptor field values on *StatementHandles* on different *ConnectionHandles*, **SQLCopyDesc** has to be called.

For a description of the fields in a descriptor header or record, see [SQLSetDescField](#). For more information on descriptors, see the Part I PDF file, Chapter 13, "Overview of Descriptors."

Copying Rows Between Tables

An application may copy data from one table to another without copying the data at the application level. To do this, the application binds the same data buffers and descriptor information to a statement that fetches the data and the statement that inserts the data into a copy. This can be accomplished either by sharing an application descriptor (binding an explicitly allocated descriptor as both the ARD to one statement and the APD in another) or by using **SQLCopyDesc** to copy the bindings between the ARD and the APD of the two statements. If the statements are on different connections, **SQLCopyDesc** must be used. In addition, **SQLCopyDesc** has to be called to copy the bindings between the IRD and the IPD of the two statements. When copying across statements on the same connection, the

SQL_ACTIVE_STATEMENTS information type returned by the driver for a call to **SQLGetInfo** must be greater than 1 for this operation to succeed. (This is not the case when copying across connections.)

Code Example

In the following example, descriptor operations are used to copy the fields of the PartsSource table into the PartsCopy table. The contents of the PartsSource table are fetched into rowset buffers in *hstmt0*. These values are used as parameters of an INSERT statement on *hstmt1* to populate the columns of the PartsCopy table. To do so, the fields of the IRD of *hstmt0* are copied to the fields of the IPD of *hstmt1*, and the fields of the ARD of *hstmt0* are copied to the fields of the APD of *hstmt1*.

```
#define ROWS 100
#define DESC_LEN 50
#define SQL_SUCCEEDED(rc) (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO)

// Template for a row
typedef struct {
    SQLINTEGER    sPartID;
    SQLINTEGER    cbPartID;
    SQLCHARszDescription[DESC_LENGTH];
    SQLINTEGER    cbDescription;
    REAL          sPrice;
    SQLINTEGER    cbPrice;
} PartsSource;

PartsSource      rget[ROWS]; // rowset buffer
SQLUSMALLINT     sts_ptr[ROWS]; // status pointer
SQLHSTMT         hstmt0, hstmt1;
SQLHDESC         hArd0, hIrd0, hApd1, hIpd1;

// ARD and IRD of hstmt0
SQLGetStmtAttr(hstmt0, SQL_ATTR_APP_ROW_DESC, &hArd0, 0, NULL);
SQLGetStmtAttr(hstmt0, SQL_ATTR_IMP_ROW_DESC, &hIrd0, 0, NULL);

// APD and IPD of hstmt1
SQLGetStmtAttr(hstmt1, SQL_ATTR_APP_PARAM_DESC, &hApd1, 0, NULL);
SQLGetStmtAttr(hstmt1, SQL_ATTR_IMP_PARAM_DESC, &hIpd1, 0, NULL);

// Use row-wise binding on hstmt0 to fetch rows
SQLSetStmtAttr(hstmt0, SQL_ATTR_ROW_BIND_TYPE, (SQLPOINTER)
sizeof(PartsSource), 0);

// Set rowset size for hstmt0
SQLSetStmtAttr(hstmt0, SQL_ATTR_ROW_ARRAY_SIZE, (SQLPOINTER) ROWS, 0);

// Execute a select statement
SQLExecDirect(hstmt0, "SELECT PARTID, DESCRIPTION, PRICE FROM PARTS ORDER BY 3,
1, 2",
SQL_NTS);

// Bind
SQLBindCol(hstmt0, 1, SQL_C_SLONG, rget[0].sPartID, 0,
&rget[0].cbPartID);
SQLBindCol(hstmt0, 2, SQL_C_CHAR, &rget[0].szDescription, DESC_LEN,
&rget[0].cbDescription);
SQLBindCol(hstmt0, 3, SQL_C_FLOAT, rget[0].sPrice,
0, &rget[0].cbPrice);
```

```

// Perform parameter bindings on hstmt1.
SQLCopyDesc(hArd0, hApd1);
SQLCopyDesc(hIrd0, hIpd1);

// Set the array status pointer of IRD
SQLSetStmtAttr(hstmt0, SQL_ATTR_ROW_STATUS_PTR, sts_ptr, SQL_IS_POINTER);

// Set the ARRAY_STATUS_PTR field of APD to be the same
// as that in IRD.
SQLSetStmtAttr(hstmt1, SQL_ATTR_PARAM_OPERATION_PTR, sts_ptr, SQL_IS_POINTER);

// Prepare an insert statement on hstmt1. PartsCopy is a copy of
// PartsSource
SQLPrepare(hstmt1, "INSERT INTO PARTS_COPY VALUES (?, ?, ?)", SQL_NTS);

// In a loop, fetch a rowset, and copy the fetched rowset to PARTS_COPY

rc = SQLFetchScroll(hstmt0, SQL_FETCH_NEXT, 0);
while (SQL_SUCCEEDED(rc)) {

    // After the call to SQLFetchScroll, the status array has row
    // statuses. This array is used as input status in the APD
    // and hence determines which elements of the rowset buffer
    // are inserted.
    SQLExecute(hstmt1);

    rc = SQLFetchScroll(hstmt0, SQL_FETCH_NEXT, 0);
} // while

```

Related Functions

For information about	See
Getting multiple descriptor fields	SQLGetDescRec
Setting a single descriptor field	SQLSetDescField
Setting multiple descriptor fields	SQLSetDescRec

SQLDataSources

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

SQLDataSources returns information about a data source. This function is implemented solely by the Driver Manager.

Syntax

SQLRETURN SQLDataSources(
SQLHENV	EnvironmentHandle,

SQLUSMALLINT	Direction,
SQLCHAR *	ServerName,
SQLSMALLINT	BufferLength1,
SQLSMALLINT *	NameLength1Ptr,
SQLCHAR *	Description,
SQLSMALLINT	BufferLength2,
SQLSMALLINT *	NameLength2Ptr);

Arguments

EnvironmentHandle

[Input]

Environment handle.

Direction

[Input]

Determines which data source the Driver Manager returns information on. Can be:

SQL_FETCH_NEXT (to fetch the next data source name in the list), SQL_FETCH_FIRST (to fetch from the beginning of the list), SQL_FETCH_FIRST_USER (to fetch the first user DSN), or SQL_FETCH_FIRST_SYSTEM (to fetch the first system DSN).

When *Direction* is set to SQL_FETCH_FIRST, subsequent calls to **SQLDataSources** with *Direction* set to SQL_FETCH_NEXT return both user and system DSNs. When *Direction* is set to SQL_FETCH_FIRST_USER, all subsequent calls to **SQLDataSources** with *Direction* set to SQL_FETCH_NEXT return only user DSNs. When *Direction* is set to SQL_FETCH_FIRST_SYSTEM, all subsequent calls to **SQLDataSources** with *Direction* set to SQL_FETCH_NEXT return only system DSNs.

ServerName

[Output]

Pointer to a buffer in which to return the data source name.

BufferLength1

[Input]

Length of the **ServerName* buffer, in bytes; this does not need to be longer than SQL_MAX_DSN_LENGTH plus the null-termination character.

NameLength1Ptr

[Output]

Pointer to a buffer in which to return the total number of bytes (excluding the null-termination byte) available to return in **ServerName*. If the number of bytes available to return is greater than or equal to *BufferLength1*, the data source name in **ServerName* is truncated to *BufferLength1* minus the length of a null-termination character.

Description

[Output]

Pointer to a buffer in which to return the description of the driver associated with the data source. For example, dBASE or SQL Server.

BufferLength2

[Input]

Length of the **Description* buffer.

NameLength2Ptr

[Output]

Pointer to a buffer in which to return the total number of bytes (excluding the null-termination byte) available to return in **Description*. If the number of bytes available to return is greater than or equal to *BufferLength2*, the driver description in **Description* is truncated to *BufferLength2* minus the length of a null-termination character.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, 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 **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_ENV and a *Handle* of *EnvironmentHandle*. 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	String data, right truncated	(DM) The buffer <i>*ServerName</i> was not large enough to return the entire data source name, so the name was truncated. The length of the entire data source name is returned in <i>*NameLength1Ptr</i> . (Function returns SQL_SUCCESS_WITH_INFO.) (DM) The buffer <i>*Description</i> was not large enough to return the entire driver description, so the description was truncated. The length of the untruncated data source description is returned in <i>*NameLength2Ptr</i> . (Function returns SQL_SUCCESS_WITH_INFO.)
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	(DM) The Driver Manager was unable to allocate memory required to support execution or completion of the function.

HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	(DM) The value specified for argument <i>BufferLength1</i> was less than 0. (DM) The value specified for argument <i>BufferLength2</i> was less than 0.
HY103	Invalid retrieval code	(DM) The value specified for the argument <i>Direction</i> was not equal to SQL_FETCH_FIRST, SQL_FETCH_FIRST_USER, SQL_FETCH_FIRST_SYSTEM, 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 standards compliance.

An application can call **SQLDataSources** multiple times to retrieve all data source names. The Driver Manager retrieves this information from the system information. When there are no more data source names, the Driver Manager returns SQL_NO_DATA. If **SQLDataSources** is called with SQL_FETCH_NEXT immediately after it returns SQL_NO_DATA, it will return the first data source name. For information about how an application uses the information returned by **SQLDataSources**, see the Part I PDF file, "Choosing a Data Source or Driver" in Chapter 6, "Connecting to a Data Source or Driver."

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

For information about	See
Discovering and listing values required to connect to a data source	SQLBrowseConnect
Connecting to a data source	SQLConnect
Connecting to a data source using a connection string or dialog box	SQLDriverConnect
Returning driver descriptions and attributes	SQLDrivers

SQLDescribeParam

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: Level 2

Summary

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

Syntax

SQLRETURN SQLDescribeParam(
SQLHSTMT	<i>StatementHandle</i> ,
SQLUSMALLINT	<i>ParameterNumber</i> ,
SQLSMALLINT *	<i>DataTypePtr</i> ,
SQLUIINTEGER *	<i>ParameterSizePtr</i> ,
SQLSMALLINT *	<i>DecimalDigitsPtr</i> ,
SQLSMALLINT *	<i>NullablePtr</i>);

Argument

StatementHandle

[Input]

Statement handle.

ParameterNumber

[Input]

Parameter marker number ordered sequentially in increasing parameter order, starting at 1.

DataTypePtr

[Output]

Pointer to a buffer in which to return the SQL data type of the parameter. This value is read from the SQL_DESC_CONCISE_TYPE record field of the IPD. This will be one of the values in the "SQL Data Types" section of Appendix D, "Data Types," in the **SOLID Programmer Guide** or a driver-specific SQL data type.

In ODBC 3.x, SQL_TYPE_DATE, SQL_TYPE_TIME, or SQL_TYPE_TIMESTAMP will be returned in **DataTypePtr* for date, time, or timestamp data, respectively; in ODBC 2.x, SQL_DATE, SQL_TIME, or SQL_TIMESTAMP will be returned. The Driver Manager performs the required mappings when an ODBC 2.x application is working with an ODBC 3.x driver or when an ODBC 3.x application is working with an ODBC 2.x driver.

When *ColumnNumber* is equal to 0 (for a bookmark column), SQL_BINARY is returned in **DataTypePtr* for variable-length bookmarks. (SQL_INTEGER is returned if bookmarks are used by an ODBC 3.x application working with an ODBC 2.x driver or by an ODBC 2.x application working with an ODBC 3.x driver.)

For more information, see “SQL Data Types” in Appendix D, “Data Types” of the **SOLID Programmer Guide**. For information about driver-specific SQL data types, see the driver's documentation.

ParameterSizePtr

[Output]

Pointer to a buffer in which to return the size of the column or expression of the corresponding parameter marker as defined by the data source. For further information concerning column size, see “Column Size, Decimal Digits, and Transfer Octet Length,” in Appendix D, “Data Types.” of the **SOLID Programmer Guide**.

DecimalDigitsPtr

[Output]

Pointer to a buffer in which to return the number of decimal digits of the column or expression of the corresponding parameter as defined by the data source. For more information on decimal digits, see “Column Size, Decimal Digits, and Transfer Octet Length,” in Appendix D, “Data Types.” of the **SOLID Programmer Guide**.

NullablePtr

[Output]

Pointer to a buffer in which to return a value that indicates whether the parameter allows NULL values. This value is read from the SQL_DESC_NULLABLE field of the IPD. 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 **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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.)
07009	Invalid descriptor index	<p>(DM) The value specified for the argument <i>ParameterNumber</i> is less than 1.</p> <p>The value specified for the argument <i>ParameterNumber</i> was greater than the number of parameters in the associated SQL statement.</p> <p>The parameter marker was part of a non-DML statement.</p> <p>The parameter marker was part of a SELECT list.</p>
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 number of parameters in the INSERT statement did not match the number of columns in the table named in the statement.
HY000	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 SQLGetDiagRec in the * <i>MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>. Then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY010	Function sequence error	<p>(DM) The function was called prior to calling SQLPrepare or SQLExecDirect for the <i>StatementHandle</i>.</p> <p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p>

HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

Parameter markers are numbered in increasing parameter order, starting with 1, 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**.

For more information, see the Part 1 PDF file, “Describing Parameters” in Chapter 9, "Executing Statements."

Code Example

The following example prompts the user for an SQL statement and then prepares that statement. Next, it calls **SQLNumParams** to determine whether the statement contains any parameters. If so, it calls **SQLDescribeParam** to describe those parameters and **SQLBindParameter** to bind them. Finally, it prompts the user for the values of any parameters and then executes the statement.

```
SQLCHAR          Statement[100];
SQLSMALLINT      NumParams, i, DataType, DecimalDigits, Nullable;
SQLINTEGER       ParamSize;
SQLHSTMT         hstmt;

// Prompt the user for an SQL statement and prepare it.
GetSQLStatement(Statement);
SQLPrepare(hstmt, Statement, SQL_NTS);

// Check to see if there are any parameters. If so, process them.
SQLNumParams(hstmt, &NumParams);
if (NumParams) {
    // Allocate memory for three arrays. The first holds pointers to buffers in
    // which
    // each parameter value will be stored in character form. The second
    // contains the
    // length of each buffer. The third contains the length/indicator value for
    // each
    // parameter.
    SQLPOINTER * PtrArray = (SQLPOINTER *) malloc(NumParams *
    sizeof(SQLPOINTER));
    SQLINTEGER * BufferLenArray = (SQLINTEGER *) malloc(NumParams *
    sizeof(SQLINTEGER));
```

```

    SQLINTEGER * LenOrIndArray = (SQLINTEGER *) malloc(NumParams *
sizeof(SQLINTEGER));

    for (i = 0; i < NumParams; i++) {
        // Describe the parameter.
        SQLDescribeParam(hstmt, i + 1, &DataType, &ParamSize, &DecimalDigits,
&Nullable);

        // Call a helper function to allocate a buffer in which to store the
parameter
        // value in character form. The function determines the size of the buffer
from
        // the SQL data type and parameter size returned by SQLDescribeParam and
returns
        // a pointer to the buffer and the length of the buffer.
        AllocParamBuffer(DataType, ParamSize, &PtrArray[i], &BufferLenArray[i]);

        // Bind the memory to the parameter. Assume that we only have input
parameters.
        SQLBindParameter(hstmt, i + 1, SQL_PARAM_INPUT, SQL_C_CHAR, DataType,
ParamSize,
            DecimalDigits, PtrArray[i], BufferLenArray[i],
            &LenOrIndArray[i]);

        // Prompt the user for the value of the parameter and store it in the memory
// allocated earlier. For simplicity, this function does not check the value
// against the information returned by SQLDescribeParam. Instead, the driver
does
        // this when the statement is executed.
        GetParamValue(PtrArray[i], BufferLenArray[i], &LenOrIndArray[i]);
    }
}

// Execute the statement.
SQLExecute(hstmt);

// Process the statement further, such as retrieving results (if any) and
closing the
// cursor (if any). Code not shown.

// Free the memory allocated for each parameter and the memory allocated for
the arrays
// of pointers, buffer lengths, and length/indicator values.
for (i = 0; i < NumParams; i++) free(PtrArray[i]);
free(PtrArray);
free(BufferLenArray);
free(LenOrIndArray);

```

Related Functions

For information about	See
Binding a buffer to a parameter	SQLBindParameter
Canceling statement processing	SQLCancel
Executing a prepared SQL statement	SQLExecute
Preparing a statement for execution	SQLPrepare

SQLDescribeCol

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: ISO 92

Summary

SQLDescribeCol returns the result descriptor—column name, type, column size, decimal digits, and nullability—for one column in the result set. This information also is available in the fields of the IRD.

Syntax

SQLRETURN SQLDescribeCol(
SQLHSTMT	StatementHandle,
SQLSMALLINT	ColumnNumber,
SQLCHAR *	ColumnName,
SQLSMALLINT	BufferLength,
SQLSMALLINT *	NameLengthPtr,
SQLSMALLINT *	DataTypePtr,
SQLINTEGER *	ColumnSizePtr,
SQLSMALLINT *	DecimalDigitsPtr,
SQLSMALLINT *	NullablePtr);

Arguments

StatementHandle

[Input]
Statement handle.

ColumnNumber

[Input]
Column number of result data, ordered sequentially in increasing column order, starting at 1. The *ColumnNumber* argument can also be set to 0 to describe the bookmark column.

ColumnName

[Output]
Pointer to a buffer in which to return the column name. This value is read from the SQL_DESC_NAME field of the IRD. If the column is unnamed or the column name cannot be determined, the driver returns an empty string.

BufferLength

[Input]
Length of the **ColumnName* buffer, in characters.

NameLengthPtr

[Output]

Pointer to a buffer in which to return the total number of bytes (excluding the null-termination byte) available to return in **ColumnName*. If the number of bytes available to return is greater than or equal to *BufferLength*, the column name in **ColumnName* is truncated to *BufferLength* minus the length of a null-termination character.

DataTypePtr

[Output]

Pointer to a buffer in which to return the SQL data type of the column. This value is read from the SQL_DESC_CONCISE_TYPE field of the IRD. This will be one of the values in the “SQL Data Types” section of Appendix D, “Data Types” of the **SOLID Programmer Guide**, or a driver-specific SQL data type. If the data type cannot be determined, the driver returns SQL_UNKNOWN_TYPE.

In ODBC 3.x, SQL_TYPE_DATE, SQL_TYPE_TIME, or SQL_TYPE_TIMESTAMP is returned in **DataTypePtr* for date, time, or timestamp data, respectively; in ODBC 2.x, SQL_DATE, SQL_TIME, or SQL_TIMESTAMP is returned. The Driver Manager performs the required mappings when an ODBC 2.x application is working with an ODBC 3.x driver or when an ODBC 3.x application is working with an ODBC 2.x driver.

When *ColumnNumber* is equal to 0 (for a bookmark column), SQL_BINARY is returned in **DataTypePtr* for variable-length bookmarks. (SQL_INTEGER is returned if bookmarks are used by an ODBC 3.x application working with an ODBC 2.x driver or by an ODBC 2.x application working with an ODBC 3.x driver.)

For more information on these data types, see “SQL Data Types” in Appendix D, “Data Types” of the **SOLID Programmer Guide**. For information about driver-specific SQL data types, see the driver's documentation.

ColumnSizePtr

[Output]

Pointer to a buffer in which to return the size of the column on the data source. If the column size cannot be determined, the driver returns 0. For more information on column size, see “Column Size, Decimal Digits, Transfer Octet Length, and Display Size” in Appendix D, “Data Types” contained on the Microsoft Web site (ODBC Programming Reference).

DecimalDigitsPtr

[Output]

Pointer to a buffer in which to return the number of decimal digits of the column on the data source. If the number of decimal digits cannot be determined or is not applicable, the driver returns 0. For more information on decimal digits, “Column Size, Decimal Digits, Transfer Octet Length, and Display Size” in Appendix D, “Data Types” contained on the Microsoft Web site (ODBC Programming Reference).

NullablePtr

[Output]

Pointer to a buffer in which to return a value that indicates whether the column allows NULL values. This value is read from the SQL_DESC_NULLABLE field of the IRD. The value is one of the following:

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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	String data, right truncated	The buffer <i>*ColumnName</i> was not large enough to return the entire column name, so the column name was truncated. The length of the untruncated column name is returned in <i>*NameLengthPtr</i> . (Function returns SQL_SUCCESS_WITH_INFO.)
07005	Prepared statement not a <i>cursor-specification</i>	The statement associated with the <i>StatementHandle</i> did not return a result set. There were no columns to describe.
07009	Invalid descriptor index	(DM) The value specified for the argument <i>ColumnNumber</i> was equal to 0, and the SQL_ATTR_USE_BOOKMARKS statement option was SQL_UB_OFF. The value specified for the argument <i>ColumnNumber</i> was greater than the number of columns in the result set.
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.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation failure	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	Asynchronous processing was enabled for the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> . Then the function was called again

		on the <i>StatementHandle</i> . The function was called, and before it completed execution, <i>SQLCancel</i> was called on the <i>StatementHandle</i> from a different thread in a multithread application.
HY010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called. (DM) The function was called prior to calling <i>SQLPrepare</i> , <i>SQLExecute</i> , or a catalog function on the statement handle. (DM) <i>SQLExecute</i> , <i>SQLExecDirect</i> , <i>SQLBulkOperations</i> , or <i>SQLSetPos</i> was called for the <i>StatementHandle</i> and returned <i>SQL_NEED_DATA</i> . This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	(DM) The value specified for argument <i>BufferLength</i> was less than 0.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through <i>SQLSetConnectAttr</i> , <i>SQL_ATTR_CONNECTION_TIMEOUT</i> .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

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 statement handle.

For performance reasons, an application should not call **SQLDescribeCol** before executing a statement.

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**. For more information, see the Part I PDF file, "Result Set Metadata" in Chapter 10, "Retrieving Results (Basic)."

SQLDescribeCol retrieves the column name, type, and length generated by a **SELECT** statement. If the column is an expression, **ColumnName* 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

For information about	See
-----------------------	-----

Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Returning information about a column in a result set	SQLColAttribute
Fetching multiple rows of data	SQLFetch
Returning the number of result set columns	SQLNumResultCols
Preparing a statement for execution	SQLPrepare

SQLDisconnect

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

SQLDisconnect closes the connection associated with a specific connection handle.

Syntax

```
SQLRETURN SQLDisconnect(  
SQLHDBCConnectionHandle);
```

Arguments

ConnectionHandle

[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 **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_DBC and a *Handle* of *ConnectionHandle*. 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 does not exist	(DM) The connection specified in the argument <i>ConnectionHandle</i> was not open.
25000	Invalid transaction state	There was a transaction in process on the connection specified by the argument <i>ConnectionHandle</i> . The transaction remains active.
HY000	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 SQLGetDiagRec in the * <i>MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error	(DM) An asynchronously executing function was called for a <i>StatementHandle</i> associated with the <i>ConnectionHandle</i> and was still executing when SQLDisconnect was called. (DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for a <i>StatementHandle</i> associated with the <i>ConnectionHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request, and the connection is still active. The connection timeout period is set through SQLSetConnectAttr, SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>ConnectionHandle</i> does not support the function.

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 connection 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 **SQLEndTran**.

If an application calls **SQLDisconnect** before it has freed all statements associated with the connection, the driver, after it successfully disconnects from the data source, frees those statements and all descriptors that have been explicitly allocated on the connection. However, if one or more of the statements associated with the connection are still executing asynchronously, **SQLDisconnect** returns SQL_ERROR with a SQLSTATE value of HY010 (Function sequence error).

For information about how an application uses **SQLDisconnect**, see the Part I PDF file, “Disconnecting from a Data Source or Driver” in Chapter 6, "Connecting to a Data Source or Driver."

Disconnecting from a Pooled Connection

If connection pooling is enabled for a shared environment and an application calls **SQLDisconnect** on a connection in that environment, the connection is returned to the connection pool and is still available to other components using the same shared environment.

Code Example

See `SQLBrowseConnect` and `SQLConnect`.

Related Functions

For information about	See
Allocating a handle	SQLAllocHandle
Connecting to a data source	SQLConnect
Connecting to a data source using a connection string or dialog box	SQLDriverConnect
Executing a commit or rollback operation	SQLEndTran
Freeing a connection handle	SQLFreeConnect

SQLDriverConnect

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ODBC

Summary

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 system information.

SQLDriverConnect provides the following connection attributes:

- 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 system information. If the application supplies a partial connection string, the driver can prompt the user for connection information.
- Establish a connection to a data source using a connection string constructed from the information in a .dsn file.

After a connection is established, **SQLDriverConnect** returns the completed connection string. The application can use this string for subsequent connection requests. For more information, see the Part I PDF file, “Connecting with SQLDriverConnect” in Chapter 6, “Connecting to a Data Source or Driver.”

Syntax

SQLRETURN SQLDriverConnect(
SQLHDBC	ConnectionHandle,
SQLHWND	WindowHandle,
SQLCHAR *	InConnectionString,
SQLSMALLINT	StringLength1,
SQLCHAR *	OutConnectionString,
SQLSMALLINT	BufferLength,
SQLSMALLINT *	StringLength2Ptr,
SQLUSMALLINT	DriverCompletion);

Arguments

ConnectionHandle

[Input]

Connection handle.

WindowHandle

[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 **SQLDriverConnect** will not present any dialog boxes.

InConnectionString

[Input]

A full connection string (see the syntax in "Comments"), a partial connection string, or an empty string.

StringLength1

[Input]

Length of **InConnectionString*, in bytes.

OutConnectionString

[Output]

Pointer to a buffer 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 1,024 bytes for this buffer.

BufferLength

[Input]

Length of the **OutConnectionString* buffer. If the **OutConnectionString* value is a Unicode string (when calling **SQLDriverConnectW**), the *BufferLength* argument must be an even number.

StringLength2Ptr

[Output]

Pointer to a buffer in which to return the total number of characters (excluding the null-termination character) available to return in **OutConnectionString*. If the number of characters available to return is greater than or equal to *BufferLength*, the completed connection string in **OutConnectionString* is truncated to *BufferLength* minus the length of a null-termination character.

DriverCompletion

[Input]

Flag that indicates whether the Driver Manager or driver must prompt for more connection information:

SQL_DRIVER_PROMPT,
SQL_DRIVER_COMPLETE,
SQL_DRIVER_COMPLETE_REQUIRED, or
SQL_DRIVER_NOPROMPT.

(For additional information, see "Comments.")

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, 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 **SQLGetDiagRec** with an *fHandleType* of SQL_HANDLE_DBC and an *hHandle* of *ConnectionHandle*. 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	String data, right truncated	The buffer <i>*OutConnectionString</i> was not large enough to return the entire connection string, so the connection string was truncated. The length of the untruncated connection string is returned in <i>*StringLength2Ptr</i> . (Function returns SQL_SUCCESS_WITH_INFO.)
01S00	Invalid connection string attribute	An invalid attribute keyword was specified in the connection string (<i>InConnectionString</i>), but the driver was able to connect to the data source anyway. (Function returns SQL_SUCCESS_WITH_INFO.)
01S02	Option value changed	The driver did not support the specified value pointed to by the <i>ValuePtr</i> argument in <i>SQLSetConnectAttr</i> and substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)

01S08	Error saving file DSN	The string in <i>*InConnectionString</i> contained a FILEDSN keyword, but the .dsn file was not saved. (Function returns SQL_SUCCESS_WITH_INFO.)
01S09	Invalid keyword	(DM) The string in <i>*InConnectionString</i> contained a SAVEFILE keyword but not a DRIVER or a FILEDSN keyword. (Function returns SQL_SUCCESS_WITH_INFO.)
08001	Client unable to establish connection	The driver was unable to establish a connection with the data source.
08002	Connection name in use	(DM) The specified <i>ConnectionHandle</i> had already been used to establish a connection with a data source, and the connection was still open.
08004	Server rejected the 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 (<i>InConnectionString</i>), violated restrictions defined by the data source.
HY000	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 SQLGetDiagRec in the <i>*szMessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	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.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	(DM) The value specified for argument <i>StringLength1</i> was less than 0 and was not equal to SQL_NTS. (DM) The value specified for argument <i>BufferLength</i> was less than 0.
HY092	Invalid attribute/option identifier	(DM) The <i>DriverCompletion</i> argument was SQL_DRIVER_PROMPT, and the <i>WindowHandle</i> argument was a null pointer.
HY110	Invalid driver completion	(DM) The value specified for the argument <i>DriverCompletion</i> was not equal to SQL_DRIVER_PROMPT, SQL_DRIVER_COMPLETE, SQL_DRIVER_COMPLETE_REQUIRED, or SQL_DRIVER_NOPROMPT.

		(DM) Connection pooling was enabled, and the value specified for the argument <i>DriverCompletion</i> was not equal to SQL_DRIVER_NOPROMPT.
HYC00	Optional feature not implemented	The driver does not support the version of ODBC behavior that the application requested.
HYT00	Timeout expired	The login timeout period expired before the connection to the data source completed. The timeout period is set through SQLSetConnectAttr, SQL_ATTR_LOGIN_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr, SQL_ATTR_CONNECTION_TIMEOUT.
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 (<i>InConnectionString</i>) was not found in the system information, and there was no default driver specification. (DM) ODBC data source and default driver information could not be found in the system information.
IM003	Specified driver could not be loaded	(DM) The driver listed in the data source specification in the system information or specified by the DRIVER keyword was not found or could not be loaded for some other reason.
IM004	Driver's SQLAllocHandle on SQL_HANDLE_ENV failed	(DM) During SQLDriverConnect, the Driver Manager called the driver's SQLAllocHandle function with an <i>fHandleType</i> of SQL_HANDLE_ENV and the driver returned an error.
IM005	Driver's SQLAllocHandle on SQL_HANDLE_DBC failed.	(DM) During SQLDriverConnect, the Driver Manager called the driver's SQLAllocHandle function with an <i>fHandleType</i> of SQL_HANDLE_DBC and the driver returned an error.
IM006	Driver's SQLSetConnectAttr failed	(DM) During SQLDriverConnect, the Driver Manager called the driver's SQLSetConnectAttr 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 <i>DriverCompletion</i> was SQL_DRIVER_NOPROMPT.
IM008	Dialog failed	The driver attempted to display its login dialog box and failed. <i>WindowHandle</i> was a null pointer, and <i>DriverCompletion</i> was not SQL_DRIVER_NO_PROMPT.
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. (DM) The string in <i>*InConnectionString</i> contained a FILEDSN keyword, but the .dsn file did not contain a DRIVER keyword or a DSN keyword.
IM014	Invalid name of file DSN	(DM) The string in <i>*InConnectionString</i> contained a FILEDSN keyword, but the name of the .dsn file was invalid.
IM015	Corrupt file data source	(DM) The string in <i>*InConnectionString</i> contained a FILEDSN keyword, but the .dsn file was unreadable.

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{ }]
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 not case-sensitive; *attribute-value* may be case-sensitive; and the value of the **DSN** keyword does not consist solely of blanks.

Characters to Avoid

Because of connection string and initialization file grammar, keywords and attribute values that contain the characters `[]{}(),;?*!=@` not enclosed with braces should be avoided. The value of the **DSN** keyword cannot consist only of blanks and should not contain leading blanks. Because of the grammar of the system information, keywords and data source names cannot contain the backslash (`\`) character.

When to Use Braces

Applications do not have to add braces around the attribute value after the **DRIVER** keyword unless the attribute contains a semicolon (`;`), in which case the braces are required. If the attribute value that the driver receives includes braces, the driver should not remove them but they should be part of the returned connection string.

A DSN or connection string value enclosed with braces (`{ }`) containing any of the characters `[]{}(),;?*!=@` is passed intact to the driver. However, when using these characters in a keyword, the Driver Manager returns an error when working with file DSNs but passes the connection string to the driver for regular connection strings. Avoid using embedded braces in a keyword value.

Keyword-Value Pairs

The connection string may include any number of driver-defined keywords. Because the **DRIVER** keyword does not use information from the system information, 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 to connect to the data source.

The following table describes the attribute values of the **DSN**, **FILEDSN**, **DRIVER**, **UID**, **PWD**, and **SAVEFILE** keywords.

Keyword	Attribute value description
DSN	Name of a data source as returned by SQLDataSources or the data sources dialog box of SQLDriverConnect .
FILEDSN	Name of a .dsn file from which a connection string will be built for the data source. These data sources are called file data sources.
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=;).
SAVEFILE	The file name of a .dsn file in which the attribute values of keywords used in making the present, successful connection should be saved.

For information about how an application chooses a data source or driver, see the Part I PDF file, "Choosing a Data Source or Driver," in Chapter 6, "Connecting to a Data Source or Driver."

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 **FILEDSN** and **DSN** keywords are mutually exclusive: whichever keyword appears first is used, and the one that appears second is ignored. The **FILEDSN** and **DRIVER** keywords, on the other hand, are not mutually exclusive. If any keyword appears in a connection string with **FILEDSN**, then the attribute value of the keyword in the connection string is used rather than the attribute value of the same keyword in the .dsn file.

If the **FILEDSN** keyword is used, the keywords specified in a .dsn file are used to create a connection string. (For more information, see "File Data Sources," later in this section.) The **UID** keyword is optional; a .dsn file may be created with only the **DRIVER** keyword. The **PWD** keyword is not stored in a .dsn file. The default directory for saving and loading a .dsn file will be a combination of the path specified by **CommonFileDir** in HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion and "ODBC\DataSources". (If **CommonFileDir** were "C:\Program Files\Common Files", the default directory would be "C:\Program Files\Common Files\ODBC\Data Sources".)

Note A .dsn file can be manipulated directly by calling **SQLReadFileDSN** and **SQLWriteFileDSN** of the installer DLL described in the Part III PDF file.

If the **SAVEFILE** keyword is used, the attribute values of keywords used in making the present, successful connection will be saved as a .dsn file with the name of the attribute value of the **SAVEFILE** keyword.

The **SAVEFILE** keyword must be used in conjunction with the **DRIVER** keyword, the **FILEDSN** keyword, or both, or the function returns SQL_SUCCESS_WITH_INFO with SQLSTATE 01S09 (Invalid keyword). The **SAVEFILE** keyword must appear before the **DRIVER** keyword in the connection string, or the results will be undefined.

Driver Manager Guidelines

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

The action of the Driver Manager is based on the value of the *DriverCompletion* argument:

SQL_DRIVER_PROMPT: If the connection string does not contain either the **DRIVER**, **DSN**, or **FILEDSN** keyword, 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. (This dialog box will not display a data source with the name "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 *DriverCompletion* 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, connects to 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 does not contain the **DRIVER** keyword, the Driver Manager determines which driver to use as follows:

If the connection string contains the **DSN** keyword, the Driver Manager retrieves the driver associated with the data source from the system information.

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 system information. (For more information, see the Part III PDF file, "Default Subkey" in Chapter 19, "Configuring Data Sources.") The Driver Manager changes the value of the **DSN** keyword in the connection string to "DEFAULT".

If the **DSN** keyword in the connection string is set to "DEFAULT", the Driver Manager retrieves the driver associated with the Default data source from the system information.

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).

File Data Sources

If the connection string specified by the application in the call to **SQLDriverConnect** contains the **FILEDSN** keyword, and this keyword is not superseded by either the **DSN** or **DRIVER** keyword, then the Driver Manager creates a connection string using the information in the .dsn file and the *InConnectionString* argument. The Driver Manager proceeds as follows:

Checks whether the file name of the .dsn file is valid. If not, it returns **SQL_ERROR** with **SQLSTATE** IM014 (Invalid name of file DSN). If the file name is an empty string (""), and **SQL_DRIVER_NOPROMPT** is not specified, then the **File-Open** dialog box is displayed. If the file name contains a valid path but no file name or an invalid file name, and **SQL_DRIVER_NOPROMPT** is not specified, then the **File-Open** dialog box is displayed with the current directory set to the one specified in the file name. If the file name is an empty string (""), or the file name contains a valid path but no file name or an invalid file name, and **SQL_DRIVER_NOPROMPT** is specified, then **SQL_ERROR** is returned with **SQLSTATE** IM014 (Invalid name of file DSN).

Reads all keywords in the [ODBC] section of the .dsn file. If the **DRIVER** keyword is not present, it returns **SQL_ERROR** with **SQLSTATE** IM012 (Driver keyword syntax error), except where the .dsn file is unshareable and thus contains only the **DSN** keyword.

If the file data source is unshareable, the Driver Manager reads the value of the **DSN** keyword and connects as necessary to the user or system data source pointed to by the unshareable file data source. Steps 3 through 5 are not performed.

Constructs a connection string for the driver. The driver connection string is the union of the keywords specified in the .dsn file and those specified in the original application connection string. Rules for the construction of the driver connection string where keywords overlap are as follows:

If the **DRIVER** keyword exists in the application connection string and the drivers specified by the **DRIVER** keywords are not the same in the .dsn file and the application connection string, then the driver information in the .dsn file is ignored and the driver information in the application connection string is used. If the drivers specified by the **DRIVER** keyword are the same in the .dsn file and the application's connection string, then where all keywords overlap, those specified in the application connection string have precedence over those specified in the .dsn file.

In the new connection string, the **FILEDSN** keyword is eliminated.

Loads the driver by looking in the registry entry **HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\<Driver Name>\Driver** where <Driver Name> is specified by the **DRIVER** keyword.

Passes the driver the new connections string.

For examples of .dsn files, see the Part I PDF file, "Connecting Using File Data Sources" in Chapter 6, "Connecting to a Data Source or Driver."

SAVEFILE Keyword

If the connection string specified by the application contains the **SAVEFILE** keyword, then the Driver Manager saves the connection string in a .dsn file. The Driver Manager proceeds as follows:

Checks whether the file name of the .dsn file included as the attribute value of the **SAVEFILE** keyword is valid. If not, it returns **SQL_ERROR** with **SQLSTATE** IM014 (Invalid name of file DSN). The validity of the file name is determined by standard system naming rules. If the file name is an empty string (""), and the *DriverCompletion* argument is not **SQL_DRIVER_NOPROMPT**, then the file name is valid. If the file name already exists, then if *DriverCompletion* is **SQL_DRIVER_NOPROMPT**, the file is overwritten. If

DriverCompletion is `SQL_DRIVER_PROMPT`, `SQL_DRIVER_COMPLETE`, or `SQL_DRIVER_COMPLETE_REQUIRED`, a dialog box prompts the user to specify whether the file should be overwritten. If No is entered, then the **File-Save** dialog box appears.

If the driver returns `SQL_SUCCESS` and the file name was not an empty string, then the Driver Manager writes the connection information returned in the *OutConnectionString* argument to the specified file with the format specified in the "Connection Strings" section earlier in this chapter.

If the driver returns `SQL_SUCCESS` and the file name was an empty string (""), then the Driver Manager calls the **File-Save** common dialog box with the *hwnd* specified and writes the connection information returned in *OutConnectionString* to the file specified in the File-Save common dialog box with the format specified in the "Connection Strings" section earlier in this chapter.

If the driver returns `SQL_SUCCESS`, it returns the *OutConnectionString* argument containing the connection string to the application.

If the driver returns `SQL_SUCCESS_WITH_INFO` or `SQL_ERROR`, then the Driver Manager returns the `SQLSTATE` to the application.

Driver Guidelines

The driver checks whether 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 system information. 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 system information as follows:

If the connection string contains the **DSN** keyword, the driver retrieves the information for the specified data source.

If the connection string does not contain the **DSN** keyword, the specified data source is not found, or the **DSN** keyword is set to "DEFAULT", the driver retrieves the information for the Default data source.

The driver uses any information it retrieves from the system information to augment the information passed to it in the connection string. If the information in the system information duplicates information in the connection string, the driver uses the information in the connection string.

Based on the value of *DriverCompletion*, 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 system information (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 **InConnectionString* and the information returned from the dialog box. It places this connection string in the **OutConnectionString* buffer.

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 **InConnectionString* to **OutConnectionString*. If any information is missing or incorrect, the driver takes the same actions as it does when *DriverCompletion* is `SQL_DRIVER_PROMPT`, except that if *DriverCompletion* 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 **InConnectionString* to **OutConnectionString*. Otherwise, the driver returns SQL_ERROR for **SQLDriverConnect**.

On successful connection to the data source, the driver also sets **StringLength2Ptr* to the length of the output connection string that is available to return in **OutConnectionString*.

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

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 system information for the driver must contain the **ConnectFunctions** keyword with the second character set to "Y".

Connecting When Connection Pooling Is Enabled

Connection pooling allows an application to reuse a connection that has already been created. When **SQLDriverConnect** is called, the Driver Manager attempts to make the connection using a connection that is part of a pool of connections in an environment that has been designated for connection pooling. For more information on connection pooling, see SQLConnect.

The following restrictions apply when an application calls **SQLDriverConnect** to connect to a pooled connection:

No connection pooling processing is performed when the **SAVEFILE** keyword is specified in the connection string.

If connection pooling is enabled, **SQLDriverConnect** can be called only with a *DriverCompletion* argument of SQL_DRIVER_NOPROMPT; if **SQLDriverConnect** is called with any other *DriverCompletion*, SQLSTATE HY110 (Invalid driver completion) will be returned.

Connection Attributes

The SQL_ATTR_LOGIN_TIMEOUT connection attribute, set using **SQLSetConnectAttr**, defines the number of seconds to wait for a login request to complete with a successful connection by the driver before returning to the application. If the user is prompted to complete the connection string, a waiting period for each login request begins when the driver starts the connection process.

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 **SQLSetConnectAttr** with the SQL_ATTR_ACCESS_MODE attribute prior to calling **SQLDriverConnect**.

If a default translation library is specified in the system information for the data source, the driver loads it. A different translation library can be loaded by calling **SQLSetConnectAttr** with the SQL_ATTR_TRANSLATE_LIB attribute. A translation option can be specified by calling **SQLSetConnectAttr** with the SQL_ATTR_TRANSLATE_OPTION option.

Code Example

See the Part I PDF file "Connecting with SQLDriverConnect" section of Chapter 6, "Connecting to a Data Source or Driver."

Related Functions

For information about	See
Allocating a handle	SQLAllocHandle
Discovering and enumerating values required to connect to a data source	SQLBrowseConnect
Connecting to a data source	SQLConnect
Disconnecting from a data source	SQLDisconnect
Returning driver descriptions and attributes	SQLDrivers
Freeing a handle	SQLFreeHandle
Setting a connection attribute	SQLSetConnectAttr

SQLDrivers

Conformance

Version Introduced: ODBC 2.0

Standards Compliance: ODBC

Summary

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

Syntax

SQLRETURN SQLDrivers (
SQLHENV	EnvironmentHandle,
SQLUSMALLINT	Direction,
SQLCHAR *	DriverDescription,
SQLSMALLINT	BufferLength1,
SQLSMALLINT *	DescriptionLengthPtr,
SQLCHAR *	DriverAttributes,
SQLSMALLINT	BufferLength2,
SQLSMALLINT *	AttributesLengthPtr);

Arguments

EnvironmentHandle

[Input]

Environment handle.

Direction

[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).

DriverDescription

[Output]

Pointer to a buffer in which to return the driver description.

BufferLength1

[Input]

Length of the **DriverDescription* buffer, in bytes.

DescriptionLengthPtr

[Output]

Pointer to a buffer in which to return the total number of bytes (excluding the null-termination byte) available to return in **DriverDescription*. If the number of bytes available to return is greater than or equal to *BufferLength1*, the driver description in **DriverDescription* is truncated to *BufferLength1* minus the length of a null-termination character.

DriverAttributes

[Output]

Pointer to a buffer in which to return the list of driver attribute value pairs (see "Comments").

BufferLength2

[Input]

Length of the **DriverAttributes* buffer, in bytes. If the **DriverDescription* value is a Unicode string (when calling **SQLDriversW**), the *BufferLength* argument must be an even number.

AttributesLengthPtr

[Output]

Pointer to a buffer in which to return the total number of bytes (excluding the null-termination byte) available to return in **DriverAttributes*. If the number of bytes available to return is greater than or equal to *BufferLength2*, the list of attribute value pairs in **DriverAttributes* is truncated to *BufferLength2* minus the length of the null-termination character.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, 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 **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_ENV and a *Handle* of *EnvironmentHandle*. 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	String data, right truncated	<p>(DM) The buffer <i>*DriverDescription</i> was not large enough to return the entire driver description, so the description was truncated. The length of the entire driver description is returned in <i>*DescriptionLengthPtr</i>. (Function returns SQL_SUCCESS_WITH_INFO.)</p> <p>(DM) The buffer <i>*DriverAttributes</i> was not large enough to return the entire list of attribute value pairs, so the list was truncated. The length of the untruncated list of attribute value pairs is returned in <i>*AttributesLengthPtr</i>. (Function returns SQL_SUCCESS_WITH_INFO.)</p>
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	(DM) The Driver Manager was unable to allocate memory required to support execution or completion of the function.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	<p>(DM) The value specified for argument <i>BufferLength1</i> was less than 0.</p> <p>(DM) The value specified for argument <i>BufferLength2</i> was less than 0 or equal to 1.</p>
HY103	Invalid retrieval code	(DM) The value specified for the argument <i>Direction</i> was not equal to SQL_FETCH_FIRST or SQL_FETCH_NEXT.

Comments

SQLDrivers returns the driver description in the **DriverDescription* buffer. It returns additional information about the driver in the **DriverAttributes* buffer as a list of keyword-value pairs. All keywords listed in the system information for drivers will be returned for all drivers, except for **CreateDSN**, which is used to prompt creation of data sources and therefore is optional. 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 file-based driver using C syntax might return the following list of attributes ("\"0" represents a null character):

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

If **DriverAttributes* 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 **AttributesLengthPtr*.

Driver attribute keywords are added from the system information when the driver is installed. For more information, see the Part III PDF file, Chapter 18, "Installing ODBC Components."

An application can call **SQLDrivers** multiple times to retrieve all driver descriptions. The Driver Manager retrieves this information from the system information. When there are no more driver descriptions, **SQLDrivers** returns SQL_NO_DATA. If **SQLDrivers** is called with SQL_FETCH_NEXT immediately after it returns SQL_NO_DATA, it returns the first driver description. For information about how an application uses the information returned by **SQLDrivers**, see the Part I PDF file, "Choosing a Data Source or Driver" in Chapter 6, "Connecting to a Data Source or Driver."

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 standards compliance.

Related Functions

For information about	See
Discovering and listing values required to connect to a data source	SQLBrowseConnect
Connecting to a data source	SQLConnect
Returning data source names	SQLDataSources
Connecting to a data source using a connection string or dialog box	SQLDriverConnect

SQLEndTran

Conformance

Version Introduced: ODBC 3.0

Standards Compliance: ISO 92

Summary

SQLEndTran requests a commit or rollback operation for all active operations on all statements associated with a connection. **SQLEndTran** can also request that a commit or rollback operation be performed for all connections associated with an environment.

Note For more information about what the Driver Manager maps this function to when an ODBC 3.x application is working with an ODBC 2.x driver, see the Part I PDF file, "Mapping Replacement Functions for Backward Compatibility of Applications" in Chapter 17, "Programming Considerations."

Syntax

SQLRETURN SQLEndTran (
SQLSMALLINT	HandleType,
SQLHANDLE	Handle,
SQLSMALLINT	CompletionType);

Arguments

HandleType

[Input]

Handle type identifier. Contains either `SQL_HANDLE_ENV` (if *Handle* is an environment handle) or `SQL_HANDLE_DBC` (if *Handle* is a connection handle).

Handle

[Input]

The handle, of the type indicated by *HandleType*, indicating the scope of the transaction. See "Comments" for more information.

CompletionType

[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 **SQLEndTran** returns `SQL_ERROR` or `SQL_SUCCESS_WITH_INFO`, an associated `SQLSTATE` value may be obtained by calling **SQLGetDiagRec** with the appropriate *HandleType* and *Handle*. The following table lists the `SQLSTATE` values commonly returned by **SQLEndTran** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of `SQLSTATE`s 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 <code>SQL_SUCCESS_WITH_INFO</code> .)
08003	Connection not open	(DM) The <i>HandleType</i> was <code>SQL_HANDLE_DBC</code> , and the <i>Handle</i> was not in a connected state.
08007	Connection failure during transaction	The <i>HandleType</i> was <code>SQL_HANDLE_DBC</code> , and the connection associated with the <i>Handle</i> failed during the execution of the function, and it cannot be determined whether the requested COMMIT or ROLLBACK occurred before the failure.
25S01	Transaction state unknown	One or more of the connections in <i>Handle</i> failed to complete the transaction with the outcome specified, and the outcome is unknown.
25S02	Transaction is still active	The driver was not able to guarantee that all work in the global transaction could be completed atomically, and the transaction is still active.

25S03	Transaction is rolled back	The driver was not able to guarantee that all work in the global transaction could be completed atomically, and all work in the transaction active in <i>Handle</i> was rolled back.
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40002	Integrity constraint violation	The <i>CompletionType</i> was SQL_COMMIT, and the commitment of changes caused integrity constraint violation. As a result, the transaction was rolled back.
HY000	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 SQLGetDiagRec in the <i>*szMessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error	(DM) An asynchronously executing function was called for a <i>StatementHandle</i> associated with the <i>ConnectionHandle</i> and was still executing when SQLEndTran was called. (DM) SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos was called for a <i>StatementHandle</i> associated with the <i>ConnectionHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY012	Invalid transaction operation code	(DM) The value specified for the argument <i>CompletionType</i> was neither SQL_COMMIT nor SQL_ROLLBACK.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY092	Invalid attribute/option identifier	(DM) The value specified for the argument <i>HandleType</i> was neither SQL_HANDLE_ENV nor SQL_HANDLE_DBC.
HYC00	Optional feature not implemented	The driver or data source does not support the ROLLBACK operation.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>ConnectionHandle</i> does not support the function.

Comments

For an ODBC 3.x driver, if *HandleType* is `SQL_HANDLE_ENV` and *Handle* is a valid environment handle, then the Driver Manager will call **SQLEndTran** in each driver associated with the environment. The *Handle* argument for the call to a driver will be the driver's environment handle. For an ODBC 2.x driver, if *HandleType* is `SQL_HANDLE_ENV` and *Handle* is a valid environment handle, and there are multiple connections in a connected state in that environment, then the Driver Manager will call **SQLTransact** in the driver once for each connection in a connected state in that environment. The *Handle* argument in each call will be the connection's handle. In either case, the driver will attempt to commit or roll back transactions, depending on the value of *CompletionType*, on all connections that are in a connected state on that environment. Connections that are not active do not affect the transaction.

Note **SQLEndTran** cannot be used to commit or roll back transactions on a shared environment. `SQLSTATE HY092` (Invalid attribute/option identifier) will be returned if **SQLEndTran** is called with *Handle* set to either the handle of a shared environment or the handle of a connection on a shared environment.

The Driver Manager will return `SQL_SUCCESS` only if it receives `SQL_SUCCESS` for each connection. If the Driver Manager receives `SQL_ERROR` on one or more connections, it returns `SQL_ERROR` to the application, and the diagnostic information is placed in the diagnostic data structure of the environment. To determine which connection or connections failed during the commit or rollback operation, the application can call **SQLGetDiagRec** for each connection.

Note The Driver Manager does not simulate a global transaction across all connections and therefore does not use two-phase commit protocols.

If *CompletionType* is `SQL_COMMIT`, **SQLEndTran** issues a commit request for all active operations on any statement associated with an affected connection. If *CompletionType* is `SQL_ROLLBACK`, **SQLEndTran** issues a rollback request for all active operations on any statement associated with an affected connection. If no transactions are active, **SQLEndTran** returns `SQL_SUCCESS` with no effect on any data sources. For more information, see the Part I PDF file, "Committing and Rolling Back Transactions" in Chapter 14, "Transactions."

If the driver is in manual-commit mode (by calling **SQLSetConnectAttr** with the `SQL_ATTR_AUTOCOMMIT` attribute set to `SQL_AUTOCOMMIT_OFF`), a new transaction is implicitly started when an SQL statement that can be contained within a transaction is executed against the current data source. For more information, see the Part I PDF file, "Commit Mode" in Chapter 14, "Transactions."

To determine how transaction operations affect cursors, an application calls **SQLGetInfo** with the `SQL_CURSOR_ROLLBACK_BEHAVIOR` and `SQL_CURSOR_COMMIT_BEHAVIOR` options. For more information, see the following paragraphs and also see the Part I PDF file, "Effect of Transactions of Cursors and Prepared Statements" in Chapter 14, "Transactions."

If the `SQL_CURSOR_ROLLBACK_BEHAVIOR` or `SQL_CURSOR_COMMIT_BEHAVIOR` value equals `SQL_CB_DELETE`, **SQLEndTran** closes and deletes all open cursors on all statements associated with the connection and discards all pending results. **SQLEndTran** leaves any statement present in an allocated (unprepared) state; the application can reuse them for subsequent SQL requests or can call **SQLFreeStmt** or **SQLFreeHandle** with a *HandleType* of `SQL_HANDLE_STMT` to deallocate them.

If the `SQL_CURSOR_ROLLBACK_BEHAVIOR` or `SQL_CURSOR_COMMIT_BEHAVIOR` value equals `SQL_CB_CLOSE`, **SQLEndTran** closes all open cursors on all statements associated with the connection. **SQLEndTran** leaves any statement present in a prepared state; the application can call **SQLExecute** for a statement associated with the connection without first calling **SQLPrepare**.

If the `SQL_CURSOR_ROLLBACK_BEHAVIOR` or `SQL_CURSOR_COMMIT_BEHAVIOR` value equals `SQL_CB_PRESERVE`, **SQLEndTran** does not affect open cursors associated with the connection. Cursors remain at the row they pointed to prior to the call to **SQLEndTran**.

For drivers and data sources that support transactions, calling **SQLEndTran** with either `SQL_COMMIT` or `SQL_ROLLBACK` when no transaction is active returns `SQL_SUCCESS` (indicating that there is no work to be committed or rolled back) and has no effect on the data source.

When a driver is in autocommit mode, the Driver Manager does not call **SQLEndTran** in the driver. **SQLEndTran** always returns `SQL_SUCCESS` regardless of whether it is called with a *CompletionType* of `SQL_COMMIT` or `SQL_ROLLBACK`.

Drivers or data sources that do not support transactions (**SQLGetInfo** option `SQL_TXN_CAPABLE` is `SQL_TC_NONE`) are effectively always in autocommit mode and therefore always return `SQL_SUCCESS` for **SQLEndTran** whether or not they are called with a *CompletionType* of `SQL_COMMIT` or `SQL_ROLLBACK`. Such drivers and data sources do not actually roll back transactions when requested to do so.

Related Functions

For information about	See
Returning information about a driver or data source	SQLGetInfo
Freeing a handle	SQLFreeHandle
Freeing a statement handle	SQLFreeStmt

SQLError

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: Deprecated

Summary

SQLError returns error or status information.

For more information about what the Driver Manager maps this function to when an ODBC 2.x application is working with an ODBC 3.x driver, see “Mapping Deprecated Functions” (Appendix G, "Driver Guidelines for Backward Compatibility") contained on the Microsoft Web site (ODBC Programming Reference).

SQLExecDirect

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: ISO 92

Summary

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

SQLRETURN SQLExecDirect(
SQLHSTMT	StatementHandle,
SQLCHAR *	StatementText,
SQLINTEGER	TextLength);

Arguments

StatementHandle

[Input]
Statement handle.

StatementText

[Input]
SQL statement to be executed.

TextLength

[Input]
Length of **StatementText*.

Returns

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

Diagnostics

When **SQLExecDirect** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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.)
01001	Cursor operation conflict	* <i>StatementText</i> contained a positioned update or delete statement, and no rows or more than one row were updated or deleted. (For more information about updates to more than one row, see the description of the SQL_ATTR_SIMULATE_CURSOR <i>Attribute</i> in SQLSetStmtAttr .) (Function returns SQL_SUCCESS_WITH_INFO.)
01003	NULL value eliminated in set function	The argument <i>StatementText</i> contained a set function (such as AVG , MAX , MIN , and so on), but not the COUNT set function, and NULL argument values were eliminated before the function was applied. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	String data, right truncated	String or binary data returned for an input/output or output parameter resulted in the truncation of nonblank character or non-NULL binary data. If it was a string value, it was right-truncated. (Function returns SQL_SUCCESS_WITH_INFO.)
01006	Privilege not revoked	* <i>StatementText</i> contained a REVOKE statement, and the user did not have the specified privilege. (Function returns SQL_SUCCESS_WITH_INFO.)
01007	Privilege not granted	* <i>StatementText</i> was a GRANT statement, and the user could not be granted the specified privilege.
01S02	Option value changed	A specified statement attribute was invalid because of implementation working conditions, so a similar value was temporarily substituted. (SQLGetStmtAttr can be called to determine what the temporarily substituted value is.) The substitute value is valid for the <i>StatementHandle</i> until the cursor is closed, at which point the statement attribute reverts to its previous value. The statement attributes that can be changed are: SQL_ATTR_CONCURRENCY SQL_ATTR_CURSOR_TYPE SQL_ATTR_KEYSET_SIZE SQL_ATTR_MAX_LENGTH SQL_ATTR_MAX_ROWS SQL_ATTR_QUERY_TIMEOUT SQL_ATTR_SIMULATE_CURSOR (Function returns SQL_SUCCESS_WITH_INFO.)
01S07	Fractional truncation	The data returned for an input/output or output parameter was truncated such that the fractional part of a numeric data type was truncated or the fractional portion of the time component of a time, timestamp, or interval data type was truncated.

		(Function returns SQL_SUCCESS_WITH_INFO.)
07002	COUNT field incorrect	<p>The number of parameters specified in SQLBindParameter was less than the number of parameters in the SQL statement contained in <i>*StatementText</i>.</p> <p>SQLBindParameter was called with <i>ParameterValuePtr</i> set to a null pointer, <i>StrLen_or_IndPtr</i> not set to SQL_NULL_DATA or SQL_DATA_AT_EXEC, and <i>InputOutputType</i> not set to SQL_PARAM_OUTPUT, so that the number of parameters specified in SQLBindParameter was greater than the number of parameters in the SQL statement contained in <i>*StatementText</i>.</p>
07006	Restricted data type attribute violation	<p>The data value identified by the <i>ValueType</i> argument in SQLBindParameter for the bound parameter could not be converted to the data type identified by the <i>ParameterType</i> argument in SQLBindParameter.</p> <p>The data value returned for a parameter bound as SQL_PARAM_INPUT_OUTPUT or SQL_PARAM_OUTPUT could not be converted to the data type identified by the <i>ValueType</i> argument in SQLBindParameter.</p> <p>(If the data values for one or more rows could not be converted but one or more rows were successfully returned, this function returns SQL_SUCCESS_WITH_INFO.)</p>
07S01	Invalid use of default parameter	A parameter value, set with SQLBindParameter , was SQL_DEFAULT_PARAM, and the corresponding parameter did not have a default value.
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	<i>*StatementText</i> 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	<i>*StatementText</i> contained a CREATE VIEW statement, and the unqualified column list (the number of columns specified for the view in the <i>column-identifier</i> arguments of the SQL statement) contained more names than the number of columns in the derived table defined by the <i>query-specification</i> argument of the SQL statement.
22001	String data, right truncation	The assignment of a character or binary value to a column resulted in the truncation of nonblank character data or non-null binary data.
22002	Indicator variable required	NULL data was bound to an output parameter whose

	but not supplied	<i>StrLen_or_IndPtr</i> set by SQLBindParameter was a null pointer.
22003	Numeric value out of range	<p>*<i>StatementText</i> contained an SQL statement that contained a bound 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.</p> <p>Returning a numeric value (as numeric or string) for one or more input/output or output parameters would have caused the whole (as opposed to fractional) part of the number to be truncated.</p>
22007	Invalid datetime format	<p>*<i>StatementText</i> contained an SQL statement that contained a date, time, or timestamp structure as a bound parameter, and the parameter was, respectively, an invalid date, time, or timestamp.</p> <p>An input/output or output parameter was bound to a date, time, or timestamp C structure, and a value in the returned parameter was, respectively, an invalid date, time, or timestamp. (Function returns SQL_SUCCESS_WITH_INFO.)</p>
22008	Datetime field overflow	<p>*<i>StatementText</i> contained an SQL statement that contained a datetime expression that, when computed, resulted in a date, time, or timestamp structure that was invalid.</p> <p>A datetime expression computed for an input/output or output parameter resulted in a date, time, or timestamp C structure that was invalid.</p>
22012	Division by zero	<p>*<i>StatementText</i> contained an SQL statement that contained an arithmetic expression that caused division by zero.</p> <p>An arithmetic expression calculated for an input/output or output parameter resulted in division by zero.</p>
22015	Interval field overflow	<p>*<i>StatementText</i> contained an exact numeric or interval parameter that, when converted to an interval SQL data type, caused a loss of significant digits.</p> <p>*<i>StatementText</i> contained an interval parameter with more than one field that, when converted to a numeric data type in a column, had no representation in the numeric data type.</p> <p>*<i>StatementText</i> contained parameter data that was assigned to an interval SQL type, and there was no representation of the value of the C type in the interval SQL type.</p> <p>Assigning an input/output or output parameter that was an exact numeric or interval SQL type to an interval C</p>

		<p>type caused a loss of significant digits.</p> <p>When an input/output or output parameter was assigned to an interval C structure, there was no representation of the data in the interval data structure.</p>
22018	Invalid character value for cast specification	<p>*<i>StatementText</i> contained a C type that was an exact or approximate numeric, a datetime, or an interval data type; the SQL type of the column was a character data type; and the value in the column was not a valid literal of the bound C type.</p> <p>When an input/output or output parameter was returned, the SQL type was an exact or approximate numeric, a datetime, or an interval data type; the C type was SQL_C_CHAR; and the value in the column was not a valid literal of the bound SQL type.</p>
22019	Invalid escape character	* <i>StatementText</i> contained an SQL statement that contained a LIKE predicate with an ESCAPE in the WHERE clause, and the length of the escape character following ESCAPE was not equal to 1.
22025	Invalid escape sequence	* <i>StatementText</i> contained an SQL statement that contained " LIKE <i>pattern value</i> ESCAPE <i>escape character</i> " in the WHERE clause, and the character following the escape character in the pattern value was not one of "%" or "_".
23000	Integrity constraint violation	* <i>StatementText</i> contained an SQL statement that 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	<p>A cursor was positioned on the <i>StatementHandle</i> by SQLFetch or SQLFetchScroll. This error is returned by the Driver Manager if SQLFetch or SQLFetchScroll has not returned SQL_NO_DATA, and is returned by the driver if SQLFetch or SQLFetchScroll has returned SQL_NO_DATA.</p> <p>A cursor was open but not positioned on the <i>StatementHandle</i>.</p> <p>*<i>StatementText</i> 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.</p>
34000	Invalid cursor name	* <i>StatementText</i> contained a positioned update or delete statement, and the cursor referenced by the statement being executed was not open.
3D000	Invalid catalog name	The catalog name specified in <i>StatementText</i> was invalid.

3F000	Invalid schema name	The schema name specified in <i>StatementText</i> was invalid.
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
42000	Syntax error or access violation	<p>*<i>StatementText</i> contained an SQL statement that was not preparable or contained a syntax error.</p> <p>The user did not have permission to execute the SQL statement contained in *<i>StatementText</i>.</p>
42S01	Base table or view already exists	* <i>StatementText</i> contained a CREATE TABLE or CREATE VIEW statement, and the table name or view name specified already exists.
42S02	Base table or view not found	<p>*<i>StatementText</i> contained a DROP TABLE or a DROP VIEW statement, and the specified table name or view name did not exist.</p> <p>*<i>StatementText</i> contained an ALTER TABLE statement, and the specified table name did not exist.</p> <p>*<i>StatementText</i> contained a CREATE VIEW statement, and a table name or view name defined by the query specification did not exist.</p> <p>*<i>StatementText</i> contained a CREATE INDEX statement, and the specified table name did not exist.</p> <p>*<i>StatementText</i> contained a GRANT or REVOKE statement, and the specified table name or view name did not exist.</p> <p>*<i>StatementText</i> contained a SELECT statement, and a specified table name or view name did not exist.</p> <p>*<i>StatementText</i> contained a DELETE, INSERT, or UPDATE statement, and the specified table name did not exist.</p> <p>*<i>StatementText</i> contained a CREATE TABLE statement, and a table specified in a constraint (referencing a table other than the one being created) did not exist.</p> <p>*<i>StatementText</i> contained a CREATE SCHEMA statement, and a specified table name or view name did not exist.</p>
42S11	Index already exists	* <i>StatementText</i> contained a CREATE INDEX statement, and the specified index name already existed.

		* <i>StatementText</i> contained a CREATE SCHEMA statement, and the specified index name already existed.
42S12	Index not found	* <i>StatementText</i> contained a DROP INDEX statement, and the specified index name did not exist.
42S21	Column already exists	* <i>StatementText</i> 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.
42S22	Column not found	<p>*<i>StatementText</i> contained a CREATE INDEX statement, and one or more of the column names specified in the column list did not exist.</p> <p>*<i>StatementText</i> contained a GRANT or REVOKE statement, and a specified column name did not exist.</p> <p>*<i>StatementText</i> contained a SELECT, DELETE, INSERT, or UPDATE statement, and a specified column name did not exist.</p> <p>*<i>StatementText</i> contained a CREATE TABLE statement, and a column specified in a constraint (referencing a table other than the one being created) did not exist.</p> <p>*<i>StatementText</i> contained a CREATE SCHEMA statement, and a specified column name did not exist.</p>
44000	WITH CHECK OPTION violation	<p>The argument <i>StatementText</i> contained an INSERT statement performed on a viewed table or a table derived from the viewed table that was created by specifying WITH CHECK OPTION, such that one or more rows affected by the INSERT statement will no longer be present in the viewed table.</p> <p>The argument <i>StatementText</i> contained an UPDATE statement performed on a viewed table or a table derived from the viewed table that was created by specifying WITH CHECK OPTION, such that one or more rows affected by the UPDATE statement will no longer be present in the viewed table.</p>
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	Asynchronous processing was enabled for the <i>StatementHandle</i> . The function was called, and before

		<p>it completed execution, SQLCancel was called on the <i>StatementHandle</i>. Then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY009	Invalid use of null pointer	(DM) * <i>StatementText</i> was a null pointer.
HY010	Function sequence error	<p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	<p>(DM) The argument <i>TextLength</i> was less than or equal to 0 but not equal to SQL_NTS.</p> <p>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, SQL_DEFAULT_PARAM, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.</p> <p>A parameter value, set with SQLBindParameter, was not a null pointer; the C data type was SQL_C_BINARY or SQL_C_CHAR; and the parameter length value was less than 0 but was not SQL_NTS, SQL_NULL_DATA, SQL_DATA_AT_EXEC, SQL_DEFAULT_PARAM, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.</p> <p>A parameter length value bound by SQLBindParameter was set to SQL_DATA_AT_EXEC; the SQL type was either SQL_LONGVARCHAR, SQL_LONGVARBINARY, or a long data source-specific data type; and the SQL_NEED_LONG_DATA_LEN information type in SQLGetInfo was "Y".</p>
HY105	Invalid parameter type	The value specified for the argument <i>InputOutputType</i> in SQLBindParameter was SQL_PARAM_OUTPUT, and the parameter was an input parameter.

HY109	Invalid cursor position	* <i>StatementText</i> contained a positioned update or delete statement, and the cursor was positioned (by SQLSetPos or SQLFetchScroll) on a row that had been deleted or could not be fetched.
HYC00	Optional feature not implemented	The combination of the current settings of the SQL_ATTR_CONCURRENCY and SQL_ATTR_CURSOR_TYPE statement attributes was not supported by the driver or data source. The SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_VARIABLE , and the SQL_ATTR_CURSOR_TYPE statement attribute was set to a cursor type for which the driver does not support bookmarks.
HYT00	Timeout expired	The query timeout period expired before the data source returned the result set. The timeout period is set through SQLSetStmtAttr , SQL_ATTR_QUERY_TIMEOUT .
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

The application calls **SQLExecDirect** to send an SQL statement to the data source. For more information about direct execution, see the Part I PDF file, “Direct Execution” in Chapter 9, “Executing Statements.” The driver modifies the statement to use the form of SQL used by the data source and then submits it to the data source. In particular, the driver modifies the escape sequences used to define certain features in SQL. For the syntax of escape sequences, see the Part I PDF file, “Escape Sequences in ODBC” in Chapter 8, “SQL Statements.”

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. For information about parameters, see the Part I PDF file, “Statement Parameters” in Chapter 9, “Executing Statements.”

If the SQL statement is a **SELECT** statement and if the application called **SQLSetCursorName** to associate a cursor with a statement, 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. For more information, see the Part I PDF file, “Manual-Commit Mode” in Chapter 14, “Transactions.”

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, an application calls **SQLEndTran**.

If **SQLExecDirect** encounters a data-at-execution parameter, it returns `SQL_NEED_DATA`. The application sends the data using **SQLParamData** and **SQLPutData**. See [SQLBindParameter](#), [SQLParamData](#), [SQLPutData](#), and the Part I PDF file, “Sending Long Data” in Chapter 9, “Executing Statements” for more information.

If **SQLExecDirect** executes a searched update or delete statement that does not affect any rows at the data source, the call to **SQLExecDirect** returns `SQL_NO_DATA`.

If the value of the `SQL_ATTR_PARAMSET_SIZE` statement attribute is greater than 1 and the SQL statement contains at least one parameter marker, **SQLExecDirect** will execute the SQL statement once for each set of parameter values from the arrays pointed to by the *ParameterValuePointer* argument in the call to **SQLBindParameter**. For more information, see the Part I PDF file “Arrays of Parameter Values” in Chapter 9, “Executing Statements.”

If bookmarks are turned on and a query is executed that cannot support bookmarks, the driver should attempt to coerce the environment to one that supports bookmarks by changing an attribute value and returning `SQLSTATE 01S02` (Option value changed). If the attribute cannot be changed, the driver should return `SQLSTATE HY024` (Invalid attribute value).

Note When using connection pooling, an application must not execute SQL statements that change the database or the context of the database, such as the **USE** *database* statement in SQL Server, which changes the catalog used by a data source.

Code Example

See [SQLBindCol](#) and [SQLGetData](#).

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Executing a commit or rollback operation	SQLEndTran
Executing a prepared SQL statement	SQLExecute
Fetching multiple rows of data	SQLFetch
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Returning a cursor name	SQLGetCursorName
Fetching part or all of a column of data	SQLGetData
Returning the next parameter to send data for	SQLParamData
Preparing a statement for execution	SQLPrepare
Sending parameter data at execution time	SQLPutData
Setting a cursor name	SQLSetCursorName
Setting a statement attribute	SQLSetStmtAttr

SQLExecute

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

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

Syntax

```
SQLRETURN SQLExecute(  
    SQLHSTMT StatementHandle);
```

Arguments

StatementHandle

[Input]

Statement handle.

Returns

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

Diagnostics

When **SQLExecute** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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.)
01001	Cursor operation conflict	The prepared statement associated with the <i>StatementHandle</i> contained a positioned update or delete statement, and no rows or more than one row were updated or deleted. (For more information about updates to more than one row, see the description of the SQL_ATTR_SIMULATE_CURSOR <i>Attribute</i> in SQLSetStmtAttr .) (Function returns SQL_SUCCESS_WITH_INFO.)

01003	NULL value eliminated in set function	The prepared statement associated with <i>StatementHandle</i> contained a set function (such as AVG , MAX , MIN , and so on), but not the COUNT set function, and NULL argument values were eliminated before the function was applied. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	String data, right truncated	String or binary data returned for an output parameter resulted in the truncation of nonblank character or non-NULL binary data. If it was a string value, it was right-truncated. (Function returns SQL_SUCCESS_WITH_INFO.)
01006	Privilege not revoked	The prepared statement associated with the <i>StatementHandle</i> was a REVOKE statement, and the user did not have the specified privilege. (Function returns SQL_SUCCESS_WITH_INFO.)
01007	Privilege not granted	The prepared statement associated with the <i>StatementHandle</i> was a GRANT statement, and the user could not be granted the specified privilege.
01S02	Option value changed	A specified statement attribute was invalid because of implementation working conditions, so a similar value was temporarily substituted. (SQLGetStmtAttr can be called to determine what the temporarily substituted value is.) The substitute value is valid for the <i>StatementHandle</i> until the cursor is closed, at which point the statement attribute reverts to its previous value. The statement attributes that can be changed are: SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ATTR_QUERY_TIMEOUT, and SQL_ATTR_SIMULATE_CURSOR. (Function returns SQL_SUCCESS_WITH_INFO.)
01S07	Fractional truncation	The data returned for an input/output or output parameter was truncated such that the fractional part of a numeric data type was truncated or the fractional portion of the time component of a time, timestamp, or interval data type was truncated. (Function returns SQL_SUCCESS_WITH_INFO.)
07002	COUNT field incorrect	The number of parameters specified in SQLBindParameter was less than the number of parameters in the SQL statement contained in * <i>StatementText</i> . SQLBindParameter was called with <i>ParameterValuePtr</i> set to a null pointer, <i>StrLen_or_IndPtr</i> not set to SQL_NULL_DATA or SQL_DATA_AT_EXEC, and <i>InputOutputType</i> not set to SQL_PARAM_OUTPUT, so that the number of

		parameters specified in SQLBindParameter was greater than the number of parameters in the SQL statement contained in <i>*StatementText</i> .
07006	Restricted data type attribute violation	<p>The data value identified by the <i>ValueType</i> argument in SQLBindParameter for the bound parameter could not be converted to the data type identified by the <i>ParameterType</i> argument in SQLBindParameter.</p> <p>The data value returned for a parameter bound as SQL_PARAM_INPUT_OUTPUT or SQL_PARAM_OUTPUT could not be converted to the data type identified by the <i>ValueType</i> argument in SQLBindParameter.</p> <p>(If the data values for one or more rows could not be converted but one or more rows were successfully returned, this function returns SQL_SUCCESS_WITH_INFO.)</p>
07S01	Invalid use of default parameter	A parameter value, set with SQLBindParameter , was SQL_DEFAULT_PARAM, and the corresponding parameter was not a parameter for an ODBC canonical procedure invocation.
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.
21S02	Degree of derived table does not match column list	The prepared statement associated with the <i>StatementHandle</i> contained a CREATE VIEW statement, and the unqualified column list (the number of columns specified for the view in the <i>column-identifier</i> arguments of the SQL statement) contained more names than the number of columns in the derived table defined by the <i>query-specification</i> argument of the SQL statement.
22001	String data, right truncation	The assignment of a character or binary value to a column resulted in the truncation of nonblank (character) or non-null (binary) characters or bytes.
22002	Indicator variable required but not supplied	NULL data was bound to an output parameter whose <i>StrLen_or_IndPtr</i> set by SQLBindParameter was a null pointer.
22003	Numeric value out of range	<p>The prepared statement associated with the <i>StatementHandle</i> contained a bound 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.</p> <p>Returning a numeric value (as numeric or string) for one or more input/output or output parameters would have caused the whole (as opposed to fractional) part of the number to be truncated.</p>

22007	Invalid datetime format	<p>The prepared statement associated with the <i>StatementHandle</i> contained an SQL statement that contained a date, time, or timestamp structure as a bound parameter, and the parameter was, respectively, an invalid date, time, or timestamp.</p> <p>An input/output or output parameter was bound to a date, time, or timestamp C structure, and a value in the returned parameter was, respectively, an invalid date, time, or timestamp. (Function returns SQL_SUCCESS_WITH_INFO.)</p>
22008	Datetime field overflow	<p>The prepared statement associated with the <i>StatementHandle</i> contained an SQL statement that contained a datetime expression that, when computed, resulted in a date, time, or timestamp structure that was invalid.</p> <p>A datetime expression computed for an input/output or output parameter resulted in a date, time, or timestamp C structure that was invalid.</p>
22012	Division by zero	<p>The prepared statement associated with the <i>StatementHandle</i> contained an arithmetic expression that caused division by zero.</p> <p>An arithmetic expression calculated for an input/output or output parameter resulted in division by zero.</p>
22015	Interval field overflow	<p>*<i>StatementText</i> contained an exact numeric or interval parameter that, when converted to an interval SQL data type, caused a loss of significant digits.</p> <p>*<i>StatementText</i> contained an interval parameter with more than one field that, when converted to a numeric data type in a column, had no representation in the numeric data type.</p> <p>*<i>StatementText</i> contained parameter data that was assigned to an interval SQL type, and there was no representation of the value of the C type in the interval SQL type.</p> <p>Assigning an input/output or output parameter that was an exact numeric or interval SQL type to an interval C type caused a loss of significant digits.</p> <p>When an input/output or output parameter was assigned to an interval C structure, there was no representation of the data in the interval data structure.</p>
22018	Invalid character value for cast specification	<p>*<i>StatementText</i> contained a C type that was an exact or approximate numeric, a datetime, or an interval data type; the SQL type of the column was a character data type; and the value in the column was not a valid literal</p>

		<p>of the bound C type.</p> <p>When an input/output or output parameter was returned, the SQL type was an exact or approximate numeric, a datetime, or an interval data type; the C type was SQL_C_CHAR; and the value in the column was not a valid literal of the bound SQL type.</p>
22019	Invalid escape character	The prepared statement associated with <i>StatementHandle</i> contained a LIKE predicate with an ESCAPE in the WHERE clause, and the length of the escape character following ESCAPE was not equal to 1.
22025	Invalid escape sequence	The prepared statement associated with <i>StatementHandle</i> contained " LIKE pattern value ESCAPE escape character" in the WHERE clause, and the character following the escape character in the pattern value was not one of "%" or "_".
23000	Integrity constraint violation	The prepared statement associated with the <i>StatementHandle</i> 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	<p>A cursor was positioned on the <i>StatementHandle</i> by SQLFetch or SQLFetchScroll. This error is returned by the Driver Manager if SQLFetch or SQLFetchScroll has not returned SQL_NO_DATA, and is returned by the driver if SQLFetch or SQLFetchScroll has returned SQL_NO_DATA.</p> <p>A cursor was open on the <i>StatementHandle</i>.</p> <p>The prepared statement associated with the <i>StatementHandle</i> 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.</p>
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
42000	Syntax error or access violation	The user did not have permission to execute the prepared statement associated with the <i>StatementHandle</i> .
44000	WITH CHECK OPTION violation	The prepared statement associated with <i>StatementHandle</i> contained an INSERT statement performed on a viewed table or a table derived from the viewed table that was created by specifying WITH

		<p>CHECK OPTION, such that one or more rows affected by the INSERT statement will no longer be present in the viewed table.</p> <p>The prepared statement associated with the <i>StatementHandle</i> contained an UPDATE statement performed on a viewed table or a table derived from the viewed table that was created by specifying WITH CHECK OPTION, such that one or more rows affected by the UPDATE statement will no longer be present in the viewed table.</p>
HY000	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 SQLGetDiagRec in the * <i>MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>. Then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY010	Function sequence error	<p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p> <p>(DM) The <i>StatementHandle</i> was not prepared.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	<p>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, SQL_DEFAULT_PARAM, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.</p> <p>A parameter value, set with SQLBindParameter, was not a null pointer; the C data type was</p>

		<p>SQL_C_BINARY or SQL_C_CHAR; and the parameter length value was less than 0 but was not SQL_NTS, SQL_NULL_DATA, SQL_DEFAULT_PARAM, or SQL_DATA_AT_EXEC, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.</p> <p>A parameter length value bound by SQLBindParameter was set to SQL_DATA_AT_EXEC; the SQL type was either SQL_LONGVARCHAR, SQL_LONGVARIABLE, or a long data source-specific data type; and the SQL_NEED_LONG_DATA_LEN information type in SQLGetInfo was "Y".</p>
HY105	Invalid parameter type	The value specified for the argument <i>InputOutputType</i> in SQLBindParameter was SQL_PARAM_OUTPUT, and the parameter was an input parameter.
HY109	Invalid cursor position	The prepared statement was a positioned update or delete statement, and the cursor was positioned (by SQLSetPos or SQLFetchScroll) on a row that had been deleted or could not be fetched.
HYC00	Optional feature not implemented	<p>The combination of the current settings of the SQL_ATTR_CONCURRENCY and SQL_ATTR_CURSOR_TYPE statement attributes was not supported by the driver or data source.</p> <p>The SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_VARIABLE, and the SQL_ATTR_CURSOR_TYPE statement attribute was set to a cursor type for which the driver does not support bookmarks.</p>
HYT00	Timeout expired	The query timeout period expired before the data source returned the result set. The timeout period is set through SQLSetStmtAttr , SQL_ATTR_QUERY_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

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

Comments

SQLExecute executes a statement prepared by **SQLPrepare**. After the application processes or discards the results from a call to **SQLExecute**, the application can call **SQLExecute** again with new parameter values. For more information about prepared execution, see the Part I PDF file, “Prepared Execution” in Chapter 9, “Executing Statements.”

To execute a **SELECT** statement more than once, the application must call **SQLCloseCursor** before reexecuting 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. For more information, see the Part I PDF file, “Manual-Commit Mode” in Chapter 14, “Transactions.”

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 **SQLEndTran**.

If **SQLExecute** encounters a data-at-execution parameter, it returns **SQL_NEED_DATA**. The application sends the data using **SQLParamData** and **SQLPutData**. See [SQLBindParameter](#), [SQLParamData](#), [SQLPutData](#), and the Part I PDF file, “Sending Long Data” in Chapter 9, “Executing Statements” for more information.

If **SQLExecute** executes a searched update or delete statement that does not affect any rows at the data source, the call to **SQLExecute** returns **SQL_NO_DATA**.

If the value of the **SQL_ATTR_PARAMSET_SIZE** statement attribute is greater than 1 and the SQL statement contains at least one parameter marker, **SQLExecute** executes the SQL statement once for each set of parameter values in the arrays pointed to by the **ParameterValuePtr* argument in the calls to **SQLBindParameter**. For more information, see Part I PDF file, “Arrays of Parameter Values” in Chapter 9, “Executing Statements.”

If bookmarks are enabled and a query is executed that cannot support bookmarks, the driver should attempt to coerce the environment to one that supports bookmarks by changing an attribute value and returning **SQLSTATE 01S02** (Option value changed). If the attribute cannot be changed, the driver should return **SQLSTATE HY024** (Invalid attribute value).

Note When using connection pooling, an application must not execute SQL statements that change the database or the context of the database, such as the **USE database** statement in SQL Server, which changes the catalog used by a data source.

Code Example

See [SQLBindParameter](#), [SQLBulkOperations](#), [SQLPutData](#), and [SQLSetPos](#).

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Closing the cursor	SQLCloseCursor
Executing a commit or rollback operation	SQLEndTran
Executing an SQL statement	SQLExecDirect

Fetching multiple rows of data	SQLFetch
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Freeing a statement handle	SQLFreeStmt
Returning a cursor name	SQLGetCursorName
Fetching part or all of a column of data	SQLGetData
Returning the next parameter to send data for	SQLParamData
Preparing a statement for execution	SQLPrepare
Sending parameter data at execution time	SQLPutData
Setting a cursor name	SQLSetCursorName
Setting a statement attribute	SQLSetStmtAttr

SQLExtendedFetch

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: Deprecated

Summary

SQLExtendedFetch fetches the specified rowset of data from the result set and returns data for all bound columns. Rowsets can be specified at an absolute or relative position or by bookmark.

Note In ODBC 3.x, **SQLExtendedFetch** has been replaced by **SQLFetchScroll**. ODBC 3.x applications should not call **SQLExtendedFetch**; instead they should call **SQLFetchScroll**. The Driver Manager maps **SQLFetchScroll** to **SQLExtendedFetch** when working with an ODBC 2.x driver. ODBC 3.x drivers should support **SQLExtendedFetch** if they want to work with ODBC 2.x applications that call it. For more information, see "Comments" and "Block Cursors, Scrollable Cursors, and Backward Compatibility" (Appendix G, "Driver Guidelines for Backward Compatibility") contained on the Microsoft Web site (ODBC Programming Reference).

Syntax

SQLRETURN SQLExtendedFetch(
SQLHSTMT	StatementHandle,
SQLUSMALLINT	FetchOrientation,
SQLINTEGER	FetchOffset,
SQLINTEGER *	RowCountPtr,
SQLUSMALLINT *	RowStatusArray);

Arguments

StatementHandle

[Input]

Statement handle.

FetchOrientation

[Input]

Type of fetch. This is the same as *FetchOrientation* in **SQLFetchScroll**.

FetchOffset

[Input]

Number of the row to fetch. This is the same as *FetchOffset* in **SQLFetchScroll**, with one exception. When *FetchOrientation* is `SQL_FETCH_BOOKMARK`, *FetchOffset* is a fixed-length bookmark, not an offset from a bookmark. In other words, **SQLExtendedFetch** retrieves the bookmark from this argument, not the `SQL_ATTR_FETCH_BOOKMARK_PTR` statement attribute. It does not support variable-length bookmarks and does not support fetching a rowset at an offset (other than 0) from a bookmark.

RowCountPtr

[Output]

Pointer to a buffer in which to return the number of rows actually fetched. This buffer is used in the same manner as the buffer specified by the `SQL_ATTR_ROWS_FETCHED_PTR` statement attribute. This buffer is used only by **SQLExtendedFetch**. It is not used by **SQLFetch** or **SQLFetchScroll**.

RowStatusArray

[Output]

Pointer to an array in which to return the status of each row. This array is used in the same manner as the array specified by the `SQL_ATTR_ROW_STATUS_PTR` statement attribute.

However, the address of this array is not stored in the `SQL_DESC_STATUS_ARRAY_PTR` field in the `IRD`. Furthermore, this array is used only by **SQLExtendedFetch** and by **SQLBulkOperations** with an *Operation* of `SQL_ADD` or **SQLSetPos** when it is called after **SQLExtendedFetch**. It is not used by **SQLFetch** or **SQLFetchScroll**, and it is not used by **SQLBulkOperations** or **SQLSetPos** when they are called after **SQLFetch** or **SQLFetchScroll**. It is also not used when **SQLBulkOperations** with an *Operation* of `SQL_ADD` is called before any fetch function is called. In other words, it is used only in statement state `S7`. It is not used in statement states `S5` or `S6`. For more information, see "Statement Transitions (Appendix B, "ODBC State Transition Tables.") on the Microsoft Web site (ODBC Programming Reference).

Applications should provide a valid pointer in the *RowStatusArray* argument; if not, the behavior of **SQLExtendedFetch** and the behavior of calls to **SQLBulkOperations** or **SQLSetPos** after a cursor has been positioned by **SQLExtendedFetch** are undefined.

Returns

`SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_NO_DATA`, `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 can 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 `SQLSTATE`s returned by the Driver Manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR`, unless noted otherwise. If an error occurs on

a single column, **SQLGetDiagField** can be called with a *DiagIdentifier* of **SQL_DIAG_COLUMN_NUMBER** to determine the column the error occurred on; and **SQLGetDiagField** can be called with a *DiagIdentifier* of **SQL_DIAG_ROW_NUMBER** to determine the row containing that column.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO .)
01004	String data, right truncated	String or binary data returned for a column resulted in the truncation of nonblank character or non-NULL binary data. If it was a string value, it was right-truncated. If it was a numeric value, the fractional part of the 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 .)
01S06	Attempt to fetch before the result set returned the first rowset	The requested rowset overlapped the start of the result set when the current position was beyond the first row, and either <i>FetchOrientation</i> was SQL_PRIOR or <i>FetchOrientation</i> was SQL_RELATIVE with a negative <i>FetchOffset</i> whose absolute value was less than or equal to the current SQL_ROWSET_SIZE . (Function returns SQL_SUCCESS_WITH_INFO .)
01S07	Fractional truncation	The data returned for a column was truncated. For numeric data types, the fractional part of the number was truncated. For time, timestamp, and interval data types containing a time component, the fractional portion of the time was truncated. (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 <i>TargetType</i> in SQLBindCol .
07009	Invalid descriptor index	Column 0 was bound with SQLBindCol , and the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF .
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.
22002	Indicator variable required but not supplied	NULL data was fetched into a column whose <i>StrLen_or_IndPtr</i> set by SQLBindCol was a null pointer. (Function returns SQL_SUCCESS_WITH_INFO .)
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. (Function returns SQL_SUCCESS_WITH_INFO .)

		For more information, see Appendix D, “Data Types” of the SOLID Programmer Guide .
22007	Invalid datetime format	A character column in the result set was bound to a date, time, or timestamp C structure, and a value in the column was, respectively, an invalid date, time, or timestamp. (Function returns SQL_SUCCESS_WITH_INFO.)
22012	Division by zero	A value from an arithmetic expression was returned, which resulted in division by zero. (Function returns SQL_SUCCESS_WITH_INFO.)
22015	Interval field overflow	Assigning from an exact numeric or interval SQL type to an interval C type caused a loss of significant digits in the leading field. When fetching data to an interval C type, there was no representation of the value of the SQL type in the interval C type. (Function returns SQL_SUCCESS_WITH_INFO.)
22018	Invalid character value for cast specification	The C type was an exact or approximate numeric, a datetime, or an interval data type; the SQL type of the column was a character data type; and the value in the column was not a valid literal of the bound C type. (Function returns SQL_SUCCESS_WITH_INFO.)
24000	Invalid cursor state	The <i>StatementHandle</i> was in an executed state, but no result set was associated with the <i>StatementHandle</i> .
HY000	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 <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	Asynchronous processing was enabled for the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> , and then the function was called again on the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.
HY010	Function sequence error	(DM) The specified <i>StatementHandle</i> was not in an executed state. The function was called without first calling SQLExecDirect , SQLExecute , or a catalog

		<p>function.</p> <p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p> <p>(DM) SQLExtendedFetch was called for the <i>StatementHandle</i> after SQLFetch or SQLFetchScroll was called and before SQLFreeStmt was called with the SQL_CLOSE option.</p> <p>(DM) SQLBulkOperations was called for a statement before SQLFetch, SQLFetchScroll, or SQLExtendedFetch was called, and then SQLExtendedFetch was called before SQLFreeStmt was called with the SQL_CLOSE option.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY106	Fetch type out of range	<p>(DM) The value specified for the argument <i>FetchOrientation</i> was invalid. (See "Comments.")</p> <p>The argument <i>FetchOrientation</i> was SQL_FETCH_BOOKMARK, and the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF.</p> <p>The value of the SQL_CURSOR_TYPE statement option was SQL_CURSOR_FORWARD_ONLY, and the value of argument <i>FetchOrientation</i> was not SQL_FETCH_NEXT.</p> <p>The argument <i>FetchOrientation</i> was SQL_FETCH_RESUME.</p>
HY107	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 attribute was greater than 0 and less than the value specified with the SQL_ROWSET_SIZE statement attribute.
HY111	Invalid bookmark value	The argument <i>FetchOrientation</i> was SQL_FETCH_BOOKMARK, and the bookmark specified in the <i>FetchOffset</i> argument was not valid.

HYC00	Optional feature not implemented	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 <i>TargetType</i> in SQLBindCol and the SQL data type of the corresponding column. This error applies only when the SQL data type of the column was mapped to a driver-specific SQL data type.
HYT00	Timeout expired	The query timeout period expired before the data source returned the result set. The timeout period is set through SQLSetStmtOption , SQL_QUERY_TIMEOUT .
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

The behavior of **SQLExtendedFetch** is identical to that of **SQLFetchScroll**, with the following exceptions:

- **SQLExtendedFetch** and **SQLFetchScroll** use different methods to return the number of rows fetched. **SQLExtendedFetch** returns the number of rows fetched in **RowCountPtr*; **SQLFetchScroll** returns the number of rows fetched directly to the buffer pointed to by **SQL_ATTR_ROWS_FETCHED_PTR**. For more information, see the *RowCountPtr* argument.
- **SQLExtendedFetch** and **SQLFetchScroll** return the status of each row in different arrays. For more information, see the *RowStatusArray* argument.
- **SQLExtendedFetch** and **SQLFetchScroll** use different methods to retrieve the bookmark when *FetchOrientation* is **SQL_FETCH_BOOKMARK**. **SQLExtendedFetch** does not support variable-length bookmarks or fetching rowsets at an offset other than 0 from a bookmark. For more information, see the *FetchOffset* argument.
- **SQLExtendedFetch** and **SQLFetchScroll** use different rowset sizes. **SQLExtendedFetch** uses the value of the **SQL_ROWSET_SIZE** statement attribute, and **SQLFetchScroll** uses the value of the **SQL_ATTR_ROW_ARRAY_SIZE** statement attribute.
- **SQLExtendedFetch** has slightly different error handling semantics than **SQLFetchScroll**. For more information, see "Error Handling" in the "Comments" section of [SQLFetchScroll](#).
- **SQLExtendedFetch** does not support binding offsets (the **SQL_ATTR_ROW_BIND_OFFSET_PTR** statement attribute).
- Calls to **SQLExtendedFetch** cannot be mixed with calls to **SQLFetch** or **SQLFetchScroll**, and if **SQLBulkOperations** is called before any fetch function is called, **SQLExtendedFetch** cannot be called until the cursor is closed and reopened. That is, **SQLExtendedFetch** can be called only in

statement state S7. For more information, see “Statement Transitions” (Appendix B, "ODBC State Transition Tables.") contained on the Microsoft Web site (ODBC Programming Reference).

- When an application calls **SQLFetchScroll** while using an ODBC 2.x driver, the Driver Manager maps this call to **SQLExtendedFetch**. For more information, see "SQLFetchScroll and ODBC 2.x Drivers" in [SQLFetchScroll](#).

In ODBC 2.x, **SQLExtendedFetch** was called to fetch multiple rows and **SQLFetch** was called to fetch a single row. In ODBC 3.x, on the other hand, **SQLFetch** can be called to fetch multiple rows.

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Performing bulk insert, update, or delete operations	SQLBulkOperations
Canceling statement processing	SQLCancel
Returning information about a column in a result set	SQLDescribeCol
Executing an SQL statement	SQLExecDirect
Executing a prepared SQL statement	SQLExecute
Returning the number of result set columns	SQLNumResultCols
Positioning the cursor, refreshing data in the rowset, or updating or deleting data in the result set	SQLSetPos
Setting a statement attribute	SQLSetStmtAttr

SQLFetch

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

SQLFetch fetches the next rowset of data from the result set and returns data for all bound columns.

Syntax

```
SQLRETURN SQLFetch(  
    SQLHSTMT StatementHandle);
```

Arguments

StatementHandle

[Input]
Statement handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, 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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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. If an error occurs on a single column, **SQLGetDiagField** can be called with a *DiagIdentifier* of SQL_DIAG_COLUMN_NUMBER to determine the column the error occurred on; and **SQLGetDiagField** can be called with a *DiagIdentifier* of SQL_DIAG_ROW_NUMBER to determine the row containing that column.

For all those SQLSTATEs that can return SQL_SUCCESS_WITH_INFO or SQL_ERROR (except 01xxx SQLSTATEs), SQL_SUCCESS_WITH_INFO is returned if an error occurs on one or more, but not all, rows of a multirow operation, and SQL_ERROR is returned if an error occurs on a single-row operation.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	String data, right truncated	String or binary data returned for a column resulted in the truncation of nonblank character or non-NULL binary data. If it was a string value, it was right-truncated.
01S01	Error in row	An error occurred while fetching one or more rows. (If this SQLSTATE is returned when an ODBC 3.x application is working with an ODBC 2.x driver, it can be ignored.)
01S07	Fractional truncation	The data returned for a column was truncated. For numeric data types, the fractional part of the number was truncated. For time, timestamp, and interval data types containing a time component, the fractional portion of the time was truncated. (Function returns SQL_SUCCESS_WITH_INFO.)
07006	Restricted data type attribute violation	The data value of a column in the result set could not be converted to the data type specified by <i>TargetType</i> in SQLBindCol . Column 0 was bound with a data type of SQL_C_BOOKMARK, and the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_VARIABLE. Column 0 was bound with a data type of SQL_C_VARBOOKMARK, and the SQL_ATTR_USE_BOOKMARKS statement attribute

		was not set to SQL_UB_VARIABLE.
07009	Invalid descriptor index	<p>The driver was an ODBC 2.x driver that does not support SQLExtendedFetch, and a column number specified in the binding for a column was 0.</p> <p>Column 0 was bound, and the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF.</p>
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 truncated	A variable-length bookmark returned for a column was truncated.
22002	Indicator variable required but not supplied	NULL data was fetched into a column whose <i>StrLen_or_IndPtr</i> set by SQLBindCol (or SQL_DESC_INDICATOR_PTR set by SQLSetDescField or SQLSetDescRec) was a null pointer.
22003	Numeric value out of range	<p>Returning the numeric value (as numeric or string) for one or more bound columns would have caused the whole (as opposed to fractional) part of the number to be truncated.</p> <p>For more information, see “Converting Data from SQL to C Data Types” in Appendix D, "Data Types" of the SOLID Programmer Guide.</p>
22007	Invalid datetime format	A character column in the result set was bound to a date, time, or timestamp C structure, and a value in the column was, respectively, an invalid date, time, or timestamp.
22012	Division by zero	A value from an arithmetic expression was returned, which resulted in division by zero.
22015	Interval field overflow	<p>Assigning from an exact numeric or interval SQL type to an interval C type caused a loss of significant digits in the leading field.</p> <p>When fetching data to an interval C type, there was no representation of the value of the SQL type in the interval C type.</p>
22018	Invalid character value for cast specification	<p>A character column in the result set was bound to a character C buffer, and the column contained a character for which there was no representation in the character set of the buffer.</p> <p>The C type was an exact or approximate numeric, a datetime, or an interval data type; the SQL type of the column was a character data type; and the value in the column was not a valid literal of the bound C type.</p>
24000	Invalid cursor state	The <i>StatementHandle</i> was in an executed state but no

		result set was associated with the <i>StatementHandle</i> .
40001	Serialization failure	The transaction in which the fetch was executed was terminated to prevent deadlock.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
HY000	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 SQLGetDiagRec in the * <i>MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>. Then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY010	Function sequence error	<p>(DM) The specified <i>StatementHandle</i> was not in an executed state. The function was called without first calling SQLExecDirect, SQLExecute or a catalog function.</p> <p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p> <p>(DM) SQLFetch was called for the <i>StatementHandle</i> after SQLExtendedFetch was called and before SQLFreeStmt with the SQL_CLOSE option was called.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	The SQL_ATTR_USE_BOOKMARK statement attribute was set to SQL_UB_VARIABLE, and column 0 was bound to a buffer whose length was not equal to the maximum length for the bookmark for this

		result set. (This length is available in the <code>SQL_DESC_OCTET_LENGTH</code> field of the IRD and can be obtained by calling SQLDescribeCol , SQLColAttribute , or SQLGetDescField .)
HY107	Row value out of range	The value specified with the <code>SQL_ATTR_CURSOR_TYPE</code> statement attribute was <code>SQL_CURSOR_KEYSET_DRIVEN</code> , but the value specified with the <code>SQL_ATTR_KEYSET_SIZE</code> statement attribute was greater than 0 and less than the value specified with the <code>SQL_ATTR_ROW_ARRAY_SIZE</code> statement attribute.
HYC00	Optional feature not implemented	The driver or data source does not support the conversion specified by the combination of the <i>TargetType</i> in SQLBindCol and the SQL data type of the corresponding column.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , <code>SQL_ATTR_CONNECTION_TIMEOUT</code> .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

SQLFetch returns the next rowset in the result set. It can be called only while a result set exists—that is, after a call that creates a result set and before the cursor over that result set is closed. If any columns are bound, it returns the data in those columns. If the application has specified a pointer to a row status array or a buffer in which to return the number of rows fetched, **SQLFetch** returns this information as well. Calls to **SQLFetch** can be mixed with calls to **SQLFetchScroll** but cannot be mixed with calls to **SQLExtendedFetch**. For more information, see the Part I PDF file, “Fetching a Row of Data” in Chapter 10, “Retrieving Results (Basic).”

If an ODBC 3.x application works with an ODBC 2.x driver, the Driver Manager maps **SQLFetch** calls to **SQLExtendedFetch** for an ODBC 2.x driver that supports **SQLExtendedFetch**. If the ODBC 2.x driver does not support **SQLExtendedFetch**, the Driver Manager maps **SQLFetch** calls to **SQLFetch** in the ODBC 2.x driver, which can fetch only a single row.

For more information, see “Block Cursors, Scrollable Cursors, and Backward Compatibility”(Appendix G, “Driver Guidelines for Backward Compatibility”) contained on the Microsoft Web site (ODBC Programmer’s Reference).

Positioning the Cursor

When the result set is created, the cursor is positioned before the start of the result set. **SQLFetch** fetches the next rowset. It is equivalent to calling **SQLFetchScroll** with *FetchOrientation* set to `SQL_FETCH_NEXT`. For more information about cursors, see the Part I PDF file, “Cursors” in Chapter 10, “Retrieving Results (Basic)” and “Block Cursors” in Chapter 11, “Retrieving Results (Advanced).”

The `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute specifies the number of rows in the rowset. If the rowset being fetched by **SQLFetch** overlaps the end of the result set, **SQLFetch** returns a partial rowset. That is, if $S + R - 1$ is greater than L , where S is the starting row of the rowset being fetched, R is the rowset size, and L is the last row in the result set, then only the first $L - S + 1$ rows of the rowset are valid. The remaining rows are empty and have a status of `SQL_ROW_NOROW`.

After **SQLFetch** returns, the current row is the first row of the rowset.

The rules listed in the following table describe cursor positioning after a call to **SQLFetch**, based on the conditions listed in the second table below.

Condition	First row of new rowset
Before start	1
$\text{CurrRowsetStart} \leq \text{LastResultRow} - \text{RowsetSize} [1]$	$\text{CurrRowsetStart} + \text{RowsetSize} [2]$
$\text{CurrRowsetStart} > \text{LastResultRow} - \text{RowsetSize} [1]$	After end
After end	After end

[1] If the rowset size is changed between fetches, this is the rowset size that was used with the previous fetch.

[2] If the rowset size is changed between fetches, this is the rowset size that was used with the new fetch.

Notation	Meaning
Before start	The block cursor is positioned before the start of the result set. If the first row of the new rowset is before the start of the result set, SQLFetch returns <code>SQL_NO_DATA</code> .
After end	The block cursor is positioned after the end of the result set. If the first row of the new rowset is after the end of the result set, SQLFetch returns <code>SQL_NO_DATA</code> .
<code>CurrRowsetStart</code>	The number of the first row in the current rowset.
<code>LastResultRow</code>	The number of the last row in the result set.
<code>RowsetSize</code>	The rowset size.

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 **SQLFetch** for different starting positions.

Current rowset	Return code	New rowset	# of rows fetched
Before start	<code>SQL_SUCCESS</code>	1 to 5	5
1 to 5	<code>SQL_SUCCESS</code>	6 to 10	5
52 to 56	<code>SQL_SUCCESS</code>	57 to 61	5
91 to 95	<code>SQL_SUCCESS</code>	96 to 100	5
93 to 97	<code>SQL_SUCCESS</code>	98 to 100. Rows 4 and 5 of the row status array are set to <code>SQL_ROW_NOROW</code> .	3
96 to 100	<code>SQL_NO_DATA</code>	None.	0

99 to 100	SQL_NO_DATA	None.	0
After end	SQL_NO_DATA	None.	0

Returning Data in Bound Columns

As **SQLFetch** returns each row, it places the data for each bound column in the buffer bound to that column. If no columns are bound, **SQLFetch** does not return any data but does move the block cursor forward. The data can still be retrieved with **SQLGetData**. If the cursor is a multirow cursor (that is, the `SQL_ATTR_ROW_ARRAY_SIZE` is greater than 1), **SQLGetData** can be called only if `SQL_GD_BLOCK` is returned when **SQLGetInfo** is called with an *InfoType* of `SQL_GETDATA_EXTENSIONS`. (For more information, see [SQLGetData](#).)

For each bound column in a row, **SQLFetch** does the following:

Sets the length/indicator buffer to `SQL_NULL_DATA` and proceeds to the next column if the data is NULL. If the data is NULL and no length/indicator buffer was bound, **SQLFetch** returns `SQLSTATE 22002` (Indicator variable required but not supplied) for the row and proceeds to the next row. For information about how to determine the address of the length/indicator buffer, see "Buffer Addresses" in [SQLBindCol](#).

If the data for the column is not NULL, **SQLFetch** proceeds to step 2.

If the `SQL_ATTR_MAX_LENGTH` statement attribute is set to a nonzero value and the column contains character or binary data, the data is truncated to `SQL_ATTR_MAX_LENGTH` bytes.

Note The `SQL_ATTR_MAX_LENGTH` statement attribute is intended to reduce network traffic. It is generally implemented by the data source, which truncates the data before returning it across the network. Drivers and data sources are not required to support it. Therefore, to guarantee that data is truncated to a particular size, an application should allocate a buffer of that size and specify the size in the *cbValueMax* argument in **SQLBindCol**.

Converts the data to the type specified by *TargetType* in **SQLBindCol**.

If the data was converted to a variable-length data type, such as character or binary, **SQLFetch** checks whether the length of the data exceeds the length of the data buffer. If the length of character data (including the null-termination character) exceeds the length of the data buffer, **SQLFetch** truncates the data to the length of the data buffer less the length of a null-termination character. It then null-terminates the data. If the length of binary data exceeds the length of the data buffer, **SQLFetch** truncates it to the length of the data buffer. The length of the data buffer is specified with *BufferLength* in **SQLBindCol**.

SQLFetch never truncates data converted to fixed-length data types; it always assumes that the length of the data buffer is the size of the data type.

Places the converted (and possibly truncated) data in the data buffer. For information about how to determine the address of the data buffer, see "Buffer Addresses" in [SQLBindCol](#).

Places the length of the data in the length/indicator buffer. If the indicator pointer and the length pointer were both set to the same buffer (as a call to **SQLBindCol** does), the length is written in the buffer for valid data and `SQL_NULL_DATA` is written in the buffer for NULL data. If no length/indicator buffer was bound, **SQLFetch** does not return the length.

For character or binary data, this is the length of the data after conversion and before truncation due to the data buffer being too small. If the driver cannot determine the length of the data after conversion, as is sometimes the case with long data, it sets the length to `SQL_NO_TOTAL`. If data was truncated due to the `SQL_ATTR_MAX_LENGTH` statement attribute, the value of this attribute—as opposed to the actual length—is placed in the length/indicator buffer. This is because this attribute is designed to truncate data on the server before conversion, so the driver has no way of figuring out what the actual length is.

For all other data types, this is the length of the data after conversion; that is, it is the size of the type to which the data was converted.

For information about how to determine the address of the length/indicator buffer, see "Buffer Addresses" in [SQLBindCol](#).

If the data is truncated during conversion without a loss of significant digits (for example, the real number 1.234 is truncated to the integer 1 when converted), **SQLFetch** returns `SQLSTATE 01S07` (Fractional truncation) and `SQL_SUCCESS_WITH_INFO`. If the data is truncated because the length of the data buffer is too small (for example, the string "abcdef" is placed in a 4-byte buffer), **SQLFetch** returns `SQLSTATE 01004` (Data truncated) and `SQL_SUCCESS_WITH_INFO`. If data is truncated due to the `SQL_ATTR_MAX_LENGTH` statement attribute, **SQLFetch** returns `SQL_SUCCESS` and does not return `SQLSTATE 01S07` (Fractional truncation) or `SQLSTATE 01004` (Data truncated). If data is truncated during conversion with a loss of significant digits (for example, if an `SQL_INTEGER` value greater than 100,000 were converted to an `SQL_C_TINYINT`), **SQLFetch** returns `SQLSTATE 22003` (Numeric value out of range) and `SQL_ERROR` (if the rowset size is 1) or `SQL_SUCCESS_WITH_INFO` (if the rowset size is greater than 1).

The contents of the bound data buffer and the length/indicator buffer are undefined if **SQLFetch** or **SQLFetchScroll** does not return `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`.

Row Status Array

The row status array is used to return the status of each row in the rowset. The address of this array is specified with the `SQL_ATTR_ROW_STATUS_PTR` statement attribute. The array is allocated by the application and must have as many elements as are specified by the `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute. Its values are set by **SQLFetch**, **SQLFetchScroll**, and **SQLBulkOperations** or **SQLSetPos** (except when they have been called after the cursor has been positioned by **SQLExtendedFetch**). If the value of the `SQL_ATTR_ROW_STATUS_PTR` statement attribute is a null pointer, these functions do not return the row status.

The contents of the row status array buffer are undefined if **SQLFetch** or **SQLFetchScroll** does not return `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`.

The following values are returned in the row status array.

Row status array value	Description
<code>SQL_ROW_SUCCESS</code>	The row was successfully fetched and has not changed since it was last fetched from this result set.
<code>SQL_ROW_SUCCESS_WITH_INFO</code>	The row was successfully fetched and has not changed since it was last fetched from this result set. However, a warning was returned about the row.
<code>SQL_ROW_ERROR</code>	An error occurred while fetching the row.
<code>SQL_ROW_UPDATED [1],[2], and [3]</code>	The row was successfully fetched and has changed since it was last fetched from this result set. If the row is fetched

	again from this result set or is refreshed by SQLSetPos , the status is changed to the row's new status.
SQL_ROW_DELETED [3]	The row has been deleted since it was last fetched from this result set.
SQL_ROW_ADDED [4]	The row was inserted by SQLBulkOperations . If the row is fetched again from this result set or is refreshed by SQLSetPos , its status is SQL_ROW_SUCCESS.
SQL_ROW_NOROW	The rowset overlapped the end of the result set, and no row was returned that corresponded to this element of the row status array.

[1] 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.

[2] Some drivers cannot detect updates to data and therefore cannot return this value. To determine whether a driver can detect updates to refetched rows, an application calls **SQLGetInfo** with the SQL_ROW_UPDATES option.

[3] **SQLFetch** can return this value only when it is intermixed with calls to **SQLFetchScroll**. The reason for this is that **SQLFetch** moves forward through the result set and when used exclusively, does not refetch any rows. Because no rows are refetched, **SQLFetch** does not detect changes made to previously fetched rows. However, if **SQLFetchScroll** positions the cursor before any previously fetched rows and **SQLFetch** is used to fetch those rows, **SQLFetch** can detect any changes to those rows.

[4] Returned by **SQLBulkOperations** only. Not set by **SQLFetch** or **SQLFetchScroll**.

Rows Fetched Buffer

The rows fetched buffer is used to return the number of rows fetched, including those rows for which no data was returned because an error occurred while they were being fetched. In other words, it is the number of rows for which the value in the row status array is not SQL_ROW_NOROW. The address of this buffer is specified with the SQL_ATTR_ROWS_FETCHED_PTR statement attribute. The buffer is allocated by the application. It is set by **SQLFetch** and **SQLFetchScroll**. If the value of the SQL_ATTR_ROWS_FETCHED_PTR statement attribute is a null pointer, these functions do not return the number of rows fetched. To determine the number of the current row in the result set, an application can call **SQLGetStmtAttr** with the SQL_ATTR_ROW_NUMBER attribute.

The contents of the rows fetched buffer are undefined if **SQLFetch** or **SQLFetchScroll** does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, except when SQL_NO_DATA is returned, in which case the value in the rows fetched buffer is set to 0.

Error Handling

Errors and warnings can apply to individual rows or to the entire function. For more information about diagnostic records, see the Part I PDF file, Chapter 15, "Overview of Diagnostics." and [SQLGetDiagRec](#).

Errors and Warnings on the Entire Function

If an error applies to the entire function, such as SQLSTATE HYT00 (Timeout expired) or SQLSTATE 24000 (Invalid cursor state), **SQLFetch** returns SQL_ERROR and the applicable SQLSTATE. The contents of the rowset buffers are undefined and the cursor position is unchanged.

If a warning applies to the entire function, **SQLFetch** returns `SQL_SUCCESS_WITH_INFO` and the applicable `SQLSTATE`. The status records for warnings that apply to the entire function are returned before the status records that apply to individual rows.

Errors and Warnings in Individual Rows

If an error (such as `SQLSTATE 22012` (Division by zero)) or a warning (such as `SQLSTATE 01004` (Data truncated)) applies to a single row, **SQLFetch** does the following:

Sets the corresponding element of the row status array to `SQL_ROW_ERROR` for errors or `SQL_ROW_SUCCESS_WITH_INFO` for warnings.

Adds zero or more status records containing `SQLSTATE`s for the error or warning.

Sets the row and column number fields in the status records. If **SQLFetch** cannot determine a row or column number, it sets that number to `SQL_ROW_NUMBER_UNKNOWN` or `SQL_COLUMN_NUMBER_UNKNOWN`, respectively. If the status record does not apply to a particular column, **SQLFetch** sets the column number to `SQL_NO_COLUMN_NUMBER`.

SQLFetch continues fetching rows until it has fetched all of the rows in the rowset. It returns `SQL_SUCCESS_WITH_INFO` unless an error occurs in every row of the rowset (not counting rows with status `SQL_ROW_NOROW`), in which case it returns `SQL_ERROR`. In particular, if the rowset size is 1 and an error occurs in that row, **SQLFetch** returns `SQL_ERROR`.

SQLFetch returns the status records in row number order. That is, it returns all status records for unknown rows (if any), then all status records for the first row (if any), then all status records for the second row (if any), and so on. The status records for each individual row are ordered according to the normal rules for ordering status records; for more information, see "Sequence of Status Records" in [SQLGetDiagField](#).

Descriptors and SQLFetch

The following sections describe how **SQLFetch** interacts with descriptors.

Argument Mappings

The driver does not set any descriptor fields based on the arguments of **SQLFetch**.

Other Descriptor Fields

The following descriptor fields are used by **SQLFetch**.

Descriptor field	Desc.	Field in	Set through
<code>SQL_DESC_ARRAY_SIZE</code>	ARD	header	<code>SQL_ATTR_ROW_ARRAY_SIZE</code> statement attribute
<code>SQL_DESC_ARRAY_STATUS_PTR</code>	IRD	header	<code>SQL_ATTR_ROW_STATUS_PTR</code> statement attribute
<code>SQL_DESC_BIND_OFFSET_PTR</code>	ARD	header	<code>SQL_ATTR_ROW_BIND_OFFSET_PTR</code> statement attribute
<code>SQL_DESC_BIND_TYPE</code>	ARD	header	<code>SQL_ATTR_ROW_BIND_TYPE</code> statement attribute
<code>SQL_DESC_COUNT</code>	ARD	header	<i>ColumnNumber</i> argument of SQLBindCol
<code>SQL_DESC_DATA_PTR</code>	ARD	records	<i>TargetValuePtr</i> argument of SQLBindCol

SQL_DESC_INDICATOR_PTR	ARD	records	<i>StrLen_or_IndPtr</i> argument in SQLBindCol
SQL_DESC_OCTET_LENGTH	ARD	records	<i>BufferLength</i> argument in SQLBindCol
SQL_DESC_OCTET_LENGTH_PTR	ARD	records	<i>StrLen_or_IndPtr</i> argument in SQLBindCol
SQL_DESC_ROWS_PROCESSED_PTR	IRD	header	SQL_ATTR_ROWS_FETCHED_PTR statement attribute
SQL_DESC_TYPE	ARD	records	<i>TargetType</i> argument in SQLBindCol

All descriptor fields can also be set through **SQLSetDescField**.

Separate Length and Indicator Buffers

Applications can bind a single buffer or two separate buffers to be used to hold length and indicator values. When an application calls **SQLBindCol**, the driver sets the SQL_DESC_OCTET_LENGTH_PTR and SQL_DESC_INDICATOR_PTR fields of the ARD to the same address, which is passed in the *StrLen_or_IndPtr* argument. When an application calls **SQLSetDescField** or **SQLSetDescRec**, it can set these two fields to different addresses.

SQLFetch determines whether the application has specified separate length and indicator buffers. In this case, when the data is not NULL, **SQLFetch** sets the indicator buffer to 0 and returns the length in the length buffer. When the data is NULL, **SQLFetch** sets the indicator buffer to SQL_NULL_DATA and does not modify the length buffer.

Code Example

See [SQLBindCol](#), [SQLColumns](#), [SQLGetData](#), and [SQLProcedures](#).

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Returning information about a column in a result set	SQLDescribeCol
Executing an SQL statement	SQLExecDirect
Executing a prepared SQL statement	SQLExecute
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Closing the cursor on the statement	SQLFreeStmt
Fetching part or all of a column of data	SQLGetData
Returning the number of result set columns	SQLNumResultCols
Preparing a statement for execution	SQLPrepare

SQLFetchScroll

Conformance

Version Introduced: ODBC 3.0
Standards Compliance: ISO 92

Summary

SQLFetchScroll fetches the specified rowset of data from the result set and returns data for all bound columns. Rowsets can be specified at an absolute or relative position or by bookmark.

When working with an ODBC 2.x driver, the Driver Manager maps this function to **SQLExtendedFetch**. For more information, see the Part I PDF file, "Mapping Replacement Functions for Backward Compatibility of Applications" in Chapter 17, "Programming Considerations."

Syntax

SQLRETURN SQLFetchScroll(
SQLHSTMT	StatementHandle,
SQLSMALLINT	FetchOrientation,
SQLINTEGER	FetchOffset);

Arguments

StatementHandle

[Input]
Statement handle.

FetchOrientation

[Input]
Type of fetch:

SQL_FETCH_NEXT
SQL_FETCH_PRIOR
SQL_FETCH_FIRST
SQL_FETCH_LAST
SQL_FETCH_ABSOLUTE
SQL_FETCH_RELATIVE
SQL_FETCH_BOOKMARK

For more information, see "Positioning the Cursor" in the "Comments" section.

FetchOffset

[Input]
Number of the row to fetch. The interpretation of this argument depends on the value of the *FetchOrientation* argument. For more information, see "Positioning the Cursor" in the "Comments" section.

Returns

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

Diagnostics

When **SQLFetchScroll** returns either SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. The following table lists the SQLSTATE values commonly returned by **SQLFetchScroll** 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. If an error occurs on a single column, **SQLGetDiagField** can be called with a *DiagIdentifier* of SQL_DIAG_COLUMN_NUMBER to determine the column the error occurred on; and **SQLGetDiagField** can be called with a *DiagIdentifier* of SQL_DIAG_ROW_NUMBER to determine the row containing that column.

For all those SQLSTATES that can return SQL_SUCCESS_WITH_INFO or SQL_ERROR (except 01xxx SQLSTATES), SQL_SUCCESS_WITH_INFO is returned if an error occurs on one or more, but not all, rows of a multirow operation, and SQL_ERROR is returned if an error occurs on a single-row operation.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	String data, right truncated	String or binary data returned for a column resulted in the truncation of nonblank character or non-NULL binary data. String values are right-truncated.
01S01	Error in row	An error occurred while fetching one or more rows. (This SQLSTATE is returned only by ODBC 2.x drivers.)
01S06	Attempt to fetch before the result set returned the first rowset	<p>The requested rowset overlapped the start of the result set when <i>FetchOrientation</i> was SQL_FETCH_PRIOR, the current position was beyond the first row, and the number of the current row is less than or equal to the rowset size.</p> <p>The requested rowset overlapped the start of the result set when <i>FetchOrientation</i> was SQL_FETCH_PRIOR, the current position was beyond the end of the result set, and the rowset size was greater than the result set size.</p> <p>The requested rowset overlapped the start of the result set when <i>FetchOrientation</i> was SQL_FETCH_RELATIVE, <i>FetchOffset</i> was negative, and the absolute value of <i>FetchOffset</i> was less than or equal to the rowset size.</p> <p>The requested rowset overlapped the start of the result set when <i>FetchOrientation</i> was SQL_FETCH_ABSOLUTE, <i>FetchOffset</i> was negative, and the absolute value of <i>FetchOffset</i> was greater than</p>

		<p>the result set size but less than or equal to the rowset size.</p> <p>(Function returns <code>SQL_SUCCESS_WITH_INFO</code>.)</p>
01S07	Fractional truncation	<p>The data returned for a column was truncated. For numeric data types, the fractional part of the number was truncated. For time, timestamp, and interval data types containing a time component, the fractional portion of the time was truncated.</p> <p>(Function returns <code>SQL_SUCCESS_WITH_INFO</code>.)</p>
07006	Restricted data type attribute violation	<p>A data value of a column in the result set could not be converted to the C data type specified by <i>TargetType</i> in SQLBindCol.</p> <p>Column 0 was bound with a data type of <code>SQL_C_BOOKMARK</code>, and the <code>SQL_ATTR_USE_BOOKMARKS</code> statement attribute was set to <code>SQL_UB_VARIABLE</code>.</p> <p>Column 0 was bound with a data type of <code>SQL_C_VARBOOKMARK</code>, and the <code>SQL_ATTR_USE_BOOKMARKS</code> statement attribute was not set to <code>SQL_UB_VARIABLE</code>.</p>
07009	Invalid descriptor index	<p>Column 0 was bound, and the <code>SQL_ATTR_USE_BOOKMARKS</code> statement attribute was set to <code>SQL_UB_OFF</code>.</p>
08S01	Communication link failure	<p>The communication link between the driver and the data source to which the driver was connected failed before the function completed processing.</p>
22001	String data, right truncation	<p>A variable-length bookmark returned for a row was truncated.</p>
22002	Indicator variable required but not supplied	<p>NULL data was fetched into a column whose <i>StrLen_or_IndPtr</i> set by SQLBindCol (or <code>SQL_DESC_INDICATOR_PTR</code> set by SQLSetDescField or SQLSetDescRec) was a null pointer.</p>
22003	Numeric value out of range	<p>Returning the numeric value (as numeric or string) for one or more bound columns would have caused the whole (as opposed to fractional) part of the number to be truncated.</p> <p>For more information, see Appendix D, "Data Types" in the SOLID Programmer Guide.</p>
22007	Invalid datetime format	<p>A character column in the result set was bound to a date, time, or timestamp C structure, and a value in the column was, respectively, an invalid date, time, or timestamp.</p>
22012	Division by zero	<p>A value from an arithmetic expression was returned,</p>

		which resulted in division by zero.
22015	Interval field overflow	<p>Assigning from an exact numeric or interval SQL type to an interval C type caused a loss of significant digits in the leading field.</p> <p>When fetching data to an interval C type, there was no representation of the value of the SQL type in the interval C type.</p>
22018	Invalid character value for cast specification	<p>A character column in the result set was bound to a character C buffer, and the column contained a character for which there was no representation in the character set of the buffer.</p> <p>The C type was an exact or approximate numeric, a datetime, or an interval data type; the SQL type of the column was a character data type; and the value in the column was not a valid literal of the bound C type.</p>
24000	Invalid cursor state	The <i>StatementHandle</i> was in an executed state, but no result set was associated with the <i>StatementHandle</i> .
40001	Serialization failure	The transaction in which the fetch was executed was terminated to prevent deadlock.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
HY000	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 SQLGetDiagRec in the * <i>MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>. Then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY010	Function sequence error	<p>(DM) The specified <i>StatementHandle</i> was not in an executed state. The function was called without first calling SQLExecDirect, SQLExecute, or a catalog function.</p> <p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p>

		<p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p> <p>(DM) SQLFetchScroll was called for a <i>StatementHandle</i> after SQLExtendedFetch was called and before SQLFreeStmt with SQL_CLOSE was called.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	The SQL_ATTR_USE_BOOKMARK statement attribute was set to SQL_UB_VARIABLE, and column 0 was bound to a buffer whose length was not equal to the maximum length for the bookmark for this result set. (This length is available in the SQL_DESC_OCTET_LENGTH field of the IRD and can be obtained by calling SQLDescribeCol , SQLColAttribute , or SQLGetDescField .)
HY106	Fetch type out of range	<p>(DM) The value specified for the argument <i>FetchOrientation</i> was invalid.</p> <p>(DM) The argument <i>FetchOrientation</i> was SQL_FETCH_BOOKMARK, and the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF.</p> <p>The value of the SQL_ATTR_CURSOR_TYPE statement attribute was SQL_CURSOR_FORWARD_ONLY, and the value of argument <i>FetchOrientation</i> was not SQL_FETCH_NEXT.</p> <p>The value of the SQL_ATTR_CURSOR_SCROLLABLE statement attribute was SQL_NONSCROLLABLE, and the value of argument <i>FetchOrientation</i> was not SQL_FETCH_NEXT.</p>
HY107	Row value out of range	The value specified with the SQL_ATTR_CURSOR_TYPE statement attribute was SQL_CURSOR_KEYSET_DRIVEN, but the value specified with the SQL_ATTR_KEYSET_SIZE statement attribute was greater than 0 and less than the value specified with the SQL_ATTR_ROW_ARRAY_SIZE statement attribute.
HY111	Invalid bookmark value	The argument <i>FetchOrientation</i> was

		SQL_FETCH_BOOKMARK, and the bookmark pointed to by the value in the SQL_ATTR_FETCH_BOOKMARK_PTR statement attribute was not valid or was a null pointer.
HYC00	Optional feature not implemented	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 <i>TargetType</i> in SQLBindCol and the SQL data type of the corresponding column. <i>FetchOrientation</i> was SQL_FETCH_BOOKMARK, <i>FetchOffset</i> was not equal to 0, and the underlying driver is an ODBC 2.x driver.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

SQLFetchScroll returns a specified rowset from the result set. Rowsets can be specified by absolute or relative position or by bookmark. **SQLFetchScroll** can be called only while a result set exists—that is, after a call that creates a result set and before the cursor over that result set is closed. If any columns are bound, it returns the data in those columns. If the application has specified a pointer to a row status array or a buffer in which to return the number of rows fetched, **SQLFetchScroll** returns this information as well. Calls to **SQLFetchScroll** can be mixed with calls to **SQLFetch** but cannot be mixed with calls to **SQLExtendedFetch**.

For more information, see the Part I PDF file, “Using Block Cursors” and “Using Scrollable Cursors” in Chapter 11, “Retrieving Results (Advanced).”

Positioning the Cursor

When the result set is created, the cursor is positioned before the start of the result set. **SQLFetchScroll** positions the block cursor based on the values of the *FetchOrientation* and *FetchOffset* arguments as shown in the following table. The exact rules for determining the start of the new rowset are shown in the next section.

FetchOrientation	Meaning
SQL_FETCH_NEXT	Return the next rowset. This is equivalent to calling SQLFetch . SQLFetchScroll ignores the value of <i>FetchOffset</i> .
SQL_FETCH_PRIOR	Return the prior rowset. SQLFetchScroll ignores the value of <i>FetchOffset</i> .
SQL_FETCH_RELATIVE	Return the rowset <i>FetchOffset</i> from the start of the current rowset.

SQL_FETCH_ABSOLUTE	Return the rowset starting at row <i>FetchOffset</i> .
SQL_FETCH_FIRST	Return the first rowset in the result set. SQLFetchScroll ignores the value of <i>FetchOffset</i> .
SQL_FETCH_LAST	Return the last complete rowset in the result set. SQLFetchScroll ignores the value of <i>FetchOffset</i> .
SQL_FETCH_BOOKMARK	Return the rowset <i>FetchOffset</i> rows from the bookmark specified by the SQL_ATTR_FETCH_BOOKMARK_PTR statement attribute.

Drivers are not required to support all fetch orientations; an application calls **SQLGetInfo** with an information type of SQL_DYNAMIC_CURSOR_ATTRIBUTES1, SQL_KEYSET_CURSOR_ATTRIBUTES1, or SQL_STATIC_CURSOR_ATTRIBUTES1 (depending on the type of the cursor) to determine which fetch orientations are supported by the driver. The application should look at the SQL_CA1_NEXT, SQL_CA1_RELATIVE, SQL_CA1_ABSOLUTE, and SQL_CA1_BOOKMARK bitmasks in these information types. Furthermore, if the cursor is forward-only and *FetchOrientation* is not SQL_FETCH_NEXT, **SQLFetchScroll** returns SQLSTATE HY106 (Fetch type out of range).

The SQL_ATTR_ROW_ARRAY_SIZE statement attribute specifies the number of rows in the rowset. If the rowset being fetched by **SQLFetchScroll** overlaps the end of the result set, **SQLFetchScroll** returns a partial rowset. That is, if $S + R - 1$ is greater than L , where S is the starting row of the rowset being fetched, R is the rowset size, and L is the last row in the result set, then only the first $L - S + 1$ rows of the rowset are valid. The remaining rows are empty and have a status of SQL_ROW_NOROW.

After **SQLFetchScroll** returns, the current row is the first row of the rowset.

Cursor Positioning Rules

The following sections describe the exact rules for each value of *FetchOrientation*. These rules use the following notation.

Notation	Meaning
Before start	The block cursor is positioned before the start of the result set. If the first row of the new rowset is before the start of the result set, SQLFetchScroll returns SQL_NO_DATA.
After end	The block cursor is positioned after the end of the result set. If the first row of the new rowset is after the end of the result set, SQLFetchScroll returns SQL_NO_DATA.
CurrRowsetStart	The number of the first row in the current rowset.
LastResultRow	The number of the last row in the result set.
RowsetSize	The rowset size.
FetchOffset	The value of the <i>FetchOffset</i> argument.
BookmarkRow	The row corresponding to the bookmark specified by the SQL_ATTR_FETCH_BOOKMARK_PTR statement attribute.

SQL_FETCH_NEXT

The following rules apply.

Condition	First row of new rowset
Before start	1
$\text{CurrRowsetStart} + \text{RowsetSize} [1] \leq \text{LastResultRow}$	$\text{CurrRowsetStart} + \text{RowsetSize} [1]$
$\text{CurrRowsetStart} + \text{RowsetSize} [1] > \text{LastResultRow}$	After end
After end	After end

[1] If the rowset size has been changed since the previous call to fetch rows, this is the rowset size that was used with the previous call.

SQL_FETCH_PRIOR

The following rules apply.

Condition	First row of new rowset
Before start	Before start
$\text{CurrRowsetStart} = 1$	Before start
$1 < \text{CurrRowsetStart} \leq \text{RowsetSize} [2]$	1[1]
$\text{CurrRowsetStart} > \text{RowsetSize} [2]$	$\text{CurrRowsetStart} - \text{RowsetSize} [2]$
After end AND $\text{LastResultRow} < \text{RowsetSize} [2]$	1[1]
After end AND $\text{LastResultRow} \geq \text{RowsetSize} [2]$	$\text{LastResultRow} - \text{RowsetSize} + 1[2]$

[1] **SQLFetchScroll** returns SQLSTATE 01S06 (Attempt to fetch before the result set returned the first rowset) and SQL_SUCCESS_WITH_INFO.

[2] If the rowset size has been changed since the previous call to fetch rows, this is the new rowset size.

SQL_FETCH_RELATIVE

The following rules apply.

Condition	First row of new rowset
(Before start AND $\text{FetchOffset} > 0$) OR (After end AND $\text{FetchOffset} < 0$)	-- [1]
BeforeStart AND $\text{FetchOffset} \leq 0$	Before start
$\text{CurrRowsetStart} = 1$ AND $\text{FetchOffset} < 0$	Before start
$\text{CurrRowsetStart} > 1$ AND $\text{CurrRowsetStart} + \text{FetchOffset} < 1$ AND $ \text{FetchOffset} > \text{RowsetSize} [3]$	Before start
$\text{CurrRowsetStart} > 1$ AND $\text{CurrRowsetStart} + \text{FetchOffset} < 1$ AND $ \text{FetchOffset} \leq \text{RowsetSize} [3]$	1 [2]

$1 \leq \text{CurrRowsetStart} + \text{FetchOffset} \leq \text{LastResultRow}$	$\text{CurrRowsetStart} + \text{FetchOffset}$
$\text{CurrRowsetStart} + \text{FetchOffset} > \text{LastResultRow}$	After end
After end AND $\text{FetchOffset} \geq 0$	After end

[1] **SQLFetchScroll** returns the same rowset as if it was called with *FetchOrientation* set to SQL_FETCH_ABSOLUTE. For more information, see the "SQL_FETCH_ABSOLUTE" section.

[2] **SQLFetchScroll** returns SQLSTATE 01S06 (Attempt to fetch before the result set returned the first rowset) and SQL_SUCCESS_WITH_INFO.

[3] If the rowset size has been changed since the previous call to fetch rows, this is the new rowset size.

SQL_FETCH_ABSOLUTE

The following rules apply.

Condition	First row of new rowset
$\text{FetchOffset} < 0$ AND $ \text{FetchOffset} \leq \text{LastResultRow}$	$\text{LastResultRow} + \text{FetchOffset} + 1$
$\text{FetchOffset} < 0$ AND $ \text{FetchOffset} > \text{LastResultRow}$ AND $ \text{FetchOffset} > \text{RowsetSize}$ [2]	Before start
$\text{FetchOffset} < 0$ AND $ \text{FetchOffset} > \text{LastResultRow}$ AND $ \text{FetchOffset} \leq \text{RowsetSize}$ [2]	1 [1]
$\text{FetchOffset} = 0$	Before start
$1 \leq \text{FetchOffset} \leq \text{LastResultRow}$	FetchOffset
$\text{FetchOffset} > \text{LastResultRow}$	After end

[1] **SQLFetchScroll** returns SQLSTATE 01S06 (Attempt to fetch before the result set returned the first rowset) and SQL_SUCCESS_WITH_INFO.

[2] If the rowset size has been changed since the previous call to fetch rows, this is the new rowset size.

An absolute fetch performed against a dynamic cursor cannot provide the required result because row positions in a dynamic cursor are undetermined. Such an operation is equivalent to a fetch first followed by a fetch relative; it is not an atomic operation, as is an absolute fetch on a static cursor.

SQL_FETCH_FIRST

The following rules apply.

Condition	First row of new rowset
Any	1

SQL_FETCH_LAST

The following rules apply.

Condition	First row of new rowset
RowsetSize [1] <= LastResultRow	LastResultRow – RowsetSize + 1[1]
RowsetSize [1] > LastResultRow	1

[1] If the rowset size has been changed since the previous call to fetch rows, this is the new rowset size.

SQL_FETCH_BOOKMARK

The following rules apply.

Condition	First row of new rowset
BookmarkRow + FetchOffset < 1	Before start
1 <= BookmarkRow + FetchOffset <= LastResultRow	BookmarkRow + FetchOffset
BookmarkRow + FetchOffset > LastResultRow	After end

For information about bookmarks, see the Part I PDF file, “Bookmarks” section in Chapter 11, “Retrieving Results (Advanced).”

Effect of Deleted, Added, and Error Rows on Cursor Movement

Static and keyset-driven cursors sometimes detect rows added to the result set and remove rows deleted from the result set. By calling **SQLGetInfo** with the **SQL_STATIC_CURSOR_ATTRIBUTES2** and **SQL_KEYSET_CURSOR_ATTRIBUTES2** options and looking at the **SQL_CA2_SENSITIVITY_ADDITIONS**, **SQL_CA2_SENSITIVITY_DELETIONS**, and **SQL_CA2_SENSITIVITY_UPDATES** bitmasks, an application determines whether the cursors implemented by a particular driver do this. For drivers that can detect deleted rows and remove them, the following paragraphs describe the effects of this behavior. For drivers that can detect deleted rows but cannot remove them, deletions have no effect on cursor movements, and the following paragraphs do not apply.

If the cursor detects rows added to the result set or removes rows deleted from the result set, it appears as if it detects these changes only when it fetches data. This includes the case when **SQLFetchScroll** is called with *FetchOrientation* set to **SQL_FETCH_RELATIVE** and *FetchOffset* set to 0 to refetch the same rowset, but does not include the case when **SQLSetPos** is called with *fOption* set to **SQL_REFRESH**. In the latter case, the data in the rowset buffers is refreshed, but not refetched, and deleted rows are not removed from the result set. Thus, when a row is deleted from or inserted into the current rowset, the cursor does not modify the rowset buffers. Instead, it detects the change when it fetches any rowset that previously included the deleted row or now includes the inserted row.

For example:

```
// Fetch the next rowset.
SQLFetchScroll(hstmt, SQL_FETCH_NEXT, 0);

// Delete third row of the rowset. Does not modify the rowset buffers.
SQLSetPos(hstmt, 3, SQL_DELETE, SQL_LOCK_NO_CHANGE);

// The third row has a status of SQL_ROW_DELETED after this call.
```

```
SQLSetPos(hstmt, 3, SQL_REFRESH, SQL_LOCK_NO_CHANGE);
```

```
// Refetch the same rowset. The third row is removed, replaced by what
// was previously the fourth row.
SQLFetchScroll(hstmt, SQL_FETCH_RELATIVE, 0);
```

When **SQLFetchScroll** returns a new rowset that has a position relative to the current rowset—that is, *FetchOrientation* is `SQL_FETCH_NEXT`, `SQL_FETCH_PRIOR`, or `SQL_FETCH_RELATIVE`—it does not include changes to the current rowset when calculating the starting position of the new rowset. However, it does include changes outside the current rowset if it is capable of detecting them. Furthermore, when **SQLFetchScroll** returns a new rowset that has a position independent of the current rowset—that is, *FetchOrientation* is `SQL_FETCH_FIRST`, `SQL_FETCH_LAST`, `SQL_FETCH_ABSOLUTE`, or `SQL_FETCH_BOOKMARK`—it includes all changes it is capable of detecting, even if they are in the current rowset.

When determining whether newly added rows are inside or outside the current rowset, a partial rowset is considered to end at the last valid row; that is, the last row for which the row status is not `SQL_ROW_NOROW`. For example, suppose the cursor is capable of detecting newly added rows, the current rowset is a partial rowset, the application adds new rows, and the cursor adds these rows to the end of the result set. If the application calls **SQLFetchScroll** with *FetchOrientation* set to `SQL_FETCH_NEXT`, **SQLFetchScroll** returns the rowset starting with the first newly added row.

For example, suppose the current rowset comprises rows 21 to 30, the rowset size is 10, the cursor removes rows deleted from the result set, and the cursor detects rows added to the result set. The following table shows the rows **SQLFetchScroll** returns in various situations.

Change	Fetch type	FetchOffset	New rowset [1]
Delete row 21	NEXT	0	31 to 40
Delete row 31	NEXT	0	32 to 41
Insert row between rows 21 and 22	NEXT	0	31 to 40
Insert row between rows 30 and 31	NEXT	0	Inserted row, 31 to 39
Delete row 21	PRIOR	0	11 to 20
Delete row 20	PRIOR	0	10 to 19
Insert row between rows 21 and 22	PRIOR	0	11 to 20
Insert row between rows 20 and 21	PRIOR	0	12 to 20, inserted row
Delete row 21	RELATIVE	0	22 to 31 [2]
Delete row 21	RELATIVE	1	22 to 31
Insert row between rows 21 and 22	RELATIVE	0	21, inserted row, 22 to 29
Insert row between rows 21 and 22	RELATIVE	1	22 to 31
Delete row 21	ABSOLUTE	21	22 to 31 [2]
Delete row 22	ABSOLUTE	21	21, 23 to 31
Insert row between rows 21 and 22	ABSOLUTE	22	Inserted row, 22 to 29

[1] This column uses the row numbers before any rows were inserted or deleted.

[2] In this case, the cursor attempts to return rows starting with row 21. Because row 21 has been deleted, the first row it returns is row 22.

Error rows (that is, rows with a status of `SQL_ROW_ERROR`) do not affect cursor movement. For example, if the current rowset starts with row 11 and the status of row 11 is `SQL_ROW_ERROR`, calling **SQLFetchScroll** with *FetchOrientation* set to `SQL_FETCH_RELATIVE` and *FetchOffset* set to 5 returns the rowset starting with row 16, just as it would if the status for row 11 was `SQL_SUCCESS`.

Returning Data in Bound Columns

SQLFetchScroll returns data in bound columns in the same way as **SQLFetch**. For more information, see "Returning Data in Bound Columns" in [SQLFetch](#).

If no columns are bound, **SQLFetchScroll** does not return data but does move the block cursor to the specified position. Whether data can be retrieved from unbound columns of a block cursor with **SQLGetData** depends on the driver. This capability is supported if a call to **SQLGetInfo** returns the `SQL_GD_BLOCK` bit for the `SQL_GETDATA_EXTENSIONS` information type.

Buffer Addresses

SQLFetchScroll uses the same formula to determine the address of data and length/indicator buffers as **SQLFetch**. For more information, see "Buffer Addresses" in [SQLBindCol](#).

Row Status Array

SQLFetchScroll sets values in the row status array in the same manner as **SQLFetch**. For more information, see "Row Status Array" in [SQLFetch](#).

Rows Fetched Buffer

SQLFetchScroll returns the number of rows fetched in the rows fetched buffer in the same manner as **SQLFetch**. For more information, see "Rows Fetched Buffer" in [SQLFetch](#).

Error Handling

When an application calls **SQLFetchScroll** in an ODBC 3.x driver, the Driver Manager calls **SQLFetchScroll** in the driver. When an application calls **SQLFetchScroll** in an ODBC 2.x driver, the Driver Manager calls **SQLExtendedFetch** in the driver. Because **SQLFetchScroll** and **SQLExtendedFetch** handle errors in a slightly different manner, the application sees slightly different error behavior when it calls **SQLFetchScroll** in ODBC 2.x and ODBC 3.x drivers.

SQLFetchScroll returns errors and warnings in the same manner as **SQLFetch**; for more information, see "Error Handling" in [SQLFetch](#). **SQLExtendedFetch** returns errors in the same manner as **SQLFetch**, with the following exceptions:

When a warning occurs that applies to a particular row in the rowset, **SQLExtendedFetch** sets the corresponding entry in the row status array to `SQL_ROW_SUCCESS`, not `SQL_ROW_SUCCESS_WITH_INFO`.

If errors occur in every row in the rowset, **SQLExtendedFetch** returns SQL_SUCCESS_WITH_INFO, not SQL_ERROR.

In each group of status records that applies to an individual row, the first status record returned by **SQLExtendedFetch** must contain SQLSTATE 01S01 (Error in row); **SQLFetchScroll** does not return this SQLSTATE. If **SQLExtendedFetch** is unable to return additional SQLSTATES, it still must return this SQLSTATE.

SQLFetchScroll and Optimistic Concurrency

If a cursor uses optimistic concurrency—that is, the SQL_ATTR_CONCURRENCY statement attribute has a value of SQL_CONCUR_VALUES or SQL_CONCUR_ROWVER—**SQLFetchScroll** updates the optimistic concurrency values used by the data source to detect whether a row has changed. This happens whenever **SQLFetchScroll** fetches a new rowset, including when it refetches the current rowset. (It is called with *FetchOrientation* set to SQL_FETCH_RELATIVE and *FetchOffset* set to 0.)

SQLFetchScroll and ODBC 2.x Drivers

When an application calls **SQLFetchScroll** in an ODBC 2.x driver, the Driver Manager maps this call to **SQLExtendedFetch**. It passes the following values for the arguments of **SQLExtendedFetch**.

SQLExtendedFetch argument	Value
StatementHandle	<i>StatementHandle</i> in SQLFetchScroll .
FetchOrientation	<i>FetchOrientation</i> in SQLFetchScroll .
FetchOffset	If <i>FetchOrientation</i> is not SQL_FETCH_BOOKMARK, the value of the <i>FetchOffset</i> argument in SQLFetchScroll is used. If <i>FetchOrientation</i> is SQL_FETCH_BOOKMARK, the value stored at the address specified by the SQL_ATTR_FETCH_BOOKMARK_PTR statement attribute is used.
RowCountPtr	The address specified by the SQL_ATTR_ROWS_FETCHED_PTR statement attribute.
RowStatusArray	The address specified by the SQL_ATTR_ROW_STATUS_PTR statement attribute.

For more information, see “Block Cursors, Scrollable Cursors, and Backward Compatibility” (Appendix G, “Driver Guidelines for Backward Compatibility”) contained on the Microsoft Web site (ODBC Programming Reference).

Descriptors and SQLFetchScroll

SQLFetchScroll interacts with descriptors in the same manner as **SQLFetch**. For more information, see the “Descriptors and SQLFetchScroll” section in [SQLFetch](#).

Code Example

See the Part I PDF file, “Column-Wise Binding” and “Row-Wise Binding” in Chapter 11, “Retrieving Results (Advanced)”, and “Positioned Update and Delete Statements” and “Updating Rows in the Rowset with SQLSetPos” in Chapter 12, “Updating Data.”

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Performing bulk insert, update, or delete operations	SQLBulkOperations
Canceling statement processing	SQLCancel
Returning information about a column in a result set	SQLDescribeCol
Executing an SQL statement	SQLExecDirect
Executing a prepared SQL statement	SQLExecute
Fetching a single row or a block of data in a forward-only direction	SQLFetch
Closing the cursor on the statement	SQLFreeStmt
Returning the number of result set columns	SQLNumResultCols
Positioning the cursor, refreshing data in the rowset, or updating or deleting data in the result set	SQLSetPos
Setting a statement attribute	SQLSetStmtAttr

SQLForeignKeys

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ODBC

Summary

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 statement.

Syntax

SQLRETURN SQLForeignKeys(
SQLHSTMT	StatementHandle,
SQLCHAR *	PKCatalogName,
SQLSMALLINT	NameLength1,
SQLCHAR *	PKSchemaName,
SQLSMALLINT	NameLength2,
SQLCHAR *	PKTableName,
SQLSMALLINT	NameLength3,

SQLCHAR *	FKCatalogName,
SQLSMALLINT	NameLength4,
SQLCHAR *	FKSchemaName,
SQLSMALLINT	NameLength5,
SQLCHAR *	FKTableName,
SQLSMALLINT	NameLength6);

Arguments

StatementHandle

[Input]

Statement handle.

PKCatalogName

[Input]

Primary key table catalog name. If a driver supports catalogs 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 catalogs. *PKCatalogName* cannot contain a string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *PKCatalogName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *PKCatalogName* is an ordinary argument; it is treated literally, and its case is significant. For more information, see the Part I PDF file, "Arguments in Catalog Functions" in Chapter 7, "Catalog Functions."

NameLength1

[Input]

Length of **PKCatalogName*, in bytes.

PKSchemaName

[Input]

Primary key table schema name. If a driver supports schemas 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 schemas. *PKSchemaName* cannot contain a string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *PKSchemaName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *PKSchemaName* is an ordinary argument; it is treated literally, and its case is significant.

NameLength2

[Input]

Length of **PKSchemaName*, in bytes.

PKTableName

[Input]

Primary key table name. *PKTableName* cannot contain a string search pattern.

If the `SQL_ATTR_METADATA_ID` statement attribute is set to `SQL_TRUE`, *PKTableName* is treated as an identifier and its case is not significant. If it is `SQL_FALSE`, *PKTableName* is an ordinary argument; it is treated literally, and its case is significant.

NameLength3

[Input]

Length of **PKTableName*.

FKCatalogName

[Input]

Foreign key table catalog name. If a driver supports catalogs 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 catalogs. *FKCatalogName* cannot contain a string search pattern.

If the `SQL_ATTR_METADATA_ID` statement attribute is set to `SQL_TRUE`, *FKCatalogName* is treated as an identifier and its case is not significant. If it is `SQL_FALSE`, *FKCatalogName* is an ordinary argument; it is treated literally, and its case is significant.

NameLength4

[Input]

Length of **FKCatalogName*.

FKSchemaName

[Input]

Foreign key table schema name. If a driver supports schemas 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 schemas. *FKSchemaName* cannot contain a string search pattern.

If the `SQL_ATTR_METADATA_ID` statement attribute is set to `SQL_TRUE`, *FKSchemaName* is treated as an identifier and its case is not significant. If it is `SQL_FALSE`, *FKSchemaName* is an ordinary argument; it is treated literally, and its case is significant.

NameLength5

[Input]

Length of **FKSchemaName*.

FKTableName

[Input]

Foreign key table name. *FKTableName* cannot contain a string search pattern.

If the `SQL_ATTR_METADATA_ID` statement attribute is set to `SQL_TRUE`, *FKTableName* is treated as an identifier and its case is not significant. If it is `SQL_FALSE`, *FKTableName* is an ordinary argument; it is treated literally, and its case is significant.

NameLength6

[Input]

Length of **FKTableName*.

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 **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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	<p>A cursor was open on the <i>StatementHandle</i>, and SQLFetch or SQLFetchScroll had been called. This error is returned by the Driver Manager if SQLFetch or SQLFetchScroll has not returned SQL_NO_DATA, and is returned by the driver if SQLFetch or SQLFetchScroll has returned SQL_NO_DATA.</p> <p>A cursor was open on the <i>StatementHandle</i>, but SQLFetch or SQLFetchScroll had not been called.</p>
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>, and then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a</p>

		multithread application.
HY009	Invalid use of null pointer	<p>(DM) The arguments <i>PKTableName</i> and <i>FKTableName</i> were both null pointers.</p> <p>(DM) The <i>SQL_ATTR_METADATA_ID</i> statement attribute was set to <i>SQL_TRUE</i>, the <i>FKCatalogName</i> or <i>PKCatalogName</i> argument was a null pointer, and the <i>SQL_CATALOG_NAME InfoType</i> returns that catalog names are supported.</p> <p>(DM) The <i>SQL_ATTR_METADATA_ID</i> statement attribute was set to <i>SQL_TRUE</i>, and the <i>FKSchemaName</i>, <i>PKSchemaName</i>, <i>FKTableName</i>, or <i>PKTableName</i> argument was a null pointer.</p>
HY010	Function sequence error	<p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned <i>SQL_NEED_DATA</i>. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	(DM) The value of one of the name length arguments was less than 0 but not equal to <i>SQL_NTS</i> .
		The value of one of the name length arguments exceeded the maximum length value for the corresponding name. (See "Comments.")
HYC00	Optional feature not implemented	<p>A catalog name was specified, and the driver or data source does not support catalogs.</p> <p>A schema name was specified, and the driver or data source does not support schemas.</p>
		<p>The combination of the current settings of the <i>SQL_ATTR_CONCURRENCY</i> and <i>SQL_ATTR_CURSOR_TYPE</i> statement attributes was not supported by the driver or data source.</p> <p>The <i>SQL_ATTR_USE_BOOKMARKS</i> statement attribute was set to <i>SQL_UB_VARIABLE</i>, and the <i>SQL_ATTR_CURSOR_TYPE</i> statement attribute was set to a cursor type for which the driver does not support bookmarks.</p>
HYT00	Timeout expired	The query timeout period expired before the data source returned the result set. The timeout period is set through SQLSetStmtAttr ,

		SQL_ATTR_QUERY_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

For information about how the information returned by this function might be used, see the Part I PDF file, “Uses of Catalog Data” in Chapter 7, “Catalog Functions.”

If **PKTableName* 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. The list of foreign keys in other tables does not include foreign keys that point to unique constraints in the specified table.

If **FKTableName* contains a table name, **SQLForeignKeys** returns a result set containing all of the foreign keys in the specified table that point to primary keys in others tables, and the primary keys in the other tables to which they refer. The list of foreign keys in the specified table does not contain foreign keys that refer to unique constraints in other tables.

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

Note For more information about the general use, arguments, and returned data of ODBC catalog functions, see the Part I PDF file, Chapter 7, “Catalog Functions.”

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_CAT, FKTABLE_SCHEM, FKTABLE_NAME, and KEY_SEQ. If the primary keys associated with a foreign key are requested, the result set is ordered by PKTABLE_CAT, PKTABLE_SCHEM, PKTABLE_NAME, and KEY_SEQ. The following table lists the columns in the result set.

The lengths of VARCHAR columns are not shown in the table; the actual lengths depend on the data source. To determine the actual lengths of the PKTABLE_CAT or FKTABLE_CAT, PKTABLE_SCHEM or FKTABLE_SCHEM, PKTABLE_NAME or FKTABLE_NAME, and PKCOLUMN_NAME or FKCOLUMN_NAME columns, an application can call **SQLGetInfo** with the SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN options.

The following columns have been renamed for ODBC 3.x. The column name changes do not affect backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
PKTABLE_QUALIFIER	PKTABLE_CAT
PKTABLE_OWNER	PKTABLE_SCHEM
FKTABLE_QUALIFIER	FK_TABLE_CAT
FKTABLE_OWNER	FKTABLE_SCHEM

The following table lists the columns in the result set. Additional columns beyond column 14 (REMARKS) can be defined by the driver. An application should gain access to driver-specific columns by counting down from the end of the result set rather than specifying an explicit ordinal position. For more information, see the Part I PDF file, "Data Returned by Catalog Functions" in Chapter 7, "Catalog Functions."

Column name	Column number	Data type	Comments
PKTABLE_CAT (ODBC 1.0)	1	Varchar	Primary key table catalog name; NULL if not applicable to the data source. If a driver supports catalogs 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 catalogs.
PKTABLE_SCHEM (ODBC 1.0)	2	Varchar	Primary key table schema name; NULL if not applicable to the data source. If a driver supports schemas 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 schemas.
PKTABLE_NAME (ODBC 1.0)	3	Varchar not NULL	Primary key table name.
PKCOLUMN_NAME (ODBC 1.0)	4	Varchar not NULL	Primary key column name. The driver returns an empty string for a column that does not have a name.
FKTABLE_CAT (ODBC 1.0)	5	Varchar	Foreign key table catalog name; NULL if not applicable to the data source. If a driver supports catalogs 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 catalogs.
FKTABLE_SCHEM (ODBC 1.0)	6	Varchar	Foreign key table schema name; NULL if not applicable to the data source. If a driver supports schemas 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 schemas.
FKTABLE_NAME (ODBC 1.0)	7	Varchar not NULL	Foreign key table name.
FKCOLUMN_NAME (ODBC 1.0)	8	Varchar not NULL	Foreign key column name. The driver returns an empty string for a column that does not have a name.
KEY_SEQ (ODBC 1.0)	9	Smallint not NULL	Column sequence number in key (starting with 1).

UPDATE_RULE (ODBC 1.0)	10	Smallint	<p>Action to be applied to the foreign key when the SQL operation is UPDATE. Can have one of the following values. (The referenced table is the table that has the primary key; the referencing table is the table that has the foreign key.)</p> <p>SQL_CASCADE: When the primary key of the referenced table is updated, the foreign key of the referencing table is also updated.</p> <p>SQL_NO_ACTION: If an update of the primary key of the referenced table would cause a "dangling reference" in the referencing table (that is, rows in the referencing table would have no counterparts in the referenced table), then the update is rejected. If an update of the foreign key of the referencing table would introduce a value that does not exist as a value of the primary key of the referenced table, then the update is rejected. (This action is the same as the SQL_RESTRICT action in ODBC 2.x.)</p> <p>SQL_SET_NULL: When one or more rows in the referenced table are updated such that one or more components of the primary key are changed, the components of the foreign key in the referencing table that correspond to the changed components of the primary key are set to NULL in all matching rows of the referencing table.</p> <p>SQL_SET_DEFAULT: When one or more rows in the referenced table are updated such that one or more components of the primary key are changed, the components of the foreign key in the referencing table that correspond to the changed components of the primary key are set to the applicable default values in all matching rows of the referencing table.</p> <p>NULL if not applicable to the data source.</p>
DELETE_RULE (ODBC 1.0)	11	Smallint	<p>Action to be applied to the foreign key when the SQL operation is DELETE. Can have one of the following values.</p>

			<p>(The referenced table is the table that has the primary key; the referencing table is the table that has the foreign key.)</p> <p>SQL_CASCADE: When a row in the referenced table is deleted, all the matching rows in the referencing tables are also deleted.</p> <p>SQL_NO_ACTION: If a delete of a row in the referenced table would cause a "dangling reference" in the referencing table (that is, rows in the referencing table would have no counterparts in the referenced table), then the update is rejected. (This action is the same as the SQL_RESTRICT action in ODBC 2.x.)</p> <p>SQL_SET_NULL: When one or more rows in the referenced table are deleted, each component of the foreign key of the referencing table is set to NULL in all matching rows of the referencing table.</p> <p>SQL_SET_DEFAULT: When one or more rows in the referenced table are deleted, each component of the foreign key of the referencing table is set to the applicable default in all matching rows of the referencing table.</p> <p>NULL if not applicable to the data source.</p>
FK_NAME (ODBC 2.0)	12	Varchar	Foreign key name. NULL if not applicable to the data source.
PK_NAME (ODBC 2.0)	13	Varchar	Primary key name. NULL if not applicable to the data source.
DEFERRABILITY (ODBC 3.0)	14	Smallint	SQL_INITIALLY_DEFERRED, SQL_INITIALLY_IMMEDIATE, SQL_NOT_DEFERRABLE.

Code Example

As illustrated in the following table, this example uses three tables, named ORDERS, LINES, and CUSTOMERS.

ORDERS	LINES	CUSTOMERS
ORDERID	ORDERID	CUSTID
CUSTID	LINES	NAME
OPENDATE	PARTID	ADDRESS

SALESPERSON	QUANTITY	PHONE
STATUS		

In the ORDERS table, CUSTID identifies the customer to whom the sale has been made. It is a foreign key that refers to CUSTID in the CUSTOMERS table.

In the LINES table, ORDERID identifies the sales order with which the line item is associated. It is a foreign key that refers to ORDERID in the ORDERS table.

This example calls **SQLPrimaryKeys** to get the primary key of the ORDERS table. The result set will have one row; the significant columns are shown in the following table.

TABLE_NAME	COLUMN_NAME	KEY_SEQ
ORDERS	ORDERID	1

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

PKTABLE_NAME	PKCOLUMN_NAME	FKTABLE_NAME	FKCOLUMN_NAME	KEY_SEQ
ORDERS	CUSTID	LINES	CUSTID	1

Finally, the example calls **SQLForeignKeys** to get the foreign keys in the ORDERS table that refer to the primary keys of other tables. The result set will have one row; the significant columns are shown in the following table.

PKTABLE_NAME	PKCOLUMN_NAME	FKTABLE_NAME	FKCOLUMN_NAME	KEY_SEQ
CUSTOMERS	CUSTID	ORDERS	CUSTID	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 */

SQLHSTMT hstmt;
SQLINTEGER cbPkTable, cbPkCol, cbFkTable, cbFkCol, cbKeySeq;
SQLSMALLINT iKeySeq;
SQLRETURN retcode;

/* Bind the columns that describe the primary and foreign keys. */
/* Ignore the table schema, name, and catalog 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, "ORDERS");

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

retcode = SQLPrimaryKeys(hstmt,
    NULL, 0, /* Catalog name */
    NULL, 0, /* Schema name */
    szTable, SQL_NTS); /* Table name */

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 ORDERS table. */

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

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

SQLFreeStmt(hstmt, SQL_CLOSE);

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

retcode = SQLForeignKeys(hstmt,
    NULL, 0, /* Primary catalog */
    NULL, 0, /* Primary schema */
    szTable, SQL_NTS, /* Primary table */
    NULL, 0, /* Foreign catalog */
    NULL, 0, /* Foreign schema */
    NULL, 0); /* Foreign table */

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 ORDERS */
    /* 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 ORDERS table. */

retcode = SQLForeignKeys(hstmt,

```

```

    NULL, 0,    /* Primary catalog */
    NULL, 0,    /* Primary schema */
    NULL, 0,    /* Primary table */
    NULL, 0,    /* Foreign catalog */
    NULL, 0,    /* Foreign schema */
    szTable, SQL_NTS); /* Foreign table */

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 ORDERS 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

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Fetching a single row or a block of data in a forward-only direction	SQLFetch
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Returning the columns of a primary key	SQLPrimaryKeys
Returning table statistics and indexes	SQLStatistics

SQLFreeConnect

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: Deprecated

Summary

In ODBC 3.x, the ODBC 2.0 function **SQLFreeConnect** has been replaced by **SQLFreeHandle**. For more information, see **SQLFreeHandle**.

Note For more information about what the Driver Manager maps this function to when an ODBC 2.x application is working with an ODBC 3.x driver, see “Mapping Deprecated Functions” (Appendix G, “Driver Guidelines for Backward Compatibility”) contained on the Microsoft Web site (ODBC Programming Reference).

SQLFreeEnv

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: Deprecated

Summary

In ODBC 3.x, the ODBC 2.0 function **SQLFreeEnv** has been replaced by **SQLFreeHandle**. For more information, see **SQLFreeHandle**.

Note For more information about what the Driver Manager maps this function to when an ODBC 2.x application is working with an ODBC 3.x driver, see “Mapping Deprecated Functions” (Appendix G, "Driver Guidelines for Backward Compatibility) contained on the Microsoft Web site (ODBC Programming Reference).

SQLFreeHandle

Conformance

Version Introduced: ODBC 3.0

Standards Compliance: ISO 92

Summary

SQLFreeHandle frees resources associated with a specific environment, connection, statement, or descriptor handle.

Note This function is a generic function for freeing handles. It replaces the ODBC 2.0 functions **SQLFreeConnect** (for freeing a connection handle) and **SQLFreeEnv** (for freeing an environment handle). **SQLFreeConnect** and **SQLFreeEnv** are both deprecated in ODBC 3.x. **SQLFreeHandle** also replaces the ODBC 2.0 function **SQLFreeStmt** (with the *SQL_DROP Option*) for freeing a statement handle. For more information, see "Comments." For more information about what the Driver Manager maps this function to when an ODBC 3.x application is working with an ODBC 2.x driver, see the Part I PDF file, "Mapping Replacement Functions for Backward Compatibility of Applications" in Chapter 17, "Programming Considerations."

Syntax

SQLRETURN SQLFreeHandle(
SQLSMALLINT	HandleType,
SQLHANDLE	Handle);

Arguments

HandleType

[Input]

The type of handle to be freed by **SQLFreeHandle**. Must be one of the following values:

SQL_HANDLE_ENV
SQL_HANDLE_DBC
SQL_HANDLE_STMT
SQL_HANDLE_DESC

If *HandleType* is not one of these values, **SQLFreeHandle** returns SQL_INVALID_HANDLE.

Handle

[Input]

The handle to be freed.

Returns

SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

If **SQLFreeHandle** returns SQL_ERROR, the handle is still valid.

Diagnostics

When **SQLFreeHandle** returns `SQL_ERROR`, an associated `SQLSTATE` value may be obtained from the diagnostic data structure for the handle that **SQLFreeHandle** attempted to free but could not. The following table lists the `SQLSTATE` values commonly returned by **SQLFreeHandle** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of `SQLSTATE`s returned by the Driver Manager. The return code associated with each `SQLSTATE` value is `SQL_ERROR`, unless noted otherwise.

SQLSTATE	Error	Description
HY000	General error	An error occurred for which there was no specific <code>SQLSTATE</code> and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by <code>SQLGetDiagRec</code> in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error	(DM) The <i>HandleType</i> argument was <code>SQL_HANDLE_ENV</code> , and at least one connection was in an allocated or connected state. <code>SQLDisconnect</code> and <code>SQLFreeHandle</code> with a <i>HandleType</i> of <code>SQL_HANDLE_DBC</code> must be called for each connection before calling <code>SQLFreeHandle</code> with a <i>HandleType</i> of <code>SQL_HANDLE_ENV</code> . (DM) The <i>HandleType</i> argument was <code>SQL_HANDLE_DBC</code> , and the function was called before calling <code>SQLDisconnect</code> for the connection. (DM) The <i>HandleType</i> argument was <code>SQL_HANDLE_STMT</code> ; an asynchronously executing function was called on the statement handle; and the function was still executing when this function was called. (DM) The <i>HandleType</i> argument was <code>SQL_HANDLE_STMT</code> ; <code>SQLExecute</code> , <code>SQLExecDirect</code> , <code>SQLBulkOperations</code> , or <code>SQLSetPos</code> was called with the statement handle and returned <code>SQL_NEED_DATA</code> . This function was called before data was sent for all data-at-execution parameters or columns. (DM) All subsidiary handles and other resources were not released before <code>SQLFreeHandle</code> was called.
HY013	Memory management error	The <i>HandleType</i> argument was <code>SQL_HANDLE_STMT</code> or <code>SQL_HANDLE_DESC</code> , and the function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY017	Invalid use of an automatically allocated descriptor handle.	(DM) The <i>Handle</i> argument was set to the handle for an automatically allocated descriptor.
HYT01	Connection timeout	The connection timeout period expired before the data

	expired	source responded to the request. The connection timeout period is set through <code>SQLSetConnectAttr</code> , <code>SQL_ATTR_CONNECTION_TIMEOUT</code> .
IM001	Driver does not support this function	(DM) The <i>HandleType</i> argument was <code>SQL_HANDLE_DESC</code> , and the driver was an ODBC 2.x driver. (DM) The <i>HandleType</i> argument was <code>SQL_HANDLE_STMT</code> , and the driver was not a valid ODBC driver.

Comments

SQLFreeHandle is used to free handles for environments, connections, statements, and descriptors, as described in the following sections. For general information about handles, see the Part I PDF file, “Handles” in Chapter 4, “ODBC Fundamentals.”

An application should not use a handle after it has been freed; the Driver Manager does not check the validity of a handle in a function call.

Freeing an Environment Handle

Prior to calling **SQLFreeHandle** with a *HandleType* of `SQL_HANDLE_ENV`, an application must call **SQLFreeHandle** with a *HandleType* of `SQL_HANDLE_DBC` for all connections allocated under the environment. Otherwise, the call to **SQLFreeHandle** returns `SQL_ERROR` and the environment and any active connection remains valid. For more information, see the Part I PDF file, “Environment Handles” in Chapter 4, “ODBC Fundamentals” and “Allocating the Environment Handle” in Chapter 6, “Connecting to a Data Source or Driver.”

When the Driver Manager processes the call to **SQLFreeHandle** with a *HandleType* of `SQL_HANDLE_ENV`, it checks the **TraceAutoStop** keyword in the [ODBC] section of the ODBC subkey of the system information. 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 subkey of the system information to 0. For information on what happens to tracing when **SQLFreeHandle** is called, see the Part I PDF file, “ODBC Subkey” in Chapter 19, “Configuring Data Sources.”

If the environment is a shared environment, the application that calls **SQLFreeHandle** with a *HandleType* of `SQL_HANDLE_ENV` no longer has access to the environment after the call, but the environment's resources are not necessarily freed. The call to **SQLFreeHandle** decrements the environment's reference count, which is maintained by the Driver Manager. If the reference count does not reach zero, the shared environment is not freed, because it is still being used by another component. If the reference count reaches zero, the shared environment's resources are freed.

Freeing a Connection Handle

Prior to calling **SQLFreeHandle** with a *HandleType* of `SQL_HANDLE_DBC`, an application must call **SQLDisconnect** for the connection if there is a connection on this handle. Otherwise, the call to **SQLFreeHandle** returns `SQL_ERROR` and the connection remains valid.

For more information, see the Part I PDF file, “Connection Handles” in Chapter 4, “ODBC Fundamentals” and “Disconnecting from a Data Source or Driver” in Chapter 6, “Connecting to a Data Source or Driver.”

Freeing a Statement Handle

A call to **SQLFreeHandle** with a *HandleType* of `SQL_HANDLE_STMT` frees all resources that were allocated by a call to **SQLAllocHandle** with a *HandleType* of `SQL_HANDLE_STMT`. When an application calls **SQLFreeHandle** to free a statement that has pending results, the pending results are deleted. When an application frees a statement handle, the driver frees the four automatically allocated descriptors associated with that handle. For more information, see the Part I PDF file, “Statement Handles” in Chapter 4, “ODBC Fundamentals” and “Freeing a Statement Handle” in Chapter 9, “Executing Statements.”

Note that **SQLDisconnect** automatically drops any statements and descriptors open on the connection.

Freeing a Descriptor Handle

A call to **SQLFreeHandle** with a *HandleType* of `SQL_HANDLE_DESC` frees the descriptor handle in *Handle*. The call to **SQLFreeHandle** does not release any memory allocated by the application that may be referenced by a pointer field (including `SQL_DESC_DATA_PTR`, `SQL_DESC_INDICATOR_PTR`, and `SQL_DESC_OCTET_LENGTH_PTR`) of any descriptor record of *Handle*. The memory allocated by the driver for fields that are not pointer fields is freed when the handle is freed. When a user-allocated descriptor handle is freed, all statements that the freed handle had been associated with revert to their respective automatically allocated descriptor handles.

Note ODBC 2.x drivers do not support freeing descriptor handles, just as they do not support allocating descriptor handles.

Note that **SQLDisconnect** automatically drops any statements and descriptors open on the connection. When an application frees a statement handle, the driver frees all the automatically generated descriptors associated with that handle.

For more information about descriptors, see the Part I PDF file, Chapter 13, “Overview of Descriptors.”

Code Example

See [SQLBrowseConnect](#) and [SQLConnect](#).

Related Functions

For information about	See
Allocating a handle	SQLAllocHandle
Canceling statement processing	SQLCancel
Setting a cursor name	SQLSetCursorName

SQLFreeStmt

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

SQLFreeStmt stops processing associated with a specific statement, closes any open cursors associated with the statement, discards pending results, or, optionally, frees all resources associated with the statement handle.

Syntax

SQLRETURN SQLFreeStmt (
SQLHSTMT	<i>StatementHandle</i> ,
SQLUSMALLINT	<i>Option</i>);

Arguments

StatementHandle

[Input]

Statement handle

Option

[Input]

One of the following options:

SQL_CLOSE: Closes the cursor associated with *StatementHandle* (if one was defined) and discards 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.

SQLCloseCursor can also be called to close a cursor. For more information, see the Part I PDF file, “Closing the Cursor” in Chapter 10, “Retrieving Results (Basic).”

SQL_DROP: This option is deprecated. A call to **SQLFreeStmt** with an *Option* of **SQL_DROP** is mapped in the Driver Manager to [SQLFreeHandle](#).

SQL_UNBIND: Sets the **SQL_DESC_COUNT** field of the ARD to 0, releasing all column buffers bound by **SQLBindCol** for the given *StatementHandle*. This does not unbind the bookmark column; to do that, the **SQL_DESC_DATA_PTR** field of the ARD for the bookmark column is set to NULL. Note that if this operation is performed on an explicitly allocated descriptor that is shared by more than one statement, the operation will affect the bindings of all statements that share the descriptor. For more information, see the Part I PDF file, “Chapter 10, “Overview of Retrieving Results (Basic).”

SQL_RESET_PARAMS: Sets the **SQL_DESC_COUNT** field of the APD to 0, releasing all parameter buffers set by **SQLBindParameter** for the given *StatementHandle*. If this operation is performed on an explicitly allocated descriptor that is shared by more than one statement, this operation will affect the bindings of all the statements that share the descriptor. For more information, see the Part I PDF file, “Binding Parameters” in Chapter 9, “Executing Statements.”

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 **SQLGetDiagRec** with a *HandleType* of **SQL_HANDLE_STMT** and a

Handle of *StatementHandle*. 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.)
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error	(DM) An asynchronously executing function was called for the <i>StatementHandle</i> and was still executing when this function was called. (DM) SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY092	Option type out of range	(DM) The value specified for the argument <i>Option</i> was not: SQL_CLOSE SQL_DROP SQL_UNBIND SQL_RESET_PARAMS
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

Calling **SQLFreeStmt** with the SQL_CLOSE option is equivalent to calling **SQLCloseCursor**, with the exception that **SQLFreeStmt** with SQL_CLOSE has no effect on the application if no cursor is open on the statement. If no cursor is open, a call to **SQLCloseCursor** returns SQLSTATE 24000 (Invalid cursor state).

An application should not use a statement handle after it has been freed; the Driver Manager does not check the validity of a handle in a function call.

Code Example

See [SQLBrowseConnect](#) and [SQLConnect](#).

Related Functions

For information about	See
Allocating a handle	SQLAllocHandle
Canceling statement processing	SQLCancel
Closing a cursor	SQLCloseCursor
Freeing a handle	SQLFreeHandle
Setting a cursor name	SQLSetCursorName

SQLGetConnectAttr

Conformance

Version Introduced: ODBC 3.0

Standards Compliance: ISO 92

Summary

SQLGetConnectAttr returns the current setting of a connection attribute.

Note For more information about what the Driver Manager maps this function to when an ODBC 3.x application is working with an ODBC 2.x driver, see the Part I PDF file, "Mapping Replacement Functions for Backward Compatibility of Applications" in Chapter 17, "Programming Considerations."

Syntax

SQLRETURN SQLGetConnectAttr(
SQLHDBC	ConnectionHandle,
SQLINTEGER	Attribute,
SQLPOINTER	ValuePtr,
SQLINTEGER	BufferLength,
SQLINTEGER *	StringLengthPtr);

Arguments

ConnectionHandle

[Input]
Connection handle.

Attribute

[Input]

Attribute to retrieve.

ValuePtr

[Output]

A pointer to memory in which to return the current value of the attribute specified by *Attribute*.

BufferLength

[Input]

If *Attribute* is an ODBC-defined attribute and *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **ValuePtr*. If *Attribute* is an ODBC-defined attribute and **ValuePtr* is an integer, *BufferLength* is ignored. If the value in **ValuePtr* is a Unicode string (when calling **SQLGetConnectAttrW**), the *BufferLength* argument must be an even number.

If *Attribute* is a driver-defined attribute, the application indicates the nature of the attribute to the Driver Manager by setting the *BufferLength* argument. *BufferLength* can have the following values:

- If **ValuePtr* is a pointer to a character string, then *BufferLength* is the length of the string or SQL_NTS.
- If **ValuePtr* is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(*length*) macro in *BufferLength*. This places a negative value in *BufferLength*.
- If **ValuePtr* is a pointer to a value other than a character string or binary string, then *BufferLength* should have the value SQL_IS_POINTER.
- If **ValuePtr* contains a fixed-length data type, then *BufferLength* is either SQL_IS_INTEGER or SQL_IS_UINTEGER, as appropriate.

StringLengthPtr

[Output]

A pointer to a buffer in which to return the total number of bytes (excluding the null-termination character) available to return in **ValuePtr*. If **ValuePtr* is a null pointer, no length is returned. If the attribute value is a character string and the number of bytes available to return is greater than *BufferLength* minus the length of the null-termination character, the data in **ValuePtr* is truncated to *BufferLength* minus the length of the null-termination character and is null-terminated by the driver.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLGetConnectAttr** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained from the diagnostic data structure by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_DBC and a *Handle* of *ConnectionHandle*. The following table lists the SQLSTATE values commonly returned by **SQLGetConnectAttr** 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	String data, right truncated	The data returned in <i>*ValuePtr</i> was truncated to be <i>BufferLength</i> minus the length of a null-termination character. The length of the untruncated string value is returned in <i>*StringLengthPtr</i> . (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection does not exist	(DM) An <i>Attribute</i> value that required an open connection was specified.
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.
HY000	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 from the diagnostic data structure by the argument <i>MessageText</i> in SQLGetDiagField describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error	(DM) SQLBrowseConnect was called for the <i>ConnectionHandle</i> and returned SQL_NEED_DATA. This function was called before SQLBrowseConnect returned SQL_SUCCESS_WITH_INFO or SQL_SUCCESS.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	(DM) <i>*ValuePtr</i> is a character string, and <i>BufferLength</i> was less than zero but not equal to SQL_NTS.
HY092	Invalid attribute/option identifier	The value specified for the argument <i>Attribute</i> was not valid for the version of ODBC supported by the driver.
HYC00	Optional feature not implemented	The value specified for the argument <i>Attribute</i> was a valid ODBC connection attribute for the version of ODBC supported by the driver, but was not supported by the driver.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver corresponding to the <i>ConnectionHandle</i> does not support the function.

Comments

For general information about connection attributes, see the Part I PDF file, "Connection Attributes" in Chapter 6, "Connecting to a Data Source or Driver."

For a list of attributes that can be set, see [SQLSetConnectAttr](#). Note that if *Attribute* specifies an attribute that returns a string, *ValuePtr* must be a pointer to a buffer for the string. The maximum length of the returned string, including the null-termination character, will be *BufferLength* bytes.

Depending on the attribute, an application does not need to establish a connection prior to calling **SQLGetConnectAttr**. However, if **SQLGetConnectAttr** is called and the specified attribute does not have a default and has not been set by a prior call to **SQLSetConnectAttr**, **SQLGetConnectAttr** will return SQL_NO_DATA.

If *Attribute* is SQL_ATTR_TRACE or SQL_ATTR_TRACEFILE, *ConnectionHandle* does not have to be valid, and **SQLGetConnectAttr** will not return SQL_ERROR or SQL_INVALID_HANDLE if *ConnectionHandle* is invalid. These attributes apply to all connections. **SQLGetConnectAttr** will return SQL_ERROR or SQL_INVALID_HANDLE if another argument is invalid.

While an application can set statement attributes using **SQLSetConnectAttr**, an application cannot use **SQLGetConnectAttr** to retrieve statement attribute values; it must call **SQLGetStmtAttr** to retrieve the setting of statement attributes.

Both SQL_ATTR_AUTO_IPD and SQL_ATTR_CONNECTION_DEAD connection attributes can be returned by a call to **SQLGetConnectAttr** but cannot be set by a call to **SQLSetConnectAttr**.

Related Functions

For information about	See
Returning the setting of a statement attribute	SQLGetStmtAttr
Setting a connection attribute	SQLSetConnectAttr
Setting an environment attribute	SQLSetEnvAttr
Setting a statement attribute	SQLSetStmtAttr

SQLGetConnectOption

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: Deprecated

Summary

In ODBC 3.x, the ODBC 2.x function **SQLGetConnectOption** has been replaced by **SQLGetConnectAttr**. For more information, see [SQLGetConnectAttr](#).

Note For more information about what the Driver Manager maps this function to when an ODBC 2.x application is working with an ODBC 3.x driver, see "Mapping Deprecated Functions" (Appendix G, "Driver Guidelines for Backward Compatibility") contained on the Microsoft Web site (ODBC Programmer'sReference).

SQLGetCursorName

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

SQLGetCursorName returns the cursor name associated with a specified statement.

Syntax

SQLRETURN SQLGetCursorName(
SQLHSTMT	StatementHandle,
SQLCHAR *	CursorName,
SQLSMALLINT	BufferLength,
SQLSMALLINT *	NameLengthPtr);

Arguments

StatementHandle

[Input]

Statement handle.

CursorName

[Output]

Pointer to a buffer in which to return the cursor name.

BufferLength

[Input]

Length of **CursorName*, in bytes. If the value in **CursorName* is a Unicode string (when calling **SQLGetCursorNameW**), the *BufferLength* argument must be an even number.

NameLengthPtr

[Output]

Pointer to memory in which to return the total number of bytes (excluding the null-termination character) available to return in **CursorName*. If the number of bytes available to return is greater than or equal to *BufferLength*, the cursor name in **CursorName* is truncated to *BufferLength* minus the length of a null-termination character.

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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	String data, right truncated	The buffer <i>*CursorName</i> was not large enough to return the entire cursor name, so the cursor name was truncated. The length of the untruncated cursor name is returned in <i>*NameLengthPtr</i> . (Function returns SQL_SUCCESS_WITH_INFO.)
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error	(DM) An asynchronously executing function was called for the <i>StatementHandle</i> and was still executing when this function was called. (DM) SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY015	No cursor name available	(DM) The driver was an ODBC 2.x driver, there was no open cursor on the statement, and no cursor name had been set with SQLSetCursorName .
HY090	Invalid string or buffer length	(DM) The value specified in the argument <i>BufferLength</i> was less than 0.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

Cursor names are used only in positioned update and delete statements (for example, **UPDATE** *table-name* ...**WHERE CURRENT OF** *cursor-name*). For more information, see the Part I PDF file, "Positioned Update and Delete Statements" in Chapter 12, "Updating Data." If the application does not call **SQLSetCursorName** to define a cursor name, the driver generates a name. This name begins with the letters SQL_CUR.

Note In ODBC 2.x, when there was no open cursor and no name had been set by a call to **SQLSetCursorName**, a call to **SQLGetCursorName** returned SQLSTATE HY015 (No cursor name available). In ODBC 3.x, this is no longer true; regardless of when **SQLGetCursorName** is called, the driver returns the cursor name.

SQLGetCursorName returns the name of a cursor whether or not the name was created explicitly or implicitly. A cursor name is implicitly generated if **SQLSetCursorName** is not called. **SQLSetCursorName** can be called to rename a cursor on a statement as long as the cursor is in an allocated or prepared state.

A cursor name that is set either explicitly or implicitly remains set until the *StatementHandle* with which it is associated is dropped, using **SQLFreeHandle** with a *HandleType* of SQL_HANDLE_STMT.

Related Functions

For information about	See
Executing an SQL statement	SQLExecDirect
Executing a prepared SQL statement	SQLExecute
Preparing a statement for execution	SQLPrepare
Setting a cursor name	SQLSetCursorName

SQLGetData

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: ISO 92

Summary

SQLGetData retrieves data for a single column in the result set. It can be called multiple times to retrieve variable-length data in parts.

Syntax

SQLRETURN SQLGetData (
SQLHSTMT	StatementHandle,
SQLUSMALLINT	ColumnNumber,
SQLSMALLINT	TargetType,
SQLPOINTER	TargetValuePtr,
SQLINTEGER	BufferLength,

SQLINTEGER *	StrLen_or_IndPtr);
--------------	--------------------

Arguments

StatementHandle

[Input]

Statement handle.

ColumnNumber

[Input]

Number of the column for which to return data. Result set columns are numbered in increasing column order starting at 1. The bookmark column is column number 0; this can be specified only if bookmarks are enabled.

TargetType

[Input]

The type identifier of the C data type of the **TargetValuePtr* buffer. For a list of valid C data types and type identifiers, see the “C Data Types” section in Appendix D, “Data Types” of **the SOLID Programmer Guide**. If *TargetType* is SQL_ARD_TYPE, the driver uses the type identifier specified in the SQL_DESC_CONCISE_TYPE field of the ARD. If it is SQL_C_DEFAULT, the driver selects the default C data type based on the SQL data type of the source.

TargetValuePtr

[Output]

Pointer to the buffer in which to return the data.

BufferLength

[Input]

Length of the **TargetValuePtr* buffer in bytes.

The driver uses *BufferLength* to avoid writing past the end of the **TargetValuePtr* buffer when returning variable-length data, such as character or binary data. Note that the driver counts the null-termination character when returning character data to **TargetValuePtr*. **TargetValuePtr* must therefore contain space for the null-termination character, or the driver will truncate the data.

When the driver returns fixed-length data, such as an integer or a date structure, the driver ignores *BufferLength* and assumes the buffer is large enough to hold the data. It is therefore important for the application to allocate a large enough buffer for fixed-length data or the driver will write past the end of the buffer.

SQLGetData returns SQLSTATE HY090 (Invalid string or buffer length) when *BufferLength* is less than 0 but not when *BufferLength* is 0.

If *TargetValuePtr* is set to a null pointer, *BufferLength* is ignored by the driver.

StrLen_or_IndPtr

[Output]

Pointer to the buffer in which to return the length or indicator value. If this is a null pointer, no length or indicator value is returned. This returns an error when the data being fetched is NULL.

SQLGetData can return the following values in the length/indicator buffer:

The length of the data available to return

SQL_NO_TOTAL

SQL_NULL_DATA

For more information, see the Part I PDF file, "Using Length/Indicator Values" section in Chapter 4, "ODBC Fundamentals" and "Comments" in this section.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, 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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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	String data, right truncated	<p>Not all of the data for the specified column, <i>ColumnNumber</i>, could be retrieved in a single call to the function. The length of the data remaining in the specified column prior to the current call to SQLGetData is returned in <i>*StrLen_or_IndPtr</i>. (Function returns SQL_SUCCESS_WITH_INFO.)</p> <p>The <i>TargetValuePtr</i> argument was a null pointer, and more data was available to return. (Function returns SQL_SUCCESS_WITH_INFO.)</p> <p>For more information on using multiple calls to SQLGetData for a single column, see "Comments."</p>
01S07	Fractional truncation	<p>The data returned for one or more columns was truncated. For numeric data types, the fractional part of the number was truncated. For time, timestamp, and interval data types containing a time component, the fractional portion of the time was truncated. (Function returns SQL_SUCCESS_WITH_INFO.)</p>
07006	Restricted data type attribute violation	The data value of a column in the result set cannot be converted to the C data type specified by the argument <i>TargetType</i> .

07009	Invalid descriptor index	<p>The value specified for the argument <i>ColumnNumber</i> was 0, and the <code>SQL_ATTR_USE_BOOKMARKS</code> statement attribute was set to <code>SQL_UB_OFF</code>.</p> <p>The value specified for the argument <i>ColumnNumber</i> was greater than the number of columns in the result set.</p> <p>(DM) The specified column was bound. This description does not apply to drivers that return the <code>SQL_GD_BOUND</code> bitmask for the <code>SQL_GETDATA_EXTENSIONS</code> option in SQLGetInfo.</p> <p>(DM) The number of the specified column was less than or equal to the number of the highest bound column. This description does not apply to drivers that return the <code>SQL_GD_ANY_COLUMN</code> bitmask for the <code>SQL_GETDATA_EXTENSIONS</code> option in SQLGetInfo.</p> <p>(DM) The application has already called SQLGetData for the current row; the number of the column specified in the current call was less than the number of the column specified in the preceding call; and the driver does not return the <code>SQL_GD_ANY_ORDER</code> bitmask for the <code>SQL_GETDATA_EXTENSIONS</code> option in SQLGetInfo.</p> <p>(DM) The <i>TargetType</i> argument was <code>SQL_ARD_TYPE</code>, and the <i>ColumnNumber</i> descriptor record in the ARD failed the consistency check.</p> <p>(DM) The <i>TargetType</i> argument was <code>SQL_ARD_TYPE</code>, and the value in the <code>SQL_DESC_COUNT</code> field of the ARD was less than the <i>ColumnNumber</i> argument.</p>
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.
22002	Indicator variable required but not supplied	<i>StrLen_or_IndPtr</i> was a null pointer and NULL data was retrieved.
22003	Numeric value out of range	<p>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.</p> <p>For more information, see Appendix D, “Data Types” in the SOLID Programmer Guide.</p>
22007	Invalid datetime format	The character column in the result set was bound to a C date, time, or timestamp structure, and the value in

		the column was an invalid date, time, or timestamp, respectively. For more information, see Appendix D, “Data Types” in the SOLID Programmer Guide .
22012	Division by zero	A value from an arithmetic expression that resulted in division by zero was returned.
22015	Interval field overflow	Assigning from an exact numeric or interval SQL type to an interval C type caused a loss of significant digits in the leading field. When returning data to an interval C type, there was no representation of the value of the SQL type in the interval C type.
22018	Invalid character value for cast specification	A character column in the result set was returned to a character C buffer, and the column contained a character for which there was no representation in the character set of the buffer. The C type was an exact or approximate numeric, a datetime, or an interval data type; the SQL type of the column was a character data type; and the value in the column was not a valid literal of the bound C type.
24000	Invalid cursor state	(DM) The function was called without first calling SQLFetch or SQLFetchScroll to position the cursor on the row of data required. (DM) The <i>StatementHandle</i> was in an executed state, but no result set was associated with the <i>StatementHandle</i> . A cursor was open on the <i>StatementHandle</i> and SQLFetch or SQLFetchScroll had been called, but the cursor was positioned before the start of the result set or after the end of the result set.
HY000	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 SQLGetDiagRec in the <i>MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY003	Invalid application buffer type	(DM) The argument <i>TargetType</i> was neither a valid data type, SQL_C_DEFAULT, nor SQL_C_ARD_TYPE. (DM) The argument <i>ColumnNumber</i> was 0, and the argument <i>TargetType</i> was not SQL_C_BOOKMARK for a fixed-length bookmark or SQL_C_VARBOOKMARK for a variable-length bookmark.
HY008	Operation canceled	Asynchronous processing was enabled for the

		<p><i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>, and then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application, and then the function was called again on the <i>StatementHandle</i>.</p>
HY009	Invalid use of null pointer	(DM) The argument <i>CursorName</i> was a null pointer.
HY010	Function sequence error	<p>(DM) The specified <i>StatementHandle</i> was not in an executed state. The function was called without first calling SQLExecDirect, SQLExecute or a catalog function.</p> <p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p> <p>(DM) The <i>StatementHandle</i> was in an executed state, but no result set was associated with the <i>StatementHandle</i>.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	<p>(DM) The value specified for argument <i>BufferLength</i> was less than 0.</p> <p>The value specified for argument <i>BufferLength</i> was less than 4, the <i>ColumnNumber</i> argument was set to 0, and the driver was an ODBC 2.x driver.</p>
HY109	Invalid cursor position	<p>The cursor was positioned (by SQLSetPos, SQLFetch, SQLFetchScroll, or SQLBulkOperations) on a row that had been deleted or could not be fetched.</p> <p>The cursor was a forward-only cursor, and the rowset size was greater than one.</p>
HYC00	Optional feature not implemented	The driver or data source does not support use of SQLGetData with multiple rows in SQLFetchScroll . This description does not apply to drivers that return the SQL_GD_BLOCK bitmask for the SQL_GETDATA_EXTENSIONS option in

		<p>SQLGetInfo.</p> <p>The driver or data source does not support the conversion specified by the combination of the <i>TargetType</i> argument and the SQL data type of the corresponding column. This error applies only when the SQL data type of the column was mapped to a driver-specific SQL data type.</p> <p>The driver supports only ODBC 2.x, and the argument <i>TargetType</i> was one of the following:</p> <p>SQL_C_GUID SQL_C_NUMERIC SQL_C_SBIGINT SQL_C_UBIGINT</p>
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver corresponding to the <i>StatementHandle</i> does not support the function.

Comments

SQLGetData returns the data in a specified column. **SQLGetData** can be called only after one or more rows have been fetched from the result set by **SQLFetch**, **SQLFetchScroll**, or **SQLExtendedFetch**. If variable-length data is too large to be returned in a single call to **SQLGetData** (due to a limitation in the application), **SQLGetData** can retrieve it in parts. It is possible to bind some columns in a row and call **SQLGetData** for others, although this is subject to some restrictions. For more information, see the Part I PDF file, "Getting Long Data" in Chapter 10, "Retrieving Results (Basic)."

Using SQLGetData

If the driver does not support extensions to **SQLGetData**, the function can return data only for unbound columns with a number greater than that of the last bound column. Furthermore, within a row of data, the value of the *ColumnNumber* argument in each call to **SQLGetData** must be greater than or equal to the value of *ColumnNumber* in the previous call; that is, data must be retrieved in increasing column number order. Finally, if no extensions are supported, **SQLGetData** cannot be called if the rowset size is greater than 1.

Drivers can relax any of these restrictions. To determine what restrictions a driver relaxes, an application calls **SQLGetInfo** with any of the following **SQL_GETDATA_EXTENSIONS** options:

- **SQL_GD_ANY_COLUMN**. If this option is returned, **SQLGetData** can be called for any unbound column, including those before the last bound column.
- **SQL_GD_ANY_ORDER**. If this option is returned, **SQLGetData** can be called for unbound columns in any order.

- **SQL_GD_BLOCK.** If this option is returned by **SQLGetInfo** for the **SQL_GETDATA_EXTENSIONS** InfoType, the driver supports calls to **SQLGetData** when the rowset size is greater than 1 and the application can call **SQLSetPos** with the **SQL_POSITION** option to position the cursor on the correct row before calling **SQLGetData**.
- **SQL_GD_BOUND.** If this option is returned, **SQLGetData** can be called for bound columns as well as unbound columns.

There are two exceptions to these restrictions and a driver's ability to relax them. First, **SQLGetData** should never be called for a forward-only cursor when the rowset size is greater than 1. Second, if a driver supports bookmarks, it must always support the ability to call **SQLGetData** for column 0, even if it does not allow applications to call **SQLGetData** for other columns before the last bound column. (When an application is working with an ODBC 2.x driver, **SQLGetData** will successfully return a bookmark when called with *ColumnNumber* equal to 0 after a call to **SQLFetch**, because **SQLFetch** is mapped by the ODBC 3.x Driver Manager to **SQLExtendedFetch** with a *FetchOrientation* of **SQL_FETCH_NEXT**, and **SQLGetData** with a *ColumnNumber* of 0 is mapped by the ODBC 3.x Driver Manager to **SQLGetStmtOption** with an *fOption* of **SQL_GET_BOOKMARK**.)

SQLGetData cannot be used to retrieve the bookmark for a row just inserted by calling **SQLBulkOperations** with the **SQL_ADD** option, because the cursor is not positioned on the row. An application can retrieve the bookmark for such a row by binding column 0 before calling **SQLBulkOperations** with **SQL_ADD**, in which case **SQLBulkOperations** returns the bookmark in the bound buffer. **SQLFetchScroll** can then be called with **SQL_FETCH_BOOKMARK** to reposition the cursor on that row.

If the *TargetType* argument is an interval data type, the default interval leading precision (2) and the default interval seconds precision (6), as set in the **SQL_DESC_DATETIME_INTERVAL_PRECISION** and **SQL_DESC_PRECISION** fields of the ARD, respectively, are used for the data. If the *TargetType* argument is an **SQL_C_NUMERIC** data type, the default precision (driver-defined) and default scale (0), as set in the **SQL_DESC_PRECISION** and **SQL_DESC_SCALE** fields of the ARD, are used for the data. If any default precision or scale is not appropriate, the application should explicitly set the appropriate descriptor field by a call to **SQLSetDescField** or **SQLSetDescRec**. It can set the **SQL_DESC_CONCISE_TYPE** field to **SQL_C_NUMERIC** and call **SQLGetData** with a *TargetType* argument of **SQL_ARD_TYPE**, which will cause the precision and scale values in the descriptor fields to be used.

Note In ODBC 2.x, applications set *TargetType* to **SQL_C_DATE**, **SQL_C_TIME**, or **SQL_C_TIMESTAMP** to indicate that **TargetValuePtr* is a date, time, or timestamp structure. In ODBC 3.x, applications set *TargetType* to **SQL_C_TYPE_DATE**, **SQL_C_TYPE_TIME**, or **SQL_C_TYPE_TIMESTAMP**. The Driver Manager makes appropriate mappings if necessary, based on the application and driver version.

Retrieving Variable-Length Data in Parts

SQLGetData can be used to retrieve data from a column that contains variable-length data in parts—that is, when the identifier of the SQL data type of the column is **SQL_CHAR**, **SQL_VARCHAR**, **SQL_LONGVARCHAR**, **SQL_WCHAR**, **SQL_WVARCHAR**, **SQL_WLONGVARCHAR**, **SQL_BINARY**, **SQL_VARBINARY**, **SQL_LONGVARBINARY**, or a driver-specific identifier for a variable-length type.

To retrieve data from a column in parts, the application calls **SQLGetData** multiple times in succession for the same column. On each call, **SQLGetData** returns the next part of the data. It is up to the application to reassemble the parts, taking care to remove the null-termination character from intermediate parts of

character data. If there is more data to return or not enough buffer was allocated for the terminating character, **SQLGetData** returns `SQL_SUCCESS_WITH_INFO` and `SQLSTATE 01004` (Data truncated). When it returns the last part of the data, **SQLGetData** returns `SQL_SUCCESS`. Neither `SQL_NO_TOTAL` nor zero can be returned on the last valid call to retrieve data from a column, because the application would then have no way of knowing how much of the data in the application buffer is valid. If **SQLGetData** is called after this, it returns `SQL_NO_DATA`. For more information, see the next section, "Retrieving Data with SQLGetData."

Variable-length bookmarks can be returned in parts by **SQLGetData**. As with other data, a call to **SQLGetData** to return variable-length bookmarks in parts will return `SQLSTATE 01004` (String data, right truncated) and `SQL_SUCCESS_WITH_INFO` when there is more data to be returned. This is different than the case when a variable-length bookmark is truncated by a call to **SQLFetch** or **SQLFetchScroll**, which returns `SQL_ERROR` and `SQLSTATE 22001` (String data, right truncated).

SQLGetData cannot be used to return fixed-length data in parts. If **SQLGetData** is called more than one time in a row for a column containing fixed-length data, it returns `SQL_NO_DATA` for all calls after the first.

Retrieving Data with SQLGetData

To return data for the specified column, **SQLGetData** performs the following sequence of steps:

Returns `SQL_NO_DATA` if it has already returned all of the data for the column.

- Sets **StrLen_or_IndPtr* to `SQL_NULL_DATA` if the data is NULL. If the data is NULL and *StrLen_or_IndPtr* was a null pointer, **SQLGetData** returns `SQLSTATE 22002` (Indicator variable required but not supplied).
- If the data for the column is not NULL, **SQLGetData** proceeds to the step below.
- If the `SQL_ATTR_MAX_LENGTH` statement attribute is set to a nonzero value, if the column contains character or binary data, and if **SQLGetData** has not previously been called for the column, the data is truncated to `SQL_ATTR_MAX_LENGTH` bytes.

Note The `SQL_ATTR_MAX_LENGTH` statement attribute is intended to reduce network traffic. It is generally implemented by the data source, which truncates the data before returning it across the network. Drivers and data sources are not required to support it. Therefore, to guarantee that data is truncated to a particular size, an application should allocate a buffer of that size and specify the size in the *BufferLength* argument.

- Converts the data to the type specified in *TargetType*. The data is given the default precision and scale for that data type. If *TargetType* is `SQL_ARD_TYPE`, the data type in the `SQL_DESC_CONCISE_TYPE` field of the ARD is used. If *TargetType* is `SQL_ARD_TYPE`, the data is given the precision and scale in the `SQL_DESC_DATETIME_INTERVAL_PRECISION`, `SQL_DESC_PRECISION`, and `SQL_DESC_SCALE` fields of the ARD, depending on the data type in the `SQL_DESC_CONCISE_TYPE` field. If any default precision or scale is not appropriate, the application should explicitly set the appropriate descriptor field by a call to **SQLSetDescField** or **SQLSetDescRec**.
- If the data was converted to a variable-length data type, such as character or binary, **SQLGetData** checks whether the length of the data exceeds *BufferLength*. If the length of character data (including the null-termination character) exceeds *BufferLength*, **SQLGetData** truncates the data to *BufferLength* less the length of a null-termination character. It then null-terminates the data. If

the length of binary data exceeds the length of the data buffer, **SQLGetData** truncates it to *BufferLength* bytes.

- If the data buffer supplied is too small to hold the null-termination character, **SQLGetData** returns SQL_SUCCESS_WITH_INFO and SQLSTATE 01004.
- **SQLGetData** never truncates data converted to fixed-length data types; it always assumes that the length of **TargetValuePtr* is the size of the data type.
- Places the converted (and possibly truncated) data in **TargetValuePtr*. Note that **SQLGetData** cannot return data out of line.
- Places the length of the data in **StrLen_or_IndPtr*. If *StrLen_or_IndPtr* was a null pointer, **SQLGetData** does not return the length.

For character or binary data, this is the length of the data after conversion and before truncation due to *BufferLength*. If the driver cannot determine the length of the data after conversion, as is sometimes the case with long data, it returns SQL_SUCCESS_WITH_INFO and sets the length to SQL_NO_TOTAL. (The last call to **SQLGetData** must always return the length of the data, not zero or SQL_NO_TOTAL.) If data was truncated due to the SQL_ATTR_MAX_LENGTH statement attribute, the value of this attribute—as opposed to the actual length—is placed in **StrLen_or_IndPtr*. This is because this attribute is designed to truncate data on the server before conversion, so the driver has no way of figuring out what the actual length is. When **SQLGetData** is called multiple times in succession for the same column, this is the length of the data available at the start of the current call; that is, the length decreases with each subsequent call.

For all other data types, this is the length of the data after conversion; that is, it is the size of the type to which the data was converted.

If the data is truncated without loss of significance during conversion (for example, the real number 1.234 is truncated when converted to the integer 1) or because *BufferLength* is too small (for example, the string "abcdef" is placed in a 4-byte buffer), **SQLGetData** returns SQLSTATE 01004 (Data truncated) and SQL_SUCCESS_WITH_INFO. If data is truncated without loss of significance due to the SQL_ATTR_MAX_LENGTH statement attribute, **SQLGetData** returns SQL_SUCCESS and does not return SQLSTATE 01004 (Data truncated).

The contents of the bound data buffer (if **SQLGetData** is called on a bound column) and the length/indicator buffer are undefined if **SQLGetData** does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO.

Successive calls to **SQLGetData** will retrieve data from the last column requested; prior offsets become invalid. For example, when the following sequence is performed:

```
SQLGetData(icol=n), SQLGetData(icol=m), SQLGetData(icol=n)
```

the second call to **SQLGetData(icol=n)** retrieves data from the start of the *n* column. Any offset in the data due to earlier calls to **SQLGetData** for the column is no longer valid.

Descriptors and SQLGetData

SQLGetData does not interact directly with any descriptor fields.

If *TargetType* is SQL_ARD_TYPE, the data type in the SQL_DESC_CONCISE_TYPE field of the ARD is used. If *TargetType* is either SQL_ARD_TYPE or SQL_C_DEFAULT, the data is given the precision and

scale in the SQL_DESC_DATETIME_INTERVAL_PRECISION, SQL_DESC_PRECISION, and SQL_DESC_SCALE fields of the ARD, depending on the data type in the SQL_DESC_CONCISE_TYPE field.

Code Example

In the following example, an application executes a **SELECT** statement to return a result set of the customer IDs, names, and phone numbers sorted by name, ID, and phone number. 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 buffers for the data and the returned number of bytes are specified in the call to **SQLGetData**. Finally, it prints each employee's name, ID, and phone number.

```
#define NAME_LEN 50
#define PHONE_LEN 50

SQLCHAR szName[NAME_LEN], szPhone[PHONE_LEN];
SQLINTEGER sCustID, cbName, cbAge, cbBirthday;
SQLRETURN retcode;
SQLHSTMT hstmt;

retcode = SQLExecDirect(hstmt,
    "SELECT CUSTID, NAME, PHONE FROM CUSTOMERS ORDER BY 2, 1, 3",
    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 */

            SQLGetData(hstmt, 1, SQL_C_ULONG, &sCustID, 0, &cbCustID);
            SQLGetData(hstmt, 2, SQL_C_CHAR, szName, NAME_LEN, &cbName);
            SQLGetData(hstmt, 3, SQL_C_CHAR, szPhone, PHONE_LEN,
                &cbPhone);

            /* Print the row of data */

            fprintf(out, "%-5d %-*s %*s", sCustID, NAME_LEN-1, szName,
                PHONE_LEN-1, szPhone);
        } else {
            break;
        }
    }
}
```

Related Functions

For information about	See
Assigning storage for a column in a result set	SQLBindCol
Performing bulk operations that do not relate to the block cursor position	SQLBulkOperations
Canceling statement processing	SQLCancel

Executing an SQL statement	SQLExecDirect
Executing a prepared SQL statement	SQLExecute
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Fetching a single row of data or a block of data in a forward-only direction	SQLFetch
Sending parameter data at execution time	SQLPutData
Positioning the cursor, refreshing data in the rowset, or updating or deleting data in the rowset	SQLSetPos

SQLGetDescField

Conformance

Version Introduced: ODBC 3.0

Standards Compliance: ISO 92

Summary

SQLGetDescField returns the current setting or value of a single field of a descriptor record.

Syntax

SQLRETURN SQLGetDescField(
SQLHDESC	DescriptorHandle,
SQLSMALLINT	RecNumber,
SQLSMALLINT	FieldIdentifier,
SQLPOINTER	ValuePtr,
SQLINTEGER	BufferLength,
SQLINTEGER *	StringLengthPtr);

Arguments

DescriptorHandle

[Input]

Descriptor handle.

RecNumber

[Input]

Indicates the descriptor record from which the application seeks information. Descriptor records are numbered from 0, with record number 0 being the bookmark record. If the *FieldIdentifier* argument indicates a header field, *RecNumber* is ignored. If *RecNumber* is less than or equal to SQL_DESC_COUNT but the row does not contain data for a column or parameter, a call to **SQLGetDescField** will return the default values of the fields. (For more information, see "Initialization of Descriptor Fields" in [SQLSetDescField](#).)

FieldIdentifier

[Input]

Indicates the field of the descriptor whose value is to be returned. For more information, see the "*FieldIdentifier* Argument" section in **SQLSetDescField**.

ValuePtr

[Output]

Pointer to a buffer in which to return the descriptor information. The data type depends on the value of *FieldIdentifier*.

BufferLength

[Input]

If *FieldIdentifier* is an ODBC-defined field and *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **ValuePtr*. If *FieldIdentifier* is an ODBC-defined field and **ValuePtr* is an integer, *BufferLength* is ignored. If the value in **ValuePtr* is of a Unicode data type (when calling **SQLGetDescFieldW**), the *BufferLength* argument must be an even number.

If *FieldIdentifier* is a driver-defined field, the application indicates the nature of the field to the Driver Manager by setting the *BufferLength* argument. *BufferLength* can have the following values:

- If **ValuePtr* is a pointer to a character string, then *BufferLength* is the length of the string or SQL_NTS.
- If **ValuePtr* is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(*length*) macro in *BufferLength*. This places a negative value in *BufferLength*.
- If **ValuePtr* is a pointer to a value other than a character string or binary string, then *BufferLength* should have the value SQL_IS_POINTER.
- If **ValuePtr* contains a fixed-length data type, then *BufferLength* is SQL_IS_INTEGER, SQL_IS_UINTEGER, SQL_IS_SMALLINT, or SQL_IS_USMALLINT, as appropriate.

StringLengthPtr

[Output]

Pointer to the buffer in which to return the total number of bytes (excluding the number of bytes required for the null-termination character) available to return in **ValuePtr*.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_NO_DATA, or SQL_INVALID_HANDLE.

SQL_NO_DATA is returned if *RecNumber* is greater than the current number of descriptor records.

SQL_NO_DATA is returned if *DescriptorHandle* is an IRD handle and the statement is in the prepared or executed state but there was no open cursor associated with it.

Diagnostics

When **SQLGetDescField** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value can be obtained by calling **SQLGetDiagRec** with a *HandleType* of

SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. The following table lists the SQLSTATE values commonly returned by **SQLGetDescField** 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	String data, right truncated	The buffer <i>*ValuePtr</i> was not large enough to return the entire descriptor field, so the field was truncated. The length of the untruncated descriptor field is returned in <i>*StringLengthPtr</i> . (Function returns SQL_SUCCESS_WITH_INFO.)
07009	Invalid descriptor index	(DM) The <i>RecNumber</i> argument was equal to 0, the SQL_ATTR_USE_BOOKMARK statement attribute was SQL_UB_OFF, and the <i>DescriptorHandle</i> argument was an IRD handle. (This error can be returned for an explicitly allocated descriptor only if the descriptor is associated with a statement handle.) The <i>FieldIdentifier</i> argument was a record field, the <i>RecNumber</i> argument was 0, and the <i>DescriptorHandle</i> argument was an IPD handle. The <i>RecNumber</i> argument was less than 0.
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.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate the memory required to support execution or completion of the function.
HY007	Associated statement is not prepared	<i>DescriptorHandle</i> was associated with a <i>StatementHandle</i> as an IRD, and the associated statement handle had not been prepared or executed.
HY010	Function sequence error	(DM) <i>DescriptorHandle</i> was associated with a <i>StatementHandle</i> for which an asynchronously executing function (not this one) was called and was still executing when this function was called. (DM) <i>DescriptorHandle</i> was associated with a <i>StatementHandle</i> for which SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos was called and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.

HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY021	Inconsistent descriptor information	The <code>SQL_DESC_TYPE</code> and <code>SQL_DESC_DATETIME_INTERVAL_CODE</code> fields do not form a valid ODBC SQL type, a valid driver-specific SQL type (for IPDs), or a valid ODBC C type (for APDs or ARDs).
HY090	Invalid string or buffer length	(DM) <i>*ValuePtr</i> was a character string, and <i>BufferLength</i> was less than zero.
HY091	Invalid descriptor field identifier	<i>FieldIdentifier</i> was not an ODBC-defined field and was not an implementation-defined value. <i>FieldIdentifier</i> was undefined for the <i>DescriptorHandle</i> .
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through <code>SQLSetConnectAttr</code> , <code>SQL_ATTR_CONNECTION_TIMEOUT</code> .
IM001	Driver does not support this function	(DM) The driver associated with the <i>DescriptorHandle</i> does not support the function.

Comments

An application can call **SQLGetDescField** to return the value of a single field of a descriptor record. A call to **SQLGetDescField** can return the setting of any field in any descriptor type, including header fields, record fields, and bookmark fields. An application can obtain the settings of multiple fields in the same or different descriptors, in arbitrary order, by making repeated calls to **SQLGetDescField**. **SQLGetDescField** can also be called to return driver-defined descriptor fields.

For performance reasons, an application should not call **SQLGetDescField** for an IRD before executing a statement.

The settings of multiple fields that describe the name, data type, and storage of column or parameter data can also be retrieved in a single call to **SQLGetDescRec**. **SQLGetStmtAttr** can be called to return the setting of a single field in the descriptor header that is also a statement attribute. **SQLColAttribute**, **SQLDescribeCol**, and **SQLDescribeParam** return record or bookmark fields.

When an application calls **SQLGetDescField** to retrieve the value of a field that is undefined for a particular descriptor type, the function returns `SQL_SUCCESS` but the value returned for the field is undefined. For example, calling **SQLGetDescField** for the `SQL_DESC_NAME` or `SQL_DESC_NULLABLE` field of an APD or ARD will return `SQL_SUCCESS` but an undefined value for the field.

When an application calls **SQLGetDescField** to retrieve the value of a field that is defined for a particular descriptor type but that has no default value and has not been set yet, the function returns `SQL_SUCCESS` but the value returned for the field is undefined. For more information on the initialization of descriptor fields and descriptions of the fields, see "Initialization of Descriptor Fields" in **SQLSetDescField**. For more information on descriptors, see the Part I PDF file, Chapter 13, "Overview of Descriptors."

Related Functions

For information about	See
Getting multiple descriptor fields	SQLGetDescRec
Setting a single descriptor field	SQLSetDescField
Setting multiple descriptor fields	SQLSetDescRec

SQLGetDescRec

Conformance

Version Introduced: ODBC 3.0

Standards Compliance: ISO 92

Summary

SQLGetDescRec returns the current settings or values of multiple fields of a descriptor record. The fields returned describe the name, data type, and storage of column or parameter data.

Syntax

SQLRETURN SQLGetDescRec(
SQLHDESC	DescriptorHandle,
SQLSMALLINT	RecNumber,
SQLCHAR *	Name,
SQLSMALLINT	BufferLength,
SQLSMALLINT *	StringLengthPtr,
SQLSMALLINT *	TypePtr,
SQLSMALLINT *	SubTypePtr,
SQLINTEGER *	LengthPtr,
SQLSMALLINT *	PrecisionPtr,
SQLSMALLINT *	ScalePtr,
SQLSMALLINT *	NullablePtr);

Arguments

DescriptorHandle

[Input]

Descriptor handle.

RecNumber

[Input]

Indicates the descriptor record from which the application seeks information. Descriptor records are numbered from 1, with record number 0 being the bookmark record. The *RecNumber* argument must be

less than or equal to the value of SQL_DESC_COUNT. If *RecNumber* is less than or equal to SQL_DESC_COUNT but the row does not contain data for a column or parameter, a call to **SQLGetDescRec** will return the default values of the fields. (For more information, see "Initialization of Descriptor Fields" in [SQLSetDescField](#).)

Name

[Output]

A pointer to a buffer in which to return the SQL_DESC_NAME field for the descriptor record.

BufferLength

[Input]

Length of the **Name* buffer, in bytes.

StringLengthPtr

[Output]

A pointer to a buffer in which to return the number of bytes of data available to return in the **Name* buffer, excluding the null-termination character. If the number of bytes was greater than or equal to *BufferLength*, the data in **Name* is truncated to *BufferLength* minus the length of a null-termination character, and is null-terminated by the driver.

TypePtr

[Output]

A pointer to a buffer in which to return the value of the SQL_DESC_TYPE field for the descriptor record.

SubTypePtr

[Output]

For records whose type is SQL_DATETIME or SQL_INTERVAL, this is a pointer to a buffer in which to return the value of the SQL_DESC_DATETIME_INTERVAL_CODE field.

LengthPtr

[Output]

A pointer to a buffer in which to return the value of the SQL_DESC_OCTET_LENGTH field for the descriptor record.

PrecisionPtr

[Output]

A pointer to a buffer in which to return the value of the SQL_DESC_PRECISION field for the descriptor record.

ScalePtr

[Output]

A pointer to a buffer in which to return the value of the SQL_DESC_SCALE field for the descriptor record.

NullablePtr

[Output]

A pointer to a buffer in which to return the value of the SQL_DESC_NULLABLE field for the descriptor record.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_NO_DATA, or SQL_INVALID_HANDLE.

SQL_NO_DATA is returned if *RecNumber* is greater than the current number of descriptor records.

SQL_NO_DATA is returned if *DescriptorHandle* is an IRD handle and the statement is in the prepared or executed state but there was no open cursor associated with it.

Diagnostics

When **SQLGetDescRec** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_DESC and a *Handle* of *DescriptorHandle*. The following table lists the SQLSTATE values commonly returned by **SQLGetDescRec** 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	String data, right truncated	The buffer <i>*Name</i> was not large enough to return the entire descriptor field, so the field was truncated. The length of the untruncated descriptor field is returned in <i>*StringLengthPtr</i> . (Function returns SQL_SUCCESS_WITH_INFO.)
07009	Invalid descriptor index	The <i>FieldIdentifier</i> argument was a record field, the <i>RecNumber</i> argument was set to 0, and the <i>DescriptorHandle</i> argument was an IPD handle. (DM) The <i>RecNumber</i> argument was set to 0, and the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF, and the <i>DescriptorHandle</i> argument was an IRD handle. The <i>RecNumber</i> argument was less than 0.
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.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate the memory required to support execution or completion of the function.

HY007	Associated statement is not prepared	<i>DescriptorHandle</i> was associated with an IRD, and the associated statement handle was not in the prepared or executed state.
HY010	Function sequence error	(DM) <i>DescriptorHandle</i> was associated with a <i>StatementHandle</i> for which an asynchronously executing function (not this one) was called and was still executing when this function was called. (DM) <i>DescriptorHandle</i> was associated with a <i>StatementHandle</i> for which SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos was called and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with <i>DescriptorHandle</i> does not support the function.

Comments

An application can call **SQLGetDescRec** to retrieve the values of the following descriptor fields for a single column or parameter:

SQL_DESC_NAME

SQL_DESC_TYPE

SQL_DESC_DATETIME_INTERVAL_CODE (for records whose type is SQL_DATETIME or SQL_INTERVAL)

SQL_DESC_OCTET_LENGTH

SQL_DESC_PRECISION

SQL_DESC_SCALE

SQL_DESC_NULLABLE

SQLGetDescRec does not retrieve the values for header fields.

An application can inhibit the return of a field's setting by setting the argument corresponding to the field to a null pointer.

When an application calls **SQLGetDescRec** to retrieve the value of a field that is undefined for a particular descriptor type, the function returns SQL_SUCCESS but the value returned for the field is undefined. For example, calling **SQLGetDescRec** for the SQL_DESC_NAME or SQL_DESC_NULLABLE field of an APD or ARD will return SQL_SUCCESS but an undefined value for the field.

When an application calls **SQLGetDescRec** to retrieve the value of a field that is defined for a particular descriptor type but that has no default value and has not been set yet, the function returns SQL_SUCCESS

but the value returned for the field is undefined. For more information, see "Initialization of Descriptor Fields" in [SQLSetDescField](#).

The values of fields can also be retrieved individually by a call to **SQLGetDescField**. For a description of the fields in a descriptor header or record, see **SQLSetDescField**. For more information on descriptors, see the Part I PDF file, Chapter 13, "Overview of Descriptors."

Related Functions

For information about	See
Binding a column	SQLBindCol
Binding a parameter	SQLBindParameter
Getting a descriptor field	SQLGetDescField
Setting multiple descriptor fields	SQLSetDescRec

SQLGetDiagField

Conformance

Version Introduced: ODBC 3.0

Standards Compliance: ISO 92

Summary

SQLGetDiagField returns the current value of a field of a record of the diagnostic data structure (associated with a specified handle) that contains error, warning, and status information.

Syntax

SQLRETURN SQLGetDiagField(
SQLSMALLINT	HandleType,
SQLHANDLE	Handle,
SQLSMALLINT	RecNumber,
SQLSMALLINT	DiagIdentifier,
SQLPOINTER	DiagInfoPtr,
SQLSMALLINT	BufferLength,
SQLSMALLINT *	StringLengthPtr);

Arguments

HandleType

[Input]

A handle type identifier that describes the type of handle for which diagnostics are required. Must be one of the following:

SQL_HANDLE_ENV

SQL_HANDLE_DBC

SQL_HANDLE_STMT
SQL_HANDLE_DESC

Handle

[Input]

A handle for the diagnostic data structure, of the type indicated by *HandleType*. If *HandleType* is SQL_HANDLE_ENV, *Handle* can be either a shared or an unshared environment handle.

RecNumber

[Input]

Indicates the status record from which the application seeks information. Status records are numbered from 1. If the *DiagIdentifier* argument indicates any field of the diagnostics header, *RecNumber* is ignored. If not, it should be greater than 0.

DiagIdentifier

[Input]

Indicates the field of the diagnostic whose value is to be returned. For more information, see the "DiagIdentifier Argument" section in "Comments."

DiagInfoPtr

[Output]

Pointer to a buffer in which to return the diagnostic information. The data type depends on the value of *DiagIdentifier*.

BufferLength

[Input]

If *DiagIdentifier* is an ODBC-defined diagnostic and *DiagInfoPtr* points to a character string or a binary buffer, this argument should be the length of **DiagInfoPtr*. If *DiagIdentifier* is an ODBC-defined field and **DiagInfoPtr* is an integer, *BufferLength* is ignored. If the value in **DiagInfoPtr* is a Unicode string (when calling **SQLGetDiagFieldW**), the *BufferLength* argument must be an even number.

If *DiagIdentifier* is a driver-defined field, the application indicates the nature of the field to the Driver Manager by setting the *BufferLength* argument. *BufferLength* can have the following values:

- If **ValuePtr* is a pointer to a character string, then *BufferLength* is the length of the string or SQL_NTS.
- If **ValuePtr* is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(*length*) macro in *BufferLength*. This places a negative value in *BufferLength*.
- If **ValuePtr* is a pointer to a value other than a character string or binary string, then *BufferLength* should have the value SQL_IS_POINTER.
- If **ValuePtr* contains a fixed-length data type, then *BufferLength* is SQL_IS_INTEGER, SQL_IS_UINTEGER, SQL_IS_SMALLINT, or SQL_IS_USMALLINT, as appropriate.

StringLengthPtr

[Output]

Pointer to a buffer in which to return the total number of bytes (excluding the number of bytes required for the null-termination character) available to return in **DiagInfoPtr*, for character data. If the number of bytes available to return is greater than *BufferLength*, the text in **DiagInfoPtr* is truncated to *BufferLength* minus the length of a null-termination character.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, or SQL_NO_DATA.

Diagnostics

SQLGetDiagField does not post diagnostic records for itself. It uses the following return values to report the outcome of its own execution:

- SQL_SUCCESS: The function successfully returned diagnostic information.
- SQL_SUCCESS_WITH_INFO: **DiagInfoPtr* was too small to hold the requested diagnostic field, so the data in the diagnostic field was truncated. To determine that a truncation occurred, the application must compare *BufferLength* to the actual number of bytes available, which is written to **StringLengthPtr*.
- SQL_INVALID_HANDLE: The handle indicated by *HandleType* and *Handle* was not a valid handle.
- SQL_ERROR: One of the following occurred:
 - *The DiagIdentifier* argument was not one of the valid values.
 - *The DiagIdentifier* argument was SQL_DIAG_CURSOR_ROW_COUNT, SQL_DIAG_DYNAMIC_FUNCTION, SQL_DIAG_DYNAMIC_FUNCTION_CODE, or SQL_DIAG_ROW_COUNT, but *Handle* was not a statement handle. (The Driver Manager returns this diagnostic.)
 - *The RecNumber* argument was negative or 0 when *DiagIdentifier* indicated a field from a diagnostic record. *RecNumber* is ignored for header fields.
 - The value requested was a character string and *BufferLength* was less than zero.
- SQL_NO_DATA: *RecNumber* was greater than the number of diagnostic records that existed for the handle specified in *Handle*. The function also returns SQL_NO_DATA for any positive *RecNumber* if there are no diagnostic records for *Handle*.

Comments

An application typically calls **SQLGetDiagField** to accomplish one of three goals:

- To obtain specific error or warning information when a function call has returned SQL_ERROR or SQL_SUCCESS_WITH_INFO (or SQL_NEED_DATA for the **SQLBrowseConnect** function).

- To find out the number of rows in the data source that were affected when insert, delete, or update operations were performed with a call to **SQLExecute**, **SQLExecDirect**, **SQLBulkOperations**, or **SQLSetPos** (from the SQL_DIAG_ROW_COUNT header field), or to find out the number of rows that exist in the current open cursor, if the driver is able to provide this information (from the SQL_DIAG_CURSOR_ROW_COUNT header field).
- To determine which function was executed by a call to **SQLExecDirect** or **SQLExecute** (from the SQL_DIAG_DYNAMIC_FUNCTION and SQL_DIAG_DYNAMIC_FUNCTION_CODE header fields).

Any ODBC function can post zero or more diagnostic records each time it is called, so an application can call **SQLGetDiagField** after any ODBC function call. There is no limit to the number of diagnostic records that can be stored at any one time. **SQLGetDiagField** retrieves only the diagnostic information most recently associated with the diagnostic data structure specified in the *Handle* argument. If the application calls an ODBC function other than **SQLGetDiagField** or **SQLGetDiagRec**, any diagnostic information from a previous call with the same handle is lost.

An application can scan all diagnostic records by incrementing *RecNumber*, as long as **SQLGetDiagField** returns SQL_SUCCESS. The number of status records is indicated in the SQL_DIAG_NUMBER header field. Calls to **SQLGetDiagField** are nondestructive to the header and record fields. The application can call **SQLGetDiagField** again at a later time to retrieve a field from a record, as long as a function other than the diagnostic functions has not been called in the interim, which would post records on the same handle.

An application can call **SQLGetDiagField** to return any diagnostic field at any time, with the exception of SQL_DIAG_CURSOR_ROW_COUNT or SQL_DIAG_ROW_COUNT, which will return SQL_ERROR if *Handle* is not a statement handle. If any other diagnostic field is undefined, the call to **SQLGetDiagField** will return SQL_SUCCESS (provided no other diagnostic is encountered) and an undefined value is returned for the field.

For more information, see the Part I PDF file, "Using SQLGetDiagRec and SQLGetDiagField" and "Implementing SQLGetDiagField" in Chapter 15, "Diagnostics."

HandleType Argument

Each handle type can have diagnostic information associated with it. The *HandleType* argument denotes the handle type of *Handle*.

Some header and record fields cannot be returned for environment, connection, statement, and descriptor handles. Those handles for which a field is not applicable are indicated in the "Header Fields" and "Record Fields" sections following.

If *HandleType* is SQL_HANDLE_ENV, *Handle* can be either a shared or unshared environment handle.

No driver-specific header diagnostic fields should be associated with an environment handle.

The only diagnostic header fields that are defined for a descriptor handle are SQL_DIAG_NUMBER and SQL_DIAG_RETURNCODE.

DiagIdentifier Argument

This argument indicates the identifier of the field required from the diagnostic data structure. If *RecNumber* is greater than or equal to 1, the data in the field describes the diagnostic information returned by a

function. If *RecNumber* is 0, the field is in the header of the diagnostic data structure and therefore contains data pertaining to the function call that returned the diagnostic information, not to the specific information.

Drivers can define driver-specific header and record fields in the diagnostic data structure.

An ODBC 3.x application working with an ODBC 2.x driver will be able to call **SQLGetDiagField** only with a *DiagIdentifier* argument of SQL_DIAG_CLASS_ORIGIN, SQL_DIAG_CLASS_SUBCLASS_ORIGIN, SQL_DIAG_CONNECTION_NAME, SQL_DIAG_MESSAGE_TEXT, SQL_DIAG_NATIVE, SQL_DIAG_NUMBER, SQL_DIAG_RETURNCODE, SQL_DIAG_SERVER_NAME, or SQL_DIAG_SQLSTATE. All other diagnostic fields will return SQL_ERROR.

Header Fields

The header fields listed in the following table can be included in the *DiagIdentifier* argument.

DiagIdentifier	Return type	Returns
SQL_DIAG_CURSOR_ROW_COUNT	SQLINTEGER	<p>This field contains the count of rows in the cursor. Its semantics depend upon the SQLGetInfo information types SQL_DYNAMIC_CURSOR_ATTRIBUTES2, SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2, SQL_KEYSET_CURSOR_ATTRIBUTES2, and SQL_STATIC_CURSOR_ATTRIBUTES2, which indicate which row counts are available for each cursor type (in the SQL_CA2_CRC_EXACT and SQL_CA2_CRC_APPROXIMATE bits).</p> <p>The contents of this field are defined only for statement handles and only after SQLExecute, SQLExecDirect, or SQLMoreResults has been called. Calling SQLGetDiagField with a <i>DiagIdentifier</i> of SQL_DIAG_CURSOR_ROW_COUNT on other than a statement handle will return SQL_ERROR.</p>
SQL_DIAG_DYNAMIC_FUNCTION	SQLCHAR *	<p>This is a string that describes the SQL statement that the underlying function executed. (See "Values of the Dynamic Function fields," later in this section, for specific values.) The contents of this field are defined only for statement handles and only after a call to SQLExecute, SQLExecDirect, or SQLMoreResults. Calling SQLGetDiagField with a <i>DiagIdentifier</i> of SQL_DIAG_DYNAMIC_FUNCTION on other than a statement handle will return</p>

		SQL_ERROR. The value of this field is undefined before a call to SQLExecute or SQLExecDirect .
SQL_DIAG_DYNAMIC_FUNCTION_CODE	SQLINTEGER	This is a numeric code that describes the SQL statement that was executed by the underlying function. (See "Values of the Dynamic Function Fields," later in this section, for specific value.) The contents of this field are defined only for statement handles and only after a call to SQLExecute , SQLExecDirect , or SQLMoreResults . Calling SQLGetDiagField with a <i>DiagIdentifier</i> of SQL_DIAG_DYNAMIC_FUNCTION_CODE on other than a statement handle will return SQL_ERROR. The value of this field is undefined before a call to SQLExecute or SQLExecDirect .
SQL_DIAG_NUMBER	SQLINTEGER	The number of status records that are available for the specified handle.
SQL_DIAG_RETURNCODE	SQLRETURN	Return code returned by the function. For a list of return codes, see the Part I PDF file, "Return Codes" in Chapter 15, "Diagnostics." The driver does not have to implement SQL_DIAG_RETURNCODE; it is always implemented by the Driver Manager. If no function has yet been called on the <i>Handle</i> , SQL_SUCCESS will be returned for SQL_DIAG_RETURNCODE.
SQL_DIAG_ROW_COUNT	SQLINTEGER	The number of rows affected by an insert, delete, or update performed by SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos . It is driver-defined after a <i>cursor specification</i> has been executed. The contents of this field are defined only for statement handles. Calling SQLGetDiagField with a <i>DiagIdentifier</i> of SQL_DIAG_ROW_COUNT on other than a statement handle will return SQL_ERROR. The data in this field is also returned in the <i>RowCountPtr</i> argument of SQLRowCount . The data in this field is reset after every nondiagnostic function call, whereas the row count returned by SQLRowCount remains the same until the statement is set back to the prepared or allocated state.

Record Fields

The record fields listed in the following table can be included in the *DiagIdentifier* argument.

DiagIdentifier	Return type	Returns
SQL_DIAG_CLASS_ORIGIN	SQLCHAR *	A string that indicates the document that defines the class portion of the SQLSTATE value in this record. Its value is "ISO 9075" for all SQLSTATEs defined by X/Open and ISO call-level interface. For ODBC-specific SQLSTATEs (all those whose SQLSTATE class is "IM"), its value is "ODBC 3.0".
SQL_DIAG_COLUMN_NUMBER	SQLINTEGER	If the SQL_DIAG_ROW_NUMBER field is a valid row number in a rowset or a set of parameters, this field contains the value that represents the column number in the result set or the parameter number in the set of parameters. Result set column numbers always start at 1; if this status record pertains to a bookmark column, the field can be zero. Parameter numbers start at 1. It has the value SQL_NO_COLUMN_NUMBER if the status record is not associated with a column number or parameter number. If the driver cannot determine the column number or parameter number that this record is associated with, this field has the value SQL_COLUMN_NUMBER_UNKNOWN. The contents of this field are defined only for statement handles.
SQL_DIAG_CONNECTION_NAME	SQLCHAR *	A string that indicates the name of the connection that the diagnostic record relates to. This field is driver-defined. For diagnostic data structures associated with the environment handle and for diagnostics that do not relate to any connection, this field is a zero-length string.
SQL_DIAG_MESSAGE_TEXT	SQLCHAR *	An informational message on the error or warning. This field is formatted as described in the Part I PDF file, "Diagnostic Messages" section of Chapter 15, "Diagnostics." There is no maximum length to the diagnostic message text.
SQL_DIAG_NATIVE	SQLINTEGER	A driver/data source-specific native error code. If there is no native error code, the driver returns 0.
SQL_DIAG_ROW_NUMBER	SQLINTEGER	This field contains the row number in the rowset, or the parameter number in the set of parameters, with which the status record is associated. Row numbers and parameter numbers start with 1. This field has the

		<p>value SQL_NO_ROW_NUMBER if this status record is not associated with a row number or parameter number. If the driver cannot determine the row number or parameter number that this record is associated with, this field has the value SQL_ROW_NUMBER_UNKNOWN.</p> <p>The contents of this field are defined only for statement handles.</p>
SQL_DIAG_SERVER_NAME	SQLCHAR *	<p>A string that indicates the server name that the diagnostic record relates to. It is the same as the value returned for a call to SQLGetInfo with the SQL_DATA_SOURCE_NAME option. For diagnostic data structures associated with the environment handle and for diagnostics that do not relate to any server, this field is a zero-length string.</p>
SQL_DIAG_SQLSTATE	SQLCHAR *	<p>A five-character SQLSTATE diagnostic code. For more information, see Part I PDF file, "SQLSTATES" in Chapter 15, "Diagnostics."</p>
SQL_DIAG_SUBCLASS_ORIGIN	SQLCHAR *	<p>A string with the same format and valid values as SQL_DIAG_CLASS_ORIGIN, that identifies the defining portion of the subclass portion of the SQLSTATE code. The ODBC-specific SQLSTATES for which "ODBC 3.0" is returned include the following:</p> <p>01S00, 01S01, 01S02, 01S06, 01S07, 07S01, 08S01, 21S01, 21S02, 25S01, 25S02, 25S03, 42S01, 42S02, 42S11, 42S12, 42S21, 42S22, HY095, HY097, HY098, HY099, HY100, HY101, HY105, HY107, HY109, HY110, HY111, HYT00, HYT01, IM001, IM002, IM003, IM004, IM005, IM006, IM007, IM008, IM010, IM011, IM012.</p>

Values of the Dynamic Function Fields

The following table describes the values of SQL_DIAG_DYNAMIC_FUNCTION and SQL_DIAG_DYNAMIC_FUNCTION_CODE that apply to each type of SQL statement executed by a call to **SQLExecute** or **SQLExecDirect**. The driver can add driver-defined values to those listed.

SQL statement executed	Value of SQL_DIAG_DYNAMIC_FUNCTION	Value of SQL_DIAG_DYNAMIC_FUNCTION_CODE
------------------------	------------------------------------	-----------------------------------------

alter-domain-statement	"ALTER DOMAIN"	SQL_DIAG_ALTER_DOMAIN
alter-table-statement	"ALTER TABLE"	SQL_DIAG_ALTER_TABLE
assertion-definition	"CREATE ASSERTION"	SQL_DIAG_CREATE_ASSERTION
character-set-definition	"CREATE CHARACTER SET"	SQL_DIAG_CREATE_CHARACTER_SET
collation-definition	"CREATE COLLATION"	SQL_DIAG_CREATE_COLLATION
create-index-statement	"CREATE INDEX"	SQL_DIAG_CREATE_INDEX
create-table-statement	"CREATE TABLE"	SQL_DIAG_CREATE_TABLE
create-view-statement	"CREATE VIEW"	SQL_DIAG_CREATE_VIEW
cursor-specification	"SELECT CURSOR"	SQL_DIAG_SELECT_CURSOR
delete-statement-positioned	"DYNAMIC DELETE CURSOR"	SQL_DIAG_DYNAMIC_DELETE_CURSOR
delete-statement-searched	"DELETE WHERE"	SQL_DIAG_DELETE_WHERE
domain-definition	"CREATE DOMAIN"	SQL_DIAG_CREATE_DOMAIN
drop-assertion-statement	"DROP ASSERTION"	SQL_DIAG_DROP_ASSERTION
drop-character-set-stmt	"DROP CHARACTER SET"	SQL_DIAG_DROP_CHARACTER_SET
drop-collation-statement	"DROP COLLATION"	SQL_DIAG_DROP_COLLATION
drop-domain-statement	"DROP DOMAIN"	SQL_DIAG_DROP_DOMAIN
drop-index-statement	"DROP INDEX"	SQL_DIAG_DROP_INDEX
drop-schema-statement	"DROP SCHEMA"	SQL_DIAG_DROP_SCHEMA
drop-table-statement	"DROP TABLE"	SQL_DIAG_DROP_TABLE
drop-translation-statement	"DROP TRANSLATION"	SQL_DIAG_DROP_TRANSLATION
drop-view-statement	"DROP VIEW"	SQL_DIAG_DROP_VIEW
grant-statement	"GRANT"	SQL_DIAG_GRANT
insert-statement	"INSERT"	SQL_DIAG_INSERT

ODBC-procedure-extension	"CALL"	SQL_DIAG_CALL
revoke-statement	"REVOKE"	SQL_DIAG_REVOKE
schema-definition	"CREATE SCHEMA"	SQL_DIAG_CREATE_SCHEMA
translation-definition	"CREATE TRANSLATION"	SQL_DIAG_CREATE_TRANSLATION
update-statement-positioned	"DYNAMIC UPDATE CURSOR"	SQL_DIAG_DYNAMIC_UPDATE_CURSOR
update-statement-searched	"UPDATE WHERE"	SQL_DIAG_UPDATE_WHERE
Unknown	empty string	SQL_DIAG_UNKNOWN_STATEMENT

Sequence of Status Records

Status records are placed in a sequence based on row number and the type of the diagnostic. The Driver Manager determines the final order in which to return status records that it generates. The driver determines the final order in which to return status records that it generates.

If diagnostic records are posted by both the Driver Manager and the driver, the Driver Manager is responsible for ordering them.

If there are two or more status records, the sequence of the records is determined first by row number. The following rules apply to determining the sequence of diagnostic records by row:

- Records that do not correspond to any row appear in front of records that correspond to a particular row, because `SQL_NO_ROW_NUMBER` is defined to be `-1`.
- Records for which the row number is unknown appear in front of all other records, because `SQL_ROW_NUMBER_UNKNOWN` is defined to be `-2`.
- For all records that pertain to specific rows, records are sorted by the value in the `SQL_DIAG_ROW_NUMBER` field. All errors and warnings of the first row affected are listed, then all errors and warnings of the next row affected, and so on.

Note The ODBC 3.x Driver Manager does not order status records in the diagnostic queue if `SQLSTATE 01S01` (Error in row) is returned by an ODBC 2.x driver or if `SQLSTATE 01S01` (Error in row) is returned by an ODBC 3.x driver when **SQLExtendedFetch** is called or **SQLSetPos** is called on a cursor that has been positioned with **SQLExtendedFetch**.

Within each row, or for all those records that do not correspond to a row or for which the row number is unknown, or for all those records with a row number equal to `SQL_NO_ROW_NUMBER`, the first record listed is determined using a set of sorting rules. After the first record, the order of the other records affecting a row is undefined. An application cannot assume that errors precede warnings after the first

record. Applications should scan the entire diagnostic data structure to obtain complete information on an unsuccessful call to a function.

The following rules are used to determine the first record within a row. The record with the highest rank is the first record. The source of a record (Driver Manager, driver, gateway, and so on) is not considered when ranking records.

Errors. Status records that describe errors have the highest rank. The following rules are applied to sort errors:

- Records that indicate a transaction failure or possible transaction failure outrank all other records.

If two or more records describe the same error condition, then SQLSTATEs defined by the X/Open CLI specification (classes 03 through HZ) outrank ODBC- and driver-defined SQLSTATEs.

- Implementation-defined No Data values.** Status records that describe driver-defined No Data values (class 02) have the second highest rank.

- Warnings.** Status records that describe warnings (class 01) have the lowest rank. If two or more records describe the same warning condition, then warning SQLSTATEs defined by the X/Open CLI specification outrank ODBC-defined and driver-defined SQLSTATEs.

Related Functions

For information about	See
Obtaining multiple fields of a diagnostic data structure	SQLGetDiagRec

SQLGetDiagRec

Conformance

Version Introduced: ODBC 3.0

Standards Compliance: ISO 92

Summary

SQLGetDiagRec returns the current values of multiple fields of a diagnostic record that contains error, warning, and status information. Unlike **SQLGetDiagField**, which returns one diagnostic field per call, **SQLGetDiagRec** returns several commonly used fields of a diagnostic record, including the SQLSTATE, the native error code, and the diagnostic message text.

Syntax

SQLRETURN SQLGetDiagRec(
SQLSMALLINT	HandleType,
SQLHANDLE	Handle,
SQLSMALLINT	RecNumber,
SQLCHAR *	Sqlstate,
SQLINTEGER *	NativeErrorPtr,

SQLCHAR *	MessageText,
SQLSMALLINT	BufferLength,
SQLSMALLINT *	TextLengthPtr);

Arguments

HandleType

[Input]

A handle type identifier that describes the type of handle for which diagnostics are required. Must be one of the following:

SQL_HANDLE_ENV
SQL_HANDLE_DBC
SQL_HANDLE_STMT
SQL_HANDLE_DESC

Handle

[Input]

A handle for the diagnostic data structure, of the type indicated by *HandleType*. If *HandleType* is SQL_HANDLE_ENV, *Handle* can be either a shared or an unshared environment handle.

RecNumber

[Input]

Indicates the status record from which the application seeks information. Status records are numbered from 1.

SQLState

[Output]

Pointer to a buffer in which to return a five-character SQLSTATE code pertaining to the diagnostic record *RecNumber*. The first two characters indicate the class; the next three indicate the subclass. This information is contained in the SQL_DIAG_SQLSTATE diagnostic field. For more information, see the Part I PDF file, "SQLSTATES" in Chapter 15, "Diagnostics."

NativeErrorPtr

[Output]

Pointer to a buffer in which to return the native error code, specific to the data source. This information is contained in the SQL_DIAG_NATIVE diagnostic field.

MessageText

[Output]

Pointer to a buffer in which to return the diagnostic message text string. This information is contained in the SQL_DIAG_MESSAGE_TEXT diagnostic field. For the format of the string, see the Part I PDF file, "Diagnostic Messages" section of Chapter 15, "Diagnostics."

BufferLength

[Input]

Length (in bytes) of the **MessageText* buffer. There is no maximum length of the diagnostic message text. If the value returned in **MessageText* is of a Unicode string (when calling **SQLGetDiagRecW**), the *BufferLength* argument must be an even number.

TextLengthPtr

[Output]

Pointer to a buffer in which to return the total number of bytes (excluding the number of bytes required for the null-termination character) available to return in **MessageText*. If the number of bytes available to return is greater than *BufferLength*, the diagnostic message text in **MessageText* is truncated to *BufferLength* minus the length of a null-termination character.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

SQLGetDiagRec does not post diagnostic records for itself. It uses the following return values to report the outcome of its own execution:

- SQL_SUCCESS: The function successfully returned diagnostic information.
- SQL_SUCCESS_WITH_INFO: The **MessageText* buffer was too small to hold the requested diagnostic message. No diagnostic records were generated. To determine that a truncation occurred, the application must compare *BufferLength* to the actual number of bytes available, which is written to **StringLengthPtr*.
- SQL_INVALID_HANDLE: The handle indicated by *HandleType* and *Handle* was not a valid handle.
- SQL_ERROR: One of the following occurred:
 - *RecNumber* was negative or 0.
 - *BufferLength* was less than zero.
- SQL_NO_DATA: *RecNumber* was greater than the number of diagnostic records that existed for the handle specified in *Handle*. The function also returns SQL_NO_DATA for any positive *RecNumber* if there are no diagnostic records for *Handle*.

Comments

An application typically calls **SQLGetDiagRec** when a previous call to an ODBC function has returned SQL_SUCCESS or SQL_SUCCESS_WITH_INFO. However, because any ODBC function can post zero or more diagnostic records each time it is called, an application can call **SQLGetDiagRec** after any ODBC function call. An application can call **SQLGetDiagRec** multiple times to return some or all of the records in the diagnostic data structure. ODBC imposes no limit to the number of diagnostic records that can be stored at any one time.

SQLGetDiagRec cannot be used to return fields from the header of the diagnostic data structure. (The *RecNumber* argument must be greater than 0.) The application should call **SQLGetDiagField** for this purpose.

SQLGetDiagRec retrieves only the diagnostic information most recently associated with the handle specified in the *Handle* argument. If the application calls another ODBC function, except **SQLGetDiagRec**, **SQLGetDiagField**, or **SQLError**, any diagnostic information from the previous calls on the same handle is lost.

An application can scan all diagnostic records by looping, incrementing *RecNumber*, as long as **SQLGetDiagRec** returns SQL_SUCCESS. Calls to **SQLGetDiagRec** are nondestructive to the header and record fields. The application can call **SQLGetDiagRec** again at a later time to retrieve a field from a record as long as no other function, except **SQLGetDiagRec**, **SQLGetDiagField**, or **SQLError**, has been called in the interim. The application can also retrieve a count of the total number of diagnostic records available by calling **SQLGetDiagField** to retrieve the value of the SQL_DIAG_NUMBER field, and then calling **SQLGetDiagRec** that many times.

For a description of the fields of the diagnostic data structure, see **SQLGetDiagField**. For more information, see the Part I PDF file, "Using SQLGetDiagRec and SQLGetDiagField" and "Implementing SQLGetDiagRec and SQLGetDiagField" in Chapter 15, "Diagnostics."

HandleType Argument

Each handle type can have diagnostic information associated with it. The *HandleType* argument denotes the handle type of the *Handle* argument.

Some header and record fields cannot be returned for environment, connection, statement, and descriptor handles. Those handles for which a field is not applicable are indicated in the "Header Fields" and "Record Fields" sections in [SQLGetDiagField](#).

A call to **SQLGetDiagRec** will return SQL_INVALID_HANDLE if *HandleType* is SQL_HANDLE_SENV, which denotes a shared environment handle. However, if *HandleType* is SQL_HANDLE_ENV, *Handle* can be either a shared or an unshared environment handle.

Related Functions

For information about	See
Obtaining a field of a diagnostic record or a field of the diagnostic header	SQLGetDiagField

SQLGetEnvAttr

Conformance

Version Introduced: ODBC 3.0
Standards Compliance: ISO 92

Summary

SQLGetEnvAttr returns the current setting of an environment attribute.

Syntax

SQLRETURN SQLGetEnvAttr(
SQLHENV	EnvironmentHandle,
SQLINTEGER	Attribute,
SQLPOINTER	ValuePtr,
SQLINTEGER	BufferLength,
SQLINTEGER *	StringLengthPtr);

Arguments

EnvironmentHandle

[Input]

Environment handle.

Attribute

[Input]

Attribute to retrieve.

ValuePtr

[Output]

Pointer to a buffer in which to return the current value of the attribute specified by *Attribute*.

BufferLength

[Input]

If *ValuePtr* points to a character string, this argument should be the length of **ValuePtr*. If **ValuePtr* is an integer, *BufferLength* is ignored. If **ValuePtr* is a Unicode string (when calling **SQLGetEnvAttrW**), the *BufferLength* argument must be an even number. If the attribute value is not a character string, *BufferLength* is unused.

StringLengthPtr

[Output]

A pointer to a buffer in which to return the total number of bytes (excluding the null-termination character) available to return in **ValuePtr*. If *ValuePtr* is a null pointer, no length is returned. If the attribute value is a character string and the number of bytes available to return is greater than or equal to *BufferLength*, the data in **ValuePtr* is truncated to *BufferLength* minus the length of a null-termination character and is null-terminated by the driver.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLGetEnvAttr** returns `SQL_ERROR` or `SQL_SUCCESS_WITH_INFO`, an associated `SQLSTATE` value can be obtained by calling **SQLGetDiagRec** with a *HandleType* of `SQL_HANDLE_ENV` and a *Handle* of *EnvironmentHandle*. The following table lists the `SQLSTATE` values commonly returned by **SQLGetEnvAttr** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of `SQLSTATE`s 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 <code>SQL_SUCCESS_WITH_INFO</code> .)
01004	String data, right truncated	The data returned in <i>*ValuePtr</i> was truncated to be <i>BufferLength</i> minus the null-termination character. The length of the untruncated string value is returned in <i>*StringLengthPtr</i> . (Function returns <code>SQL_SUCCESS_WITH_INFO</code> .)
HY000	General error	An error occurred for which there was no specific <code>SQLSTATE</code> and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY092	Invalid attribute/option identifier	The value specified for the argument <i>Attribute</i> was not valid for the version of ODBC supported by the driver.
HYC00	Optional feature not implemented	The value specified for the argument <i>Attribute</i> was a valid ODBC environment attribute for the version of ODBC supported by the driver but was not supported by the driver.
IM001	Driver does not support this function	(DM) The driver corresponding to the <i>EnvironmentHandle</i> does not support the function.

Comments

For a list of attributes, see [SQLSetEnvAttr](#). There are no driver-specific environment attributes. If *Attribute* specifies an attribute that returns a string, *ValuePtr* must be a pointer to a buffer in which to return the string. The maximum length of the string, including the null-termination byte, will be *BufferLength* bytes.

SQLGetEnvAttr can be called at any time between the allocation and the freeing of an environment handle. All environment attributes successfully set by the application for the environment persist until **SQLFreeHandle** is called on the *EnvironmentHandle* with a *HandleType* of `SQL_HANDLE_ENV`. More than one environment handle can be allocated simultaneously in ODBC 3.x. An environment attribute on one environment is not affected when another environment has been allocated.

Note The `SQL_ATTR_OUTPUT_NTS` environment attribute is supported by standards-compliant applications. When **SQLGetEnvAttr** is called, the ODBC 3.x Driver Manager always returns `SQL_TRUE`

for this attribute. SQL_ATTR_OUTPUT_NTS can be set to SQL_TRUE only by a call to **SQLSetEnvAttr**.

Related Functions

For information about	See
Returning the setting of a connection attribute	SQLGetConnectAttr
Returning the setting of a statement attribute	SQLGetStmtAttr
Setting a connection attribute	SQLSetConnectAttr
Setting an environment attribute	SQLSetEnvAttr
Setting a statement attribute	SQLSetStmtAttr

SQLGetFunctions

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

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

SQLRETURN SQLGetFunctions(
SQLHDBC	ConnectionHandle,
SQLUSMALLINT	FunctionId,
SQLUSMALLINT *	SupportedPtr);

Arguments

ConnectionHandle

[Input]

Connection handle.

FunctionId

[Input]

A **#define** value that identifies the ODBC function of interest; SQL_API_ODBC3_ALL_FUNCTIONS or SQL_API_ALL_FUNCTIONS. SQL_API_ODBC3_ALL_FUNCTIONS is used by an ODBC 3.x application to determine support of ODBC 3.x and earlier functions. SQL_API_ALL_FUNCTIONS is used by an ODBC 2.x application to determine support of ODBC 2.x and earlier functions.

For a list of **#define** values that identify ODBC functions, see the tables in "Comments."

SupportedPtr

[Output]

If *FunctionId* identifies a single ODBC function, *SupportedPtr* points to a single **SQLUSMALLINT** value that is **SQL_TRUE** if the specified function is supported by the driver, and **SQL_FALSE** if it is not supported.

If *FunctionId* is **SQL_API_ODBC3_ALL_FUNCTIONS**, *SupportedPtr* points to a **SQLSMALLINT** array with a number of elements equal to **SQL_API_ODBC3_ALL_FUNCTIONS_SIZE**. This array is treated by the Driver Manager as a 4,000-bit bitmap that can be used to determine whether an ODBC 3.x or earlier function is supported. The **SQL_FUNC_EXISTS** macro is called to determine function support. (See "Comments.") An ODBC 3.x application can call **SQLGetFunctions** with **SQL_API_ODBC3_ALL_FUNCTIONS** against either an ODBC 3.x or ODBC 2.x driver.

If *FunctionId* is **SQL_API_ALL_FUNCTIONS**, *SupportedPtr* points to an **SQLUSMALLINT** array of 100 elements. The array is indexed by **#define** values used by *FunctionId* to identify each ODBC function; some elements of the array are unused and reserved for future use. An element is **SQL_TRUE** if it identifies an ODBC 2.x or earlier function supported by the driver. It is **SQL_FALSE** if it identifies an ODBC function not supported by the driver or does not identify an ODBC function.

The arrays returned in **SupportedPtr* use zero-based indexing.

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of **SQL_HANDLE_DBC** and a *Handle* of *ConnectionHandle*. 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 **SQLSTATE**s 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.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error	(DM) SQLGetFunctions was called before

		SQLConnect, SQLBrowseConnect, or SQLDriverConnect. (DM) SQLBrowseConnect was called for the <i>ConnectionHandle</i> and returned SQL_NEED_DATA. This function was called before SQLBrowseConnect returned SQL_SUCCESS_WITH_INFO or SQL_SUCCESS.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY095	Function type out of range	(DM) An invalid <i>FunctionId</i> value was specified.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.

Comments

SQLGetFunctions always returns that **SQLGetFunctions**, **SQLDataSources**, and **SQLDrivers** are supported. It does this because these functions are implemented in the Driver Manager. The Driver Manager will map an ANSI function to the corresponding Unicode function if the Unicode function exists and will map a Unicode function to the corresponding ANSI function if the ANSI function exists. For information about how applications use **SQLGetFunctions**, see the Part I PDF file, "Interface Conformance Levels" in Chapter 4, "ODBC Fundamentals."

The following is a list of valid values for *FunctionId* for functions that conform to the ISO 92 standards-compliance level:

SQL_API_SQLALLOCHANDLE	SQL_API_SQLGETDESCFIELD
SQL_API_SQLBINDCOL	SQL_API_SQLGETDESCREC
SQL_API_SQLCANCEL	SQL_API_SQLGETDIAGFIELD
SQL_API_SQLCLOSECURSOR	SQL_API_SQLGETDIAGREC
SQL_API_SQLCOLATTRIBUTE	SQL_API_SQLGETENVATTR
SQL_API_SQLCONNECT	SQL_API_SQLGETFUNCTIONS
SQL_API_SQLCOPYDESC	SQL_API_SQLGETINFO
SQL_API_SQLDATASOURCES	SQL_API_SQLGETSTMTATTR
SQL_API_SQLDESCRIBECOL	SQL_API_SQLGETTYPEINFO
SQL_API_SQLDISCONNECT	SQL_API_SQLNUMRESULTCOLS
SQL_API_SQLDRIVERS	SQL_API_SQLPARAMDATA
SQL_API_SQLENDTRAN	SQL_API_SQLPREPARE
SQL_API_SQLEXECDIRECT	SQL_API_SQLPUTDATA
SQL_API_SQLEXECUTE	SQL_API_SQLROWCOUNT
SQL_API_SQLFETCH	SQL_API_SQLSETCONNECTATTR
SQL_API_SQLFETCHSCROLL	SQL_API_SQLSETCURSORNAME
SQL_API_SQLFREEHANDLE	SQL_API_SQLSETDESCFIELD

SQL_API_SQLFREESTMT	SQL_API_SQLSETDESCREC
SQL_API_SQLGETCONNECTATTR	SQL_API_SQLSETENVATTR
SQL_API_SQLGETCURSORNAME	SQL_API_SQLSETSTMTATTR
SQL_API_SQLGETDATA	

The following is a list of valid values for *FunctionId* for functions conforming to the X/Open standards–compliance level:

SQL_API_SQLCOLUMNS	SQL_API_SQLSTATISTICS
SQL_API_SQLSPECIALCOLUMNS	SQL_API_SQLTABLES

The following is a list of valid values for *FunctionId* for functions conforming to the ODBC standards–compliance level.

SQL_API_SQLBINDPARAMETER	SQL_API_SQLNATIVESQL
SQL_API_SQLBROWSECONNECT	SQL_API_SQLNUMPARAMS
SQL_API_SQLBULKOPERATIONS [1]	SQL_API_SQLPRIMARYKEYS
SQL_API_SQLCOLUMNPRIVILEGES	SQL_API_SQLPROCEDURECOLUMNS
SQL_API_SQLDESCRIBEPARAM	SQL_API_SQLPROCEDURES
SQL_API_SQLDRIVERCONNECT	SQL_API_SQLSETPOS
SQL_API_SQLFOREIGNKEYS	SQL_API_SQLTABLEPRIVILEGES
SQL_API_SQLMORERESULTS	

[1] When working with an ODBC 2.x driver, **SQLBulkOperations** will be returned as supported only if both of the following are true: the ODBC 2.x driver supports **SQLSetPos**, and the information type SQL_POS_OPERATIONS returns the SQL_POS_ADD bit as set.

SQL_FUNC_EXISTS Macro

The SQL_FUNC_EXISTS(*SupportedPtr*, *FunctionID*) macro is used to determine support of ODBC 3.x or earlier functions after **SQLGetFunctions** has been called with a *FunctionId* argument of SQL_API_ODBC3_ALL_FUNCTIONS. The application calls SQL_FUNC_EXISTS with the *SupportedPtr* argument set to the *SupportedPtr* passed in *SQLGetFunctions*, and with the *FunctionID* argument set to the **#define** for the function. SQL_FUNC_EXISTS returns SQL_TRUE if the function is supported, and SQL_FALSE otherwise.

Note When working with an ODBC 2.x driver, the ODBC 3.x Driver Manager will return SQL_TRUE for **SQLAllocHandle** and **SQLFreeHandle** because **SQLAllocHandle** is mapped to **SQLAllocEnv**, **SQLAllocConnect**, or **SQLAllocStmt**, and because **SQLFreeHandle** is mapped to **SQLFreeEnv**, **SQLFreeConnect**, or **SQLFreeStmt**. **SQLAllocHandle** or **SQLFreeHandle** with a *HandleType* argument of SQL_HANDLE_DESC is not supported, however, even though SQL_TRUE is returned for the functions, because there is no ODBC 2.x function to map to in this case.

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.

```

SQLUSMALLINT 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.

```

#define FUNCTIONS 100

SQLUSMALLINT fExists[FUNCTIONS];

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

For information about	See
Returning the setting of a connection attribute	SQLGetConnectAttr
Returning information about a driver or data source	SQLGetInfo
Returning the setting of a statement attribute	SQLGetStmtAttr

SQLGetInfo

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: ISO 92

Summary

SQLGetInfo returns general information about the driver and data source associated with a connection.

Syntax

SQLRETURN SQLGetInfo(
-----------------------	--

SQLHDBC	ConnectionHandle,
SQLUSMALLINT	InfoType,
SQLPOINTER	InfoValuePtr,
SQLSMALLINT	BufferLength,
SQLSMALLINT *	StringLengthPtr);

Arguments

ConnectionHandle

[Input]
Connection handle.

InfoType

[Input]
Type of information.

InfoValuePtr

[Output]
Pointer to a buffer in which to return the information. Depending on the *InfoType* requested, the information returned will be one of the following: a null-terminated character string, an SQLUSMALLINT value, an SQLINTEGER bitmask, an SQLINTEGER flag, or a SQLINTEGER binary value.

If the *InfoType* argument is SQL_DRIVER_HDESC or SQL_DRIVER_HSTMT, the *InfoValuePtr* argument is both input and output. (See the SQL_DRIVER_HDESC or SQL_DRIVER_HSTMT descriptors later in this function description for more information.)

BufferLength

[Input]
Length of the **InfoValuePtr* buffer. If the value in **InfoValuePtr* is not a character string or if *InfoValuePtr* is a null pointer, the *BufferLength* argument is ignored. The driver assumes that the size of **InfoValuePtr* is SQLUSMALLINT or SQLINTEGER, based on the *InfoType*. If **InfoValuePtr* is a Unicode string (when calling **SQLGetInfoW**), the *BufferLength* argument must be an even number; if not, SQLSTATE HY090 (Invalid string or buffer length) is returned.

StringLengthPtr

[Output]
Pointer to a buffer in which to return the total number of bytes (excluding the null-termination character for character data) available to return in **InfoValuePtr*.

For character data, if the number of bytes available to return is greater than or equal to *BufferLength*, the information in **InfoValuePtr* is truncated to *BufferLength* bytes minus the length of a null-termination character and is null-terminated by the driver.

For all other types of data, the value of *BufferLength* is ignored and the driver assumes the size of **InfoValuePtr* is SQLUSMALLINT or SQLINTEGER, depending on the *InfoType*.

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_DBC and a *Handle* of *ConnectionHandle*. 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	String data, right truncated	The buffer <i>*InfoValuePtr</i> was not large enough to return all of the requested information, so the information was truncated. The length of the requested information in its untruncated form is returned in <i>*StringLengthPtr</i> . (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection does not exist	(DM) The type of information requested in <i>InfoType</i> requires an open connection. Of the information types reserved by ODBC, only SQL_ODBC_VER can be returned without an open connection.
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.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY024	Invalid attribute value	(DM) The <i>InfoType</i> argument was SQL_DRIVER_HSTMT, and the value pointed to by <i>InfoValuePtr</i> was not a valid statement handle. (DM) The <i>InfoType</i> argument was SQL_DRIVER_HDESC, and the value pointed to by <i>InfoValuePtr</i> was not a valid descriptor handle.
HY090	Invalid string or buffer length	(DM) The value specified for argument <i>BufferLength</i> was less than 0. (DM) The value specified for <i>BufferLength</i> was an

		odd number, and <i>*InfoValuePtr</i> was of a Unicode data type.
HY096	Information type out of range	The value specified for the argument <i>InfoType</i> was not valid for the version of ODBC supported by the driver.
HYC00	Optional field not implemented	The value specified for the argument <i>InfoType</i> was a driver-specific value that is not supported by the driver.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver corresponding to the <i>ConnectionHandle</i> does not support the function.

Comments

The currently defined information types are shown in "Information Types," later in this section; it is expected that more will be defined to take advantage of different data sources. A range of information types is reserved by ODBC; driver developers must reserve values for their own driver-specific use from X/Open. **SQLGetInfo** performs no Unicode conversion or *thunking* (see Appendix A of the *ODBC Programmer's Reference*) for driver-defined *InfoTypes*. For more information, see the Part I PDF file, "Driver-Specific Data Types, Descriptor Types, Information Types, Diagnostic Types, and Attributes" in Chapter 17, "Programming Considerations." The format of the information returned in **InfoValuePtr* depends on the *InfoType* requested. **SQLGetInfo** will return information in one of five different formats:

- A null-terminated character string
- An SQLUSMALLINT value
- An SQLINTEGER bitmask
- An SQLINTEGER value
- A SQLINTEGER 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 **InfoValuePtr* accordingly. For an example of how an application could retrieve data from a SQLINTEGER bitmask, see "Code Example."

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

Format of <i>*InfoValuePtr</i>	Returned value
Character string ("Y" or "N")	"N"
Character string (not "Y" or "N")	Empty string
SQLUSMALLINT	0
SQLINTEGER bitmask or SQLINTEGER binary value	0L

For example, if a data source does not support procedures, **SQLGetInfo** returns the values listed in the following table for the values of *InfoType* that are related to procedures.

InfoType	Returned value
SQL_PROCEDURES	"N"
SQL_ACCESSIBLE_PROCEDURES	"N"
SQL_MAX_PROCEDURE_NAME_LEN	0
SQL_PROCEDURE_TERM	Empty string

SQLGetInfo returns SQLSTATE HY096 (Invalid argument value) for values of *InfoType* 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 HYC00 (Optional feature not implemented) for values of *InfoType* that are in the range of information types reserved for driver-specific use but are not supported by the driver.

All calls to **SQLGetInfo** require an open connection, except when the *InfoType* is SQL_ODBC_VER, which returns the version of the Driver Manager.

Information Types

This section lists the information types supported by **SQLGetInfo**. Information types are grouped categorically and listed alphabetically. Information types that were added or renamed for ODBC 3.x are also listed.

Driver Information

The following values of the *InfoType* argument return information about the ODBC driver, such as the number of active statements, the data source name, and the interface standards compliance level:

SQL_ACTIVE_ENVIRONMENTS	SQL_GETDATA_EXTENSIONS
SQL_ASYNC_MODE	SQL_INFO_SCHEMA_VIEWS
SQL_BATCH_ROW_COUNT	SQL_KEYSET_CURSOR_ATTRIBUTES1
SQL_BATCH_SUPPORT	SQL_KEYSET_CURSOR_ATTRIBUTES2
SQL_DATA_SOURCE_NAME	SQL_MAX_ASYNC_CONCURRENT_STATEMENTS
SQL_DRIVER_HDBC	SQL_MAX_CONCURRENT_ACTIVITIES
SQL_DRIVER_HDESC	SQL_MAX_DRIVER_CONNECTIONS
SQL_DRIVER_HENV	SQL_ODBC_INTERFACE_CONFORMANCE
SQL_DRIVER_HLIB	SQL_ODBC_STANDARD_CLI_CONFORMANCE
SQL_DRIVER_HSTMT	SQL_ODBC_VER
SQL_DRIVER_NAME	SQL_PARAM_ARRAY_ROW_COUNTS
SQL_DRIVER_ODBC_VER	SQL_PARAM_ARRAY_SELECTS
SQL_DRIVER_VER	SQL_ROW_UPDATES
SQL_DYNAMIC_CURSOR_ATTRIBUTES1	SQL_SEARCH_PATTERN_ESCAPE
SQL_DYNAMIC_CURSOR_ATTRIBUTES2	SQL_SERVER_NAME
SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1	SQL_STATIC_CURSOR_ATTRIBUTES1

BUTES1	
SQL_FORWARD_ONLY_CURSOR_ATTRI BUTES2	SQL_STATIC_CURSOR_ATTRIBUTES2
SQL_FILE_USAGE	

DBMS Product Information

The following values of the *InfoType* argument 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 the *InfoType* argument return information about the data source, such as cursor characteristics and transaction capabilities:

SQL_ACCESSIBLE_PROCEDURES	SQL_MULT_RESULT_SETS
SQL_ACCESSIBLE_TABLES	SQL_MULTIPLE_ACTIVE_TXN
SQL_BOOKMARK_PERSISTENCE	SQL_NEED_LONG_DATA_LEN
SQL_CATALOG_TERM	SQL_NULL_COLLATION
SQL_COLLATION_SEQ	SQL_PROCEDURE_TERM
SQL_CONCAT_NULL_BEHAVIOR	SQL_SCHEMA_TERM
SQL_CURSOR_COMMIT_BEHAVIOR	SQL_SCROLL_OPTIONS
SQL_CURSOR_ROLLBACK_BEHAVIO R	SQL_TABLE_TERM
SQL_CURSOR_SENSITIVITY	SQL_TXN_CAPABLE
SQL_DATA_SOURCE_READ_ONLY	SQL_TXN_ISOLATION_OPTION
SQL_DEFAULT_TXN_ISOLATION	SQL_USER_NAME
SQL_DESCRIBE_PARAMETER	

Supported SQL

The following values of the *InfoType* argument return information about the SQL statements supported by the data source. The SQL syntax of each feature described by these information types is the SQL-92 syntax. These information types do not exhaustively describe the entire SQL-92 grammar. Instead, they describe those parts of the grammar for which data sources commonly offer different levels of support. Specifically, most of the DDL statements in SQL-92 are covered.

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

SQLAggregate_FUNCTIONS	SQL_DROP_TABLE
SQL_ALTER_DOMAIN	SQL_DROP_TRANSLATION

SQL_ALTER_SCHEMA	SQL_DROP_VIEW
SQL_ALTER_TABLE	SQL_EXPRESSIONS_IN_ORDERBY
SQL_ANSI_SQL_DATETIME_LITERALS	SQL_GROUP_BY
SQL_CATALOG_LOCATION	SQL_IDENTIFIER_CASE
SQL_CATALOG_NAME	SQL_IDENTIFIER_QUOTE_CHAR
SQL_CATALOG_NAME_SEPARATOR	SQL_INDEX_KEYWORDS
SQL_CATALOG_USAGE	SQL_INSERT_STATEMENT
SQL_COLUMN_ALIAS	SQL_INTEGRITY
SQL_CORRELATION_NAME	SQL_KEYWORDS
SQL_CREATE_ASSERTION	SQL_LIKE_ESCAPE_CLAUSE
SQL_CREATE_CHARACTER_SET	SQL_NON_NULLABLE_COLUMNS
SQL_CREATE_COLLATION	SQL_SQL_CONFORMANCE
SQL_CREATE_DOMAIN	SQL_OJ_CAPABILITIES
SQL_CREATE_SCHEMA	SQL_ORDER_BY_COLUMNS_IN_SELECT
SQL_CREATE_TABLE	SQL_OUTER_JOINS
SQL_CREATE_TRANSLATION	SQL_PROCEDURES
SQL_DDL_INDEX	SQL_QUOTED_IDENTIFIER_CASE
SQL_DROP_ASSERTION	SQL_SCHEMA_USAGE
SQL_DROP_CHARACTER_SET	SQL_SPECIAL_CHARACTERS
SQL_DROP_COLLATION	SQL_SUBQUERIES
SQL_DROP_DOMAIN	SQL_UNION
SQL_DROP_SCHEMA	

SQL Limits

The following values of the *InfoType* argument 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 can be imposed by either the driver or the data source.

SQL_MAX_BINARY_LITERAL_LEN	SQL_MAX_IDENTIFIER_LEN
SQL_MAX_CATALOG_NAME_LEN	SQL_MAX_INDEX_SIZE
SQL_MAX_CHAR_LITERAL_LEN	SQL_MAX_PROCEDURE_NAME_LEN
SQL_MAX_COLUMN_NAME_LEN	SQL_MAX_ROW_SIZE
SQL_MAX_COLUMNS_IN_GROUP_BY	SQL_MAX_ROW_SIZE_INCLUDES_LONG
SQL_MAX_COLUMNS_IN_INDEX	SQL_MAX_SCHEMA_NAME_LEN
SQL_MAX_COLUMNS_IN_ORDER_BY	SQL_MAX_STATEMENT_LEN
SQL_MAX_COLUMNS_IN_SELECT	SQL_MAX_TABLE_NAME_LEN

SQL_MAX_COLUMNS_IN_TABLE	SQL_MAX_TABLES_IN_SELECT
SQL_MAX_CURSOR_NAME_LEN	SQL_MAX_USER_NAME_LEN

Scalar Function Information

The following values of the *InfoType* argument return information about the scalar functions supported by the data source and the driver. For more information about scalar functions, see Appendix E, “Scalar Functions” in the **SOLID Programmer Guide**.

SQL_CONVERT_FUNCTIONS	SQL_TIMEDATE_ADD_INTERVALS
SQL_NUMERIC_FUNCTIONS	SQL_TIMEDATE_DIFF_INTERVALS
SQL_STRING_FUNCTIONS	SQL_TIMEDATE_FUNCTIONS
SQL_SYSTEM_FUNCTIONS	

Conversion Information

The following values of the *InfoType* argument 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_LONGVARIABLE
SQL_CONVERT_BINARY	SQL_CONVERT_LONGVARCHAR
SQL_CONVERT_BIT	SQL_CONVERT_NUMERIC
SQL_CONVERT_CHAR	SQL_CONVERT_REAL
SQL_CONVERT_DATE	SQL_CONVERT_SMALLINT
SQL_CONVERT_DECIMAL	SQL_CONVERT_TIME
SQL_CONVERT_DOUBLE	SQL_CONVERT_TIMESTAMP
SQL_CONVERT_FLOAT	SQL_CONVERT_TINYINT
SQL_CONVERT_INTEGER	SQL_CONVERT_VARBINARY
SQL_CONVERT_INTERVAL_YEAR_MONTH	SQL_CONVERT_VARCHAR
SQL_CONVERT_INTERVAL_DAY_TIME	

Information Types Added for ODBC 3.x

The following values of the *InfoType* argument have been added for ODBC 3.x:

SQL_ACTIVE_ENVIRONMENTS	SQL_DROP_ASSERTION
SQLAggregate_FUNCTIONS	SQL_DROP_CHARACTER_SET
SQL_ALTER_DOMAIN	SQL_DROP_COLLATION
SQL_ALTER_SCHEMA	SQL_DROP_DOMAIN
SQL_ANSI_SQL_DATETIME_LITERALS	SQL_DROP_SCHEMA
SQL_ASYNC_MODE	SQL_DROP_TABLE
SQL_BATCH_ROW_COUNT	SQL_DROP_TRANSLATION
SQL_BATCH_SUPPORT	SQL_DROP_VIEW

SQL_CATALOG_NAME	SQL_DYNAMIC_CURSOR_ATTRIBUTES1
SQL_COLLATION_SEQ	SQL_DYNAMIC_CURSOR_ATTRIBUTES2
SQL_CONVERT_INTERVAL_YEAR_MONTH	SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1
SQL_CONVERT_INTERVAL_DAY_TIME	SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2
SQL_CREATE_ASSERTION	SQL_INFO_SCHEMA_VIEWS
SQL_CREATE_CHARACTER_SET	SQL_INSERT_STATEMENT
SQL_CREATE_COLLATION	SQL_KEYSET_CURSOR_ATTRIBUTES1
SQL_CREATE_DOMAIN	SQL_KEYSET_CURSOR_ATTRIBUTES2
SQL_CREATE_SCHEMA	SQL_MAX_ASYNC_CONCURRENT_STATEMENTS
SQL_CREATE_TABLE	SQL_MAX_IDENTIFIER_LEN
SQL_CREATE_TRANSLATION	SQL_PARAM_ARRAY_ROW_COUNTS
SQL_CURSOR_SENSITIVITY	SQL_PARAM_ARRAY_SELECTS
SQL_DDL_INDEX	SQL_STATIC_CURSOR_ATTRIBUTES1
SQL_DESCRIBE_PARAMETER	SQL_STATIC_CURSOR_ATTRIBUTES2
SQL_DM_VER	SQL_XOPEN_CLI_YEAR
SQL_DRIVER_HDESC	

Information Types Renamed for ODBC 3.x

The following values of the *InfoType* argument have been renamed for ODBC 3.x.

ODBC 2.0 <i>InfoType</i>	ODBC 3.x InfoType
SQL_ACTIVE_CONNECTIONS	SQL_MAX_DRIVER_CONNECTIONS
SQL_ACTIVE_STATEMENTS	SQL_MAX_CONCURRENT_ACTIVITIES
SQL_MAX_OWNER_NAME_LEN	SQL_MAX_SCHEMA_NAME_LEN
SQL_MAX_QUALIFIER_NAME_LEN	SQL_MAX_CATALOG_NAME_LEN
SQL_ODBC_SQL_OPT_IEF	SQL_INTEGRITY
SQL_OWNER_TERM	SQL_SCHEMA_TERM
SQL_OWNER_USAGE	SQL_SCHEMA_USAGE
SQL_QUALIFIER_LOCATION	SQL_CATALOG_LOCATION
SQL_QUALIFIER_NAME_SEPARATOR	SQL_CATALOG_NAME_SEPARATOR
SQL_QUALIFIER_TERM	SQL_CATALOG_TERM
SQL_QUALIFIER_USAGE	SQL_CATALOG_USAGE

Information Types Deprecated in ODBC 3.x

The following values of the *InfoType* argument have been deprecated in ODBC 3.x. ODBC 3.x drivers must continue to support these information types for backward compatibility with ODBC 2.x applications. (For more information on these types, see “SQLGetInfoSupport” (Appendix G, "Driver Guidelines for Backward Compatibility.") contained on the Microsoft Web site (ODBC Programmer’s Reference).

SQL_FETCH_DIRECTION	SQL_POS_OPERATIONS
SQL_LOCK_TYPES	SQL_POSITIONED_STATEMENTS
SQL_ODBC_API_CONFORMANCE	SQL_SCROLL_CONCURRENCY
SQL_ODBC_SQL_CONFORMANCE	SQL_STATIC_SENSITIVITY

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_ENVIRONMENTS (ODBC 3.0)	An SQLUSMALLINT value specifying the maximum number of active environments that the driver can support. If there is no specified limit or the limit is unknown, this value is set to zero.
SQLAggregate_FUNCTIONS (ODBC 3.0)	An SQLINTEGER bitmask enumerating support for aggregation functions: SQL_AF_ALL SQL_AF_AVG SQL_AF_COUNT SQL_AF_DISTINCT SQL_AF_MAX SQL_AF_MIN SQL_AF_SUM An SQL-92 Entry level–conformant driver will always return all of these options as supported.
SQL ALTER DOMAIN (ODBC 3.0)	An SQLINTEGER bitmask enumerating the clauses in the ALTER DOMAIN statement, as defined in SQL-92, supported by the data source. An SQL-92 Full level–compliant driver will always return all of the bitmasks. A return value of "0" means that the ALTER DOMAIN statement is not supported. The SQL-92 or FIPS conformance level at which this feature needs to be supported is shown in parentheses next to each

	<p>bitmask.</p> <p>The following bitmasks are used to determine which clauses are supported:</p> <p>SQL_AD_ADD_DOMAIN_CONSTRAINT = Adding a domain constraint is supported (Full level)</p> <p>SQL_AD_ADD_DOMAIN_DEFAULT = <alter domain> <set domain default clause> is supported (Full level)</p> <p>SQL_AD_CONSTRAINT_NAME_DEFINITION = <constraint name definition clause> is supported for naming domain constraint (Intermediate level)</p> <p>SQL_AD_DROP_DOMAIN_CONSTRAINT = <drop domain constraint clause> is supported (Full level)</p> <p>SQL_AD_DROP_DOMAIN_DEFAULT = <alter domain> <drop domain default clause> is supported (Full level)</p> <p>The following bits specify the supported <constraint attributes> if <add domain constraint> is supported (the SQL_AD_ADD_DOMAIN_CONSTRAINT bit is set):</p> <p>SQL_AD_ADD_CONSTRAINT_DEFERRABLE (Full level)</p> <p>SQL_AD_ADD_CONSTRAINT_NON_DEFERRABLE (Full level)</p> <p>SQL_AD_ADD_CONSTRAINT_INITIALLY_DEFERRED (Full level)</p> <p>SQL_AD_ADD_CONSTRAINT_INITIALLY_IMMEDIATE (Full level)</p>
SQL_ALTER_TABLE (ODBC 2.0)	<p>An SQLINTEGER bitmask enumerating the clauses in the ALTER TABLE statement supported by the data source.</p> <p>The SQL-92 or FIPS conformance level at which this feature needs to be supported is shown in parentheses next to each bitmask.</p> <p>The following bitmasks are used to determine which clauses are supported:</p> <p>SQL_AT_ADD_COLUMN_COLLATION = <add column> clause is supported, with facility to specify column collation (Full level) (ODBC 3.0)</p> <p>SQL_AT_ADD_COLUMN_DEFAULT = <add column> clause is supported, with facility to specify column defaults (FIPS Transitional level) (ODBC 3.0)</p> <p>SQL_AT_ADD_COLUMN_SINGLE = <add column> is supported (FIPS Transitional level) (ODBC 3.0)</p>

	<p>SQL_AT_ADD_CONSTRAINT = <add column> clause is supported, with facility to specify column constraints (FIPS Transitional level) (ODBC 3.0)</p> <p>SQL_AT_ADD_TABLE_CONSTRAINT = <add table constraint> clause is supported (FIPS Transitional level) (ODBC 3.0)</p> <p>SQL_AT_CONSTRAINT_NAME_DEFINITION = <constraint name definition> is supported for naming column and table constraints (Intermediate level) (ODBC 3.0)</p> <p>SQL_AT_DROP_COLUMN_CASCADE = <drop column> CASCADE is supported (FIPS Transitional level) (ODBC 3.0)</p> <p>SQL_AT_DROP_COLUMN_DEFAULT = <alter column> <drop column default clause> is supported (Intermediate level) (ODBC 3.0)</p> <p>SQL_AT_DROP_COLUMN_RESTRICT = <drop column> RESTRICT is supported (FIPS Transitional level) (ODBC 3.0)</p> <p>SQL_AT_DROP_TABLE_CONSTRAINT_CASCADE (ODBC 3.0)</p> <p>SQL_AT_DROP_TABLE_CONSTRAINT_RESTRICT = <drop column> RESTRICT is supported (FIPS Transitional level) (ODBC 3.0)</p> <p>SQL_AT_SET_COLUMN_DEFAULT = <alter column> <set column default clause> is supported (Intermediate level) (ODBC 3.0)</p> <p>The following bits specify the support <constraint attributes> if specifying column or table constraints is supported (the SQL_AT_ADD_CONSTRAINT bit is set):</p> <p>SQL_AT_CONSTRAINT_INITIALLY_DEFERRED (Full level) (ODBC 3.0)</p> <p>SQL_AT_CONSTRAINT_INITIALLY_IMMEDIATE (Full level) (ODBC 3.0)</p> <p>SQL_AT_CONSTRAINT_DEFERRABLE (Full level) (ODBC 3.0)</p> <p>SQL_AT_CONSTRAINT_NON_DEFERRABLE (Full level) (ODBC 3.0)</p>
SQL_ASYNC_MODE (ODBC 3.0)	<p>An SQLINTEGER value indicating the level of asynchronous support in the driver:</p> <p>SQL_AM_CONNECTION = Connection level asynchronous execution is supported. Either all statement handles associated with a given connection handle are in asynchronous mode or all are in synchronous mode. A statement handle on a</p>

	<p>connection cannot be in asynchronous mode while another statement handle on the same connection is in synchronous mode, and vice versa.</p> <p>SQL_AM_STATEMENT = Statement level asynchronous execution is supported. Some statement handles associated with a connection handle can be in asynchronous mode, while other statement handles on the same connection are in synchronous mode.</p> <p>SQL_AM_NONE = Asynchronous mode is not supported.</p>
SQL_BATCH_ROW_COUNT (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the behavior of the driver with respect to the availability of row counts. The following bitmasks are used in conjunction with the information type:</p> <p>SQL_BRC_ROLLED_UP = Row counts for consecutive INSERT, DELETE, or UPDATE statements are rolled up into one. If this bit is not set, then row counts are available for each individual statement.</p> <p>SQL_BRC_PROCEDURES = Row counts, if any, are available when a batch is executed in a stored procedure. If row counts are available, they can be rolled up or individually available, depending on the SQL_BRC_ROLLED_UP bit.</p> <p>SQL_BRC_EXPLICIT = Row counts, if any, are available when a batch is executed directly by calling SQLExecute or SQLExecDirect. If row counts are available, they can be rolled up or individually available, depending on the SQL_BRC_ROLLED_UP bit.</p>
SQL_BATCH_SUPPORT (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the driver's support for batches. The following bitmasks are used to determine which level is supported:</p> <p>SQL_BS_SELECT_EXPLICIT = The driver supports explicit batches that can have result-set generating statements.</p> <p>SQL_BS_ROW_COUNT_EXPLICIT = The driver supports explicit batches that can have row-count generating statements.</p> <p>SQL_BS_SELECT_PROC = The driver supports explicit procedures that can have result-set generating statements.</p> <p>SQL_BS_ROW_COUNT_PROC = The driver supports explicit procedures that can have row-count generating statements.</p>
SQL_BOOKMARK_PERSISTENCE (ODBC 2.0)	<p>An SQLINTEGER bitmask enumerating the operations through which bookmarks persist.</p> <p>The following bitmasks are used in conjunction with the flag to determine through which options bookmarks persist:</p>

	<p>SQL_BP_CLOSE = Bookmarks are valid after an application calls SQLFreeStmt with the SQL_CLOSE option, or SQLCloseCursor to close the cursor associated with a statement.</p> <p>SQL_BP_DELETE = The bookmark for a row is valid after that row has been deleted.</p> <p>SQL_BP_DROP = Bookmarks are valid after an application calls SQLFreeHandle with a <i>HandleType</i> of SQL_HANDLE_STMT to drop a statement.</p> <p>SQL_BP_TRANSACTION = Bookmarks are valid after an application commits or rolls back a transaction.</p> <p>SQL_BP_UPDATE = The bookmark for a row is valid after any column in that row has been updated, including key columns.</p> <p>SQL_BP_OTHER_HSTMT = A bookmark associated with one statement can be used with another statement. Unless SQL_BP_CLOSE or SQL_BP_DROP is specified, the cursor on the first statement must be open.</p>
SQL_CATALOG_LOCATION (ODBC 2.0)	<p>An SQLUSMALLINT value indicating the position of the catalog in a qualified table name:</p> <p>SQL_CL_START SQL_CL_END</p> <p>For example, an Xbase driver returns SQL_CL_START because the directory (catalog) name is at the start of the table name, as in \EMPDATA\EMP.DBF. An ORACLE Server driver returns SQL_CL_END because the catalog is at the end of the table name, as in ADMIN.EMP@EMPDATA.</p> <p>An SQL-92 Full level-conformant driver will always return SQL_CL_START. A value of 0 is returned if catalogs are not supported by the data source. To find out whether catalogs are supported, an application calls SQLGetInfo with the SQL_CATALOG_NAME information type.</p> <p>This <i>InfoType</i> has been renamed for ODBC 3.0 from the ODBC 2.0 <i>InfoType</i> SQL_QUALIFIER_LOCATION.</p>
SQL_CATALOG_NAME (ODBC 3.0)	<p>A character string: "Y" if the server supports catalog names, or "N" if it does not.</p> <p>An SQL-92 Full level-conformant driver will always return "Y".</p>
SQL_CATALOG_NAME_SEPARATOR (ODBC 1.0)	<p>A character string: the character or characters that the data source defines as the separator between a catalog name and the qualified name element that follows or precedes it.</p>

	<p>An empty string is returned if catalogs are not supported by the data source. To find out whether catalogs are supported, an application calls SQLGetInfo with the SQL_CATALOG_NAME information type. An SQL-92 Full level-conformant driver will always return ".".</p> <p>This <i>InfoType</i> has been renamed for ODBC 3.0 from the ODBC 2.0 <i>InfoType</i> SQL_QUALIFIER_NAME_SEPARATOR.</p>
SQL_CATALOG_TERM (ODBC 1.0)	<p>A character string with the data source vendor's name for a catalog; for example, "database" or "directory". This string can be in upper, lower, or mixed case.</p> <p>An empty string is returned if catalogs are not supported by the data source. To find out whether catalogs are supported, an application calls SQLGetInfo with the SQL_CATALOG_NAME information type. An SQL-92 Full level-conformant driver will always return "catalog".</p> <p>This <i>InfoType</i> has been renamed for ODBC 3.0 from the ODBC 2.0 <i>InfoType</i> SQL_QUALIFIER_TERM.</p>
SQL_CATALOG_USAGE (ODBC 2.0)	<p>An SQLINTEGER bitmask enumerating the statements in which catalogs can be used.</p> <p>The following bitmasks are used to determine where catalogs can be used:</p> <p>SQL_CU_DML_STATEMENTS = Catalogs are supported in all Data Manipulation Language statements: SELECT, INSERT, UPDATE, DELETE, and if supported, SELECT FOR UPDATE and positioned update and delete statements.</p> <p>SQL_CU_PROCEDURE_INVOCATION = Catalogs are supported in the ODBC procedure invocation statement.</p> <p>SQL_CU_TABLE_DEFINITION = Catalogs are supported in all table definition statements: CREATE TABLE, CREATE VIEW, ALTER TABLE, DROP TABLE, and DROP VIEW.</p> <p>SQL_CU_INDEX_DEFINITION = Catalogs are supported in all index definition statements: CREATE INDEX and DROP INDEX.</p> <p>SQL_CU_PRIVILEGE_DEFINITION = Catalogs are supported in all privilege definition statements: GRANT and REVOKE.</p> <p>A value of 0 is returned if catalogs are not supported by the data source. To find out whether catalogs are supported, an application calls SQLGetInfo with the SQL_CATALOG_NAME information type. An SQL-92 Full</p>

	<p>level-conformant driver will always return a bitmask with all of these bits set.</p> <p>This <i>InfoType</i> has been renamed for ODBC 3.0 from the ODBC 2.0 <i>InfoType</i> SQL_QUALIFIER_USAGE.</p>
SQL_COLLATION_SEQ (ODBC 3.0)	<p>The name of the collation sequence. This is a character string that indicates the name of the default collation for the default character set for this server (for example, 'ISO 8859-1' or EBCDIC). If this is unknown, an empty string will be returned. An SQL-92 Full level-conformant driver will always return a non-empty string.</p>
SQL_COLUMN_ALIAS (ODBC 2.0)	<p>A character string: "Y" if the data source supports column aliases; otherwise, "N".</p> <p>A column alias is an alternate name that can be specified for a column in the select list by using an AS clause. An SQL-92 Entry level-conformant driver will always return "Y".</p>
SQL_CONCAT_NULL_BEHAVIOR (ODBC 1.0)	<p>An SQLUSMALLINT value indicating how the data source handles the concatenation of NULL valued character data type columns with non-NULL valued character data type columns:</p> <p>SQL_CB_NULL = Result is NULL valued.</p> <p>SQL_CB_NON_NULL = Result is concatenation of non-NULL valued column or columns.</p> <p>An SQL-92 Entry level-conformant driver will always return SQL_CB_NULL.</p>
SQL_CONVERT_BIGINT SQL_CONVERT_BINARY SQL_CONVERT_BIT SQL_CONVERT_CHAR SQL_CONVERT_GUID SQL_CONVERT_DATE SQL_CONVERT_DECIMAL SQL_CONVERT_DOUBLE SQL_CONVERT_FLOAT SQL_CONVERT_INTEGER SQL_CONVERT_INTERVAL_YEAR_MONTH SQL_CONVERT_INTERVAL_DAY_TIME 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	<p>An SQLINTEGER bitmask. The bitmask indicates the conversions supported by the data source with the CONVERT scalar function for data of the type named in the <i>InfoType</i>. If the bitmask equals zero, the data source does not support any conversions from data of the named type, including conversion to the same data type.</p> <p>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 <i>InfoType</i> of SQL_CONVERT_INTEGER. The application performs an AND operation with the returned bitmask and SQL_CVT_BIGINT. If the resulting value is nonzero, the conversion is supported.</p> <p>The following bitmasks are used to determine which conversions are supported:</p> <p>SQL_CVT_BIGINT (ODBC 1.0) SQL_CVT_BINARY (ODBC 1.0) SQL_CVT_BIT (ODBC 1.0) SQL_CVT_GUID (ODBC 3.5) SQL_CVT_CHAR (ODBC 1.0)</p>

(ODBC 1.0)	SQL_CVT_DATE (ODBC 1.0) SQL_CVT_DECIMAL (ODBC 1.0) SQL_CVT_DOUBLE (ODBC 1.0) SQL_CVT_FLOAT (ODBC 1.0) SQL_CVT_INTEGER (ODBC 1.0) SQL_CVT_INTERVAL_YEAR_MONTH (ODBC 3.0) SQL_CVT_INTERVAL_DAY_TIME (ODBC 3.0) SQL_CVT_LONGVARBINARY (ODBC 1.0) SQL_CVT_LONGVARCHAR (ODBC 1.0) SQL_CVT_NUMERIC (ODBC 1.0) SQL_CVT_REAL (ODBC 1.0) SQL_CVT_SMALLINT (ODBC 1.0) SQL_CVT_TIME (ODBC 1.0) SQL_CVT_TIMESTAMP (ODBC 1.0) SQL_CVT_TINYINT (ODBC 1.0) SQL_CVT_VARBINARY (ODBC 1.0) SQL_CVT_VARCHAR (ODBC 1.0)
SQL_CONVERT_FUNCTIONS (ODBC 1.0)	An SQLINTEGER 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_CAST SQL_FN_CVT_CONVERT
SQL_CORRELATION_NAME (ODBC 1.0)	An SQLUSMALLINT value indicating whether 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. An SQL-92 Entry level-conformant driver will always return SQL_CN_ANY.
SQL_CREATE_ASSERTION (ODBC 3.0)	An SQLINTEGER bitmask enumerating the clauses in the CREATE ASSERTION statement, as defined in SQL-92, supported by the data source. The following bitmasks are used to determine which clauses are supported: SQL_CA_CREATE_ASSERTION The following bits specify the supported constraint attribute if the ability to specify constraint attributes explicitly is supported (see the SQL_ALTER_TABLE and

	<p>SQL_CREATE_TABLE information types):</p> <p>SQL_CA_CONSTRAINT_INITIALLY_DEFERRED SQL_CA_CONSTRAINT_INITIALLY_IMMEDIATE SQL_CA_CONSTRAINT_DEFERRABLE SQL_CA_CONSTRAINT_NON_DEFERRABLE</p> <p>An SQL-92 Full level-conformant driver will always return all of these options as supported. A return value of "0" means that the CREATE ASSERTION statement is not supported.</p>
SQL_CREATE_CHARACTER_SET (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses in the CREATE CHARACTER SET statement, as defined in SQL-92, supported by the data source.</p> <p>The following bitmasks are used to determine which clauses are supported:</p> <p>SQL_CCS_CREATE_CHARACTER_SET SQL_CCS_COLLATE_CLAUSE SQL_CCS_LIMITED_COLLATION</p> <p>An SQL-92 Full level-conformant driver will always return all of these options as supported. A return value of "0" means that the CREATE CHARACTER SET statement is not supported.</p>
SQL_CREATE_COLLATION (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses in the CREATE COLLATION statement, as defined in SQL-92, supported by the data source.</p> <p>The following bitmask is used to determine which clauses are supported:</p> <p>SQL_CCOL_CREATE_COLLATION</p> <p>An SQL-92 Full level-conformant driver will always return this option as supported. A return value of "0" means that the CREATE COLLATION statement is not supported.</p>
SQL_CREATE_DOMAIN (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses in the CREATE DOMAIN statement, as defined in SQL-92, supported by the data source.</p> <p>The following bitmasks are used to determine which clauses are supported:</p> <p>SQL_CDO_CREATE_DOMAIN = The CREATE DOMAIN statement is supported (Intermediate level).</p> <p>SQL_CDO_CONSTRAINT_NAME_DEFINITION = <constraint name definition> is supported for naming domain constraints (Intermediate level).</p> <p>The following bits specify the ability to create column</p>

	<p>constraints:</p> <p>SQL_CDO_DEFAULT = Specifying domain constraints is supported (Intermediate level)</p> <p>SQL_CDO_CONSTRAINT = Specifying domain defaults is supported (Intermediate level)</p> <p>SQL_CDO_COLLATION = Specifying domain collation is supported (Full level)</p> <p>The following bits specify the supported constraint attributes if specifying domain constraints is supported (SQL_CDO_DEFAULT is set):</p> <p>SQL_CDO_CONSTRAINT_INITIALLY_DEFERRED (Full level)</p> <p>SQL_CDO_CONSTRAINT_INITIALLY_IMMEDIATE (Full level)</p> <p>SQL_CDO_CONSTRAINT_DEFERRABLE (Full level)</p> <p>SQL_CDO_CONSTRAINT_NON_DEFERRABLE (Full level)</p> <p>A return value of "0" means that the CREATE DOMAIN statement is not supported.</p>
SQL_CREATE_SCHEMA (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses in the CREATE SCHEMA statement, as defined in SQL-92, supported by the data source.</p> <p>The following bitmasks are used to determine which clauses are supported:</p> <p>SQL_CS_CREATE_SCHEMA SQL_CS_AUTHORIZATION SQL_CS_DEFAULT_CHARACTER_SET</p> <p>An SQL-92 Intermediate level-conformant driver will always return the SQL_CS_CREATE_SCHEMA and SQL_CS_AUTHORIZATION options as supported. These must also be supported at the SQL-92 Entry level, but not necessarily as SQL statements. An SQL-92 Full level-conformant driver will always return all of these options as supported.</p>
SQL_CREATE_TABLE (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses in the CREATE TABLE statement, as defined in SQL-92, supported by the data source.</p> <p>The SQL-92 or FIPS conformance level at which this feature needs to be supported is shown in parentheses next to each bitmask.</p> <p>The following bitmasks are used to determine which clauses are supported:</p> <p>SQL_CT_CREATE_TABLE = The CREATE TABLE</p>

	<p>statement is supported. (Entry level)</p> <p>SQL_CT_TABLE_CONSTRAINT = Specifying table constraints is supported (FIPS Transitional level)</p> <p>SQL_CT_CONSTRAINT_NAME_DEFINITION = The <constraint name definition> clause is supported for naming column and table constraints (Intermediate level)</p> <p>The following bits specify the ability to create temporary tables:</p> <p>SQL_CT_COMMIT_PRESERVE = Deleted rows are preserved on commit. (Full level)</p> <p>SQL_CT_COMMIT_DELETE = Deleted rows are deleted on commit. (Full level)</p> <p>SQL_CT_GLOBAL_TEMPORARY = Global temporary tables can be created. (Full level)</p> <p>SQL_CT_LOCAL_TEMPORARY = Local temporary tables can be created. (Full level)</p> <p>The following bits specify the ability to create column constraints:</p> <p>SQL_CT_COLUMN_CONSTRAINT = Specifying column constraints is supported (FIPS Transitional level)</p> <p>SQL_CT_COLUMN_DEFAULT = Specifying column defaults is supported (FIPS Transitional level)</p> <p>SQL_CT_COLUMN_COLLATION = Specifying column collation is supported (Full level)</p> <p>The following bits specify the supported constraint attributes if specifying column or table constraints is supported:</p> <p>SQL_CT_CONSTRAINT_INITIALLY_DEFERRED (Full level)</p> <p>SQL_CT_CONSTRAINT_INITIALLY_IMMEDIATE (Full level)</p> <p>SQL_CT_CONSTRAINT_DEFERRABLE (Full level)</p> <p>SQL_CT_CONSTRAINT_NON_DEFERRABLE (Full level)</p>
<p>SQL_CREATE_TRANSLATION (ODBC 3.0)</p>	<p>An SQLINTEGER bitmask enumerating the clauses in the CREATE TRANSLATION statement, as defined in SQL-92, supported by the data source.</p> <p>The following bitmask is used to determine which clauses are supported:</p> <p>SQL_CTR_CREATE_TRANSLATION</p> <p>An SQL-92 Full level-conformant driver will always return these options as supported. A return value of "0" means that the CREATE TRANSLATION statement is not supported.</p>

<p>SQL_CREATE_VIEW (ODBC 3.0)</p>	<p>An SQLINTEGER bitmask enumerating the clauses in the CREATE VIEW statement, as defined in SQL-92, supported by the data source.</p> <p>The following bitmasks are used to determine which clauses are supported:</p> <p>SQL_CV_CREATE_VIEW SQL_CV_CHECK_OPTION SQL_CV_CASCADE SQL_CV_LOCAL</p> <p>A return value of "0" means that the CREATE VIEW statement is not supported.</p> <p>An SQL-92 Entry level-conformant driver will always return the SQL_CV_CREATE_VIEW and SQL_CV_CHECK_OPTION options as supported.</p> <p>An SQL-92 Full level-conformant driver will always return all of these options as supported.</p>
<p>SQL_CURSOR_COMMIT_BEHAVIOR (ODBC 1.0)</p>	<p>An SQLUSMALLINT value indicating how a COMMIT operation affects cursors and prepared statements in the data source:</p> <p>SQL_CB_DELETE = Close cursors and delete prepared statements. To use the cursor again, the application must reprepare and reexecute the statement.</p> <p>SQL_CB_CLOSE = Close cursors. For prepared statements, the application can call SQLExecute on the statement without calling SQLPrepare again.</p> <p>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 statement without repreparing it.</p>
<p>SQL_CURSOR_ROLLBACK_BEHAVIOR (ODBC 1.0)</p>	<p>An SQLUSMALLINT value indicating how a ROLLBACK operation affects cursors and prepared statements in the data source:</p> <p>SQL_CB_DELETE = Close cursors and delete prepared statements. To use the cursor again, the application must reprepare and reexecute the statement.</p> <p>SQL_CB_CLOSE = Close cursors. For prepared statements, the application can call SQLExecute on the statement without calling SQLPrepare again.</p> <p>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</p>

	the statement without reparing it.
SQL_CURSOR_ROLLBACK_SQL_CURSOR_SENSITIVITY (ODBC 3.0)	<p>An SQLINTEGER value indicating the support for cursor sensitivity:</p> <p>SQL_INSENSITIVE = All cursors on the statement handle show the result set without reflecting any changes made to it by any other cursor within the same transaction.</p> <p>SQL_UNSPECIFIED = It is unspecified whether cursors on the statement handle make visible the changes made to a result set by another cursor within the same transaction. Cursors on the statement handle may make visible none, some, or all such changes.</p> <p>SQL_SENSITIVE = Cursors are sensitive to changes made by other cursors within the same transaction.</p> <p>An SQL-92 Entry level–conformant driver will always return the SQL_UNSPECIFIED option as supported.</p> <p>An SQL-92 Full level–conformant driver will always return the SQL_INSENSITIVE option as supported.</p>
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 <i>szDSN</i> 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)	<p>A character string. "Y" if the data source is set to READ ONLY mode, "N" if it is otherwise.</p> <p>This characteristic pertains only to the data source itself; it is not a characteristic of the driver that enables access to the data source. A driver that is read/write can be used with a data source that is read-only. If a driver is read-only, all of its data sources must be read-only and must return SQL_DATA_SOURCE_READ_ONLY.</p>
SQL_DATABASE_NAME (ODBC 1.0)	<p>A character string with the name of the current database in use, if the data source defines a named object called "database".</p> <p>Note In ODBC 3.x, the value returned for this <i>InfoType</i> can also be returned by calling SQLGetConnectAttr with an <i>Attribute</i> argument of SQL_ATTR_CURRENT_CATALOG.</p>
SQL_DATETIME_LITERALS (ODBC 3.0)	An SQLINTEGER bitmask enumerating the SQL-92 datetime literals supported by the data source. Note that these are the datetime literals listed in the SQL-92 specification and are separate from the datetime literal escape clauses defined by ODBC. For more information about the ODBC datetime literal escape clauses, see the Part I PDF file, "Date, Time, and

	<p>Timestamp Literals” in Chapter 8, "SQL Statements."</p> <p>A FIPS Transitional level–conformant driver will always return the "1" value in the bitmask for the bits listed below. A value of "0" means that SQL-92 datetime literals are not supported.</p> <p>The following bitmasks are used to determine which literals are supported:</p> <p>SQL_DL_SQL92_DATE SQL_DL_SQL92_TIME SQL_DL_SQL92_TIMESTAMP SQL_DL_SQL92_INTERVAL_YEAR SQL_DL_SQL92_INTERVAL_MONTH SQL_DL_SQL92_INTERVAL_DAY SQL_DL_SQL92_INTERVAL_HOUR SQL_DL_SQL92_INTERVAL_MINUTE SQL_DL_SQL92_INTERVAL_SECOND SQL_DL_SQL92_INTERVAL_YEAR_TO_MONTH SQL_DL_SQL92_INTERVAL_DAY_TO_HOUR SQL_DL_SQL92_INTERVAL_DAY_TO_MINUTE SQL_DL_SQL92_INTERVAL_DAY_TO_SECOND SQL_DL_SQL92_INTERVAL_HOUR_TO_MINUTE SQL_DL_SQL92_INTERVAL_HOUR_TO_SECOND SQL_DL_SQL92_INTERVAL_MINUTE_TO_SECOND</p>
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_DDL_INDEX (ODBC 3.0)	<p>An SQLINTEGER value that indicates support for creation and dropping of indexes:</p> <p>SQL_DI_CREATE_INDEX SQL_DI_DROP_INDEX</p>
SQL_DEFAULT_TXN_ISOLATION (ODBC 1.0)	<p>An SQLINTEGER value 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:</p> <p>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.</p>

	<p>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.</p> <p>Phantom Transaction 1 reads a set of rows that satisfy some search criteria. Transaction 2 generates one or more rows (through either inserts or updates) that match the search criteria. If transaction 1 reexecutes the statement that reads the rows, it receives a different set of rows.</p> <p>If the data source supports transactions, the driver returns one of the following bitmasks:</p> <p>SQL_TXN_READ_UNCOMMITTED = Dirty reads, nonrepeatable reads, and phantoms are possible.</p> <p>SQL_TXN_READ_COMMITTED = Dirty reads are not possible. Nonrepeatable reads and phantoms are possible.</p> <p>SQL_TXN_REPEATABLE_READ = Dirty reads and nonrepeatable reads are not possible. Phantoms are possible.</p> <p>SQL_TXN_SERIALIZABLE = Transactions are serializable. Serializable transactions do not allow dirty reads, nonrepeatable reads, or phantoms.</p>
SQL_DESCRIBE_PARAMETER (ODBC 3.0)	<p>A character string: "Y" if parameters can be described; "N", if not.</p> <p>An SQL-92 Full level–conformant driver will usually return "Y" because it will support the DESCRIBE INPUT statement. Because this does not directly specify the underlying SQL support, however, describing parameters might not be supported, even in a SQL-92 Full level–conformant driver.</p>
SQL_DM_VER (ODBC 3.0)	<p>A character string with the version of the Driver Manager. The version is of the form ##.##.####.####, where:</p> <p>The first set of two digits is the major ODBC version, as given by the constant SQL_SPEC_MAJOR.</p> <p>The second set of two digits is the minor ODBC version, as given by the constant SQL_SPEC_MINOR.</p> <p>The third set of four digits is the Driver Manager major build number.</p> <p>The last set of four digits is the Driver Manager minor build number.</p>
SQL_DRIVER_HDBC SQL_DRIVER_HENV (ODBC 1.0)	<p>An SQLINTEGER value, the driver's environment handle or connection handle, determined by the argument <i>InfoType</i>.</p> <p>These information types are implemented by the Driver</p>

	Manager alone.
SQL_DRIVER_HDESC (ODBC 3.0)	<p>An SQLINTEGER value, the driver's descriptor handle determined by the Driver Manager's descriptor handle, which must be passed on input in <i>*InfoValuePtr</i> from the application. In this case, <i>InfoValuePtr</i> is both an input and output argument. The input descriptor handle passed in <i>*InfoValuePtr</i> must have been either explicitly or implicitly allocated on the <i>ConnectionHandle</i>.</p> <p>The application should make a copy of the Driver Manager's descriptor handle before calling SQLGetInfo with this information type, to ensure that the handle is not overwritten on output.</p> <p>This information type is implemented by the Driver Manager alone.</p>
SQL_DRIVER_HLIB (ODBC 2.0)	<p>An SQLINTEGER value, the <i>hinst</i> from the load library returned to the Driver Manager when it loaded the driver DLL (on a Microsoft® Windows® platform) or equivalent on a non-Windows platform. The handle is valid only for the connection handle specified in the call to SQLGetInfo.</p> <p>This information type is implemented by the Driver Manager alone.</p>
SQL_DRIVER_HSTMT (ODBC 1.0)	<p>An SQLINTEGER value, the driver's statement handle determined by the Driver Manager statement handle, which must be passed on input in <i>*InfoValuePtr</i> from the application. In this case, <i>InfoValuePtr</i> is both an input and an output argument. The input statement handle passed in <i>*InfoValuePtr</i> must have been allocated on the argument <i>ConnectionHandle</i>.</p> <p>The application should make a copy of the Driver Manager's statement handle before calling SQLGetInfo with this information type, to ensure that the handle is not overwritten on output.</p> <p>This information type is implemented by the Driver Manager alone.</p>
SQL_DRIVER_NAME (ODBC 1.0)	A character string with the file name of the driver used to 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 <i>##.##</i> , where the first two digits are the major version and the next two digits are the minor version. <i>SQL_SPEC_MAJOR</i> and <i>SQL_SPEC_MINOR</i> define the major and minor version numbers. For the version of ODBC described in this manual, these are 3 and 0, and the driver should return "03.00".
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 <i>##.##.####</i> , 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_DROP_ASSERTION (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses in the DROP ASSERTION statement, as defined in SQL-92, supported by the data source.</p> <p>The following bitmask is used to determine which clauses are supported:</p> <p>SQL_DA_DROP_ASSERTION</p> <p>An SQL-92 Full level–conformant driver will always return this option as supported.</p>
SQL_DROP_CHARACTER_SET (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses in the DROP CHARACTER SET statement, as defined in SQL-92, supported by the data source.</p> <p>The following bitmask is used to determine which clauses are supported:</p> <p>SQL_DCS_DROP_CHARACTER_SET</p> <p>An SQL-92 Full level–conformant driver will always return this option as supported.</p>
SQL_DROP_COLLATION (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses in the DROP COLLATION statement, as defined in SQL-92, supported by the data source.</p> <p>The following bitmask is used to determine which clauses are supported:</p> <p>SQL_DC_DROP_COLLATION</p> <p>An SQL-92 Full level–conformant driver will always return this option as supported.</p>
SQL_DROP_DOMAIN (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses in the DROP DOMAIN statement, as defined in SQL-92, supported by the data source.</p> <p>The following bitmasks are used to determine which clauses are supported:</p> <p>SQL_DD_DROP_DOMAIN SQL_DD_CASCADE SQL_DD_RESTRICT</p> <p>An SQL-92 Intermediate level–conformant driver will always return all of these options as supported.</p>
SQL_DROP_SCHEMA (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses in the DROP SCHEMA statement, as defined in SQL-92, supported by the data source.</p>

	<p>The following bitmasks are used to determine which clauses are supported:</p> <p>SQL_DS_DROP_SCHEMA SQL_DS_CASCADE SQL_DS_RESTRICT</p> <p>An SQL-92 Intermediate level–conformant driver will always return all of these options as supported.</p>
SQL_DROP_TABLE (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses in the DROP TABLE statement, as defined in SQL-92, supported by the data source.</p> <p>The following bitmasks are used to determine which clauses are supported:</p> <p>SQL_DT_DROP_TABLE SQL_DT_CASCADE SQL_DT_RESTRICT</p> <p>An FIPS Transitional level–conformant driver will always return all of these options as supported.</p>
SQL_DROP_TRANSLATION (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses in the DROP TRANSLATION statement, as defined in SQL-92, supported by the data source.</p> <p>The following bitmask is used to determine which clauses are supported:</p> <p>SQL_DTR_DROP_TRANSLATION</p> <p>An SQL-92 Full level–conformant driver will always return this option as supported.</p>
SQL_DROP_VIEW (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses in the DROP VIEW statement, as defined in SQL-92, supported by the data source.</p> <p>The following bitmasks are used to determine which clauses are supported:</p> <p>SQL_DV_DROP_VIEW SQL_DV_CASCADE SQL_DV_RESTRICT</p> <p>An FIPS Transitional level–conformant driver will always return all of these options as supported.</p>
SQL_DYNAMIC_CURSOR_ATTRIBUTES1 (ODBC 3.0)	<p>An SQLINTEGER bitmask that describes the attributes of a dynamic cursor that are supported by the driver. This bitmask contains the first subset of attributes; for the second subset, see SQL_DYNAMIC_CURSOR_ATTRIBUTES2.</p> <p>The following bitmasks are used to determine which attributes</p>

	<p>are supported:</p> <p>SQL_CA1_NEXT = A <i>FetchOrientation</i> argument of SQL_FETCH_NEXT is supported in a call to SQLFetchScroll when the cursor is a dynamic cursor.</p> <p>SQL_CA1_ABSOLUTE = <i>FetchOrientation</i> arguments of SQL_FETCH_FIRST, SQL_FETCH_LAST, and SQL_FETCH_ABSOLUTE are supported in a call to SQLFetchScroll when the cursor is a dynamic cursor. (The rowset that will be fetched is independent of the current cursor position.)</p> <p>SQL_CA1_RELATIVE = <i>FetchOrientation</i> arguments of SQL_FETCH_PRIOR and SQL_FETCH_RELATIVE are supported in a call to SQLFetchScroll when the cursor is a dynamic cursor. (The rowset that will be fetched is dependent on the current cursor position. Note that this is separated from SQL_FETCH_NEXT because in a forward-only cursor, only SQL_FETCH_NEXT is supported.)</p> <p>SQL_CA1_BOOKMARK = A <i>FetchOrientation</i> argument of SQL_FETCH_BOOKMARK is supported in a call to SQLFetchScroll when the cursor is a dynamic cursor.</p> <p>SQL_CA1_LOCK_EXCLUSIVE = A <i>LockType</i> argument of SQL_LOCK_EXCLUSIVE is supported in a call to SQLSetPos when the cursor is a dynamic cursor.</p> <p>SQL_CA1_LOCK_NO_CHANGE = A <i>LockType</i> argument of SQL_LOCK_NO_CHANGE is supported in a call to SQLSetPos when the cursor is a dynamic cursor.</p> <p>SQL_CA1_LOCK_UNLOCK = A <i>LockType</i> argument of SQL_LOCK_UNLOCK is supported in a call to SQLSetPos when the cursor is a dynamic cursor.</p> <p>SQL_CA1_POS_POSITION = An <i>Operation</i> argument of SQL_POSITION is supported in a call to SQLSetPos when the cursor is a dynamic cursor.</p> <p>SQL_CA1_POS_UPDATE = An <i>Operation</i> argument of SQL_UPDATE is supported in a call to SQLSetPos when the cursor is a dynamic cursor.</p> <p>SQL_CA1_POS_DELETE = An <i>Operation</i> argument of SQL_DELETE is supported in a call to SQLSetPos when the cursor is a dynamic cursor.</p> <p>SQL_CA1_POS_REFRESH = An <i>Operation</i> argument of SQL_REFRESH is supported in a call to SQLSetPos when the cursor is a dynamic cursor.</p>
--	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<p>SQL_CA1_POSITIONED_UPDATE = An UPDATE WHERE CURRENT OF SQL statement is supported when the cursor is a dynamic cursor. (An SQL-92 Entry level–conformant driver will always return this option as supported.)</p> <p>SQL_CA1_POSITIONED_DELETE = A DELETE WHERE CURRENT OF SQL statement is supported when the cursor is a dynamic cursor. (An SQL-92 Entry level–conformant driver will always return this option as supported.)</p> <p>SQL_CA1_SELECT_FOR_UPDATE = A SELECT FOR UPDATE SQL statement is supported when the cursor is a dynamic cursor. (An SQL-92 Entry level–conformant driver will always return this option as supported.)</p> <p>SQL_CA1_BULK_ADD = An <i>Operation</i> argument of SQL_ADD is supported in a call to SQLBulkOperations when the cursor is a dynamic cursor.</p> <p>SQL_CA1_BULK_UPDATE_BY_BOOKMARK = An <i>Operation</i> argument of SQL_UPDATE_BY_BOOKMARK is supported in a call to SQLBulkOperations when the cursor is a dynamic cursor.</p> <p>SQL_CA1_BULK_DELETE_BY_BOOKMARK = An <i>Operation</i> argument of SQL_DELETE_BY_BOOKMARK is supported in a call to SQLBulkOperations when the cursor is a dynamic cursor.</p> <p>SQL_CA1_BULK_FETCH_BY_BOOKMARK = An <i>Operation</i> argument of SQL_FETCH_BY_BOOKMARK is supported in a call to SQLBulkOperations when the cursor is a dynamic cursor.</p> <p>An SQL-92 Intermediate level–conformant driver will usually return the SQL_CA1_NEXT, SQL_CA1_ABSOLUTE, and SQL_CA1_RELATIVE options as supported, because it supports scrollable cursors through the embedded SQL FETCH statement. Because this does not directly determine the underlying SQL support, however, scrollable cursors may not be supported, even for an SQL-92 Intermediate level–conformant driver.</p>
SQL_DYNAMIC_CURSOR_ATTRIBUTES2 (ODBC 3.0)	<p>An SQLINTEGER bitmask that describes the attributes of a dynamic cursor that are supported by the driver. This bitmask contains the second subset of attributes; for the first subset, see SQL_DYNAMIC_CURSOR_ATTRIBUTES1.</p> <p>The following bitmasks are used to determine which attributes are supported:</p> <p>SQL_CA2_READ_ONLY_CONCURRENCY = A read-only dynamic cursor, in which no updates are allowed, is supported.</p>

	<p>(The <code>SQL_ATTR_CONCURRENCY</code> statement attribute can be <code>SQL_CONCUR_READ_ONLY</code> for a dynamic cursor).</p> <p><code>SQL_CA2_LOCK_CONCURRENCY</code> = A dynamic cursor that uses the lowest level of locking sufficient to ensure that the row can be updated is supported. (The <code>SQL_ATTR_CONCURRENCY</code> statement attribute can be <code>SQL_CONCUR_LOCK</code> for a dynamic cursor.) These locks must be consistent with the transaction isolation level set by the <code>SQL_ATTR_TXN_ISOLATION</code> connection attribute.</p> <p><code>SQL_CA2_OPT_ROWVER_CONCURRENCY</code> = A dynamic cursor that uses the optimistic concurrency control comparing row versions is supported. (The <code>SQL_ATTR_CONCURRENCY</code> statement attribute can be <code>SQL_CONCUR_ROWVER</code> for a dynamic cursor.)</p> <p><code>SQL_CA2_OPT_VALUES_CONCURRENCY</code> = A dynamic cursor that uses the optimistic concurrency control comparing values is supported. (The <code>SQL_ATTR_CONCURRENCY</code> statement attribute can be <code>SQL_CONCUR_VALUES</code> for a dynamic cursor.)</p> <p><code>SQL_CA2_SENSITIVITY_ADDITIONS</code> = Added rows are visible to a dynamic cursor; the cursor can scroll to those rows. (Where these rows are added to the cursor is driver-dependent.)</p> <p><code>SQL_CA2_SENSITIVITY_DELETIONS</code> = Deleted rows are no longer available to a dynamic cursor, and do not leave a "hole" in the result set; after the dynamic cursor scrolls from a deleted row, it cannot return to that row.</p> <p><code>SQL_CA2_SENSITIVITY_UPDATES</code> = Updates to rows are visible to a dynamic cursor; if the dynamic cursor scrolls from and returns to an updated row, the data returned by the cursor is the updated data, not the original data.</p> <p><code>SQL_CA2_MAX_ROWS_SELECT</code> = The <code>SQL_ATTR_MAX_ROWS</code> statement attribute affects SELECT statements when the cursor is a dynamic cursor.</p> <p><code>SQL_CA2_MAX_ROWS_INSERT</code> = The <code>SQL_ATTR_MAX_ROWS</code> statement attribute affects INSERT statements when the cursor is a dynamic cursor.</p> <p><code>SQL_CA2_MAX_ROWS_DELETE</code> = The <code>SQL_ATTR_MAX_ROWS</code> statement attribute affects DELETE statements when the cursor is a dynamic cursor.</p> <p><code>SQL_CA2_MAX_ROWS_UPDATE</code> = The <code>SQL_ATTR_MAX_ROWS</code> statement attribute affects</p>
--	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<p>UPDATE statements when the cursor is a dynamic cursor.</p> <p>SQL_CA2_MAX_ROWS_CATALOG = The SQL_ATTR_MAX_ROWS statement attribute affects CATALOG result sets when the cursor is a dynamic cursor.</p> <p>SQL_CA2_MAX_ROWS_AFFECTS_ALL = The SQL_ATTR_MAX_ROWS statement attribute affects SELECT, INSERT, DELETE, and UPDATE statements, and CATALOG result sets, when the cursor is a dynamic cursor.</p> <p>SQL_CA2_CRC_EXACT = The exact row count is available in the SQL_DIAG_CURSOR_ROW_COUNT diagnostic field when the cursor is a dynamic cursor.</p> <p>SQL_CA2_CRC_APPROXIMATE = An approximate row count is available in the SQL_DIAG_CURSOR_ROW_COUNT diagnostic field when the cursor is a dynamic cursor.</p> <p>SQL_CA2_SIMULATE_NON_UNIQUE = The driver does not guarantee that simulated positioned update or delete statements will affect only one row when the cursor is a dynamic cursor; it is the application's responsibility to guarantee this. (If a statement affects more than one row, SQLExecute or SQLExecDirect returns SQLSTATE 01001 [Cursor operation conflict].) To set this behavior, the application calls SQLSetStmtAttr with the SQL_ATTR_SIMULATE_CURSOR attribute set to SQL_SC_NON_UNIQUE.</p> <p>SQL_CA2_SIMULATE_TRY_UNIQUE = The driver attempts to guarantee that simulated positioned update or delete statements will affect only one row when the cursor is a dynamic cursor. 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 01001 [Cursor operation conflict].) To set this behavior, the application calls SQLSetStmtAttr with the SQL_ATTR_SIMULATE_CURSOR attribute set to SQL_SC_TRY_UNIQUE.</p> <p>SQL_CA2_SIMULATE_UNIQUE = The driver guarantees that simulated positioned update or delete statements will affect only one row when the cursor is a dynamic cursor. If the driver cannot guarantee this for a given statement, SQLExecDirect or SQLPrepare return SQLSTATE 01001 (Cursor operation conflict). To set this behavior, the application calls SQLSetStmtAttr with the SQL_ATTR_SIMULATE_CURSOR attribute set to</p>
--	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	SQL_SC_UNIQUE.
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_FILE_USAGE (ODBC 2.0)	<p>An SQLUSMALLINT value indicating how a single-tier driver directly treats files in a data source:</p> <p>SQL_FILE_NOT_SUPPORTED = The driver is not a single-tier driver. For example, an ORACLE driver is a two-tier driver.</p> <p>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.</p> <p>SQL_FILE_CATALOG = A single-tier driver treats files in a data source as a catalog. For example, a Microsoft® Access driver treats each Microsoft Access file as a complete database.</p> <p>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 MicrosoftAccess users generally think of data as stored in tables.</p> <p>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.</p>
SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1 (ODBC 3.0)	<p>An SQLINTEGER bitmask that describes the attributes of a forward-only cursor that are supported by the driver. This bitmask contains the first subset of attributes; for the second subset, see SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2.</p> <p>The following bitmasks are used to determine which attributes are supported:</p> <p>SQL_CA1_NEXT SQL_CA1_LOCK_EXCLUSIVE SQL_CA1_LOCK_NO_CHANGE SQL_CA1_LOCK_UNLOCK SQL_CA1_POS_POSITION SQL_CA1_POS_UPDATE SQL_CA1_POS_DELETE SQL_CA1_POS_REFRESH SQL_CA1_POSITIONED_UPDATE SQL_CA1_POSITIONED_DELETE SQL_CA1_SELECT_FOR_UPDATE SQL_CA1_BULK_ADD SQL_CA1_BULK_UPDATE_BY_BOOKMARK SQL_CA1_BULK_DELETE_BY_BOOKMARK</p>

	<p>SQL_CA1_BULK_FETCH_BY_BOOKMARK</p> <p>For descriptions of these bitmasks, see SQL_DYNAMIC_CURSOR_ATTRIBUTES1 (and substitute "forward-only cursor" for "dynamic cursor" in the descriptions).</p>
<p>SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2 (ODBC 3.0)</p>	<p>An SQLINTEGER bitmask that describes the attributes of a forward-only cursor that are supported by the driver. This bitmask contains the second subset of attributes; for the first subset, see SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1.</p> <p>The following bitmasks are used to determine which attributes are supported:</p> <p>SQL_CA2_READ_ONLY_CONCURRENCY SQL_CA2_LOCK_CONCURRENCY SQL_CA2_OPT_ROWVER_CONCURRENCY SQL_CA2_OPT_VALUES_CONCURRENCY SQL_CA2_SENSITIVITY_ADDITIONS SQL_CA2_SENSITIVITY_DELETIONS SQL_CA2_SENSITIVITY_UPDATES SQL_CA2_MAX_ROWS_SELECT SQL_CA2_MAX_ROWS_INSERT SQL_CA2_MAX_ROWS_DELETE SQL_CA2_MAX_ROWS_UPDATE SQL_CA2_MAX_ROWS_CATALOG SQL_CA2_MAX_ROWS_AFFECTS_ALL SQL_CA2_CRC_EXACT SQL_CA2_CRC_APPROXIMATE SQL_CA2_SIMULATE_NON_UNIQUE SQL_CA2_SIMULATE_TRY_UNIQUE SQL_CA2_SIMULATE_UNIQUE</p> <p>For descriptions of these bitmasks, see SQL_DYNAMIC_CURSOR_ATTRIBUTES2 (and substitute "forward-only cursor" for "dynamic cursor" in the descriptions).</p>
<p>SQL_GETDATA_EXTENSIONS (ODBC 2.0)</p>	<p>An SQLINTEGER bitmask enumerating extensions to SQLGetData.</p> <p>The following bitmasks are used in conjunction with the flag to determine what common extensions the driver supports for SQLGetData:</p> <p>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.</p> <p>SQL_GD_ANY_ORDER = SQLGetData can be called for</p>

	<p>unbound columns in any order. Note that SQLGetData can be called only for columns after the last bound column unless SQL_GD_ANY_COLUMN is also returned.</p> <p>SQL_GD_BLOCK = SQLGetData can be called for an unbound column in any row in a block (where the rowset size is greater than 1) of data after positioning to that row with SQLSetPos.</p> <p>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.</p> <p>SQLGetData is required to return data only 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.</p> <p>If a driver supports bookmarks (either fixed-length or variable-length), it must support calling SQLGetData on column 0. This support is required regardless of what the driver returns for a call to SQLGetInfo with the SQL_GETDATA_EXTENSIONS <i>InfoType</i>.</p>
<p>SQL_GROUP_BY (ODBC 2.0)</p>	<p>An SQLUSMALLINT value specifying the relationship between the columns in the GROUP BY clause and the nonaggregated columns in the select list:</p> <p>SQL_GB_COLLATE = A COLLATE clause can be specified at the end of each grouping column. (ODBC 3.0)</p> <p>SQL_GB_NOT_SUPPORTED = GROUP BY clauses are not supported. (ODBC 2.0)</p> <p>SQL_GB_GROUP_BY_EQUALS_SELECT = The GROUP BY clause must contain all nonaggregated columns in the select list. It cannot contain any other columns. For example, SELECT DEPT, MAX(SALARY) FROM EMPLOYEE GROUP BY DEPT. (ODBC 2.0)</p> <p>SQL_GB_GROUP_BY_CONTAINS_SELECT = The GROUP BY clause must contain all nonaggregated 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. (ODBC 2.0)</p> <p>SQL_GB_NO_RELATION = The columns in the GROUP BY clause and the select list are not related. The meaning of nongrouped, nonaggregated columns in the select list is data source-dependent. For example, SELECT DEPT, SALARY FROM EMPLOYEE GROUP BY DEPT, AGE. (ODBC 2.0)</p>

	<p>An SQL-92 Entry level–conformant driver will always return the SQL_GB_GROUP_BY_EQUALS_SELECT option as supported. An SQL-92 Full level–conformant driver will always return the SQL_GB_COLLATE option as supported. If none of the options is supported, the GROUP BY clause is not supported by the data source.</p>
SQL_IDENTIFIER_CASE (ODBC 1.0)	<p>An SQLUSMALLINT value as follows:</p> <p>SQL_IC_UPPER = Identifiers in SQL are not case-sensitive and are stored in uppercase in system catalog.</p> <p>SQL_IC_LOWER = Identifiers in SQL are not case-sensitive and are stored in lowercase in system catalog.</p> <p>SQL_IC_SENSITIVE = Identifiers in SQL are case-sensitive and are stored in mixed case in system catalog.</p> <p>SQL_IC_MIXED = Identifiers in SQL are not case-sensitive and are stored in mixed case in system catalog.</p> <p>Because identifiers in SQL-92 are never case-sensitive, a driver that conforms strictly to SQL-92 (any level) will never return the SQL_IC_SENSITIVE option as supported.</p>
SQL_IDENTIFIER_QUOTE_CHAR (ODBC 1.0)	<p>The character string used as the starting and ending delimiter of a quoted (delimited) identifier 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.</p> <p>This character string can also be used for quoting catalog function arguments when the connection attribute SQL_ATTR_METADATA_ID is set to SQL_TRUE.</p> <p>Because the identifier quote character in SQL-92 is the double quotation mark ("), a driver that conforms strictly to SQL-92 will always return the double quotation mark character.</p>
SQL_INDEX_KEYWORDS (ODBC 3.0)	<p>An SQLINTEGER bitmask that enumerates keywords in the CREATE INDEX statement that are supported by the driver:</p> <p>SQL_IK_NONE = None of the keywords is supported.</p> <p>SQL_IK_ASC = ASC keyword is supported.</p> <p>SQL_IK_DESC = DESC keyword is supported.</p> <p>SQL_IK_ALL = All keywords are supported.</p> <p>To see if the CREATE INDEX statement is supported, an application calls SQLGetInfo with the SQL_DLL_INDEX information type.</p>
SQL_INFO_SCHEMA_VIEWS	<p>An SQLINTEGER bitmask enumerating the views in the</p>

(ODBC 3.0)	<p>INFORMATION_SCHEMA that are supported by the driver. The views in, and the contents of, INFORMATION_SCHEMA are as defined in SQL-92.</p> <p>The SQL-92 or FIPS conformance level at which this feature needs to be supported is shown in parentheses next to each bitmask.</p> <p>The following bitmasks are used to determine which views are supported:</p> <p>SQL_ISV_ASSERTIONS = Identifies the catalog's assertions that are owned by a given user. (Full level)</p> <p>SQL_ISV_CHARACTER_SETS = Identifies the catalog's character sets that are accessible to a given user. (Intermediate level)</p> <p>SQL_ISV_CHECK_CONSTRAINTS = Identifies the CHECK constraints that are owned by a given user. (Intermediate level)</p> <p>SQL_ISV_COLLATIONS = Identifies the character collations for the catalog that are accessible to a given user. (Full level)</p> <p>SQL_ISV_COLUMN_DOMAIN_USAGE = Identifies columns for the catalog that are dependent on domains defined in the catalog and are owned by a given user. (Intermediate level)</p> <p>SQL_ISV_COLUMN_PRIVILEGES = Identifies the privileges on columns of persistent tables that are available to or granted by a given user. (FIPS Transitional level)</p> <p>SQL_ISV_COLUMNS = Identifies the columns of persistent tables that are accessible to a given user. (FIPS Transitional level)</p> <p>SQL_ISV_CONSTRAINT_COLUMN_USAGE = Similar to CONSTRAINT_TABLE_USAGE view, columns are identified for the various constraints that are owned by a given user. (Intermediate level)</p> <p>SQL_ISV_CONSTRAINT_TABLE_USAGE = Identifies the tables that are used by constraints (referential, unique, and assertions), and are owned by a given user. (Intermediate level)</p> <p>SQL_ISV_DOMAIN_CONSTRAINTS = Identifies the domain constraints (of the domains in the catalog) that are accessible to a given user. (Intermediate level)</p> <p>SQL_ISV_DOMAINS = Identifies the domains defined in a catalog that are accessible to the user. (Intermediate level)</p> <p>SQL_ISV_KEY_COLUMN_USAGE = Identifies columns</p>
------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<p>defined in the catalog that are constrained as keys by a given user. (Intermediate level)</p> <p>SQL_ISV_REFERENTIAL_CONSTRAINTS = Identifies the referential constraints that are owned by a given user. (Intermediate level)</p> <p>SQL_ISV_SCHEMATA = Identifies the schemas that are owned by a given user. (Intermediate level)</p> <p>SQL_ISV_SQL_LANGUAGES = Identifies the SQL conformance levels, options, and dialects supported by the SQL implementation. (Intermediate level)</p> <p>SQL_ISV_TABLE_CONSTRAINTS = Identifies the table constraints that are owned by a given user. (Intermediate level)</p> <p>SQL_ISV_TABLE_PRIVILEGES = Identifies the privileges on persistent tables that are available to or granted by a given user. (FIPS Transitional level)</p> <p>SQL_ISV_TABLES = Identifies the persistent tables defined in a catalog that are accessible to a given user. (FIPS Transitional level)</p> <p>SQL_ISV_TRANSLATIONS = Identifies character translations for the catalog that are accessible to a given user. (Full level)</p> <p>SQL_ISV_USAGE_PRIVILEGES = Identifies the USAGE privileges on catalog objects that are available to or owned by a given user. (FIPS Transitional level)</p> <p>SQL_ISV_VIEW_COLUMN_USAGE = Identifies the columns on which the catalog's views that are owned by a given user are dependent. (Intermediate level)</p> <p>SQL_ISV_VIEW_TABLE_USAGE = Identifies the tables on which the catalog's views that are owned by a given user are dependent. (Intermediate level)</p> <p>SQL_ISV_VIEWS = Identifies the viewed tables defined in this catalog that are accessible to a given user. (FIPS Transitional level)</p>
SQL_INSERT_STATEMENT (ODBC 3.0)	<p>An SQLINTEGER bitmask that indicates support for INSERT statements:</p> <p>SQL_IS_INSERT_LITERALS</p> <p>SQL_IS_INSERT_SEARCHED</p> <p>SQL_IS_SELECT_INTO</p>

	An SQL-92 Entry level–conformant driver will always return all of these options as supported.
SQL_INTEGRITY (ODBC 1.0)	<p>A character string: "Y" if the data source supports the Integrity Enhancement Facility; "N" if it does not.</p> <p>This <i>InfoType</i> has been renamed for ODBC 3.0 from the ODBC 2.0 <i>InfoType</i> SQL_ODBC_SQL_OPT_IEF.</p>
SQL_KEYSET_CURSOR_ATTRIBUTES1 (ODBC 3.0)	<p>An SQLINTEGER bitmask that describes the attributes of a keyset cursor that are supported by the driver. This bitmask contains the first subset of attributes; for the second subset, see SQL_KEYSET_CURSOR_ATTRIBUTES2.</p> <p>The following bitmasks are used to determine which attributes are supported:</p> <p>SQL_CA1_NEXT SQL_CA1_ABSOLUTE SQL_CA1_RELATIVE SQL_CA1_BOOKMARK SQL_CA1_LOCK_EXCLUSIVE SQL_CA1_LOCK_NO_CHANGE SQL_CA1_LOCK_UNLOCK SQL_CA1_POS_POSITION SQL_CA1_POS_UPDATE SQL_CA1_POS_DELETE SQL_CA1_POS_REFRESH SQL_CA1_POSITIONED_UPDATE SQL_CA1_POSITIONED_DELETE SQL_CA1_SELECT_FOR_UPDATE SQL_CA1_BULK_ADD SQL_CA1_BULK_UPDATE_BY_BOOKMARK SQL_CA1_BULK_DELETE_BY_BOOKMARK SQL_CA1_BULK_FETCH_BY_BOOKMARK</p> <p>For descriptions of these bitmasks, see SQL_DYNAMIC_CURSOR_ATTRIBUTES1 (and substitute "keyset-driven cursor" for "dynamic cursor" in the descriptions).</p> <p>An SQL-92 Intermediate level–conformant driver will usually return the SQL_CA1_NEXT, SQL_CA1_ABSOLUTE, and SQL_CA1_RELATIVE options as supported, because the driver supports scrollable cursors through the embedded SQL FETCH statement. Because this does not directly determine the underlying SQL support, however, scrollable cursors may not be supported, even for an SQL-92 Intermediate level–conformant driver.</p>
SQL_KEYSET_CURSOR_ATTRIBUTES2 (ODBC 3.0)	An SQLINTEGER bitmask that describes the attributes of a keyset cursor that are supported by the driver. This bitmask contains the second subset of attributes; for the first subset, see

	<p>SQL_KEYSET_CURSOR_ATTRIBUTES1.</p> <p>The following bitmasks are used to determine which attributes are supported:</p> <p>SQL_CA2_READ_ONLY_CONCURRENCY SQL_CA2_LOCK_CONCURRENCY SQL_CA2_OPT_ROWVER_CONCURRENCY SQL_CA2_OPT_VALUES_CONCURRENCY SQL_CA2_SENSITIVITY_ADDITIONS SQL_CA2_SENSITIVITY_DELETIONS SQL_CA2_SENSITIVITY_UPDATES SQL_CA2_MAX_ROWS_SELECT SQL_CA2_MAX_ROWS_INSERT SQL_CA2_MAX_ROWS_DELETE SQL_CA2_MAX_ROWS_UPDATE SQL_CA2_MAX_ROWS_CATALOG SQL_CA2_MAX_ROWS_AFFECTS_ALL SQL_CA2_CRC_EXACT SQL_CA2_CRC_APPROXIMATE SQL_CA2_SIMULATE_NON_UNIQUE SQL_CA2_SIMULATE_TRY_UNIQUE SQL_CA2_SIMULATE_UNIQUE</p> <p>For descriptions of these bitmasks, see SQL_DYNAMIC_CURSOR_ATTRIBUTES1 (and substitute "keyset-driven cursor" for "dynamic cursor" in the descriptions).</p>
SQL_KEYWORDS (ODBC 2.0)	<p>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. This list represents all the reserved keywords; interoperable applications should not use these words in object names.</p> <p>For a list of ODBC keywords, see "List of Reserved Keywords" in Appendix C, "SQL Minimum Grammar" of the SOLID Programmer Guide. The #define value SQL_ODBC_KEYWORDS contains a comma-separated list of ODBC keywords.</p>
SQL_LIKE_ESCAPE_CLAUSE (ODBC 2.0)	<p>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.</p>
SQL_MAX_ASYNC_CONCURRENT_STATEMENTS (ODBC 3.0)	<p>An SQLINTEGER value specifying the maximum number of active concurrent statements in asynchronous mode that the driver can support on a given connection. If there is no specific limit or the limit is unknown, this value is zero.</p>
SQL_MAX_BINARY_LITERAL_LEN (ODBC 2.0)	<p>An SQLINTEGER value specifying the maximum length (number of hexadecimal characters, excluding the literal prefix</p>

	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_CATALOG_NAME_LEN (ODBC 1.0)	An SQLUSMALLINT value specifying the maximum length of a catalog name in the data source. If there is no maximum length or the length is unknown, this value is set to zero. An FIPS Full level–conformant driver will return at least 128. This <i>InfoType</i> has been renamed for ODBC 3.0 from the ODBC 2.0 <i>InfoType</i> SQL_MAX_QUALIFIER_NAME_LEN.
SQL_MAX_CHAR_LITERAL_LEN (ODBC 2.0)	An SQLINTEGER 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)	An SQLUSMALLINT 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. An FIPS Entry level–conformant driver will return at least 18. An FIPS Intermediate level–conformant driver will return at least 128.
SQL_MAX_COLUMNS_IN_GROUP_BY (ODBC 2.0)	An SQLUSMALLINT 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. An FIPS Entry level–conformant driver will return at least 6. An FIPS Intermediate level–conformant driver will return at least 15.
SQL_MAX_COLUMNS_IN_INDEX (ODBC 2.0)	An SQLUSMALLINT 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)	An SQLUSMALLINT 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. An FIPS Entry level–conformant driver will return at least 6. An FIPS Intermediate level–conformant driver will return at least 15.
SQL_MAX_COLUMNS_IN_SELECT (ODBC 2.0)	An SQLUSMALLINT 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. An FIPS Entry level–conformant driver will return at least 100. An FIPS Intermediate level–conformant driver will return at least 250.
SQL_MAX_COLUMNS_IN_TABLE (ODBC 2.0)	An SQLUSMALLINT value specifying the maximum number of columns allowed in a table. If there is no specified limit or

	<p>the limit is unknown, this value is set to zero.</p> <p>An FIPS Entry level–conformant driver will return at least 100. An FIPS Intermediate level–conformant driver will return at least 250.</p>
SQL_MAX_CONCURRENT_ACTIVITIES (ODBC 1.0)	<p>An SQLUSMALLINT value specifying the maximum number of active statements that the driver can support for a connection. A statement is defined as active if it has results pending, with the term "results" meaning rows from a SELECT operation or rows affected by an INSERT, UPDATE, or DELETE operation (such as a row count), or if it is in a NEED_DATA state. 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.</p> <p>This <i>InfoType</i> has been renamed for ODBC 3.0 from the ODBC 2.0 <i>InfoType</i> SQL_ACTIVE_STATEMENTS.</p>
SQL_MAX_CURSOR_NAME_LEN (ODBC 1.0)	<p>An SQLUSMALLINT 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.</p> <p>An FIPS Entry level–conformant driver will return at least 18. An FIPS Intermediate level–conformant driver will return at least 128.</p>
SQL_MAX_DRIVER_CONNECTIONS (ODBC 1.0)	<p>An SQLUSMALLINT value specifying the maximum number of active connections that the driver can support for an environment. 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.</p> <p>This <i>InfoType</i> has been renamed for ODBC 3.0 from the ODBC 2.0 <i>InfoType</i> SQL_ACTIVE_CONNECTIONS.</p>
SQL_MAX_IDENTIFIER_LEN (ODBC 3.0)	<p>An SQLUSMALLINT that indicates the maximum size in characters that the data source supports for user-defined names.</p> <p>An FIPS Entry level–conformant driver will return at least 18. An FIPS Intermediate level–conformant driver will return at least 128.</p>
SQL_MAX_INDEX_SIZE (ODBC 2.0)	<p>An SQLINTEGER 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.</p>
SQL_MAX_PROCEDURE_NAME_LEN (ODBC 1.0)	<p>An SQLUSMALLINT 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.</p>
SQL_MAX_ROW_SIZE (ODBC 2.0)	<p>An SQLINTEGER 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.</p> <p>An FIPS Entry level–conformant driver will return at least 2,000. An FIPS Intermediate level–conformant driver will</p>

	return at least 8,000.
SQL_MAX_ROW_SIZE_INCLUDES_LONG (ODBC 3.0)	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_LONGVARIABLE columns in the row; "N" otherwise.
SQL_MAX_SCHEMA_NAME_LEN (ODBC 1.0)	<p>An SQLUSMALLINT value specifying the maximum length of a schema name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.</p> <p>An FIPS Entry level–conformant driver will return at least 18. An FIPS Intermediate level–conformant driver will return at least 128.</p> <p>This <i>InfoType</i> has been renamed for ODBC 3.0 from the ODBC 2.0 <i>InfoType</i> SQL_MAX_OWNER_NAME_LEN.</p>
SQL_MAX_STATEMENT_LEN (ODBC 2.0)	An SQLINTEGER 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)	<p>An SQLUSMALLINT 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.</p> <p>An FIPS Entry level–conformant driver will return at least 18. An FIPS Intermediate level–conformant driver will return at least 128.</p>
SQL_MAX_TABLES_IN_SELECT (ODBC 2.0)	<p>An SQLUSMALLINT 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.</p> <p>An FIPS Entry level–conformant driver will return at least 15. An FIPS Intermediate level–conformant driver will return at least 50.</p>
SQL_MAX_USER_NAME_LEN (ODBC 2.0)	An SQLUSMALLINT 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)	<p>A character string: "Y" if the data source supports multiple result sets, "N" if it does not.</p> <p>For more information on multiple result sets, see the Part I PDF file, Chapter 11, “Overview of Retrieving Results (Advanced).</p>
SQL_MULTIPLE_ACTIVE_TXN (ODBC 1.0)	<p>A character string: "Y" if the driver supports more than one active transaction at the same time, "N" if only one transaction can be active at any time.</p> <p>The information returned for this information type does not apply in the case of distributed transactions.</p>

<p>SQL_NEED_LONG_DATA_LEN (ODBC 2.0)</p>	<p>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.</p>
<p>SQL_NON_NULLABLE_COLUMNS (ODBC 1.0)</p>	<p>An SQLUSMALLINT value specifying whether the data source supports NOT NULL in columns:</p> <p>SQL_NNC_NULL = All columns must be nullable.</p> <p>SQL_NNC_NON_NULL = Columns cannot be nullable. (The data source supports the NOT NULL column constraint in CREATE TABLE statements.)</p> <p>An SQL-92 Entry level-conformant driver will return SQL_NNC_NON_NULL.</p>
<p>SQL_NULL_COLLATION (ODBC 2.0)</p>	<p>An SQLUSMALLINT value specifying where NULLs are sorted in a result set:</p> <p>SQL_NC_END = NULLs are sorted at the end of the result set, regardless of the ASC or DESC keywords.</p> <p>SQL_NC_HIGH = NULLs are sorted at the high end of the result set, depending on the ASC or DESC keywords.</p> <p>SQL_NC_LOW = NULLs are sorted at the low end of the result set, depending on the ASC or DESC keywords.</p> <p>SQL_NC_START = NULLs are sorted at the start of the result set, regardless of the ASC or DESC keywords.</p>
<p>SQL_NUMERIC_FUNCTIONS (ODBC 1.0)</p> <p>The information type was introduced in ODBC 1.0; each bitmask is labeled with the version in which it was introduced.</p>	<p>An SQLINTEGER bitmask enumerating the scalar numeric functions supported by the driver and associated data source.</p> <p>The following bitmasks are used to determine which numeric functions are supported:</p> <p>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)</p>

	<p>SQL_FN_NUM_POWER (ODBC 2.0)</p> <p>SQL_FN_NUM_RADIANS (ODBC 2.0)</p> <p>SQL_FN_NUM_RAND (ODBC 1.0)</p> <p>SQL_FN_NUM_ROUND (ODBC 2.0)</p> <p>SQL_FN_NUM_SIGN (ODBC 1.0)</p> <p>SQL_FN_NUM_SIN (ODBC 1.0)</p> <p>SQL_FN_NUM_SQRT (ODBC 1.0)</p> <p>SQL_FN_NUM_TAN (ODBC 1.0)</p> <p>SQL_FN_NUM_TRUNCATE (ODBC 2.0)</p>
SQL_ODBC_INTERFACE_CONFORMANCE (ODBC 3.0)	<p>An SQLINTEGER value indicating the level of the ODBC 3.x interface that the driver conforms to.</p> <p>SQL_OIC_CORE: The minimum level that all ODBC drivers are expected to conform to. This level includes basic interface elements such as connection functions, functions for preparing and executing an SQL statement, basic result set metadata functions, basic catalog functions, and so on.</p> <p>SQL_OIC_LEVEL1: A level including the core standards compliance level functionality, plus scrollable cursors, bookmarks, positioned updates and deletes, and so on.</p> <p>SQL_OIC_LEVEL2: A level including level 1 standards compliance level functionality, plus advanced features such as sensitive cursors; update, delete, and refresh by bookmarks; stored procedure support; catalog functions for primary and foreign keys; multicatalog support; and so on.</p> <p>For more information, see the Part I PDF file, "Interface Conformance Level" in Chapter 4, "ODBC Fundamentals."</p>
SQL_ODBC_VER (ODBC 1.0)	<p>A character string with the version of ODBC to which the Driver Manager conforms. The version is of the form ###.###.0000, 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.</p>
SQL_OJ_CAPABILITIES (ODBC 2.01)	<p>An SQLINTEGER bitmask enumerating the types of outer joins supported by the driver and data source. The following bitmasks are used to determine which types are supported:</p> <p>SQL_OJ_LEFT = Left outer joins are supported.</p> <p>SQL_OJ_RIGHT = Right outer joins are supported.</p> <p>SQL_OJ_FULL = Full outer joins are supported.</p> <p>SQL_OJ_NESTED = Nested outer joins are supported.</p> <p>SQL_OJ_NOT_ORDERED = The column names in the ON clause of the outer join do not have to be in the same order as their respective table names in the OUTER JOIN clause.</p> <p>SQL_OJ_INNER = The inner table (the right table in a left</p>

	<p>outer join or the left table in a right outer join) can also be used in an inner join. This does not apply to full outer joins, which do not have an inner table.</p> <p>SQL_OJ_ALL_COMPARISON_OPS = The comparison operator in the ON clause can be any of the ODBC comparison operators. If this bit is not set, only the equals (=) comparison operator can be used in outer joins.</p> <p>If none of these options is returned as supported, no outer join clause is supported.</p> <p>For information on the support of relational join operators in a SELECT statement, as defined by SQL-92, see SQL_SQL92_RELATIONAL_JOIN_OPERATORS.</p>
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_PARAM_ARRAY_ROW_COUNTS (ODBC 3.0)	<p>An SQLINTEGER enumerating the driver's properties regarding the availability of row counts in a parameterized execution. Has the following values:</p> <p>SQL_PARC_BATCH = Individual row counts are available for each set of parameters. This is conceptually equivalent to the driver generating a batch of SQL statements, one for each parameter set in the array. Extended error information can be retrieved by using the SQL_PARAM_STATUS_PTR descriptor field.</p> <p>SQL_PARC_NO_BATCH = There is only one row count available, which is the cumulative row count resulting from the execution of the statement for the entire array of parameters. This is conceptually equivalent to treating the statement along with the entire parameter array as one atomic unit. Errors are handled the same as if one statement were executed.</p>
SQL_PARAM_ARRAY_SELECTS (ODBC 3.0)	<p>An SQLINTEGER enumerating the driver's properties regarding the availability of result sets in a parameterized execution. Has the following values:</p> <p>SQL_PAS_BATCH = There is one result set available per set of parameters. This is conceptually equivalent to the driver generating a batch of SQL statements, one for each parameter set in the array.</p> <p>SQL_PAS_NO_BATCH = There is only one result set available, which represents the cumulative result set resulting from the execution of the statement for the entire array of parameters. This is conceptually equivalent to treating the statement along with the entire parameter array as one atomic unit.</p> <p>SQL_PAS_NO_SELECT = A driver does not allow a result-set generating statement to be executed with an array of</p>

	parameters.
SQL_PROCEDURE_TERM (ODBC 1.0)	A character string with the data source vendor's name for a procedure; for example, "database procedure", "stored procedure", "procedure", "package", or "stored query".
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_POS_OPERATIONS (ODBC 2.0)	<p>An SQLINTEGER bitmask enumerating the support operations in SQLSetPos.</p> <p>The following bitmasks are used in conjunction with the flag to determine which options are supported.</p> <p>SQL_POS_POSITION (ODBC 2.0) SQL_POS_REFRESH (ODBC 2.0) SQL_POS_UPDATE (ODBC 2.0) SQL_POS_DELETE (ODBC 2.0) SQL_POS_ADD (ODBC 2.0)</p>
SQL_QUOTED_IDENTIFIER_CASE (ODBC 2.0)	<p>An SQLUSMALLINT value as follows:</p> <p>SQL_IC_UPPER = Quoted identifiers in SQL are not case-sensitive and are stored in uppercase in the system catalog.</p> <p>SQL_IC_LOWER = Quoted identifiers in SQL are not case-sensitive and are stored in lowercase in the system catalog.</p> <p>SQL_IC_SENSITIVE = Quoted identifiers in SQL are case-sensitive and are stored in mixed case in the system catalog. (In an SQL-92-compliant database, quoted identifiers are always case-sensitive.)</p> <p>SQL_IC_MIXED = Quoted identifiers in SQL are not case-sensitive and are stored in mixed case in the system catalog.</p> <p>An SQL-92 Entry level-conformant driver will always return SQL_IC_SENSITIVE.</p>
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 updates made to a row by any user since the row was last fetched. (This applies only to updates, not to deletions or insertions.) The driver can return the SQL_ROW_UPDATED flag to the row status array when SQLFetchScroll is called. Otherwise, "N".
SQL_SCHEMA_TERM (ODBC 1.0)	<p>A character string with the data source vendor's name for an schema; for example, "owner", "Authorization ID", or "Schema".</p> <p>The character string can be returned in upper, lower, or mixed case.</p> <p>An SQL-92 Entry level-conformant driver will always return "schema".</p>

	<p>This <i>InfoType</i> has been renamed for ODBC 3.0 from the ODBC 2.0 <i>InfoType</i> SQL_OWNER_TERM.</p>
<p>SQL_SCHEMA_USAGE (ODBC 2.0)</p>	<p>An SQLINTEGER bitmask enumerating the statements in which schemas can be used:</p> <p>SQL_SU_DML_STATEMENTS = Schemas are supported in all Data Manipulation Language statements: SELECT, INSERT, UPDATE, DELETE, and if supported, SELECT FOR UPDATE and positioned update and delete statements.</p> <p>SQL_SU_PROCEDURE_INVOCATION = Schemas are supported in the ODBC procedure invocation statement.</p> <p>SQL_SU_TABLE_DEFINITION = Schemas are supported in all table definition statements: CREATE TABLE, CREATE VIEW, ALTER TABLE, DROP TABLE, and DROP VIEW.</p> <p>SQL_SU_INDEX_DEFINITION = Schemas are supported in all index definition statements: CREATE INDEX and DROP INDEX.</p> <p>SQL_SU_PRIVILEGE_DEFINITION = Schemas are supported in all privilege definition statements: GRANT and REVOKE.</p> <p>An SQL-92 Entry level-conformant driver will always return the SQL_SU_DML_STATEMENTS, SQL_SU_TABLE_DEFINITION, and SQL_SU_PRIVILEGE_DEFINITION options, as supported.</p> <p>This <i>InfoType</i> has been renamed for ODBC 3.0 from the ODBC 2.0 <i>InfoType</i> SQL_OWNER_USAGE.</p>
<p>SQL_SCROLL_OPTIONS (ODBC 1.0)</p> <p>The information type was introduced in ODBC 1.0; each bitmask is labeled with the version in which it was introduced.</p>	<p>An SQLINTEGER bitmask enumerating the scroll options supported for scrollable cursors.</p> <p>The following bitmasks are used to determine which options are supported:</p> <p>SQL_SO_FORWARD_ONLY = The cursor only scrolls forward. (ODBC 1.0)</p> <p>SQL_SO_STATIC = The data in the result set is static. (ODBC 2.0)</p> <p>SQL_SO_KEYSET_DRIVEN = The driver saves and uses the keys for every row in the result set. (ODBC 1.0)</p> <p>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)</p>

	<p>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)</p> <p>For information about scrollable cursors, see the Part I PDF file, "Scrollable Cursors" in Chapter 11, "Retrieving Results (Advanced)"</p>
SQL_SEARCH_PATTERN_ESCAPE (ODBC 1.0)	<p>A character string specifying what the driver supports as an escape character that permits the use of the pattern match metacharacters underscore (_) and percent sign (%) 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.</p> <p>Because this information type does not indicate general support of the escape character in the LIKE predicate, SQL-92 does not include requirements for this character string.</p> <p>This <i>InfoType</i> is limited to catalog functions. For a description of the use of the escape character in search pattern strings, see the Part I PDF file, "Pattern Value Arguments" in Chapter 7, "Catalog Functions."</p>
SQL_SERVER_NAME (ODBC 1.0)	<p>A character string with the actual data source-specific server name; useful when a data source name is used during SQLConnect, SQLDriverConnect, and SQLBrowseConnect.</p>
SQL_SPECIAL_CHARACTERS (ODBC 2.0)	<p>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 identifier name, such as a table name, column column name, or index name, on the data source. For example, "\$%^". If an identifier contains one or more of these characters, the identifier must be a delimited identifier.</p>
SQL_SQL_CONFORMANCE (ODBC 3.0)	<p>An SQLINTEGER value indicating the level of SQL-92 supported by the driver:</p> <p>SQL_SC_SQL92_ENTRY = Entry level SQL-92 compliant.</p> <p>SQL_SC_FIPS127_2_TRANSITIONAL = FIPS 127-2 transitional level compliant.</p> <p>SQL_SC_SQL92_FULL = Full level SQL-92 compliant.</p> <p>SQL_SC_SQL92_INTERMEDIATE = Intermediate level SQL-92 compliant.</p>
SQL_SQL92_DATETIME_FUNCTIONS (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the datetime scalar functions that are supported by the driver and the associated data source, as defined in SQL-92.</p>

	<p>The following bitmasks are used to determine which datetime functions are supported:</p> <p>SQL_SDF_CURRENT_DATE SQL_SDF_CURRENT_TIME SQL_SDF_CURRENT_TIMESTAMP</p>
SQL_SQL92_FOREIGN_KEY_DELETE_RULE (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the rules supported for a foreign key in a DELETE statement, as defined in SQL-92.</p> <p>The following bitmasks are used to determine which clauses are supported by the data source:</p> <p>SQL_SFKD_CASCADE SQL_SFKD_NO_ACTION SQL_SFKD_SET_DEFAULT SQL_SFKD_SET_NULL</p> <p>An FIPS Transitional level–conformant driver will always return all of these options as supported.</p>
SQL_SQL92_FOREIGN_KEY_UPDATE_RULE (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the rules supported for a foreign key in an UPDATE statement, as defined in SQL-92.</p> <p>The following bitmasks are used to determine which clauses are supported by the data source:</p> <p>SQL_SFKU_CASCADE SQL_SFKU_NO_ACTION SQL_SFKU_SET_DEFAULT SQL_SFKU_SET_NULL</p> <p>An SQL-92 Full level–conformant driver will always return all of these options as supported.</p>
SQL_SQL92_GRANT (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses supported in the GRANT statement, as defined in SQL-92.</p> <p>The SQL-92 or FIPS conformance level at which this feature needs to be supported is shown in parentheses next to each bitmask.</p> <p>The following bitmasks are used to determine which clauses are supported by the data source:</p> <p>SQL_SG_DELETE_TABLE (Entry level) SQL_SG_INSERT_COLUMN (Intermediate level) SQL_SG_INSERT_TABLE (Entry level) SQL_SG_REFERENCES_TABLE (Entry level) SQL_SG_REFERENCES_COLUMN (Entry level) SQL_SG_SELECT_TABLE (Entry level) SQL_SG_UPDATE_COLUMN (Entry level) SQL_SG_UPDATE_TABLE (Entry level)</p>

	SQL_SG_USAGE_ON_DOMAIN (FIPS Transitional level) SQL_SG_USAGE_ON_CHARACTER_SET (FIPS Transitional level) SQL_SG_USAGE_ON_COLLATION (FIPS Transitional level) SQL_SG_USAGE_ON_TRANSLATION (FIPS Transitional level) SQL_SG_WITH_GRANT_OPTION (Entry level)
SQL_SQL92_NUMERIC_VALUE_FUNCTIONS (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the numeric value scalar functions that are supported by the driver and the associated data source, as defined in SQL-92.</p> <p>The following bitmasks are used to determine which numeric functions are supported:</p> <p>SQL_SNVF_BIT_LENGTH SQL_SNVF_CHAR_LENGTH SQL_SNVF_CHARACTER_LENGTH SQL_SNVF_EXTRACT SQL_SNVF_OCTET_LENGTH SQL_SNVF_POSITION</p>
SQL_SQL92_PREDICATES (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the predicates supported in a SELECT statement, as defined in SQL-92.</p> <p>The SQL-92 or FIPS conformance level at which this feature needs to be supported is shown in parentheses next to each bitmask.</p> <p>The following bitmasks are used to determine which options are supported by the data source:</p> <p>SQL_SP_BETWEEN (Entry level) SQL_SP_COMPARISON (Entry level) SQL_SP_EXISTS (Entry level) SQL_SP_IN (Entry level) SQL_SP_ISNOTNULL (Entry level) SQL_SP_ISNULL (Entry level) SQL_SP_LIKE (Entry level) SQL_SP_MATCH_FULL (Full level) SQL_SP_MATCH_PARTIAL (Full level) SQL_SP_MATCH_UNIQUE_FULL (Full level) SQL_SP_MATCH_UNIQUE_PARTIAL (Full level) SQL_SP_OVERLAPS (FIPS Transitional level) SQL_SP_QUANTIFIED_COMPARISON (Entry level) SQL_SP_UNIQUE (Entry level)</p>
SQL_SQL92_RELATIONAL_JOIN_OPERATORS (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the relational join operators supported in a SELECT statement, as defined in SQL-92.</p> <p>The SQL-92 or FIPS conformance level at which this feature needs to be supported is shown in parentheses next to each</p>

	<p>bitmask.</p> <p>The following bitmasks are used to determine which options are supported by the data source:</p> <p>SQL_SRJO_CORRESPONDING_CLAUSE (Intermediate level) SQL_SRJO_CROSS_JOIN (Full level) SQL_SRJO_EXCEPT_JOIN (Intermediate level) SQL_SRJO_FULL_OUTER_JOIN (Intermediate level) SQL_SRJO_INNER_JOIN (FIPS Transitional level) SQL_SRJO_INTERSECT_JOIN (Intermediate level) SQL_SRJO_LEFT_OUTER_JOIN (FIPS Transitional level) SQL_SRJO_NATURAL_JOIN (FIPS Transitional level) SQL_SRJO_RIGHT_OUTER_JOIN (FIPS Transitional level) SQL_SRJO_UNION_JOIN (Full level)</p> <p>SQL_SRJO_INNER_JOIN indicates support for the INNER JOIN syntax, not for the inner join capability. Support for the INNER JOIN syntax is FIPS TRANSITIONAL, while support for the inner join capability is ENTRY.</p>
SQL_SQL92_REVOKE (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the clauses supported in the REVOKE statement, as defined in SQL-92, supported by the data source.</p> <p>The SQL-92 or FIPS conformance level at which this feature needs to be supported is shown in parentheses next to each bitmask.</p> <p>The following bitmasks are used to determine which clauses are supported by the data source:</p> <p>SQL_SR_CASCADE (FIPS Transitional level) SQL_SR_DELETE_TABLE (Entry level) SQL_SR_GRANT_OPTION_FOR (Intermediate level) SQL_SR_INSERT_COLUMN (Intermediate level) SQL_SR_INSERT_TABLE (Entry level) SQL_SR_REFERENCES_COLUMN (Entry level) SQL_SR_REFERENCES_TABLE (Entry level) SQL_SR_RESTRICT (FIPS Transitional level) SQL_SR_SELECT_TABLE (Entry level) SQL_SR_UPDATE_COLUMN (Entry level) SQL_SR_UPDATE_TABLE (Entry level) SQL_SR_USAGE_ON_DOMAIN (FIPS Transitional level) SQL_SR_USAGE_ON_CHARACTER_SET (FIPS Transitional level) SQL_SR_USAGE_ON_COLLATION (FIPS Transitional level) SQL_SR_USAGE_ON_TRANSLATION (FIPS Transitional level)</p>
SQL_SQL92_ROW_VALUE_CONSTRUCTOR	An SQLINTEGER bitmask enumerating the row value

(ODBC 3.0)	<p>constructor expressions supported in a SELECT statement, as defined in SQL-92. The following bitmasks are used to determine which options are supported by the data source:</p> <p>SQL_SRVC_VALUE_EXPRESSION SQL_SRVC_NULL SQL_SRVC_DEFAULT SQL_SRVC_ROW_SUBQUERY</p>
SQL_SQL92_STRING_FUNCTIONS (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the string scalar functions that are supported by the driver and the associated data source, as defined in SQL-92.</p> <p>The following bitmasks are used to determine which string functions are supported:</p> <p>SQL_SSF_CONVERT SQL_SSF_LOWER SQL_SSF_UPPER SQL_SSF_SUBSTRING SQL_SSF_TRANSLATE SQL_SSF_TRIM_BOTH SQL_SSF_TRIM_LEADING SQL_SSF_TRIM_TRAILING</p>
SQL_SQL92_VALUE_EXPRESSIONS (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the value expressions supported, as defined in SQL-92.</p> <p>The SQL-92 or FIPS conformance level at which this feature needs to be supported is shown in parentheses next to each bitmask.</p> <p>The following bitmasks are used to determine which options are supported by the data source:</p> <p>SQL_SVE_CASE (Intermediate level) SQL_SVE_CAST (FIPS Transitional level) SQL_SVE_COALESCE (Intermediate level) SQL_SVE_NULLIF (Intermediate level)</p>
SQL_STANDARD_CLI_CONFORMANCE (ODBC 3.0)	<p>An SQLINTEGER bitmask enumerating the CLI standard or standards to which the driver conforms. The following bitmasks are used to determine which levels the driver conforms to:</p> <p>SQL_SCC_XOPEN_CLI_VERSION1: The driver conforms to the X/Open CLI version 1.</p> <p>SQL_SCC_ISO92_CLI: The driver conforms to the ISO 92 CLI.</p>
SQL_STATIC_CURSOR_ATTRIBUTES1 (ODBC 3.0)	<p>An SQLINTEGER bitmask that describes the attributes of a static cursor that are supported by the driver. This bitmask contains the first subset of attributes; for the second subset, see SQL_STATIC_CURSOR_ATTRIBUTES2.</p>

	<p>The following bitmasks are used to determine which attributes are supported:</p> <p>SQL_CA1_NEXT SQL_CA1_ABSOLUTE SQL_CA1_RELATIVE SQL_CA1_BOOKMARK SQL_CA1_LOCK_NO_CHANGE SQL_CA1_LOCK_EXCLUSIVE SQL_CA1_LOCK_UNLOCK SQL_CA1_POS_POSITION SQL_CA1_POS_UPDATE SQL_CA1_POS_DELETE SQL_CA1_POS_REFRESH SQL_CA1_POSITIONED_UPDATE SQL_CA1_POSITIONED_DELETE SQL_CA1_SELECT_FOR_UPDATE SQL_CA1_BULK_ADD SQL_CA1_BULK_UPDATE_BY_BOOKMARK SQL_CA1_BULK_DELETE_BY_BOOKMARK SQL_CA1_BULK_FETCH_BY_BOOKMARK</p> <p>For descriptions of these bitmasks, see SQL_DYNAMIC_CURSOR_ATTRIBUTES1 (and substitute "static cursor" for "dynamic cursor" in the descriptions).</p> <p>An SQL-92 Intermediate level-conformant driver will usually return the SQL_CA1_NEXT, SQL_CA1_ABSOLUTE, and SQL_CA1_RELATIVE options as supported, because the driver supports scrollable cursors through the embedded SQL FETCH statement. Because this does not directly determine the underlying SQL support, however, scrollable cursors may not be supported, even for an SQL-92 Intermediate level-conformant driver.</p>
SQL_STATIC_CURSOR_ATTRIBUTES2 (ODBC 3.0)	<p>An SQLINTEGER bitmask that describes the attributes of a static cursor that are supported by the driver. This bitmask contains the second subset of attributes; for the first subset, see SQL_STATIC_CURSOR_ATTRIBUTES1.</p> <p>The following bitmasks are used to determine which attributes are supported:</p> <p>SQL_CA2_READ_ONLY_CONCURRENCY SQL_CA2_LOCK_CONCURRENCY SQL_CA2_OPT_ROWVER_CONCURRENCY SQL_CA2_OPT_VALUES_CONCURRENCY SQL_CA2_SENSITIVITY_ADDITIONS SQL_CA2_SENSITIVITY_DELETIONS SQL_CA2_SENSITIVITY_UPDATES SQL_CA2_MAX_ROWS_SELECT SQL_CA2_MAX_ROWS_INSERT</p>

	<p>SQL_CA2_MAX_ROWS_DELETE SQL_CA2_MAX_ROWS_UPDATE SQL_CA2_MAX_ROWS_CATALOG SQL_CA2_MAX_ROWS_AFFECTS_ALL SQL_CA2_CRC_EXACT SQL_CA2_CRC_APPROXIMATE SQL_CA2_SIMULATE_NON_UNIQUE SQL_CA2_SIMULATE_TRY_UNIQUE SQL_CA2_SIMULATE_UNIQUE</p> <p>For descriptions of these bitmasks, see SQL_DYNAMIC_CURSOR_ATTRIBUTES2 (and substitute "static cursor" for "dynamic cursor" in the descriptions).</p>
<p>SQL_STRING_FUNCTIONS (ODBC 1.0)</p> <p>The information type was introduced in ODBC 1.0; each bitmask is labeled with the version in which it was introduced.</p>	<p>An SQLINTEGER bitmask enumerating the scalar string functions supported by the driver and associated data source.</p> <p>The following bitmasks are used to determine which string functions are supported:</p> <p>SQL_FN_STR_ASCII (ODBC 1.0) SQL_FN_STR_BIT_LENGTH (ODBC 3.0) SQL_FN_STR_CHAR (ODBC 1.0) SQL_FN_STR_CHAR_LENGTH (ODBC 3.0) SQL_FN_STR_CHARACTER_LENGTH (ODBC 3.0) 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_LTRIM (ODBC 1.0) SQL_FN_STR_OCTET_LENGTH (ODBC 3.0) SQL_FN_STR_POSITION (ODBC 3.0) 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)</p> <p>If an application can call the LOCATE scalar function with the <i>string_exp1</i>, <i>string_exp2</i>, and <i>start</i> arguments, the driver returns the SQL_FN_STR_LOCATE bitmask. If an application can call the LOCATE scalar function with only the <i>string_exp1</i> and <i>string_exp2</i> arguments, the driver returns the SQL_FN_STR_LOCATE_2 bitmask. Drivers that fully</p>

	<p>support the LOCATE scalar function return both bitmasks.</p> <p>(For more information, see Appendix E, “Scalar Functions” in the SOLID Programmer Guide.)</p>
SQL_SUBQUERIES (ODBC 2.0)	<p>An SQLINTEGER bitmask enumerating the predicates that support subqueries:</p> <p>SQL_SQ_CORRELATED_SUBQUERIES SQL_SQ_COMPARISON SQL_SQ_EXISTS SQL_SQ_IN SQL_SQ_QUANTIFIED</p> <p>The SQL_SQ_CORRELATED_SUBQUERIES bitmask indicates that all predicates that support subqueries support correlated subqueries.</p> <p>An SQL-92 Entry level–conformant driver will always return a bitmask in which all of these bits are set.</p>
SQL_SYSTEM_FUNCTIONS (ODBC 1.0)	<p>An SQLINTEGER bitmask enumerating the scalar system functions supported by the driver and associated data source.</p> <p>The following bitmasks are used to determine which system functions are supported:</p> <p>SQL_FN_SYS_DBNAME SQL_FN_SYS_IFNULL SQL_FN_SYS_USERNAME</p>
SQL_TABLE_TERM (ODBC 1.0)	<p>A character string with the data source vendor's name for a table; for example, "table" or "file".</p> <p>This character string can be in upper, lower, or mixed case.</p> <p>An SQL-92 Entry level–conformant driver will always return "table".</p>
SQL_TIMEDATE_ADD_INTERVALS (ODBC 2.0)	<p>An SQLINTEGER bitmask enumerating the timestamp intervals supported by the driver and associated data source for the TIMESTAMPADD scalar function.</p> <p>The following bitmasks are used to determine which intervals are supported:</p> <p>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</p>

	An FIPS Transitional level–conformant driver will always return a bitmask in which all of these bits are set.
SQL_TIMEDATE_DIFF_INTERVALS (ODBC 2.0)	<p>An SQLINTEGER bitmask enumerating the timestamp intervals supported by the driver and associated data source for the TIMESTAMPDIFF scalar function.</p> <p>The following bitmasks are used to determine which intervals are supported:</p> <p>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</p> <p>An FIPS Transitional level–conformant driver will always return a bitmask in which all of these bits are set.</p>
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.	<p>An SQLINTEGER bitmask enumerating the scalar date and time functions supported by the driver and associated data source.</p> <p>The following bitmasks are used to determine which date and time functions are supported:</p> <p>SQL_FN_TD_CURRENT_DATE ODBC 3.0) SQL_FN_TD_CURRENT_TIME (ODBC 3.0) SQL_FN_TD_CURRENT_TIMESTAMP (ODBC 3.0) 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_EXTRACT (ODBC 3.0) 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)</p>
SQL_TXN_CAPABLE	An SQLUSMALLINT value describing the transaction support

<p>(ODBC 1.0)</p> <p>The information type was introduced in ODBC 1.0; each return value is labeled with the version in which it was introduced.</p>	<p>in the driver or data source:</p> <p>SQL_TC_NONE = Transactions not supported. (ODBC 1.0)</p> <p>SQL_TC_DML = Transactions can contain only Data Manipulation Language (DML) statements (SELECT, INSERT, UPDATE, DELETE). Data Definition Language (DDL) statements encountered in a transaction cause an error. (ODBC 1.0)</p> <p>SQL_TC_DDL_COMMIT = Transactions can contain only DML statements. DDL statements (CREATE TABLE, DROP INDEX, and so on) encountered in a transaction cause the transaction to be committed. (ODBC 2.0)</p> <p>SQL_TC_DDL_IGNORE = Transactions can contain only DML statements. DDL statements encountered in a transaction are ignored. (ODBC 2.0)</p> <p>SQL_TC_ALL = Transactions can contain DDL statements and DML statements in any order. (ODBC 1.0)</p> <p>(Because support of transactions is mandatory in SQL-92, an SQL-92 conformant driver [any level] will never return SQL_TC_NONE.)</p>
<p>SQL_TXN_ISOLATION_OPTION (ODBC 1.0)</p>	<p>An SQLINTEGER bitmask enumerating the transaction isolation levels available from the driver or data source.</p> <p>The following bitmasks are used in conjunction with the flag to determine which options are supported:</p> <p>SQL_TXN_READ_UNCOMMITTED SQL_TXN_READ_COMMITTED SQL_TXN_REPEATABLE_READ SQL_TXN_SERIALIZABLE</p> <p>For descriptions of these isolation levels, see the description of SQL_DEFAULT_TXN_ISOLATION.</p> <p>To set the transaction isolation level, an application calls SQLSetConnectAttr to set the SQL_ATTR_TXN_ISOLATION attribute. For more information, see SQLSetConnectAttr.</p> <p>An SQL-92 Entry level–conformant driver will always return SQL_TXN_SERIALIZABLE as supported. A FIPS Transitional level–conformant driver will always return all of these options as supported.</p>
<p>SQL_UNION (ODBC 2.0)</p>	<p>An SQLINTEGER bitmask enumerating the support for the UNION clause:</p> <p>SQL_U_UNION = The data source supports the UNION clause.</p>

	<p>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.)</p> <p>An SQL-92 Entry level-conformant driver will always return both of these options as supported.</p>
SQL_USER_NAME (ODBC 1.0)	A character string with the name used in a particular database, which can be different from the login name.
SQL_XOPEN_CLI_YEAR (ODBC 3.0)	A character string that indicates the year of publication of the X/Open specification with which the version of the ODBC Driver Manager fully complies.

Code Example

SQLGetInfo returns lists of supported options as an SQLINTEGER bitmask in **InfoValuePtr*. 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 connection:

```
SQLINTEGER    fFuncs;

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

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

Related Functions

For information about	See
Returning the setting of a connection attribute	SQLGetConnectAttr
Determining whether a driver supports a function	SQLGetFunctions
Returning the setting of a statement attribute	SQLGetStmtAttr
Returning information about a data source's data types	SQLGetTypeInfo

SQLGetStmtAttr

Conformance

Version Introduced: ODBC 3.0
Standards Compliance: ISO 92

Summary

SQLGetStmtAttr returns the current setting of a statement attribute.

Note For more information about what the Driver Manager maps this function to when an ODBC 3.x application is working with an ODBC 2.x driver, see the Part I PDF file, "Mapping Replacement Functions for Backward Compatibility of Applications" in Chapter 17, "Programming Considerations."

Syntax

SQLRETURN SQLGetStmtAttr(
SQLHSTMT	StatementHandle,
SQLINTEGER	Attribute,
SQLPOINTER	ValuePtr,
SQLINTEGER	BufferLength,
SQLINTEGER *	StringLengthPtr);

Arguments

StatementHandle

[Input]
Statement handle.

Attribute

[Input]
Attribute to retrieve.

ValuePtr

[Output]
Pointer to a buffer in which to return the value of the attribute specified in *Attribute*.

BufferLength

[Input]
If *Attribute* is an ODBC-defined attribute and *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **ValuePtr*. If *Attribute* is an ODBC-defined attribute and **ValuePtr* is an integer, *BufferLength* is ignored. If the value returned in **ValuePtr* is a Unicode string (when calling **SQLGetStmtAttrW**), the *BufferLength* argument must be an even number

If *Attribute* is a driver-defined attribute, the application indicates the nature of the attribute to the Driver Manager by setting the *BufferLength* argument. *BufferLength* can have the following values:

- If **ValuePtr* is a pointer to a character string, then *BufferLength* is the length of the string or SQL_NTS.
- If **ValuePtr* is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(*length*) macro in *BufferLength*. This places a negative value in *BufferLength*.
- If **ValuePtr* is a pointer to a value other than a character string or binary string, then *BufferLength* should have the value SQL_IS_POINTER.

- If **ValuePtr* contains a fixed-length data type, then *BufferLength* is either SQL_IS_INTEGER or SQL_IS_UINTEGER, as appropriate.

StringLengthPtr

[Output]

A pointer to a buffer in which to return the total number of bytes (excluding the null-termination character) available to return in **ValuePtr*. If *ValuePtr* is a null pointer, no length is returned. If the attribute value is a character string, and the number of bytes available to return is greater than or equal to *BufferLength*, the data in **ValuePtr* is truncated to *BufferLength* minus the length of a null-termination character and is null-terminated by the driver.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLGetStmtAttr** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. The following table lists the SQLSTATE values commonly returned by **SQLGetStmtAttr** 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	String data, right truncated	The data returned in <i>*ValuePtr</i> was truncated to be <i>BufferLength</i> minus the length of a null-termination character. The length of the untruncated string value is returned in <i>*StringLengthPtr</i> . (Function returns SQL_SUCCESS_WITH_INFO.)
24000	Invalid cursor state	The argument <i>Attribute</i> was SQL_ATTR_ROW_NUMBER 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.
HY000	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 SQLGetDiagRec in the argument <i>MessageText</i> describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error	(DM) An asynchronously executing function was called for the <i>StatementHandle</i> and was still executing when this function was called. (DM) SQLExecute , SQLExecDirect ,

		SQLBulkOperations , or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA . This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	(DM) <i>*ValuePtr</i> is a character string, and <i>BufferLength</i> was less than zero, but not equal to SQL_NTS .
HY092	Invalid attribute/option identifier	The value specified for the argument <i>Attribute</i> was not valid for the version of ODBC supported by the driver.
HY109	Invalid cursor position	The <i>Attribute</i> argument was SQL_ATTR_ROW_NUMBER and the row had been deleted or could not be fetched.
HYC00	Optional feature not implemented	The value specified for the argument <i>Attribute</i> was a valid ODBC statement attribute for the version of ODBC supported by the driver, but was not supported by the driver.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver corresponding to the <i>StatementHandle</i> does not support the function.

Comments

For general information about statement attributes, see the Part I PDF file, "Statement Attributes" in Chapter 9, "Executing Statements."

A call to **SQLGetStmtAttr** returns in **ValuePtr* the value of the statement attribute specified in *Attribute*. That value can either be a 32-bit value or a null-terminated character string. If the value is a null-terminated string, the application specifies the maximum length of that string in the *BufferLength* argument, and the driver returns the length of that string in the **StringLengthPtr* buffer. If the value is a 32-bit value, the *BufferLength* and *StringLengthPtr* arguments are not used.

To allow applications calling **SQLGetStmtAttr** to work with ODBC 2.x drivers, a call to **SQLGetStmtAttr** is mapped in the Driver Manager to **SQLGetStmtOption**.

The following statement attributes are read-only, so can be retrieved by **SQLGetStmtAttr**, but not set by **SQLSetStmtAttr**:

SQL_ATTR_IMP_PARAM_DESC
SQL_ATTR_IMP_ROW_DESC
SQL_ATTR_ROW_NUMBER

For a list of attributes that can be set and retrieved, see **SQLSetStmtAttr**.

Related Functions

For information about	See
Returning the setting of a connection attribute	SQLGetConnectAttr
Setting a connection attribute	SQLSetConnectAttr
Setting a statement attribute	SQLSetStmtAttr

SQLGetStmtOption

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: Deprecated

Summary

In ODBC 3.x, the ODBC 2.0 function **SQLGetStmtOption** has been replaced by **SQLGetStmtAttr**. For more information, see [SQLGetStmtAttr](#).

Note For more information about what the Driver Manager maps this function to when an ODBC 2.x application is working with an ODBC 3.x driver, see “Mapping Deprecated Functions” (Appendix G, "Driver Guidelines for Backward Compatibility") contained on the Microsoft Web site (ODBC Programming Reference).

SQLGetTypeInfo

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: ISO 92

Summary

SQLGetTypeInfo returns information about data types supported by the data source. The driver returns the information in the form of an SQL result set. The data types are intended for use in Data Definition Language (DDL) statements.

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

Syntax

SQLRETURN SQLGetTypeInfo(SQLHSTMT SQLSMALLINT	 StatementHandle, DataType);
------------------------------------------------------	------------------------------------

Arguments

StatementHandle

[Input]

Statement handle for the result set.

DataType

[Input]

The SQL data type. This must be one of the values in the "SQL Data Types" section of Appendix D, "Data Types" in the **SOLID Programmer Guide** or a driver-specific SQL data type. SQL_ALL_TYPES specifies that information about all data types should be returned.

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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.)
01S02	Option value changed	A specified statement attribute was invalid because of implementation working conditions, so a similar value was temporarily substituted. (Call SQLGetStmtAttr to determine the temporarily substituted value.) The substitute value is valid for the <i>StatementHandle</i> until the cursor is closed. The statement attributes that can be changed are: SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ATTR_QUERY_TIMEOUT, and SQL_ATTR_SIMULATE_CURSOR. (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	A cursor was open on the <i>StatementHandle</i> , and SQLFetch or SQLFetchScroll had been called. This

		<p>error is returned by the Driver Manager if SQLFetch or SQLFetchScroll has not returned SQL_NO_DATA, and is returned by the driver if SQLFetch or SQLFetchScroll has returned SQL_NO_DATA.</p> <p>A result set was open on the <i>StatementHandle</i>, but SQLFetch or SQLFetchScroll had not been called.</p>
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function and the state of the transaction cannot be determined.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY004	Invalid SQL data type	The value specified for the argument <i>DataType</i> was neither a valid ODBC SQL data type identifier nor a driver-specific data type identifier supported by the driver.
HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>, then the function was called and, before it completed execution, SQLCancel was called on the <i>StatementHandle</i>. Then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called and, before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY010	Function sequence error	<p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HYC00	Optional feature not implemented	The combination of the current settings of the SQL_ATTR_CONCURRENCY and SQL_ATTR_CURSOR_TYPE statement attributes

		was not supported by the driver or data source. The <code>SQL_ATTR_USE_BOOKMARKS</code> statement attribute was set to <code>SQL_UB_VARIABLE</code> , and the <code>SQL_ATTR_CURSOR_TYPE</code> statement attribute was set to a cursor type for which the driver does not support bookmarks.
HYT00	Timeout expired	The query timeout period expired before the data source returned the result set. The timeout period is set through <code>SQLSetStmtAttr</code> , <code>SQL_ATTR_QUERY_TIMEOUT</code> .
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through <code>SQLSetConnectAttr</code> , <code>SQL_ATTR_CONNECTION_TIMEOUT</code> .
IM001	Driver does not support this function	(DM) The driver corresponding to the <i>StatementHandle</i> does not support the function.

Comments

SQLGetTypeInfo returns the results as a standard result set, ordered by `DATA_TYPE` and then by how closely the data type maps to the corresponding ODBC SQL data type. Data types defined by the data source take precedence over user-defined data types. For example, suppose that a data source defined `INTEGER` and `COUNTER` data types, where `COUNTER` is auto-incrementing, and that a user-defined data type `WHOLENUM` has also been defined. These would be returned in the order `INTEGER`, `WHOLENUM`, and `COUNTER`, because `WHOLENUM` maps closely to the ODBC SQL data type `SQL_INTEGER`, while the auto-incrementing data type, even though supported by the data source, does not map closely to an ODBC SQL data type. For information about how this information might be used, see the Part I PDF file, “DDL Statements” in Chapter 8, “SQL Statements.”

If the *DataType* argument specifies a data type which is valid for the version of ODBC supported by the driver, but is not supported by the driver, then it will return an empty result set.

Note For more information about the general use, arguments, and returned data of ODBC catalog functions, see the Part I PDF file, Chapter 7, “Catalog Functions.”

The following columns have been renamed for ODBC 3.x. The column name changes do not affect backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
PRECISION	COLUMN_SIZE
MONEY	FIXED_PREC_SCALE
AUTO_INCREMENT	AUTO_UNIQUE_VALUE

The following columns have been added to the results set returned by **SQLGetTypeInfo** for ODBC 3.x:

`SQL_DATA_TYPE`
`INTERVAL_PRECISION`
`SQL_DATETIME_SUB`
`NUM_PREC_RADIX`

The following table lists the columns in the result set. Additional columns beyond column 19 (INTERVAL_PRECISION) can be defined by the driver. An application should gain access to driver-specific columns by counting down from the end of the result set rather than specifying an explicit ordinal position. For more information, see the Part I PDF file, "Data Returned by Catalog Functions" in Chapter 7, "Catalog Functions."

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 data types returned by **SQLGetTypeInfo** are those supported by the data source. They are intended for use in Data Definition Language (DDL) statements. Drivers can return result-set data using data types other than the types returned by **SQLGetTypeInfo**. In creating the result set for a catalog function, the driver might use a data type that is not supported by the data source.

Column name	Column number	Data type	Comments
TYPE_NAME (ODBC 2.0)	1	Varchar 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 (ODBC 2.0)	2	Smallint not NULL	SQL data type. This can be an ODBC SQL data type or a driver-specific SQL data type. For datetime or interval data types, this column returns the concise data type (such as SQL_TYPE_TIME or SQL_INTERVAL_YEAR_TO_MONTH). For a list of valid ODBC SQL data types, see "SQL Data Types" in Appendix D, "Data Types" of the SOLID Programmer Guide . For information about driver-specific SQL data types, see the driver's documentation.
COLUMN_SIZE (ODBC 2.0)	3	Integer	The maximum column size that the server supports for this data type. For numeric data, this is the maximum precision. For string data, this is the length in characters. For datetime data types, this is the length in characters of the string representation (assuming the maximum allowed precision of the fractional seconds component). NULL is returned for data types where column size is not applicable. For interval data types, this is the number of characters in the character representation of the

			interval literal (as defined by the interval leading precision).
LITERAL_PREFIX (ODBC 2.0)	4	Varchar	Character or characters used to prefix a literal; for example, a single quotation mark (') 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 (ODBC 2.0)	5	Varchar	Character or characters used to terminate a literal; for example, a single quotation mark (') for character data types; NULL is returned for data types where a literal suffix is not applicable.
CREATE_PARAMS (ODBC 2.0)	6	Varchar	<p>A list of keywords, separated by commas, corresponding to each parameter that the application may specify in parentheses when using the name that is returned in the TYPE_NAME field. The keywords in the list can be any of the following: length, precision, or scale. They appear in the order that the syntax requires them to be used. For example, CREATE_PARAMS for DECIMAL would be "precision,scale"; CREATE_PARAMS for VARCHAR would equal "length." NULL is returned if there are no parameters for the data type definition; for example, INTEGER.</p> <p>The driver supplies the CREATE_PARAMS text in the language of the country where it is used.</p>
NULLABLE (ODBC 2.0)	7	Smallint not NULL	<p>Whether the data type accepts a NULL value:</p> <p>SQL_NO_NULLS if the data type does not accept NULL values.</p> <p>SQL_NULLABLE if the data type accepts NULL values.</p> <p>SQL_NULLABLE_UNKNOWN if it is not known whether the column accepts NULL values.</p>
CASE_SENSITIVE (ODBC 2.0)	8	Smallint not NULL	<p>Whether a character data type is case-sensitive in collations and comparisons:</p> <p>SQL_TRUE if the data type is a</p>

			<p>character data type and is case-sensitive.</p> <p>SQL_FALSE if the data type is not a character data type or is not case-sensitive.</p>
<p>SEARCHABLE (ODBC 2.0)</p>	9	Smallint not NULL	<p>How the data type is used in a WHERE clause:</p> <p>SQL_PRED_NONE if the column cannot be used in a WHERE clause. (This is the same as the SQL_UNSEARCHABLE value in ODBC 2.x.)</p> <p>SQL_PRED_CHAR if the column can be used in a WHERE clause, but only with the LIKE predicate. (This is the same as the SQL_LIKE_ONLY value in ODBC 2.x.)</p> <p>SQL_PRED_BASIC if the column can be used in a WHERE clause with all the comparison operators except LIKE (comparison, quantified comparison, BETWEEN, DISTINCT, IN, MATCH, and UNIQUE). (This is the same as the SQL_ALL_EXCEPT_LIKE value in ODBC 2.x.)</p> <p>SQL_SEARCHABLE if the column can be used in a WHERE clause with any comparison operator.</p>
<p>UNSIGNED_ATTRIBUTE (ODBC 2.0)</p>	10	Smallint	<p>Whether the data type is unsigned:</p> <p>SQL_TRUE if the data type is unsigned.</p> <p>SQL_FALSE if the data type is signed.</p> <p>NULL is returned if the attribute is not applicable to the data type or the data type is not numeric.</p>
<p>FIXED_PREC_SCALE (ODBC 2.0)</p>	11	Smallint not NULL	<p>Whether the data type has predefined fixed precision and scale (which are data source-specific), such as a money data type:</p> <p>SQL_TRUE if it has predefined fixed precision and scale.</p>

			SQL_FALSE if it does not have predefined fixed precision and scale.
AUTO_UNIQUE_VALUE (ODBC 2.0)	12	Smallint	<p>Whether the data type is autoincrementing:</p> <p>SQL_TRUE if the data type is autoincrementing.</p> <p>SQL_FALSE if the data type is not autoincrementing.</p> <p>NULL is returned if the attribute is not applicable to the data type or the data type is not numeric.</p> <p>An application can insert values into a column having this attribute, but typically cannot update the values in the column.</p> <p>When an insert is made into an auto-increment column, a unique value is inserted into the column at insert time. The increment is not defined, but is data source-specific. An application should not assume that an auto-increment column starts at any particular point or increments by any particular value.</p>
LOCAL_TYPE_NAME (ODBC 2.0)	13	Varchar	<p>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.</p>
MINIMUM_SCALE (ODBC 2.0)	14	Smallint	<p>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_TYPE_TIMESTAMP column might have a fixed scale for fractional seconds. NULL is returned where scale is not applicable. For more information, see Appendix D, "Data Types" of the SOLID Programmer Guide.</p>
MAXIMUM_SCALE (ODBC 2.0)	15	Smallint	<p>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</p>

			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 COLUMN_SIZE column. For more information, see Appendix D, "Data Types" of the SOLID Programmer Guide .
SQL_DATA_TYPE (ODBC 3.0)	16	Smallint NOT NULL	<p>The value of the SQL data type as it appears in the SQL_DESC_TYPE field of the descriptor. This column is the same as the DATA_TYPE column, except for interval and datetime data types.</p> <p>For interval and datetime data types, the SQL_DATA_TYPE field in the result set will return SQL_INTERVAL or SQL_DATETIME, and the SQL_DATETIME_SUB field will return the subcode for the specific interval or datetime data type. (See Appendix D, "Data Types" of the SOLID Programmer Guide)</p>
SQL_DATETIME_SUB (ODBC 3.0)	17	Smallint	<p>When the value of SQL_DATA_TYPE is SQL_DATETIME or SQL_INTERVAL, this column contains the datetime/interval subcode. For data types other than datetime and interval, this field is NULL.</p> <p>For interval or datetime data types, the SQL_DATA_TYPE field in the result set will return SQL_INTERVAL or SQL_DATETIME, and the SQL_DATETIME_SUB field will return the subcode for the specific interval or datetime data type.</p>
NUM_PREC_RADIX (ODBC 3.0)	18	Integer	If the data type is an approximate numeric type, this column contains the value 2 to indicate that COLUMN_SIZE specifies a number of bits. For exact numeric types, this column contains the value 10 to indicate that COLUMN_SIZE specifies a number of decimal digits. Otherwise, this column is NULL.
INTERVAL_PRECISION (ODBC 3.0)	19	Smallint	If the data type is an interval data type, then this column contains the value of the interval leading precision. Otherwise, this column is NULL.

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

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Returning information about a column in a result set	SQLColAttribute
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Fetching a single row or a block of data in a forward-only direction	SQLFetch
Returning information about a driver or data source	SQLGetInfo

SQLMoreResults

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ODBC

Summary

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

Syntax

```
SQLRETURN SQLMoreResults(
    SQLHSTMT StatementHandle);
```

Arguments

StatementHandle

[Input]

Statement handle.

Returns

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

Diagnostics

When **SQLMoreResults** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value can be obtained by calling **SQLGetDiagRec** with a *HandleType* of

SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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.)
01S02	Option value has changed	The value of a statement attribute changed as the batch was being processed. (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.
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function and the state of the transaction cannot be determined.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	Asynchronous processing was enabled for the <i>StatementHandle</i> . The function was called and, before it completed execution, SQLCancel was called on the <i>StatementHandle</i> . Then the function was called again on the <i>StatementHandle</i> . The function was called and, before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.
HY010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called. (DM) SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.

HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

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 (numbered in increasing parameter order, in the order that they appear in the batch), or in procedures, they can return multiple result sets or row counts. For information about batches of statements and arrays of parameters, see the Part I PDF file, “Batches of SQL Statements” and “Arrays of Parameter Values” in Chapter 9, “Executing Statements.”

After executing the batch, the application is positioned on the first result set. The application can call **SQLBindCol**, **SQLBulkOperations**, **SQLFetch**, **SQLGetData**, **SQLFetchScroll**, **SQLSetPos**, and all the metadata functions, on the first or any subsequent result sets, just as it would if there were just a single result set. Once it is done with the first result set, the application calls **SQLMoreResults** to move to the next result set. If another result set or count is available, **SQLMoreResults** returns **SQL_SUCCESS** and initializes the result set or count for additional processing. If any row count–generating statements appear in between result set–generating statements, they can be stepped over by calling **SQLMoreResults**. After calling **SQLMoreResults** for **UPDATE**, **INSERT**, or **DELETE** statements, an application can call **SQLRowCount**.

If there was a current result set with unfetched rows, **SQLMoreResults** discards that result set and makes the next result set or count available. If all results have been processed, **SQLMoreResults** returns **SQL_NO_DATA**. For some drivers, output parameters and return values are not available until all result sets and row counts have been processed. For such drivers, output parameters and return values become available when **SQLMoreResults** returns **SQL_NO_DATA**.

Any bindings that were established for the previous result set still remain valid. If the column structures are different for this result set, then calling **SQLFetch** or **SQLFetchScroll** may result in an error or truncation. To prevent this, the application has to call **SQLBindCol** to explicitly rebind as appropriate (or do so by setting descriptor fields). Alternatively, the application can call **SQLFreeStmt** with an *Option* of **SQL_UNBIND** to unbind all the column buffers.

The values of statement attributes, such as cursor type, cursor concurrency, keyset size, or maximum length, may change as the application navigates through the batch by calls to **SQLMoreResults**. If this happens, **SQLMoreResults** will return **SQL_SUCCESS_WITH_INFO** and **SQLSTATE 01S02** (Option value has changed).

Calling **SQLCloseCursor**, or **SQLFreeStmt** with an *Option* of **SQL_CLOSE**, discards all the result sets and row counts that were available as a result of the execution of the batch. The statement handle returns to either the allocated or prepared state. Calling **SQLCancel** to cancel an asynchronously executing function when a batch has been executed and the statement handle is in the executed, cursor-positioned, or asynchronous state results in all the results sets and row counts generated by the batch being discarded if the cancel call was successful. The statement then returns to the prepared or allocated state.

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**.

For more information, see the Part I PDF file, "Multiple Results" in Chapter 11, "Retrieving Results (Advanced)."

If a searched update or delete statement in a batch of statements does not affect any rows at the data source, **SQLMoreResults** returns **SQL_SUCCESS**. This is different from the case of a searched update or delete statement that is executed through **SQLExecDirect**, **SQLExecute**, or **SQLParamData**, which returns **SQL_NO_DATA** if it does not affect any rows at the data source. If an application calls **SQLRowCount** to retrieve the row count after a call to **SQLMoreResults** has not affected any rows, **SQLRowCount** will return **SQL_NO_DATA**.

For additional information about the valid sequencing of result-processing functions, see Appendix B, "ODBC State Transition Tables" contained on the Microsoft Web site (ODBC Programming Reference).

Availability of Row Counts

When a batch contains multiple consecutive row count-generating statements, it is possible that these row counts are rolled up into just one row count. For example, if a batch has five insert statements, then certain data sources are capable of returning five individual row counts. Certain other data sources return only one row count that represents the sum of the five individual row counts.

When a batch contains a combination of result set-generating and row count-generating statements, row counts may or may not be available at all. The behavior of the driver with respect to the availability of row counts is enumerated in the **SQL_BATCH_ROW_COUNT** information type available through a call to **SQLGetInfo**. For example, suppose that the batch contains a **SELECT**, followed by two **INSERT**s and another **SELECT**. Then the following cases are possible:

The row counts corresponding to the two **INSERT** statements are not available at all. The first call to **SQLMoreResults** will position you on the result set of the second **SELECT** statement.

The row counts corresponding to the two **INSERT** statements are available individually. (A call to **SQLGetInfo** does not return the **SQL_BRC_ROLLED_UP** bit for the **SQL_BATCH_ROW_COUNT** information type.) The first call to **SQLMoreResults** will position you on the row count of the first **INSERT**, and the second call will position you on the row count of the second **INSERT**. The third call to **SQLMoreResults** will position you on the result set of the second **SELECT** statement.

The row counts corresponding to the two **INSERT**s are rolled up into one single row count that is available. (A call to **SQLGetInfo** returns the **SQL_BRC_ROLLED_UP** bit for the **SQL_BATCH_ROW_COUNT** information type.) The first call to **SQLMoreResults** will position you on the rolled-up row count, and the second call to **SQLMoreResults** will position you on the result set of the second **SELECT**.

Certain drivers make row counts available only for explicit batches and not for stored procedures.

Related Functions

For information about	See
Canceling statement processing	SQLCancel
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Fetching a single row or a block of data in a forward-only direction	SQLFetch

SQLNativeSql

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ODBC

Summary

SQLNativeSql returns the SQL string as modified by the driver. **SQLNativeSql** does not execute the SQL statement.

Syntax

SQLRETURN SQLNativeSql(
SQLHDBC	ConnectionHandle,
SQLCHAR *	InStatementText,
SQLINTEGER	TextLength1,
SQLCHAR *	OutStatementText,
SQLINTEGER	BufferLength,
SQLINTEGER *	TextLength2Ptr);

Arguments

ConnectionHandle

[Input]

Connection handle.

InStatementText

[Input]

SQL text string to be translated.

TextLength1

[Input]

Length of **InStatementText* text string.

OutStatementText

[Output]

Pointer to a buffer in which to return the translated SQL string.

BufferLength

[Input]

Length of the **OutStatementText* buffer. If the value returned in **InStatementText* is a Unicode string (when calling **SQLNativeSqlW**), the *BufferLength* argument must be an even number.

TextLength2Ptr

[Output]

Pointer to a buffer in which to return the total number of bytes (excluding the null-termination byte) available to return in **OutStatementText*. If the number of bytes available to return is greater than or equal to *BufferLength*, the translated SQL string in **OutStatementText* is truncated to *BufferLength* minus the length of a null-termination character.

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_DBC and a *Handle* of *ConnectionHandle*. 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	String data, right truncated	The buffer <i>*OutStatementText</i> was not large enough to return the entire SQL string, so the SQL string was truncated. The length of the untruncated SQL string is returned in <i>*TextLength2Ptr</i> . (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection does not exist	The <i>ConnectionHandle</i> 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.
22007	Invalid datetime format	<i>*InStatementText</i> contained an escape clause with an invalid date, time, or timestamp value.
24000	Invalid cursor state	The cursor referred to in the statement was positioned before the start of the result set or after the end of the result set. This error may not be returned by a driver having a native DBMS cursor implementation.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.

HY009	Invalid use of null pointer	(DM) * <i>InStatementText</i> was a null pointer.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	(DM) The argument <i>TextLength1</i> was less than 0, but not equal to SQL_NTS.
		(DM) The argument <i>BufferLength</i> was less than 0 and the argument <i>OutStatementText</i> was not a null pointer.
HY109	Invalid cursor position	The current row of the cursor had been deleted or had not been fetched. This error may not be returned by a driver having a native DBMS cursor implementation.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>ConnectionHandle</i> does not support the function.

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 Microsoft 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
```

For more information, see the Part I PDF file, “Direct Execution” and “Prepared Execution” in Chapter 9, “Executing Statements.”

Related Functions

None.

SQLNumParams

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

SQLNumParams returns the number of parameters in an SQL statement.

Syntax

SQLRETURN SQLNumParams(
SQLHSTMT	StatementHandle,
SQLSMALLINT *	ParameterCountPtr);

Arguments

StatementHandle

[Input]

Statement handle.

ParameterCountPtr

[Output]

Pointer to a buffer in which to return the 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 **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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.)
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.
HY000	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 SQLGetDiagRec in the * <i>MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	Asynchronous processing was enabled for the <i>StatementHandle</i> . The function was called and, before

		<p>it completed execution, SQLCancel was called on the <i>StatementHandle</i>; the function was then called again on the <i>StatementHandle</i>.</p> <p>The function was called and, before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY010	Function sequence error	<p>(DM) The function was called prior to calling SQLPrepare or SQLExecDirect for the <i>StatementHandle</i>.</p> <p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned <code>SQL_NEED_DATA</code>. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , <code>SQL_ATTR_CONNECTION_TIMEOUT</code> .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

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

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

The number of parameters returned by **SQLNumParams** is the same value as the `SQL_DESC_COUNT` field of the IPD.

For more information, see the Part I PDF file, "Describing Parameters" in Chapter 9, "Executing Statements."

Related Functions

For information about	See
Binding a buffer to a parameter	SQLBindParameter
Returning information about a parameter in a	SQLDescribeParam

statement	
-----------	--

SQLNumResultCols

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

SQLNumResultCols returns the number of columns in a result set.

Syntax

SQLRETURN SQLNumResultCols(
SQLHSTMT	StatementHandle,
SQLSMALLINT *	ColumnCountPtr);

Arguments

StatementHandle

[Input]

Statement handle.

ColumnCountPtr

[Output]

Pointer to a buffer in which to return the number of columns in the result set. This count does not include a bound bookmark column.

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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.)
08S01	Communication link	The communication link between the driver and the

	failure	data source to which the driver was connected failed before the function completed processing.
HY000	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 SQLGetDiagRec in the * <i>MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	Asynchronous processing was enabled for the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> ; the function was then called again on the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.
HY010	Function sequence error	(DM) The function was called prior to calling SQLPrepare or SQLExecDirect for the <i>StatementHandle</i> . (DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called. (DM) SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

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 statement.

Comments

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

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

The number of columns returned by **SQLNumResultCols** is the same value as the SQL_DESC_COUNT field of the IRD.

For more information, see the Part I PDF file, “Was a Result Set Created” and “How is Metadata Used” in Chapter 10, "Retrieving Results (Basic)."

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Returning information about a column in a result set	SQLColAttribute
Returning information about a column in a result set	SQLDescribeCol
Executing an SQL statement	SQLExecDirect
Executing a prepared SQL statement	SQLExecute
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Fetching a single row or a block of data in a forward-only direction	SQLFetch
Fetching part or all of a column of data	SQLGetData
Preparing an SQL statement for execution	SQLPrepare
Setting cursor scrolling options	SQLSetStmtAttr

SQLParamData

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

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

Syntax

SQLRETURN SQLParamData(
SQLHSTMT	StatementHandle,
SQLPOINTER *	ValuePtrPtr);

Arguments

StatementHandle

[Input]

Statement handle.

ValuePtrPtr

[Output]

Pointer to a buffer in which to return the address of the *ParameterValuePtr* buffer specified in **SQLBindParameter** (for parameter data) or the address of the *TargetValuePtr* buffer specified in **SQLBindCol** (for column data), as contained in the SQL_DESC_DATA_PTR descriptor record field.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_NO_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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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.)
07006	Restricted data type attribute violation	<p>The data value identified by the <i>ValueType</i> argument in SQLBindParameter for the bound parameter could not be converted to the data type identified by the <i>ParameterType</i> argument in SQLBindParameter.</p> <p>The data value returned for a parameter bound as SQL_PARAM_INPUT_OUTPUT or SQL_PARAM_OUTPUT could not be converted to the data type identified by the <i>ValueType</i> argument in SQLBindParameter.</p> <p>(If the data values for one or more rows could not be converted, but one or more rows were successfully returned, this function returns SQL_SUCCESS_WITH_INFO.)</p>
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

		<p>SQL_LONGVARCHAR, SQL_LONGVARBINARY, or a long data source-specific data type) than was specified with the <i>StrLen_or_IndPtr</i> argument in SQLBindParameter.</p> <p>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 SQLBulkOperations or updated with SQLSetPos.</p>
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>; the function was then called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY010	Function sequence error	<p>(DM) The previous function call was not a call to SQLExecDirect, SQLExecute, SQLBulkOperations, or SQLSetPos where the return code was SQL_NEED_DATA, or the previous function call was a call to SQLPutData.</p> <p>The previous function call was a call to SQLParamData.</p> <p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. SQLCancel was called</p>

		before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver that corresponds to the <i>StatementHandle</i> does not support the function.

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 **SQLBulkOperations** or being updated with **SQLSetPos**, it can return any SQLSTATE that can be returned by **SQLBulkOperations** or **SQLSetPos**.

Comments

SQLParamData can be called to supply data-at-execution data for two uses: parameter data to be used in a call to **SQLExecute** or **SQLExecDirect**, or column data to be used when a row is updated or added by a call to **SQLBulkOperations** or updated by a call to **SQLSetPos**. At execution time, **SQLParamData** returns to the application an indicator of which data the driver requires.

When an application calls **SQLExecute**, **SQLExecDirect**, **SQLBulkOperations**, or **SQLSetPos**, the driver returns **SQL_NEED_DATA** if it needs data-at-execution data. An application then calls **SQLParamData** to determine which data to send. If the driver requires parameter data, the driver returns in the **ValuePtrPtr* output buffer the value that the application placed in the rowset buffer. The application can use this value to determine which parameter data the driver is requesting. If the driver requires column data, the driver returns in the **ValuePtrPtr* buffer the address that the column was originally bound to, in the following manner:

$$\text{Bound Address} + \text{Binding Offset} + ((\text{Row Number} - 1) \times \text{Element Size})$$

where the variables are defined as indicated in the following table.

Variable	Description
Bound Address	The address specified with the <i>TargetValuePtr</i> argument in SQLBindCol .
Binding Offset	The value stored at the address specified with the SQL_ATTR_ROW_BIND_OFFSET_PTR statement attribute.
Row Number	The 1-based number of the row in the rowset. For single-row fetches, which are the default, this is 1.
Element Size	The value of the SQL_ATTR_ROW_BIND_TYPE statement attribute for both data and length/indicator buffers.

It also returns `SQL_NEED_DATA`, which is an indicator to the application that it should call **SQLPutData** to send the data.

The application calls **SQLPutData** as many times as necessary to send the data-at-execution data for the column or parameter. After all of the data has been sent for the column or parameter, the application calls **SQLParamData** again. If **SQLParamData** again returns `SQL_NEED_DATA`, data needs to be sent for another parameter or column, so the application again calls **SQLPutData**. If all data-at-execution data has been sent for all parameters or columns, then **SQLParamData** returns `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`, the value in **ValuePtrPtr* is undefined, and the SQL statement can be executed or the **SQLBulkOperations** or **SQLSetPos** call can be processed.

If **SQLParamData** supplies parameter data for a searched update or delete statement that does not affect any rows at the data source, the call to **SQLParamData** returns `SQL_NO_DATA`.

For more information on how data-at-execution parameter data is passed at statement execution time, see "Passing Parameter Values" in [SQLBindParameter](#) and in the Part I PDF file, "Sending Long Data" in Chapter 9, "Executing Statements." For more information on how data-at-execution column data is updated or added, see the section "Using SQLSetPos" in [SQLSetPos](#), "Performing Bulk Updates Using Bookmarks" in [SQLBulkOperations](#), and in the Part I PDF file, "Long Data and SQLSetPos and SQLBulkOperations" in Chapter 12, "Updating Data."

Code Example

See [SQLPutData](#).

Related Functions

For information about	See
Binding a buffer to a parameter	SQLBindParameter
Canceling statement processing	SQLCancel
Returning information about a parameter in a statement	SQLDescribeParam
Executing an SQL statement	SQLExecDirect
Executing a prepared SQL statement	SQLExecute
Sending parameter data at execution time	SQLPutData

SQLParamOptions

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: Deprecated

Summary

The ODBC 2.0 function **SQLParamOptions** has been replaced in ODBC 3.x by calls to `SQLSetStmtAttr`.

Note For more information about what the Driver Manager maps this function to when an ODBC 2.x application is working with an ODBC 3.x driver, see "Mapping Deprecated Functions" (Appendix G, "Driver Guidelines for Backward Compatibility") on the Microsoft Web site (ODBC Programming Reference).

SQLPrepare

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

SQLPrepare prepares an SQL string for execution.

Syntax

SQLRETURN SQLPrepare (
SQLHSTMT	StatementHandle,
SQLCHAR *	StatementText,
SQLINTEGER	TextLength);

Arguments

StatementHandle

[Input]

Statement handle.

StatementText

[Input]

SQL text string.

TextLength

[Input]

Length of **StatementText*.

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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.)
01S02	Option value changed	A specified statement attribute was invalid because of implementation working conditions, so a similar value was temporarily substituted. (SQLGetStmtAttr can be called to determine what the temporarily substituted value is.) The substitute value is valid for the <i>StatementHandle</i> until the cursor is closed. The statement attributes that can be changed are: SQL_ATTR_CONCURRENCY SQL_ATTR_CURSOR_TYPE SQL_ATTR_KEYSET_SIZE SQL_ATTR_MAX_LENGTH SQL_ATTR_MAX_ROWS SQL_ATTR_QUERY_TIMEOUT SQL_ATTR_SIMULATE_CURSOR (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	* <i>StatementText</i> 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	* <i>StatementText</i> 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.
22018	Invalid character value for cast specification	* <i>StatementText</i> contained an SQL statement that contained a literal or parameter, and the value was incompatible with the data type of the associated table column.
22019	Invalid escape character	The argument <i>StatementText</i> contained a LIKE predicate with an ESCAPE in the WHERE clause, and the length of the escape character following ESCAPE was not equal to 1.
22025	Invalid escape sequence	The argument <i>StatementText</i> contained " LIKE pattern value ESCAPE escape character" in the WHERE clause, and the character following the escape character in the pattern value was neither "%" nor "_".
24000	Invalid cursor state	(DM) A cursor was open on the <i>StatementHandle</i> , and SQLFetch or SQLFetchScroll had been called. A cursor was open on the <i>StatementHandle</i> , but SQLFetch or SQLFetchScroll had not been called.
34000	Invalid cursor name	* <i>StatementText</i> contained a positioned DELETE or a positioned UPDATE , and the cursor referenced by the statement being prepared was not open.
3D000	Invalid catalog name	The catalog name specified in <i>StatementText</i> was

		invalid.
3F000	Invalid schema name	The schema name specified in <i>StatementText</i> was invalid.
42000	Syntax error or access violation	<p>*<i>StatementText</i> contained an SQL statement that was not preparable or contained a syntax error.</p> <p>*<i>StatementText</i> contained a statement for which the user did not have the required privileges.</p>
42S01	Base table or view already exists	* <i>StatementText</i> contained a CREATE TABLE or CREATE VIEW statement, and the table name or view name specified already exists.
42S02	Base table or view not found	<p>*<i>StatementText</i> contained a DROP TABLE or a DROP VIEW statement, and the specified table name or view name did not exist.</p> <p>*<i>StatementText</i> contained an ALTER TABLE statement, and the specified table name did not exist.</p> <p>*<i>StatementText</i> contained a CREATE VIEW statement, and a table name or view name defined by the query specification did not exist.</p> <p>*<i>StatementText</i> contained a CREATE INDEX statement, and the specified table name did not exist.</p> <p>*<i>StatementText</i> contained a GRANT or REVOKE statement, and the specified table name or view name did not exist.</p> <p>*<i>StatementText</i> contained a SELECT statement, and a specified table name or view name did not exist.</p> <p>*<i>StatementText</i> contained a DELETE, INSERT, or UPDATE statement, and the specified table name did not exist.</p> <p>*<i>StatementText</i> contained a CREATE TABLE statement, and a table specified in a constraint (referencing a table other than the one being created) did not exist.</p>
42S11	Index already exists	* <i>StatementText</i> contained a CREATE INDEX statement, and the specified index name already existed.
42S12	Index not found	* <i>StatementText</i> contained a DROP INDEX statement, and the specified index name did not exist.
42S21	Column already exists	* <i>StatementText</i> 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.
42S22	Column not found	* <i>StatementText</i> contained a CREATE INDEX

		<p>statement, and one or more of the column names specified in the column list did not exist.</p> <p>*<i>StatementText</i> contained a GRANT or REVOKE statement, and a specified column name did not exist.</p> <p>*<i>StatementText</i> contained a SELECT, DELETE, INSERT, or UPDATE statement, and a specified column name did not exist.</p> <p>*<i>StatementText</i> contained a CREATE TABLE statement, and a column specified in a constraint (referencing a table other than the one being created) did not exist.</p>
HY000	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 SQLGetDiagRec in the * <i>MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>, and then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY009	Invalid use of null pointer	(DM) <i>StatementText</i> was a null pointer.
HY010	Function sequence error	<p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	(DM) The argument <i>TextLength</i> was less than or equal to 0 but not equal to SQL_NTS.
HYC00	Optional feature not implemented	<p>The concurrency setting was invalid for the type of cursor defined.</p> <p>The SQL_ATTR_USE_BOOKMARKS statement</p>

		attribute was set to <code>SQL_UB_VARIABLE</code> , and the <code>SQL_ATTR_CURSOR_TYPE</code> statement attribute was set to a cursor type for which the driver does not support bookmarks.
HYT00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through <code>SQLSetStmtAttr</code> , <code>SQL_ATTR_QUERY_TIMEOUT</code> .
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through <code>SQLSetConnectAttr</code> , <code>SQL_ATTR_CONNECTION_TIMEOUT</code> .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

The application calls **SQLPrepare** to send an SQL statement to the data source for preparation. For more information about prepared execution, see the Part I PDF file, "Prepared Execution" in Chapter 9, "Executing Statements." The application can include one or more parameter markers in the SQL statement. To include a parameter marker, the application embeds a question mark (?) into the SQL string at the appropriate position. For information about parameters, see the Part I PDF file, "Statement Parameters" in Chapter 9, "Executing Statements."

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 **SQLEndTran**.

The driver can modify the statement to use the form of SQL used by the data source and then submit it to the data source for preparation. In particular, the driver modifies the escape sequences used to define SQL syntax for certain features. (For a description of SQL statement grammar, see the Part I PDF file, "Escape Sequences in ODBC" in Chapter 8, "SQL Statements," and Appendix C, "SQL Minimum Grammar" in the **SOLID Programmer Guide**. For the driver, a statement handle 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 the statement handle to refer to the statement in later function calls. The prepared statement associated with the statement handle can be reexecuted by calling **SQLExecute** until the application frees the statement with a call to **SQLFreeStmt** with the `SQL_DROP` option or until the statement handle 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. For some implementations, calling **SQLDescribeCol** or **SQLDescribeParam** after **SQLPrepare** might not be as efficient as calling the function after **SQLExecute** or **SQLExecDirect**.

Some drivers cannot return syntax errors or access violations when the application calls **SQLPrepare**. A driver can 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**, **SQLColAttribute**, and **SQLExecute**.

Depending on the capabilities of the driver and data source, parameter information (such as data types) might be checked when the statement is prepared (if all parameters have been bound) or when it is executed (if all parameters have not been bound). 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 statement. This prevents errors that are due to old parameter information being applied to the new statement.

Important Committing a transaction, either by explicitly calling **SQLEndTran** or by working in autocommit mode, can cause the data source to delete the access plans for all statements on a connection. For more information, see the **SQL_CURSOR_COMMIT_BEHAVIOR** and **SQL_CURSOR_ROLLBACK_BEHAVIOR** information types in [SQLGetInfo](#) and in the Part I PDF file, “Effect of Transactions on Cursors and Prepared Statements” in Chapter 14, “Transactions.”

Code Example

See [SQLBindParameter](#), [SQLPutData](#), and [SQLSetPos](#).

Related Functions

For information about	See
Allocating a statement handle	SQLAllocHandle
Binding a buffer to a column in a result set	SQLBindCol
Binding a buffer to a parameter	SQLBindParameter
Canceling statement processing	SQLCancel
Executing a commit or rollback operation	SQLEndTran
Executing an SQL statement	SQLExecDirect
Executing a prepared SQL statement	SQLExecute
Returning the number of rows affected by a statement	SQLRowCount
Setting a cursor name	SQLSetCursorName

SQLPrimaryKeys

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ODBC

Summary

SQLPrimaryKeys returns the column names that make up 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

SQLRETURN SQLPrimaryKeys(
SQLHSTMT	StatementHandle,
SQLCHAR *	CatalogName,
SQLSMALLINT	NameLength1,
SQLCHAR *	SchemaName,
SQLSMALLINT	NameLength2,
SQLCHAR *	TableName,
SQLSMALLINT	NameLength3);

Arguments

StatementHandle

[Input]
Statement handle.

CatalogName

[Input]
Catalog name. If a driver supports catalogs 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 catalogs. *CatalogName* cannot contain a string search pattern.

If the `SQL_ATTR_METADATA_ID` statement attribute is set to `SQL_TRUE`, *CatalogName* is treated as an identifier and its case is not significant. If it is `SQL_FALSE`, *CatalogName* is an ordinary argument; it is treated literally, and its case is significant. For more information, see the Part 1 PDF file, "Arguments in Catalog Functions" in Chapter 7, "Catalog Functions."

NameLength1

[Input]
Length in bytes of **CatalogName*.

SchemaName

[Input]
Schema name. If a driver supports schemas 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 schemas. *SchemaName* cannot contain a string search pattern.

If the `SQL_ATTR_METADATA_ID` statement attribute is set to `SQL_TRUE`, *SchemaName* is treated as an identifier and its case is not significant. If it is `SQL_FALSE`, *SchemaName* is an ordinary argument; it is treated literally, and its case is not significant.

NameLength2

[Input]
Length in bytes of **SchemaName*.

TableName

[Input]
Table name. This argument cannot be a null pointer. *TableName* cannot contain a string search pattern.

If the `SQL_ATTR_METADATA_ID` statement attribute is set to `SQL_TRUE`, *TableName* is treated as an identifier and its case is not significant. If it is `SQL_FALSE`, *TableName* is an ordinary argument; it is treated literally, and its case is not significant.

NameLength3

[Input]
Length in bytes of **TableName*.

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of **SQL_HANDLE_STMT** and a *Handle* of *StatementHandle*. 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 **SQLSTATE**s 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 <i>StatementHandle</i> , and SQLFetch or SQLFetchScroll had been called. A cursor was open on the <i>StatementHandle</i> , but SQLFetch or SQLFetchScroll had not been called.
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	Asynchronous processing was enabled for the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> . Then the function was called again on the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.
HY009	Invalid use of null pointer	(DM) The <i>TableName</i> argument was a null pointer. (DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE , the <i>CatalogName</i> argument was a null pointer, and SQLGetInfo with the SQL_CATALOG_NAME information type returns that catalog names are supported. (DM) The SQL_ATTR_METADATA_ID statement

		attribute was set to SQL_TRUE, and the <i>SchemaName</i> argument was a null pointer.
HY010	Function sequence error	<p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	<p>(DM) The value of one of the name length arguments was less than 0 but not equal to SQL_NTS, and the associated name argument is not a null pointer.</p> <p>The value of one of the name length arguments exceeded the maximum length value for the corresponding name.</p>
HYC00	Optional feature not implemented	<p>A catalog was specified, and the driver or data source does not support catalogs.</p> <p>A schema was specified and the driver or data source does not support schemas.</p> <p>The combination of the current settings of the SQL_ATTR_CONCURRENCY and SQL_ATTR_CURSOR_TYPE statement attributes was not supported by the driver or data source.</p> <p>The SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_VARIABLE, and the SQL_ATTR_CURSOR_TYPE statement attribute was set to a cursor type for which the driver does not support bookmarks.</p>
HYT00	Timeout expired	The timeout period expired before the data source returned the requested result set. The timeout period is set through SQLSetStmtAttr , SQL_ATTR_QUERY_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

SQLPrimaryKeys returns the results as a standard result set, ordered by TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and KEY_SEQ. For information about how this information might be used, see the Part I PDF file, “Uses of Catalog Data” in Chapter 7, “Catalog Functions.”

The following columns have been renamed for ODBC 3.x. The column name changes do not affect backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM

To determine the actual lengths of the TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and COLUMN_NAME columns, call **SQLGetInfo** with the SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN options.

Note For more information about the general use, arguments, and returned data of ODBC catalog functions, see the Part I PDF file, Chapter 7, “Catalog Functions.”

The following table lists the columns in the result set. Additional columns beyond column 6 (PK_NAME) can be defined by the driver. An application should gain access to driver-specific columns by counting down from the end of the result set rather than specifying an explicit ordinal position. For more information, see the Part I PDF file, “Data Returned by Catalog Functions” in Chapter 7, “Catalog Functions.”

Column name	Column number	Data type	Comments
TABLE_CAT (ODBC 1.0)	1	Varchar	Primary key table catalog name; NULL if not applicable to the data source. If a driver supports catalogs 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 catalogs.
TABLE_SCHEM (ODBC 1.0)	2	Varchar	Primary key table schema name; NULL if not applicable to the data source. If a driver supports schemas 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 schemas.
TABLE_NAME (ODBC 1.0)	3	Varchar not NULL	Primary key table name.
COLUMN_NAME (ODBC 1.0)	4	Varchar not NULL	Primary key column name. The driver returns an empty string for a column that does not have a name.
KEY_SEQ	5	Smallint	Column sequence number in key (starting

(ODBC 1.0)		not NULL	with 1).
PK_NAME (ODBC 2.0)	6	Varchar	Primary key name. NULL if not applicable to the data source.

Code Example

See [SQLForeignKeys](#).

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Fetching a single row or a block of data in a forward-only direction	SQLFetch
Returning the columns of foreign keys	SQLForeignKeys
Returning table statistics and indexes	SQLStatistics

SQLProcedureColumns

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ODBC

Summary

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 statement.

Syntax

SQLRETURN SQLProcedureColumns(
SQLHSTMT	StatementHandle,
SQLCHAR *	CatalogName,
SQLSMALLINT	NameLength1,
SQLCHAR *	SchemaName,
SQLSMALLINT	NameLength2,
SQLCHAR *	ProcName,
SQLSMALLINT	NameLength3,
SQLCHAR *	ColumnName,
SQLSMALLINT	NameLength4);

Arguments

StatementHandle

[Input]

Statement handle.

CatalogName

[Input]

Procedure catalog name. If a driver supports catalogs 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 catalogs. *CatalogName* cannot contain a string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *CatalogName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *CatalogName* is an ordinary argument; it is treated literally, and its case is significant. For more information, see the Part I PDF File, "Arguments in Catalog Functions" in Chapter 7, "Catalog Functions."

NameLength1

[Input]

Length of **CatalogName*.

SchemaName

[Input]

String search pattern for procedure schema names. If a driver supports schemas 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 schemas.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *SchemaName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *SchemaName* is a pattern value argument; it is treated literally, and its case is significant.

NameLength2

[Input]

Length of **SchemaName*.

ProcName

[Input]

String search pattern for procedure names.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *ProcName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *ProcName* is a pattern value argument; it is treated literally, and its case is significant.

NameLength3

[Input]

Length of **ProcName*.

ColumnName

[Input]

String search pattern for column names.

If the `SQL_ATTR_METADATA_ID` statement attribute is set to `SQL_TRUE`, *ColumnName* is treated as an identifier and its case is not significant. If it is `SQL_FALSE`, *ColumnName* is a pattern value argument; it is treated literally, and its case is significant.

NameLength4

[Input]

Length of **ColumnName*.

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of `SQL_HANDLE_STMT` and a *Handle* of *StatementHandle*. 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 `SQLSTATE`s 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 <code>SQL_SUCCESS_WITH_INFO</code> .)
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	<p>A cursor was open on the <i>StatementHandle</i>, and SQLFetch or SQLFetchScroll had been called. This error is returned by the Driver Manager if SQLFetch or SQLFetchScroll has not returned <code>SQL_NO_DATA</code>, and is returned by the driver if SQLFetch or SQLFetchScroll has returned <code>SQL_NO_DATA</code>.</p> <p>A cursor was open on the <i>StatementHandle</i>, but SQLFetch or SQLFetchScroll had not been called.</p>
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
HY000	General error	An error occurred for which there was no specific <code>SQLSTATE</code> and for which no implementation-specific

		SQLSTATE was defined. The error message returned by SQLERROR in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>. Then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY009	Invalid use of null pointer	<p>(DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE, the <i>CatalogName</i> argument was a null pointer, and the SQL_CATALOG_NAME <i>InfoType</i> returns that catalog names are supported.</p> <p>(DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE, and the <i>SchemaName</i>, <i>ProcName</i>, or <i>ColumnName</i> argument was a null pointer.</p>
HY010	Function sequence error	<p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY090	Invalid string or buffer length	<p>(DM) The value of one of the name length arguments was less than 0 but not equal to SQL_NTS.</p> <p>The value of one of the name length arguments exceeded the maximum length value for the corresponding catalog, schema, procedure, or column name.</p>
HYC00	Optional feature not implemented	<p>A procedure catalog was specified, and the driver or data source does not support catalogs.</p> <p>A procedure schema was specified, and the driver or data source does not support schemas.</p> <p>A string search pattern was specified for the procedure schema, procedure name, or column name, and the data source does not support search patterns for one or more</p>

		<p>of those arguments.</p> <p>The combination of the current settings of the <code>SQL_ATTR_CONCURRENCY</code> and <code>SQL_ATTR_CURSOR_TYPE</code> statement attributes was not supported by the driver or data source.</p> <p>The <code>SQL_ATTR_USE_BOOKMARKS</code> statement attribute was set to <code>SQL_UB_VARIABLE</code>, and the <code>SQL_ATTR_CURSOR_TYPE</code> statement attribute was set to a cursor type for which the driver does not support bookmarks.</p>
HYT00	Timeout expired	The timeout period expired before the data source returned the result set. The timeout period is set through SQLSetStmtAttr , <code>SQL_ATTR_QUERY_TIMEOUT</code> .
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , <code>SQL_ATTR_CONNECTION_TIMEOUT</code> .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

This function is typically used before statement execution to retrieve information about procedure parameters and the columns that make up the result set or sets returned by the procedure, if any. For more information, see the Part I PDF file, “Procedures” in Chapter 9, “Executing Statements.”

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

The *SchemaName*, *ProcName*, and *ColumnName* arguments accept search patterns. For more information about valid search patterns, see the Part I PDF file, “Pattern Value Arguments” in Chapter 7, “Catalog Functions.”

Note For more information about the general use, arguments, and returned data of ODBC catalog functions, see the Part I PDF file, Chapter 7, “Catalog Functions.”

SQLProcedureColumns returns the results as a standard result set, ordered by `PROCEDURE_CAT`, `PROCEDURE_SCHEM`, `PROCEDURE_NAME`, and `COLUMN_TYPE`. Column names are returned for each procedure in the following order: the name of the return value, the names of each parameter in the procedure invocation (in call order), and then the names of each column in the result set returned by the procedure (in column order).

Applications should bind driver-specific columns relative to the end of the result set. For more information, see the Part I PDF file, “Data Returned by Catalog Functions” in Chapter 7, “Catalog Functions.”

To determine the actual lengths of the `PROCEDURE_CAT`, `PROCEDURE_SCHEM`, `PROCEDURE_NAME`, and `COLUMN_NAME` columns, an application can call **SQLGetInfo** with the

SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_PROCEDURE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN options.

The following columns have been renamed for ODBC 3.x. The column name changes do not affect backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
PROCEDURE_QUALIFIER	PROCEDURE_CAT
PROCEDURE_OWNER	PROCEDURE_SCHEM
PRECISION	COLUMN_SIZE
LENGTH	BUFFER_LENGTH
SCALE	DECIMAL_DIGITS
RADIX	NUM_PREC_RADIX

The following columns have been added to the results set returned by **SQLProcedureColumns** for ODBC 3.x:

COLUMN_DEF
 DATETIME_CODE
 CHAR_OCTET_LENGTH
 ORDINAL_POSITION
 IS_NULLABLE

The following table lists the columns in the result set. Additional columns beyond column 19 (IS_NULLABLE) can be defined by the driver. An application should gain access to driver-specific columns by counting down from the end of the result set rather than specifying an explicit ordinal position. For more information, see the Part I PDF file, "Data Returned by Catalog Functions" in Chapter 7, "Catalog Functions."

Column name	Column number	Data type	Comments
PROCEDURE_CAT (ODBC 2.0)	1	Varchar	Procedure catalog name; NULL if not applicable to the data source. If a driver supports catalogs 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 catalogs.
PROCEDURE_SCHEM (ODBC 2.0)	2	Varchar	Procedure schema name; NULL if not applicable to the data source. If a driver supports schemas 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 schemas.
PROCEDURE_NAME (ODBC 2.0)	3	Varchar not NULL	Procedure name. An empty string is returned for a procedure that does not have a name.
COLUMN_NAME	4	Varchar	Procedure column name. The driver

(ODBC 2.0)		not NULL	returns an empty string for a procedure column that does not have a name.
COLUMN_TYPE (ODBC 2.0)	5	Smallint not NULL	<p>Defines the procedure column as a parameter or a result set column:</p> <p>SQL_PARAM_TYPE_UNKNOWN: The procedure column is a parameter whose type is unknown. (ODBC 1.0)</p> <p>SQL_PARAM_INPUT: The procedure column is an input parameter. (ODBC 1.0)</p> <p>SQL_PARAM_INPUT_OUTPUT: The procedure column is an input/output parameter. (ODBC 1.0)</p> <p>SQL_PARAM_OUTPUT: The procedure column is an output parameter. (ODBC 2.0)</p> <p>SQL_RETURN_VALUE: The procedure column is the return value of the procedure. (ODBC 2.0)</p> <p>SQL_RESULT_COL: The procedure column is a result set column. (ODBC 1.0)</p>
DATA_TYPE (ODBC 2.0)	6	Smallint not NULL	SQL data type. This can be an ODBC SQL data type or a driver-specific SQL data type. For datetime and interval data types, this column returns the concise data types (for example, SQL_TYPE_TIME or SQL_INTERVAL_YEAR_TO_MONTH). For a list of valid ODBC SQL data types, see “SQL Data Types” in Appendix D, “Data Types” of the SOLID Programmer Guide . For information about driver-specific SQL data types, see the driver's documentation.
TYPE_NAME (ODBC 2.0)	7	Varchar not NULL	Data source–dependent data type name; for example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINARY", or "CHAR () FOR BIT DATA".
COLUMN_SIZE (ODBC 2.0)	8	Integer	The column size of the procedure column on the data source. NULL is returned for data types where column size is not applicable. For more information concerning precision, see in Appendix D, “Data Types” of the SOLID Programmer Guide .
BUFFER_LENGTH (ODBC 2.0)	9	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 Appendix D, "Data Types" of the SOLID Programmer Guide .
DECIMAL_DIGITS (ODBC 2.0)	10	Smallint	The decimal digits of the procedure column on the data source. NULL is returned for data types where decimal digits is not applicable. For more information concerning decimal digits, see Appendix D, "Data Types" of the SOLID Programmer Guide .
NUM_PREC_RADIX (ODBC 2.0)	11	Smallint	<p>For numeric data types, either 10 or 2.</p> <p>If 10, the values in COLUMN_SIZE and DECIMAL_DIGITS give the number of decimal digits allowed for the column. For example, a DECIMAL(12,5) column would return a NUM_PREC_RADIX of 10, a COLUMN_SIZE of 12, and a DECIMAL_DIGITS of 5; a FLOAT column could return a NUM_PREC_RADIX of 10, a COLUMN_SIZE of 15, and a DECIMAL_DIGITS of NULL.</p> <p>If 2, the values in COLUMN_SIZE and DECIMAL_DIGITS give the number of bits allowed in the column. For example, a FLOAT column could return a NUM_PREC_RADIX of 2, a COLUMN_SIZE of 53, and a DECIMAL_DIGITS of NULL.</p> <p>NULL is returned for data types where NUM_PREC_RADIX is not applicable.</p>
NULLABLE (ODBC 2.0)	12	Smallint not NULL	<p>Whether the procedure column accepts a NULL value:</p> <p>SQL_NO_NULLS: The procedure column does not accept NULL values.</p> <p>SQL_NULLABLE: The procedure column accepts NULL values.</p> <p>SQL_NULLABLE_UNKNOWN: It is not known if the procedure column accepts NULL values.</p>
REMARKS (ODBC 2.0)	13	Varchar	A description of the procedure column.

COLUMN_DEF (ODBC 3.0)	14	Varchar	<p>The default value of the column.</p> <p>If NULL was specified as the default value, this column is the word NULL, not enclosed in quotation marks. If the default value cannot be represented without truncation, this column contains TRUNCATED, with no enclosing single quotation marks. If no default value was specified, this column is NULL.</p> <p>The value of COLUMN_DEF can be used in generating a new column definition, except when it contains the value TRUNCATED.</p>
SQL_DATA_TYPE (ODBC 3.0)	15	Smallint not NULL	<p>The value of the SQL data type as it appears in the SQL_DESC_TYPE field of the descriptor. This column is the same as the DATA_TYPE column, except for datetime and interval data types.</p> <p>For datetime and interval data types, the SQL_DATA_TYPE field in the result set will return SQL_INTERVAL or SQL_DATETIME, and the SQL_DATETIME_SUB field will return the subcode for the specific interval or datetime data type. (See Appendix D, "Data Types" of the SOLID Programmer Guide.)</p>
SQL_DATETIME_SUB (ODBC 3.0)	16	Smallint	The subtype code for datetime and interval data types. For other data types, this column returns a NULL.
CHAR_OCTET_LENGTH (ODBC 3.0)	17	Integer	The maximum length in bytes of a character or binary data type column. For all other data types, this column returns a NULL.
ORDINAL_POSITION (ODBC 3.0)	18	Integer not NULL	For input and output parameters, the ordinal position of the parameter in the procedure definition (in increasing parameter order, starting at 1). For a return value (if any), 0 is returned. For result-set columns, the ordinal position of the column in the result set, with the first column in the result set being number 1. If there are multiple result sets, column ordinal positions are returned in a driver-specific manner.
IS_NULLABLE (ODBC 3.0)	19	Varchar	"NO" if the column does not include NULLs.

			<p>"YES" if the column can include NULLs.</p> <p>This column returns a zero-length string if nullability is unknown.</p> <p>ISO rules are followed to determine nullability. An ISO SQL-compliant DBMS cannot return an empty string.</p> <p>The value returned for this column is different from the value returned for the NULLABLE column. (See the description of the NULLABLE column.)</p>
--	--	--	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Code Example

See the Part I PDF file, "Procedure Calls" in Chapter 8, "SQL Statements."

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Fetching a single row or a block of data in a forward-only direction	SQLFetch
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Returning a list of procedures in a data source	SQLProcedures

SQLProcedures

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ODBC

Summary

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. For more information on procedures, see the Part I PDF file, "Procedures" section in Chapter 9, "Executing Statements."

Syntax

SQLRETURN SQLProcedures(
SQLHSTMT	StatementHandle,

SQLCHAR *	CatalogName,
SQLSMALLINT	NameLength1,
SQLCHAR *	SchemaName,
SQLSMALLINT	NameLength2,
SQLCHAR *	ProcName,
SQLSMALLINT	NameLength3);

Arguments

StatementHandle

[Input]

Statement handle.

CatalogName

[Input]

Procedure catalog. If a driver supports catalogs 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 catalogs.

CatalogName cannot contain a string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *CatalogName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *CatalogName* is an ordinary argument; it is treated literally, and its case is significant. For more information, see the Part I PDF file, "Arguments in Catalog Functions" in Chapter 7, "Catalog Functions."

NameLength1

[Input]

Length in bytes of **CatalogName*.

SchemaName

[Input]

String search pattern for procedure schema names. If a driver supports schemas 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 schemas.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *SchemaName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *SchemaName* is a pattern value argument; it is treated literally, and its case is significant.

NameLength2

[Input]

Length in bytes of **SchemaName*.

ProcName

[Input]

String search pattern for procedure names.

If the `SQL_ATTR_METADATA_ID` statement attribute is set to `SQL_TRUE`, *ProcName* is treated as an identifier and its case is not significant. If it is `SQL_FALSE`, *ProcName* is a pattern value argument; it is treated literally, and its case is significant.

NameLength3

[Input]

Length in bytes of **ProcName*.

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of `SQL_HANDLE_STMT` and a *Handle* of *StatementHandle*. 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 `SQLSTATE`s 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 <code>SQL_SUCCESS_WITH_INFO</code> .)
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	A cursor was open on the <i>StatementHandle</i> , and SQLFetch or SQLFetchScroll had been called. This error is returned by the Driver Manager if SQLFetch or SQLFetchScroll has not returned <code>SQL_NO_DATA</code> , and is returned by the driver if SQLFetch or SQLFetchScroll has returned <code>SQL_NO_DATA</code> . A cursor was open on the <i>StatementHandle</i> , but SQLFetch or SQLFetchScroll had not been called.
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
HY000	General error	An error occurred for which there was no specific <code>SQLSTATE</code> and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.

HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>. Then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY009	Invalid use of null pointer	<p>(DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE, the <i>CatalogName</i> argument was a null pointer, and the SQL_CATALOG_NAME <i>InfoType</i> returns that catalog names are supported.</p> <p>(DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE, and the <i>SchemaName</i> or <i>ProcName</i> argument was a null pointer.</p>
HY010	Function sequence error	<p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	<p>(DM) The value of one of the name length arguments was less than 0 but not equal to SQL_NTS.</p> <p>The value of one of the name length arguments exceeded the maximum length value for the corresponding name.</p>
HYC00	Optional feature not implemented	<p>A procedure catalog was specified, and the driver or data source does not support catalogs.</p> <p>A procedure schema was specified, and the driver or data source does not support schemas.</p> <p>A string search pattern was specified for the procedure schema or procedure name, and the data source does not support search patterns for one or more of those arguments.</p> <p>The combination of the current settings of the</p>

		SQL_ATTR_CONCURRENCY and SQL_ATTR_CURSOR_TYPE statement attributes was not supported by the driver or data source. The SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_VARIABLE, and the SQL_ATTR_CURSOR_TYPE statement attribute was set to a cursor type for which the driver does not support bookmarks.
HYT00	Timeout expired	The query timeout period expired before the data source returned the requested result set. The timeout period is set through SQLSetStmtAttr , SQL_ATTR_QUERY_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support this function.

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 that it cannot execute. For information about how this information might be used, see the Part I PDF file, "Procedures" in Chapter 9, "Executing Statements."

Note For more information about the general use, arguments, and returned data of ODBC catalog functions, see the Part I PDF file, Chapter 7, "Catalog Functions."

SQLProcedures returns the results as a standard result set, ordered by PROCEDURE_CAT, PROCEDURE_SCHEMA, and PROCEDURE_NAME.

Note **SQLProcedures** might not return all procedures. Applications can use any valid procedure, regardless of whether it is returned by **SQLProcedures**.

The following columns have been renamed for ODBC 3.x. The column name changes do not affect backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
PROCEDURE_QUALIFIER	PROCEDURE_CAT
PROCEDURE_OWNER	PROCEDURE_SCHEMA

To determine the actual lengths of the PROCEDURE_CAT, PROCEDURE_SCHEMA, and PROCEDURE_NAME columns, an application can call **SQLGetInfo** with the SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, and SQL_MAX_PROCEDURE_NAME_LEN options.

The following table lists the columns in the result set. Additional columns beyond column 8 (PROCEDURE_TYPE) can be defined by the driver. An application should gain access to driver-specific columns by counting down from the end of the result set rather than specifying an explicit ordinal position. For more information, see the Part I PDF file, "Data Returned by Catalog Functions" in Chapter 7, "Catalog Functions."

Column name	Column number	Data type	Comments
PROCEDURE_CAT (ODBC 2.0)	1	Varchar	Procedure catalog identifier; NULL if not applicable to the data source. If a driver supports catalogs 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 catalogs.
PROCEDURE_SCHEM (ODBC 2.0)	2	Varchar	Procedure schema identifier; NULL if not applicable to the data source. If a driver supports schemas 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 schemas.
PROCEDURE_NAME (ODBC 2.0)	3	Varchar not NULL	Procedure identifier.
NUM_INPUT_PARAMS (ODBC 2.0)	4	N/A	Reserved for future use. Applications should not rely on the data returned in these result columns.
NUM_OUTPUT_PARAMS (ODBC 2.0)	5	N/A	Reserved for future use. Applications should not rely on the data returned in these result columns.
NUM_RESULT_SETS (ODBC 2.0)	6	N/A	Reserved for future use. Applications should not rely on the data returned in these result columns.
REMARKS (ODBC 2.0)	7	Varchar	A description of the procedure.
PROCEDURE_TYPE (ODBC 2.0)	8	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.

The *SchemaName* and *ProcName* arguments accept search patterns. For more information about valid search patterns, see the Part I PDF file, “Pattern Value Arguments” in Chapter 7, “Catalog Functions.”

Code Example

See the Part I PDF file, “Procedure Calls” in Chapter 8, “SQL Statements.”

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Fetching a single row or a block of data in a forward-only direction	SQLFetch
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Returning information about a driver or data source	SQLGetInfo
Returning the parameters and result set columns of a procedure	SQLProcedureColumns
Syntax for invoking stored procedures	See the Part I PDF file, Chapter 8, “Executing Statements”

SQLPutData

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

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_LONGVARIABLE` or `SQL_LONGVARIABLE` types). **SQLPutData** supports binding to a Unicode C data type, even if the underlying driver does not support Unicode data.

Syntax

SQLRETURN SQLPutData (
SQLHSTMT	StatementHandle,
SQLPOINTER	DataPtr,
SQLINTEGER	StrLen_or_Ind);

Arguments

StatementHandle

[Input]
Statement handle.

DataPtr

[Input]
Pointer to a buffer containing the actual data for the parameter or column. The data must be in the C data type specified in the *ValueType* argument of **SQLBindParameter** (for parameter data) or the *TargetType* argument of **SQLBindCol** (for column data).

StrLen_or_Ind

[Input]
Length of **DataPtr*. 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. *StrLen_or_Ind* is ignored unless it meets one of the following conditions:

StrLen_or_Ind 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.

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 *StrLen_or_Ind* is not SQL_NULL_DATA or SQL_DEFAULT_PARAM, the driver assumes that the size of the **DataPtr* buffer is the size of the C data type specified with *ValueType* or *TargetType* and sends the entire data value. For more information, see "Converting Data from C to SQL Data Types" in Appendix D, "Data Types" of the **SOLID Programmer Guide**.

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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	String data, right truncated	String or binary data returned for an output parameter resulted in the truncation of nonblank character or non-NULL binary data. If it was a string value, it was right-truncated. (Function returns SQL_SUCCESS_WITH_INFO.)
07006	Restricted data type attribute violation	The data value identified by the <i>ValueType</i> argument in SQLBindParameter for the bound parameter could not be converted to the data type identified by

		the <i>ParameterType</i> argument in SQLBindParameter .
07S01	Invalid use of default parameter	A parameter value, set with SQLBindParameter , was SQL_DEFAULT_PARAM, and the corresponding parameter did not have a default value.
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	<p>The assignment of a character or binary value to a column resulted in the truncation of nonblank (character) or non-null (binary) characters or bytes.</p> <p>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 <i>StrLen_or_IndPtr</i> argument in SQLBindParameter.</p> <p>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 SQLBulkOperations or updated with SQLSetPos.</p>
22003	Numeric value out of range	<p>The data sent for a bound 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.</p> <p>Returning a numeric value (as numeric or string) for one or more input/output or output parameters would have caused the whole (as opposed to fractional) part of the number to be truncated.</p>
22007	Invalid datetime format	<p>The data sent for a parameter or column that was bound to a date, time, or timestamp structure was, respectively, an invalid date, time, or timestamp.</p> <p>An input/output or output parameter was bound to a date, time, or timestamp C structure, and a value in the returned parameter was, respectively, an invalid date, time, or timestamp. (Function returns SQL_SUCCESS_WITH_INFO.)</p>
22008	Datetime field overflow	A datetime expression computed for an input/output or output parameter resulted in a date, time, or

		timestamp C structure that was invalid.
22012	Division by zero	An arithmetic expression calculated for an input/output or output parameter resulted in division by zero.
22015	Interval field overflow	<p>The data sent for an exact numeric or interval column or parameter to an interval SQL data type caused a loss of significant digits.</p> <p>Data was sent for an interval column or parameter with more than one field, was converted to a numeric data type, and had no representation in the numeric data type.</p> <p>The data sent for column or parameter data was assigned to an interval SQL type, and there was no representation of the value of the C type in the interval SQL type.</p> <p>The data sent for an exact numeric or interval C column or parameter to an interval C type caused a loss of significant digits.</p> <p>The data sent for column or parameter data was assigned to an interval C structure, and there was no representation of the data in the interval data structure.</p>
22018	Invalid character value for cast specification	<p>The C type was an exact or approximate numeric, a datetime, or an interval data type; the SQL type of the column was a character data type; and the value in the column or parameter was not a valid literal of the bound C type.</p> <p>The SQL type was an exact or approximate numeric, a datetime, or an interval data type; the C type was SQL_C_CHAR; and the value in the column or parameter was not a valid literal of the bound SQL type.</p>
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>. Then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed</p>

		execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.
HY009	Invalid use of null pointer	(DM) The argument <i>DataPtr</i> was a null pointer, and the argument <i>StrLen_or_Ind</i> was not 0, SQL_DEFAULT_PARAM , or SQL_NULL_DATA .
HY010	Function sequence error	(DM) The previous function call was not a call to SQLPutData or SQLParamData . (DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY019	Non-character and non-binary data sent in pieces	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.
HY020	Attempt to concatenate a null value	SQLPutData was called more than once since the call that returned SQL_NEED_DATA , and in one of those calls, the <i>StrLen_or_Ind</i> argument contained SQL_NULL_DATA or SQL_DEFAULT_PARAM .
HY090	Invalid string or buffer length	The argument <i>DataPtr</i> was not a null pointer, and the argument <i>StrLen_or_Ind</i> was less than 0 but not equal to SQL_NTS or SQL_NULL_DATA .
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

If **SQLPutData** 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 **SQLBulkOperations** or being updated with **SQLSetPos**, it can return any SQLSTATE that can be returned by **SQLBulkOperations** or **SQLSetPos**.

Comments

SQLPutData can be called to supply data-at-execution data for two uses: parameter data to be used in a call to **SQLExecute** or **SQLExecDirect**, or column data to be used when a row is updated or added by a call to **SQLBulkOperations** or is updated by a call to **SQLSetPos**.

When an application calls **SQLParamData** to determine which data it should send, the driver returns an indicator that the application can use to determine which parameter data to send or where column data can

be found. It also returns `SQL_NEED_DATA`, which is an indicator to the application that it should call **SQLPutData** to send the data. In the *DataPtr* argument to **SQLPutData**, the application passes a pointer to the buffer containing the actual data for the parameter or column.

When the driver returns `SQL_SUCCESS` for **SQLPutData**, the application calls **SQLParamData** again. **SQLParamData** returns `SQL_NEED_DATA` if more data needs to be sent, in which case the application calls **SQLPutData** again. It returns `SQL_SUCCESS` if all data-at-execution data has been sent. The application then calls **SQLParamData** again. If the driver returns `SQL_NEED_DATA` and another indicator in **ValuePtrPtr*, it requires data for another parameter or column and **SQLPutData** is called again. If the driver returns `SQL_SUCCESS`, then all data-at-execution data has been sent and the SQL statement can be executed or the **SQLBulkOperations** or **SQLSetPos** call can be processed.

For more information on how data-at-execution parameter data is passed at statement execution time, see "Passing Parameter Values" in [SQLBindParameter](#) and the Part I PDF file, "Sending Long Data" in Chapter 9, "Executing Statements." For more information on how data-at-execution column data is updated or added, see the section "Using SQLSetPos" in [SQLSetPos](#), "Performing Bulk Updates Using Bookmarks" in [SQLBulkOperations](#), and the Part I PDF file, "Long Data and SQLSetPos and SQLBulkOperation" in Chapter 12, "Updating Data."

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 HY019` (Non-character and non-binary data sent in pieces).

Code Example

In the following example, an application prepares an SQL statement to insert data into the `PICTURES` table. The statement contains parameters for the `PARTID` and `PICTURE` 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 second parameter will be passed at execution time and passes the value 2 for later retrieval by **SQLParamData**. This value will identify the parameter that is being processed.

The application calls **GetNextID** to get the next available part ID number. It then calls **SQLExecute** to execute the statement. **SQLExecute** returns `SQL_NEED_DATA` when it needs data for the second parameter. 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, at which point it returns `SQL_SUCCESS`.

For the second parameter, **InitUserData** calls a routine to prompt the user for the name of a file containing a bitmap photo of the part 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 MAX_DATA_LEN 1024
SQLINTEGER cbPartID = 0, cbPhotoParam, cbData;
SQLINTEGER sPartID;
szPhotoFile;
SQLPOINTER pToken, InitValue;
SQLCHAR Data[MAX_DATA_LEN];
SQLRETURN retcode;
```

```

SQLHSTMT  hstmt;

retcode = SQLPrepare(hstmt,
    "INSERT INTO PICTURES (PARTID, PICTURE) VALUES
    (?, ?)", SQL_NTS);
if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {

    /* Bind the parameters. For parameter 2, pass */
    /* the parameter number in ParameterValuePtr instead of a buffer */
    /* address. */

    SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_ULONG,
        SQL_INTEGER, 0, 0, &sPartID, 0, &cbPartID);
    SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT,
        SQL_C_BINARY, SQL_LONGVARIABLE,
        0, 0, (SQLPOINTER) 2, 0, &cbPhotoParam);

    /* Set values so data for parameter 2 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. */

    cbPhotoParam = SQL_LEN_DATA_AT_EXEC(0);

    sPartID = 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 */

    while (retcode == SQL_NEED_DATA) {
        retcode = SQLParamData(hstmt, &pToken);
        if (retcode == SQL_NEED_DATA) {
            InitUserData((SQLSMALLINT)pToken, InitValue);
            while (GetUserData(InitValue, (SQLSMALLINT)pToken, Data,
                &cbData))
                SQLPutData(hstmt, Data, cbData);
        }
    }
}

VOID InitUserData(sParam, InitValue)
SQLPOINTER InitValue;
{
    SQLCHAR szPhotoFile[MAX_FILE_NAME_LEN];

    /* 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)
    SQLPOINTER InitValue;
    SQLCHAR * Data;
    SQLINTEGER * cbData;
    BOOL Done;

    {

        /* 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);
    }

```

Related Functions

For information about	See
Binding a buffer to a parameter	SQLBindParameter
Canceling statement processing	SQLCancel
Executing an SQL statement	SQLExecDirect
Executing a prepared SQL statement	SQLExecute
Returning the next parameter to send data for	SQLParamData

SQLRowCount

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: ISO 92

Summary

SQLRowCount returns the number of rows affected by an **UPDATE**, **INSERT**, or **DELETE** statement; an SQL_ADD, SQL_UPDATE_BY_BOOKMARK, or SQL_DELETE_BY_BOOKMARK operation in **SQLBulkOperations**; or an SQL_UPDATE or SQL_DELETE operation in **SQLSetPos**.

Syntax

SQLRETURN SQLRowCount (
SQLHSTMT	StatementHandle,
SQLINTEGER *	RowCountPtr);

Arguments

StatementHandle

[Input]

Statement handle.

RowCountPtr

[Output]

Points to a buffer in which to return a row count. For **UPDATE**, **INSERT**, and **DELETE** statements, for the **SQL_ADD**, **SQL_UPDATE_BY_BOOKMARK**, and **SQL_DELETE_BY_BOOKMARK** operations in **SQLBulkOperations**, and for the **SQL_UPDATE** or **SQL_DELETE** operations in **SQLSetPos**, the value returned in **RowCountPtr* is either the number of rows affected by the request or -1 if the number of affected rows is not available.

When **SQLExecute**, **SQLExecDirect**, **SQLBulkOperations**, **SQLSetPos**, or **SQLMoreResults** is called, the **SQL_DIAG_ROW_COUNT** field of the diagnostic data structure is set to the row count, and the row count is cached in an implementation-dependent way. **SQLRowCount** returns the cached row count value. The cached row count value is valid until the statement handle is set back to the prepared or allocated state, the statement is reexecuted, or **SQLCloseCursor** is called. Note that if a function has been called since the **SQL_DIAG_ROW_COUNT** field was set, the value returned by **SQLRowCount** might be different from the value in the **SQL_DIAG_ROW_COUNT** field because the **SQL_DIAG_ROW_COUNT** field is reset to 0 by any function call.

For other statements and functions, the driver may define the value returned in **RowCountPtr*. 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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of **SQL_HANDLE_STMT** and a *Handle* of *StatementHandle*. 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 **SQLSTATE**s 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 .)
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer

		describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error	<p>(DM) The function was called prior to calling SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos for the <i>StatementHandle</i>.</p> <p>(DM) An asynchronously executing function was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

If the last SQL statement executed on the statement handle was not an **UPDATE**, **INSERT**, or **DELETE** statement or if the *Operation* argument in the previous call to **SQLBulkOperations** was not SQL_ADD, SQL_UPDATE_BY_BOOKMARK, or SQL_DELETE_BY_BOOKMARK, or if the *Operation* argument in the previous call to **SQLSetPos** was not SQL_UPDATE or SQL_DELETE, the value of **RowCountPtr* is driver-defined. For more information, see the Part I PDF file, "Determining the Number of Affected Rows" in Chapter 12, "Updating Data."

Related Functions

For information about	See
Executing an SQL statement	SQLExecDirect
Executing a prepared SQL statement	SQLExecute

SQLSetConnectAttr

Conformance

Version Introduced: ODBC 3.0
Standards Compliance: ISO 92

Summary

SQLSetConnectAttr sets attributes that govern aspects of connections.

Note For more information about what the Driver Manager maps this function to when an ODBC 3.x application is working with an ODBC 2.x driver, see the Part I PDF file, "Mapping Replacement Functions for Backward Compatibility of Applications" in Chapter 17, "Programming Considerations."

Syntax

SQLRETURN SQLSetConnectAttr(
SQLHDBC	ConnectionHandle,
SQLINTEGER	Attribute,
SQLPOINTER	ValuePtr,
SQLINTEGER	StringLength);

Arguments

ConnectionHandle

[Input]
Connection handle.

Attribute

[Input]
Attribute to set, listed in "Comments."

ValuePtr

[Input]
Pointer to the value to be associated with *Attribute*. Depending on the value of *Attribute*, *ValuePtr* will be a 32-bit unsigned integer value or will point to a null-terminated character string. Note that if the *Attribute* argument is a driver-specific value, the value in *ValuePtr* may be a signed integer.

StringLength

[Input]
If *Attribute* is an ODBC-defined attribute and *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **ValuePtr*. If *Attribute* is an ODBC-defined attribute and *ValuePtr* is an integer, *StringLength* is ignored.

If *Attribute* is a driver-defined attribute, the application indicates the nature of the attribute to the Driver Manager by setting the *StringLength* argument. *StringLength* can have the following values:

If *ValuePtr* is a pointer to a character string, then *StringLength* is the length of the string or SQL_NTS.

If *ValuePtr* is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(*length*) macro in *StringLength*. This places a negative value in *StringLength*. If *ValuePtr* is a pointer to a value other than a character string or a binary string, then *StringLength* should have the value SQL_IS_POINTER.

If *ValuePtr* contains a fixed-length value, then *StringLength* is either SQL_IS_INTEGER or SQL_IS_UNSIGNED, as appropriate.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLSetConnectAttr** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_DBC and a *Handle* of *ConnectionHandle*. The following table lists the SQLSTATE values commonly returned by **SQLSetConnectAttr** 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.

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 value specified in <i>ValuePtr</i> and substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
08002	Connection name in use	The <i>Attribute</i> argument was SQL_ATTR_ODBC_CURSORS, and the driver was already connected to the data source.
08003	Connection does not exist	(DM) An <i>Attribute</i> value was specified that required an open connection, but the <i>ConnectionHandle</i> 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.
24000	Invalid cursor state	The <i>Attribute</i> argument was SQL_ATTR_CURRENT_CATALOG, and a result set was pending.
3D000	Invalid catalog name	The <i>Attribute</i> argument was SQL_CURRENT_CATALOG, and the specified catalog name was invalid.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY009	Invalid use of null pointer	The <i>Attribute</i> argument identified a connection attribute that required a string value, and the <i>ValuePtr</i> argument was a null pointer.
HY010	Function sequence error	(DM) An asynchronously executing function was called for a <i>StatementHandle</i> associated with the

		<p><i>ConnectionHandle</i> and was still executing when SQLSetConnectAttr was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for a <i>StatementHandle</i> associated with the <i>ConnectionHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p> <p>(DM) SQLBrowseConnect was called for the <i>ConnectionHandle</i> and returned SQL_NEED_DATA. This function was called before SQLBrowseConnect returned SQL_SUCCESS_WITH_INFO or SQL_SUCCESS.</p>
HY011	Attribute cannot be set now	The <i>Attribute</i> argument was SQL_ATTR_TXN_ISOLATION, and a transaction was open.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY024	Invalid attribute value	<p>Given the specified <i>Attribute</i> value, an invalid value was specified in <i>ValuePtr</i>. (The Driver Manager returns this SQLSTATE only for connection and statement attributes that accept a discrete set of values, such as SQL_ATTR_ACCESS_MODE or SQL_ATTR_ASYNC_ENABLE. For all other connection and statement attributes, the driver must verify the value specified in <i>ValuePtr</i>.)</p> <p>The <i>Attribute</i> argument was SQL_ATTR_TRACEFILE or SQL_ATTR_TRANSLATE_LIB, and <i>ValuePtr</i> was an empty string.</p>
HY090	Invalid string or buffer length	(DM) * <i>ValuePtr</i> is a character string, and the <i>StringLength</i> argument was less than 0 but was not SQL_NTS.
HY092	Invalid attribute/option identifier	<p>(DM) The value specified for the argument <i>Attribute</i> was not valid for the version of ODBC supported by the driver.</p> <p>(DM) The value specified for the argument <i>Attribute</i> was a read-only attribute.</p>
HYC00	Optional feature not implemented	The value specified for the argument <i>Attribute</i> was a valid ODBC connection or statement attribute for the version of ODBC supported by the driver but was not supported by the driver.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr ,

		SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>ConnectionHandle</i> 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 be returned only when <i>Attribute</i> is SQL_ATTR_TRANSLATE_LIB.

When *Attribute* is a statement attribute, **SQLSetConnectAttr** can return any SQLSTATEs returned by **SQLSetStmtAttr**.

Comments

For general information about connection attributes, see the Part I PDF file, “Connection Attributes” in Chapter 6, “Connecting to a Data Source or Driver.”

The currently defined attributes and the version of ODBC in which they were introduced are shown in the table later in this section; it is expected that more attributes will be defined to take advantage of different data sources. A range of attributes is reserved by ODBC; driver developers must reserve values for their own driver-specific use from X/Open.

Note The ability to set statement attributes at the connection level by calling **SQLSetConnectAttr** has been deprecated in ODBC 3.x. ODBC 3.x applications should never set statement attributes at the connection level. ODBC 3.x statement attributes cannot be set at the connection level, with the exception of the SQL_ATTR_METADATA_ID and SQL_ATTR_ASYNC_ENABLE attributes, which are both connection attributes and statement attributes and can be set at either the connection level or the statement level.

ODBC 3.x drivers need only support this functionality if they should work with ODBC 2.x applications that set ODBC 2.x statement options at the connection level. For more information, see “SQLSetConnectOption Mapping” (Appendix G, “Driver Guidelines for Backward Compatibility”) contained on the Microsoft Web site (ODBC Programmer’s Reference).

An application can call **SQLSetConnectAttr** at any time between the time the connection is allocated and freed. All connection and statement attributes successfully set by the application for the connection persist until **SQLFreeHandle** is called on the connection. For example, if an application calls **SQLSetConnectAttr** before connecting to a data source, the attribute persists even if **SQLSetConnectAttr** fails in the driver when the application connects to the data source; if an application sets a driver-specific attribute, the attribute persists even if the application connects to a different driver on the connection.

Some connection attributes can be set only before a connection has been made; others can be set only after a connection has been made. The following table indicates those connection attributes that must be set either before or after a connection has been made. *Either* indicates that the attribute can be set either before or after connection.

Attribute	Set before or after connection?
SQL_ATTR_ACCESS_MODE	Either [1]
SQL_ATTR_ASYNC_ENABLE	Either [2]
SQL_ATTR_AUTOCOMMIT	Either
SQL_ATTR_CONNECTION_TI	Either

MEOUT	
SQL_ATTR_CURRENT_CATALOG	Either [1]
SQL_ATTR_LOGIN_TIMEOUT	Before
SQL_ATTR_METADATA_ID	Either
SQL_ATTR_ODBC_CURSORS	Before
SQL_ATTR_PACKET_SIZE	Before
SQL_ATTR_QUIET_MODE	Either
SQL_ATTR_TRACE	Either
SQL_ATTR_TRACEFILE	Either
SQL_ATTR_TRANSLATE_LIB	After
SQL_ATTR_TRANSLATE_OPTION	After
SQL_ATTR_TXN_ISOLATION	Either [3]

[1] SQL_ATTR_ACCESS_MODE and SQL_ATTR_CURRENT_CATALOG can be set before or after connecting, depending on the driver. However, interoperable applications set them before connecting because some drivers do not support changing these after connecting.

[2] SQL_ATTR_ASYNC_ENABLE must be set before there is an active statement.

[3] SQL_ATTR_TXN_ISOLATION can be set only if there are no open transactions on the connection. Some connection attributes support substitution of a similar value if the data source does not support the value specified in **ValuePtr*. In such cases, the driver returns SQL_SUCCESS_WITH_INFO and SQLSTATE 01S02 (Option value changed). For example, if *Attribute* is SQL_ATTR_PACKET_SIZE and **ValuePtr* exceeds the maximum packet size, the driver substitutes the maximum size. To determine the substituted value, an application calls **SQLGetConnectAttr**.

The format of information set in the **ValuePtr* buffer depends on the specified *Attribute*.

SQLSetConnectAttr will accept attribute 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 attribute's description. Character strings pointed to by the *ValuePtr* argument of **SQLSetConnectAttr** have a length of *StringLength* bytes.

The *StringLength* argument is ignored if the length is defined by the attribute, as is the case for all attributes introduced in ODBC 2.x or earlier.

Attribute	<i>ValuePtr</i> contents
SQL_ATTR_ACCESS_MODE (ODBC 1.0)	An SQLINTEGER 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. <code>SQL_MODE_READ_WRITE</code> is the default.
<code>SQL_ATTR_ASYNC_ENABLE</code> (ODBC 3.0)	<p>An SQLINTEGER value that specifies whether a function called with a statement on the specified connection is executed asynchronously:</p> <p><code>SQL_ASYNC_ENABLE_OFF</code> = Off (the default) <code>SQL_ASYNC_ENABLE_ON</code> = On</p> <p>Setting <code>SQL_ASYNC_ENABLE_ON</code> enables asynchronous execution for all future statement handles allocated on this connection. It is driver-defined whether this enables asynchronous execution for existing statement handles associated with this connection. An error is returned if asynchronous execution is enabled while there is an active statement on the connection.</p> <p>This attribute can be set whether SQLGetInfo with the <code>SQL_ASYNC_MODE</code> information type returns <code>SQL_AM_CONNECTION</code> or <code>SQL_AM_STATEMENT</code>.</p> <p>After a function has been called asynchronously, only the original function, SQLAllocHandle, SQLCancel, SQLGetDiagField, or SQLGetDiagRec can be called on the statement or the connection associated with <i>StatementHandle</i>, until the original function returns a code other than <code>SQL_STILL_EXECUTING</code>. Any other function called on <i>StatementHandle</i> or the connection associated with <i>StatementHandle</i> returns <code>SQL_ERROR</code> with an <code>SQLSTATE</code> of HY010 (Function sequence error). Functions can be called on other statements. For more information, see the Part I PDF file, "Asynchronous Execution" in Chapter 9, "Executing Statements."</p> <p>In general, applications should execute functions asynchronously only on single-thread operating systems. On multithread operating systems, applications should execute functions on separate threads rather than executing them asynchronously on the same thread. Drivers that operate only on multithread operating systems do not need to support asynchronous execution.</p> <p>The following functions can be executed asynchronously:</p> <p><code>SQLBulkOperations</code> <code>SQLColAttribute</code> <code>SQLColumnPrivileges</code> <code>SQLColumns</code> <code>SQLCopyDesc</code> <code>SQLDescribeCol</code> <code>SQLDescribeParam</code> <code>SQLExecDirect</code> <code>SQLExecute</code> <code>SQLFetch</code> <code>SQLFetchScroll</code> <code>SQLForeignKeys</code> <code>SQLGetData</code></p>

	SQLGetDescField [1] SQLGetDescRec [1] SQLGetDiagField SQLGetDiagRec SQLGetTypeInfo SQLMoreResults SQLNumParams SQLNumResultCols SQLParamData SQLPrepare SQLPrimaryKeys SQLProcedureColumns SQLProcedures SQLPutData SQLSetPos SQLSpecialColumns SQLStatistics SQLTablePrivileges SQLTables
SQL_ATTR_AUTO_IPD (ODBC 3.0)	<p>A read-only SQLINTEGER value that specifies whether automatic population of the IPD after a call to SQLPrepare is supported:</p> <p>SQL_TRUE = Automatic population of the IPD after a call to SQLPrepare is supported by the driver.</p> <p>SQL_FALSE = Automatic population of the IPD after a call to SQLPrepare is not supported by the driver. Servers that do not support prepared statements will not be able to populate the IPD automatically.</p> <p>If SQL_TRUE is returned for the SQL_ATTR_AUTO_IPD connection attribute, the statement attribute SQL_ATTR_ENABLE_AUTO_IPD can be set to turn automatic population of the IPD on or off. If SQL_ATTR_AUTO_IPD is SQL_FALSE, SQL_ATTR_ENABLE_AUTO_IPD cannot be set to SQL_TRUE. The default value of SQL_ATTR_ENABLE_AUTO_IPD is equal to the value of SQL_ATTR_AUTO_IPD.</p> <p>This connection attribute can be returned by SQLGetConnectAttr but cannot be set by SQLSetConnectAttr.</p>
SQL_ATTR_AUTOCOMMIT (ODBC 1.0)	<p>An SQLINTEGER value that specifies whether to use autocommit or manual-commit mode:</p> <p>SQL_AUTOCOMMIT_OFF = The driver uses manual-commit mode, and the application must explicitly commit or roll back transactions with SQLEndTran.</p> <p>SQL_AUTOCOMMIT_ON = The driver uses autocommit mode. Each statement is committed immediately after it is executed. This is the default. Any open transactions on the connection are</p>

	<p>committed when SQL_ATTR_AUTOCOMMIT is set to SQL_AUTOCOMMIT_ON to change from manual-commit mode to autocommit mode.</p> <p>For more information, see the Part I PDF file, “Commit Mode” in Chapter 14, "Transactions."</p> <p>Important Some data sources delete the access plans and close the cursors for all statements on a connection each time a statement is committed; autocommit mode can cause this to happen after each nonquery statement is executed or when the cursor is closed for a query. For more information, see the SQL_CURSOR_COMMIT_BEHAVIOR and SQL_CURSOR_ROLLBACK_BEHAVIOR information types in SQLGetInfo and the Part I PDF file, “Effect of Transactions of Cursors and Prepared Statements” in Chapter 14, "Transactions."</p> <p>When a batch is executed in autocommit mode, two things are possible. The entire batch can be treated as an autocommittable unit, or each statement in a batch is treated as an autocommittable unit. Certain data sources can support both these behaviors and may provide a way of choosing one or the other. It is driver-defined whether a batch is treated as an autocommittable unit or whether each individual statement within the batch is autocommittable.</p>
SQL_ATTR_CONNECTION_DEAD (ODBC 3.5)	An SQLINTEGER value that indicates the state of the connection. If SQL_CD_TRUE, the connection has been lost. If SQL_CD_FALSE, the connection is still active.
SQL_ATTR_CONNECTION_TIMEOUT (ODBC 3.0)	<p>An SQLINTEGER value corresponding to the number of seconds to wait for any request on the connection to complete before returning to the application. The driver should return SQLSTATE HYT00 (Timeout expired) anytime that it is possible to time out in a situation not associated with query execution or login.</p> <p>If <i>ValuePtr</i> is equal to 0 (the default), there is no timeout.</p>
SQL_ATTR_CURRENT_CATALOG (ODBC 2.0)	A character string containing the name of the catalog to be used by the data source. For example, in SQL Server, the catalog is a database, so the driver sends a USE database statement to the data source, where <i>database</i> is the database specified in <i>*ValuePtr</i> . For a single-tier driver, the catalog might be a directory, so the driver changes its current directory to the directory specified in <i>*ValuePtr</i> .
SQL_ATTR_LOGIN_TIMEOUT (ODBC 1.0)	<p>An SQLINTEGER 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. If <i>ValuePtr</i> is 0, the timeout is disabled and a connection attempt will wait indefinitely.</p> <p>If the specified timeout exceeds the maximum login timeout in the data source, the driver substitutes that value and returns</p>

	SQLSTATE 01S02 (Option value changed).
SQL_ATTR_METADATA_ID (ODBC 3.0)	<p>An SQLINTEGER value that determines how the string arguments of catalog functions are treated.</p> <p>If SQL_TRUE, the string argument of catalog functions are treated as identifiers. The case is not significant. For nondelimited strings, the driver removes any trailing spaces and the string is folded to uppercase. For delimited strings, the driver removes any leading or trailing spaces and takes literally whatever is between the delimiters. If one of these arguments is set to a null pointer, the function returns SQL_ERROR and SQLSTATE HY009 (Invalid use of null pointer).</p> <p>If SQL_FALSE, the string arguments of catalog functions are not treated as identifiers. The case is significant. They can either contain a string search pattern or not, depending on the argument.</p> <p>The default value is SQL_FALSE.</p> <p>The <i>TableType</i> argument of SQLTables, which takes a list of values, is not affected by this attribute.</p> <p>SQL_ATTR_METADATA_ID can also be set on the statement level. (It is the only connection attribute that is also a statement attribute.)</p> <p>For more information, see the Part I PDF file, “Arguments in Catalog Function,” in Chapter 7, "Catalog Functions."</p>
SQL_ATTR_ODBC_CURSORS (ODBC 2.0)	<p>An SQLINTEGER value specifying how the Driver Manager uses the ODBC cursor library:</p> <p>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 SQLFetchScroll, the Driver Manager uses the scrolling capabilities of the driver. Otherwise, it uses the ODBC cursor library.</p> <p>SQL_CUR_USE_ODBC = The Driver Manager uses the ODBC cursor library.</p> <p>SQL_CUR_USE_DRIVER = The Driver Manager uses the scrolling capabilities of the driver. This is the default setting.</p> <p>For more information about the ODBC cursor library, see Appendix F, “ODBC Cursor Library” contained on the Microsoft Web site (ODBC Programming Reference).</p>
SQL_ATTR_PACKET_SIZE (ODBC 2.0)	<p>An SQLINTEGER value specifying the network packet size in bytes.</p> <p>Note Many data sources either do not support this option or only can return but not set the network packet size.</p>

	<p>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).</p> <p>If the application sets packet size after a connection has already been made, the driver will return SQLSTATE HY011 (Attribute cannot be set now).</p>
SQL_ATTR_QUIET_MODE (ODBC 2.0)	<p>A 32-bit window handle (<i>hwnd</i>).</p> <p>If the window handle is a null pointer, the driver does not display any dialog boxes.</p> <p>If the window handle is not a null pointer, it should be the parent window handle of the application. This is the default. The driver uses this handle to display dialog boxes.</p> <p>Note The SQL_ATTR_QUIET_MODE connection attribute does not apply to dialog boxes displayed by SQLDriverConnect.</p>
SQL_ATTR_TRACE (ODBC 1.0)	<p>An SQLINTEGER value telling the Driver Manager whether to perform tracing:</p> <p>SQL_OPT_TRACE_OFF = Tracing off (the default)</p> <p>SQL_OPT_TRACE_ON = Tracing on</p> <p>When tracing is on, the Driver Manager writes each ODBC function call to the trace file.</p> <p>Note When tracing is on, the Driver Manager can return SQLSTATE IM013 (Trace file error) from any function.</p> <p>An application specifies a trace file with the SQL_ATTR_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 in the root directory.</p> <p>An application can set the variable ODBCSharedTraceFlag to enable tracing dynamically. Tracing is then enabled for all ODBC applications currently running. If an application turns tracing off, it is turned off only for that application.</p> <p>If the Trace keyword in the system information is set to 1 when an application calls SQLAllocHandle with a <i>HandleType</i> of SQL_HANDLE_ENV, tracing is enabled for all handles. It is enabled only for the application that called SQLAllocHandle.</p> <p>Calling SQLSetConnectAttr with an <i>Attribute</i> of SQL_ATTR_TRACE does not require that the <i>ConnectionHandle</i> argument be valid and will not return SQL_ERROR if <i>ConnectionHandle</i> is NULL. This attribute applies to all connections.</p>

SQL_ATTR_TRACEFILE (ODBC 1.0)	<p>A null-terminated character string containing the name of the trace file.</p> <p>The default value of the SQL_ATTR_TRACEFILE attribute is specified with the TraceFile keyword in the system information. For more information, see the Part III PDF file, “ODBC Subkey” in Chapter 19, “Configuring Data Sources.”</p> <p>Calling SQLSetConnectAttr with an <i>Attribute</i> of SQL_ATTR_TRACEFILE does not require the <i>ConnectionHandle</i> argument to be valid and will not return SQL_ERROR if <i>ConnectionHandle</i> is invalid. This attribute applies to all connections.</p>
SQL_ATTR_TRANSLATE_LIB (ODBC 1.0)	<p>A null-terminated character string containing the name of a library containing the functions SQLDriverToDataSource and SQLDataSourceToDriver that the driver accesses to perform tasks such as character set translation. This option may be specified only if the driver has connected to the data source. The setting of this attribute will persist across connections. For more information about translating data, see the Part I PDF file, “Translation DLLs” in Chapter 17, “Programming Considerations,” and the Part III PDF file, Chapter 24, “Translation DLL Function Reference.”</p>
SQL_ATTR_TRANSLATE_OPTION (ODBC 1.0)	<p>A 32-bit flag value that is passed to the translation DLL. This attribute can be specified only if the driver has connected to the data source. For information about translating data, see the Part I PDF file, “Translation DLLs” in Chapter 17, “Programming Considerations.”</p>
SQL_ATTR_TXN_ISOLATION (ODBC 1.0)	<p>A 32-bit bitmask that sets the transaction isolation level for the current connection. An application must call SQLEndTran to commit or roll back all open transactions on a connection, before calling SQLSetConnectAttr with this option.</p> <p>The valid values for <i>ValuePtr</i> can be determined by calling SQLGetInfo with <i>InfoType</i> equal to SQL_TXN_ISOLATION_OPTIONS.</p> <p>For a description of transaction isolation levels, see the description of the SQL_DEFAULT_TXN_ISOLATION information type in SQLGetInfo and the Part I PDF file, “Transaction Isolation Levels” in Chapter 14, “Transactions.”</p>

[1] These functions can be called asynchronously only if the descriptor is an implementation descriptor, not an application descriptor.

Code Example

See SQLConnect.

Related Functions

For information about	See
Allocating a handle	SQLAllocHandle
Returning the setting of a connection attribute	SQLSetConnectAttr

SQLSetConnectOption

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: Deprecated

Summary

In ODBC 3.x, the ODBC 2.0 function **SQLSetConnectOption** has been replaced by **SQLSetConnectAttr**. For more information, see **SQLSetConnectAttr**.

Note For more information about what the Driver Manager maps this function to when an ODBC 2.x application is working with an ODBC 3.x driver, see “Mapping Deprecated Functions” (Appendix G, “Driver Guidelines for Backward Compatibility”) contained on the Microsoft Web site (ODBC Programmer’s Reference).

SQLSetCursorName

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

SQLSetCursorName associates a cursor name with an active statement. If an application does not call **SQLSetCursorName**, the driver generates cursor names as needed for SQL statement processing.

Syntax

SQLRETURN SQLSetCursorName(
SQLHSTMT	StatementHandle,
SQLCHAR *	CursorName,
SQLSMALLINT	NameLength);

Arguments

StatementHandle

[Input]

Statement handle.

CursorName

[Input]

Cursor name. For efficient processing, the cursor name should not include any leading or trailing spaces in the cursor name, and if the cursor name includes a delimited identifier, the delimiter should be positioned as the first character in the cursor name.

NameLength

[Input]

Length of **CursorName*.

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 can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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.)
01004	String data, right truncated	The cursor name exceeded the maximum limit, so only the maximum allowable number of characters was used.
24000	Invalid cursor state	The statement corresponding to <i>StatementHandle</i> was already in an executed or cursor-positioned state.
34000	Invalid cursor name	The cursor name specified in <i>*CursorName</i> was invalid because it exceeded the maximum length as defined by the driver, or it started with "SQLCUR" or "SQL_CUR."
3C000	Duplicate cursor name	The cursor name specified in <i>*CursorName</i> already exists.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY009	Invalid use of null pointer	(DM) The argument <i>CursorName</i> was a null pointer.
HY010	Function sequence error	(DM) An asynchronously executing function was called for the <i>StatementHandle</i> and was still executing when this function was called.

		(DM) SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA . This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	(DM) The argument <i>NameLength</i> was less than 0 but not equal to SQL_NTS .
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

Cursor names are used only in positioned update and delete statements (for example, **UPDATE** *table-name* ...**WHERE CURRENT OF** *cursor-name*). For more information, see the Part 1 PDF file, "Positioned Update and Delete Statements" in Chapter 12, "Updating Data." If the application does not call **SQLSetCursorName** to define a cursor name, on execution of a query 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 connection 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. In ODBC 3.x, if a cursor name is a quoted identifier, it is treated in a case-sensitive manner and it can contain characters that the syntax of SQL would not permit or would treat specially, such as blanks or reserved keywords. If a cursor name must be treated in a case-sensitive manner, it must be passed as a quoted identifier.

A cursor name that is set either explicitly or implicitly remains set until the statement with which it is associated is dropped, using **SQLFreeHandle**. **SQLSetCursorName** can be called to rename a cursor on a statement as long as the cursor is in an allocated or prepared state.

Code Example

In the following example, an application uses **SQLSetCursorName** to set a cursor name for a statement. It then uses that statement to retrieve results from the **CUSTOMERS** table. Finally, it performs a positioned update to change the phone number of John Smith. Note that the application uses different statement handles for the **SELECT** and **UPDATE** statements.

For another code example, see **SQLSetPos**.

```
#define NAME_LEN 50
#define PHONE_LEN 10

SQLHSTMT hstmtSelect,
SQLHSTMT hstmtUpdate;
```

```

SQLRETURN    retcode;
SQLHDBC hdbc;
SQLCHAR szName[NAME_LEN], szPhone[PHONE_LEN];
SQLINTEGER  cbName, cbPhone;

/* Allocate the statements and set the cursor name. */

SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmtSelect);
SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmtUpdate);
SQLSetCursorName(hstmtSelect, "C1", SQL_NTS);

/* SELECT the result set and bind its columns to local buffers. */

SQLExecDirect(hstmtSelect,
    "SELECT NAME, PHONE FROM CUSTOMERS",
    SQL_NTS);
SQLBindCol(hstmtSelect, 1, SQL_C_CHAR, szName, NAME_LEN, &cbName);
SQLBindCol(hstmtSelect, 2, SQL_C_CHAR, szPhone, PHONE_LEN, &cbPhone);

/* Read through the result set until the cursor is */
/* positioned on the row for John Smith. */

do
    retcode = SQLFetch(hstmtSelect);
while ((retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) &&
    (strcmp(szName, "Smith, John") != 0));

/* Perform a positioned update of John Smith's name. */

if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
    SQLExecDirect(hstmtUpdate,
        "UPDATE EMPLOYEE SET PHONE=\"2064890154\" WHERE CURRENT OF C1",
        SQL_NTS);
}

```

Related Functions

For information about	See
Executing an SQL statement	SQLExecDirect
Executing a prepared SQL statement	SQLExecute
Returning a cursor name	SQLGetCursorName
Setting cursor scrolling options	SQLSetScrollOptions

SQLSetDescField

Conformance

Version Introduced: ODBC 3.0

Standards Compliance: ISO 92

Summary

SQLSetDescField sets the value of a single field of a descriptor record.

Syntax

SQLRETURN SQLSetDescField(
SQLHDESC	DescriptorHandle,
SQLSMALLINT	RecNumber,
SQLSMALLINT	FieldIdentifier,
SQLPOINTER	ValuePtr,
SQLINTEGER	BufferLength);

Arguments

DescriptorHandle

[Input]

Descriptor handle.

RecNumber

[Input]

Indicates the descriptor record containing the field that the application seeks to set. Descriptor records are numbered from 0, with record number 0 being the bookmark record. The *RecNumber* argument is ignored for header fields.

FieldIdentifier

[Input]

Indicates the field of the descriptor whose value is to be set. For more information, see "*FieldIdentifier* Argument" in the "Comments" section.

ValuePtr

[Input]

Pointer to a buffer containing the descriptor information, or a 4-byte value. The data type depends on the value of *FieldIdentifier*. If *ValuePtr* is a 4-byte value, either all four of the bytes are used or just two of the four are used, depending on the value of the *FieldIdentifier* argument.

BufferLength

[Input]

If *FieldIdentifier* is an ODBC-defined field and *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **ValuePtr*. If *FieldIdentifier* is an ODBC-defined field and *ValuePtr* is an integer, *BufferLength* is ignored.

If *FieldIdentifier* is a driver-defined field, the application indicates the nature of the field to the Driver Manager by setting the *BufferLength* argument. *BufferLength* can have the following values:

If *ValuePtr* is a pointer to a character string, then *BufferLength* is the length of the string or SQL_NTS.

If *ValuePtr* is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(*length*) macro in *BufferLength*. This places a negative value in *BufferLength*.

If *ValuePtr* is a pointer to a value other than a character string or a binary string, then *BufferLength* should have the value SQL_IS_POINTER.

If *ValuePtr* contains a fixed-length value, then *BufferLength* is either SQL_IS_INTEGER, SQL_IS_UINTEGER, SQL_IS_SMALLINT, or SQL_IS_USMALLINT, as appropriate.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLSetDescField** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_DESC and a *Handle* of *DescriptorHandle*. The following table lists the SQLSTATE values commonly returned by **SQLSetDescField** 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 value specified in * <i>ValuePtr</i> (if <i>ValuePtr</i> was a pointer) or the value in <i>ValuePtr</i> (if <i>ValuePtr</i> was a 4-byte value), or * <i>ValuePtr</i> was invalid because of implementation working conditions, so the driver substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
07009	Invalid descriptor index	<p>The <i>FieldIdentifier</i> argument was a record field, the <i>RecNumber</i> argument was 0, and the <i>DescriptorHandle</i> argument referred to an IPD handle.</p> <p>The <i>RecNumber</i> argument was less than 0, and the <i>DescriptorHandle</i> argument referred to an ARD or an APD.</p> <p>The <i>RecNumber</i> argument was greater than the maximum number of columns or parameters that the data source can support, and the <i>DescriptorHandle</i> argument referred to an APD or ARD.</p> <p>(DM) The <i>FieldIdentifier</i> argument was SQL_DESC_COUNT, and *<i>ValuePtr</i> argument was less than 0.</p> <p>The <i>RecNumber</i> argument was equal to 0, and the <i>DescriptorHandle</i> argument referred to an implicitly allocated APD. (This error does not occur with an explicitly allocated application descriptor, because it is not known whether an explicitly allocated application descriptor is an APD or ARD until execute time.)</p>
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	The <i>FieldIdentifier</i> argument was SQL_DESC_NAME,

	truncated	and the <i>BufferLength</i> argument was a value larger than SQL_MAX_IDENTIFIER_LEN.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error	(DM) The <i>DescriptorHandle</i> was associated with a <i>StatementHandle</i> for which an asynchronously executing function (not this one) was called and was still executing when this function was called. (DM) SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos was called for the <i>StatementHandle</i> with which the <i>DescriptorHandle</i> was associated and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY016	Cannot modify an implementation row descriptor	The <i>DescriptorHandle</i> argument was associated with an IRD, and the <i>FieldIdentifier</i> argument was not SQL_DESC_ARRAY_STATUS_PTR or SQL_DESC_ROWS_PROCESSED_PTR.
HY021	Inconsistent descriptor information	The SQL_DESC_TYPE and SQL_DESC_DATETIME_INTERVAL_CODE fields do not form a valid ODBC SQL type or a valid driver-specific SQL type (for IPDs) or a valid ODBC C type (for APDs or ARDs). Descriptor information checked during a consistency check was not consistent. (See "Consistency Check" in SQLSetDescRec .)
HY090	Invalid string or buffer length	(DM) <i>*ValuePtr</i> is a character string, and <i>BufferLength</i> was less than zero but was not equal to SQL_NTS. (DM) The driver was an ODBC 2.x driver, the descriptor was an ARD, the <i>ColumnNumber</i> argument was set to 0, and the value specified for the argument <i>BufferLength</i> was not equal to 4.
HY091	Invalid descriptor field identifier	The value specified for the <i>FieldIdentifier</i> argument was not an ODBC-defined field and was not an implementation-defined value. The <i>FieldIdentifier</i> argument was invalid for the <i>DescriptorHandle</i> argument.

		The <i>FieldIdentifier</i> argument was a read-only, ODBC-defined field.
HY092	Invalid attribute/option identifier	The value in * <i>ValuePtr</i> was not valid for the <i>FieldIdentifier</i> argument. The <i>FieldIdentifier</i> argument was SQL_DESC_UNNAMED, and <i>ValuePtr</i> was SQL_NAMED.
HY105	Invalid parameter type	(DM) The value specified for the SQL_DESC_PARAMETER_TYPE field was invalid. (For more information, see the " <i>InputOutputType Argument</i> " section in SQLBindParameter .)
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>DescriptorHandle</i> does not support the function.

Comments

An application can call **SQLSetDescField** to set any descriptor field one at a time. One call to **SQLSetDescField** sets a single field in a single descriptor. This function can be called to set any field in any descriptor type, provided the field can be set. (See the table later in this section.)

Note If a call to **SQLSetDescField** fails, the contents of the descriptor record identified by the *RecNumber* argument are undefined.

Other functions can be called to set multiple descriptor fields with a single call of the function. The **SQLSetDescRec** function sets a variety of fields that affect the data type and buffer bound to a column or parameter (the SQL_DESC_TYPE, SQL_DESC_DATETIME_INTERVAL_CODE, SQL_DESC_OCTET_LENGTH, SQL_DESC_PRECISION, SQL_DESC_SCALE, SQL_DESC_DATA_PTR, SQL_DESC_OCTET_LENGTH_PTR, and SQL_DESC_INDICATOR_PTR fields). **SQLBindCol** or **SQLBindParameter** can be used to make a complete specification for the binding of a column or parameter. These functions set a specific group of descriptor fields with one function call.

SQLSetDescField can be called to change the binding buffers by adding an offset to the binding pointers (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, or SQL_DESC_OCTET_LENGTH_PTR). This changes the binding buffers without calling **SQLBindCol** or **SQLBindParameter**, which allows an application to change SQL_DESC_DATA_PTR without changing other fields, such as SQL_DESC_DATA_TYPE.

If an application calls **SQLSetDescField** to set any field other than SQL_DESC_COUNT or the deferred fields SQL_DESC_DATA_PTR, SQL_DESC_OCTET_LENGTH_PTR, or SQL_DESC_INDICATOR_PTR, the record becomes unbound.

Descriptor header fields are set by calling **SQLSetDescField** with the appropriate *FieldIdentifier*. Many header fields are also statement attributes, so they can also be set by a call to **SQLSetStmtAttr**. This allows applications to set a descriptor field without first obtaining a descriptor handle. When **SQLSetDescField** is called to set a header field, the *RecNumber* argument is ignored.

A *RecNumber* of 0 is used to set bookmark fields.

Note The statement attribute `SQL_ATTR_USE_BOOKMARKS` should always be set before calling **SQLSetDescField** to set bookmark fields. While this is not mandatory, it is strongly recommended.

Sequence of Setting Descriptor Fields

When setting descriptor fields by calling **SQLSetDescField**, the application must follow a specific sequence:

The application must first set the `SQL_DESC_TYPE`, `SQL_DESC_CONCISE_TYPE`, or `SQL_DESC_DATETIME_INTERVAL_CODE` field.

After one of these fields has been set, the application can set an attribute of a data type, and the driver sets data type attribute fields to the appropriate default values for the data type. Automatic defaulting of type attribute fields ensures that the descriptor is always ready to use once the application has specified a data type. If the application explicitly sets a data type attribute, it is overriding the default attribute.

After one of the fields listed in step 1 has been set, and data type attributes have been set, the application can set `SQL_DESC_DATA_PTR`. This prompts a consistency check of descriptor fields. If the application changes the data type or attributes after setting the `SQL_DESC_DATA_PTR` field, the driver sets `SQL_DESC_DATA_PTR` to a null pointer, unbinding the record. This forces the application to complete the proper steps in sequence, before the descriptor record is usable.

Initialization of Descriptor Fields

When a descriptor is allocated, the fields in the descriptor can be initialized to a default value, be initialized without a default value, or be undefined for the type of descriptor. The following tables indicate the initialization of each field for each type of descriptor, with "D" indicating that the field is initialized with a default, and "ND" indicating that the field is initialized without a default. If a number is shown, the default value of the field is that number. The tables also indicate whether a field is read/write (R/W) or read-only (R).

The fields of an IRD have a default value only after the statement has been prepared or executed and the IRD has been populated, not when the statement handle or descriptor has been allocated. Until the IRD has been populated, any attempt to gain access to a field of an IRD will return an error.

Some descriptor fields are defined for one or more, but not all, of the descriptor types (ARDs and IRDs, and APDs and IPDs). When a field is undefined for a type of descriptor, it is not needed by any of the functions that use that descriptor.

The fields that can be accessed by **SQLGetDescField** cannot necessarily be set by **SQLSetDescField**. Fields that can be set by **SQLSetDescField** are listed in the following tables.

The initialization of header fields is outlined in the table that follows.

Header field name	Type	R/W	Default
<code>SQL_DESC_ALLOC_TYPE</code>	<code>SQLSMALLINT</code>	ARD: R APD: R IRD: R IPD: R	ARD: <code>SQL_DESC_ALLOC_AUTO</code> for implicit or <code>SQL_DESC_ALLOC_USER</code> for explicit APD: <code>SQL_DESC_ALLOC_AUTO</code> for implicit or

			SQL_DESC_ALLOC_USER for explicit IRD: SQL_DESC_ALLOC_AUTO IPD: SQL_DESC_ALLOC_AUTO
SQL_DESC_ARRAY_SIZE	SQLINTEGER	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD: [1] APD: [1] IRD: Unused IPD: Unused
SQL_DESC_ARRAY_STATUS_PTR	SQLUSMALLINT*	ARD: R/W APD: R/W IRD: R/W IPD: R/W	ARD: Null ptr APD: Null ptr IRD: Null ptr IPD: Null ptr
SQL_DESC_BIND_OFFSET_PTR	SQLINTEGER*	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD: Null ptr APD: Null ptr IRD: Unused IPD: Unused
SQL_DESC_BIND_TYPE	SQLINTEGER	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD: SQL_BIND_BY_COLUMN APD: SQL_BIND_BY_COLUMN IRD: Unused IPD: Unused
SQL_DESC_COUNT	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: 0 APD: 0 IRD: D IPD: 0
SQL_DESC_ROWS_PROCESSED_PTR	SQLINTEGER*	ARD: Unused APD: Unused IRD: R/W IPD: R/W	ARD: Unused APD: Unused IRD: Null ptr IPD: Null ptr

[1] These fields are defined only when the IPD is automatically populated by the driver. If not, they are undefined. If an application attempts to set these fields, SQLSTATE HY091 (Invalid descriptor field identifier) will be returned.

The initialization of record fields is as shown in the following table.

Record field name	Type	R/W	Default
SQL_DESC_AUTO_UNIQUE_VALUE	SQLINTEGER	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused

SQL_DESC_BASE_COLUMN_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_BASE_TABLE_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_CASE_SENSITIVE	SQLINTEGER	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: D IPD: D [1]
SQL_DESC_CATALOG_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_CONCISE_TYPE	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: SQL_C_ DEFAULT APD: SQL_C_ DEFAULT IRD: D IPD: ND
SQL_DESC_DATA_PTR	SQLPOINTER	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD: Null ptr APD: Null ptr IRD: Unused IPD: Unused [2]
SQL_DESC_DATETIME_INTERVAL_CODE	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_DATETIME_INTERVAL_PRECISION	SQLINTEGER	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_DISPLAY_SIZE	SQLINTEGER	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_FIXED_PREC_SCALE	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: D IPD: D [1]
SQL_DESC_INDICATOR_PTR	SQLINTEGER *	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD: Null ptr APD: Null ptr IRD: Unused IPD: Unused
SQL_DESC_LABEL	SQLCHAR *	ARD: Unused APD: Unused IRD: R	ARD: Unused APD: Unused IRD: D

		IPD: Unused	IPD: Unused
SQL_DESC_LENGTH	SQLINTEGER	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_LITERAL_PREFIX	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_LITERAL_SUFFIX	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_LOCAL_TYPE_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: D IPD: D [1]
SQL_DESC_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_NULLABLE	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: R	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_NUM_PREC_RADIX	SQLINTEGER	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_OCTET_LENGTH	SQLINTEGER	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_OCTET_LENGTH_PTR	SQLINTEGER *	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD: Null ptr APD: Null ptr IRD: Unused IPD: Unused
SQL_DESC_PARAMETER_TYPE	SQLSMALLINT	ARD: Unused APD: Unused IRD: Unused IPD: R/W	ARD: Unused APD: Unused IRD: Unused IPD: D=SQL_PARAM_INPUT
SQL_DESC_PRECISION	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_ROWVER	SQLSMALLINT	ARD: Unused	ARD: Unused

		APD: Unused IRD: R IPD: R	APD: Unused IRD: ND IPD: ND
SQL_DESC_SCALE	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_SCHEMA_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_SEARCHABLE	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_TABLE_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_TYPE	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: SQL_C_DEFAULT APD: SQL_C_DEFAULT IRD: D IPD: ND
SQL_DESC_TYPE_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: D IPD: D [1]
SQL_DESC_UNNAMED	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_UNSIGNED	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: D IPD: D [1]
SQL_DESC_UPDATABLE	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused

[1] These fields are defined only when the IPD is automatically populated by the driver. If not, they are undefined. If an application attempts to set these fields, SQLSTATE HY091 (Invalid descriptor field identifier) will be returned.

[2] The `SQL_DESC_DATA_PTR` field in the IPD can be set to force a consistency check. In a subsequent call to **SQLGetDescField** or **SQLGetDescRec**, the driver is not required to return the value that `SQL_DESC_DATA_PTR` was set to.

FieldIdentifier Argument

The *FieldIdentifier* argument indicates the descriptor field to be set. A descriptor contains the *descriptor header*, consisting of the header fields described in the next section, "Header Fields," and zero or more *descriptor records*, consisting of the record fields described in the section following the "Header Fields" section.

Header Fields

Each descriptor has a header consisting of the following fields:

SQL_DESC_ALLOC_TYPE [All]

This read-only `SQLSMALLINT` header field specifies whether the descriptor was allocated automatically by the driver or explicitly by the application. The application can obtain, but not modify, this field. The field is set to `SQL_DESC_ALLOC_AUTO` by the driver if the descriptor was automatically allocated by the driver. It is set to `SQL_DESC_ALLOC_USER` by the driver if the descriptor was explicitly allocated by the application.

SQL_DESC_ARRAY_SIZE [Application descriptors]

In ARDs, this `SQLINTEGER` header field specifies the number of rows in the rowset. This is the number of rows to be returned by a call to **SQLFetch** or **SQLFetchScroll** or to be operated on by a call to **SQLBulkOperations** or **SQLSetPos**.

In APDs, this `SQLINTEGER` header field specifies the number of values for each parameter.

The default value of this field is 1. If `SQL_DESC_ARRAY_SIZE` is greater than 1, `SQL_DESC_DATA_PTR`, `SQL_DESC_INDICATOR_PTR`, and `SQL_DESC_OCTET_LENGTH_PTR` of the APD or ARD point to arrays. The cardinality of each array is equal to the value of this field.

This field in the ARD can also be set by calling **SQLSetStmtAttr** with the `SQL_ATTR_ROW_ARRAY_SIZE` attribute. This field in the APD can also be set by calling **SQLSetStmtAttr** with the `SQL_ATTR_PARAMSET_SIZE` attribute.

SQL_DESC_ARRAY_STATUS_PTR [All]

For each descriptor type, this `SQLUSMALLINT *` header field points to an array of `SQLUSMALLINT` values. These arrays are named as follows: row status array (IRD), parameter status array (IPD), row operation array (ARD), and parameter operation array (APD).

In the IRD, this header field points to a row status array containing status values after a call to **SQLBulkOperations**, **SQLFetch**, **SQLFetchScroll**, or **SQLSetPos**. The array has as many elements as there are rows in the rowset. The application must allocate an array of `SQLUSMALLINT`s and set this field to point to the array. The field is set to a null pointer by default. The driver will populate the array—unless the `SQL_DESC_ARRAY_STATUS_PTR` field is set to a null pointer, in which case no status values are generated and the array is not populated.

Caution Driver behavior is undefined if the application sets the elements of the row status array pointed to by the `SQL_DESC_ARRAY_STATUS_PTR` field of the IRD.

The array is initially populated by a call to **SQLBulkOperations**, **SQLFetch**, **SQLFetchScroll**, or **SQLSetPos**. If the call did not return **SQL_SUCCESS** or **SQL_SUCCESS_WITH_INFO**, the contents of the array pointed to by this field are undefined. The elements in the array can contain the following values:

SQL_ROW_SUCCESS: The row was successfully fetched and has not changed since it was last fetched.

SQL_ROW_SUCCESS_WITH_INFO: The row was successfully fetched and has not changed since it was last fetched. However, a warning was returned about the row.

SQL_ROW_ERROR: An error occurred while fetching the row.

SQL_ROW_UPDATED: The row was successfully fetched and has been updated since it was last fetched. If the row is fetched again, its status is **SQL_ROW_SUCCESS**.

SQL_ROW_DELETED: The row has been deleted since it was last fetched.

SQL_ROW_ADDED: The row was inserted by **SQLBulkOperations**. If the row is fetched again, its status is **SQL_ROW_SUCCESS**.

SQL_ROW_NOROW: The rowset overlapped the end of the result set, and no row was returned that corresponded to this element of the row status array.

- This field in the IRD can also be set by calling **SQLSetStmtAttr** with the **SQL_ATTR_ROW_STATUS_PTR** attribute.
- The **SQL_DESC_ARRAY_STATUS_PTR** field of the IRD is valid only after **SQL_SUCCESS** or **SQL_SUCCESS_WITH_INFO** has been returned. If the return code is not one of these, the location pointed to by **SQL_DESC_ROWS_PROCESSED_PTR** is undefined.
- In the IPD, this header field points to a parameter status array containing status information for each set of parameter values after a call to **SQLExecute** or **SQLExecDirect**. If the call to **SQLExecute** or **SQLExecDirect** did not return **SQL_SUCCESS** or **SQL_SUCCESS_WITH_INFO**, the contents of the array pointed to by this field are undefined. The application must allocate an array of **SQLUSMALLINT**s and set this field to point to the array. The driver will populate the array—unless the **SQL_DESC_ARRAY_STATUS_PTR** field is set to a null pointer, in which case no status values are generated and the array is not populated. The elements in the array can contain the following values:

SQL_PARAM_SUCCESS: The SQL statement was successfully executed for this set of parameters.

SQL_PARAM_SUCCESS_WITH_INFO: The SQL statement was successfully executed for this set of parameters; however, warning information is available in the diagnostics data structure.

SQL_PARAM_ERROR: An error occurred in processing this set of parameters. Additional error information is available in the diagnostics data structure.

SQL_PARAM_UNUSED: This parameter set was unused, possibly due to the fact that some previous parameter set caused an error that aborted further processing, or because **SQL_PARAM_IGNORE** was set for that set of parameters in the array specified by the **SQL_DESC_ARRAY_STATUS_PTR** field of the APD.

SQL_PARAM_DIAG_UNAVAILABLE: Diagnostic information is not available. An example of this is when the driver treats arrays of parameters as a monolithic unit and so does not generate this level of error information.

This field in the IPD can also be set by calling **SQLSetStmtAttr** with the **SQL_ATTR_PARAM_STATUS_PTR** attribute.

In the ARD, this header field points to a row operation array of values that can be set by the application to indicate whether this row is to be ignored for **SQLSetPos** operations. The elements in the array can contain the following values:

SQL_ROW_PROCEED: The row is included in the bulk operation using **SQLSetPos**. (This setting does not guarantee that the operation will occur on the row. If the row has the status **SQL_ROW_ERROR** in the IRD row status array, the driver might not be able to perform the operation in the row.)

SQL_ROW_IGNORE: The row is excluded from the bulk operation using **SQLSetPos**.

If no elements of the array are set, all rows are included in the bulk operation. If the value in the **SQL_DESC_ARRAY_STATUS_PTR** field of the ARD is a null pointer, all rows are included in the bulk operation; the interpretation is the same as if the pointer pointed to a valid array and all elements of the array were **SQL_ROW_PROCEED**. If an element in the array is set to **SQL_ROW_IGNORE**, the value in the row status array for the ignored row is not changed.

This field in the ARD can also be set by calling **SQLSetStmtAttr** with the **SQL_ATTR_ROW_OPERATION_PTR** attribute.

In the APD, this header field points to a parameter operation array of values that can be set by the application to indicate whether this set of parameters is to be ignored when **SQLExecute** or **SQLExecDirect** is called. The elements in the array can contain the following values:

SQL_PARAM_PROCEED: The set of parameters is included in the **SQLExecute** or **SQLExecDirect** call.

SQL_PARAM_IGNORE: The set of parameters is excluded from the **SQLExecute** or **SQLExecDirect** call.

If no elements of the array are set, all sets of parameters in the array are used in the **SQLExecute** or **SQLExecDirect** calls. If the value in the **SQL_DESC_ARRAY_STATUS_PTR** field of the APD is a null pointer, all sets of parameters are used; the interpretation is the same as if the pointer pointed to a valid array and all elements of the array were **SQL_PARAM_PROCEED**.

This field in the APD can also be set by calling **SQLSetStmtAttr** with the **SQL_ATTR_PARAM_OPERATION_PTR** attribute.

SQL_DESC_BIND_OFFSET_PTR [Application descriptors]

This **SQLINTEGER *** header field points to the binding offset. It is set to a null pointer by default. If this field is not a null pointer, the driver dereferences the pointer and adds the dereferenced value to each of the deferred fields that has a non-null value in the descriptor record (**SQL_DESC_DATA_PTR**, **SQL_DESC_INDICATOR_PTR**, and **SQL_DESC_OCTET_LENGTH_PTR**) at fetch time and uses the new pointer values when binding.

The binding offset is always added directly to the values in the **SQL_DESC_DATA_PTR**, **SQL_DESC_INDICATOR_PTR**, and **SQL_DESC_OCTET_LENGTH_PTR** fields. If the offset is changed to a different value, the new value is still added directly to the value in each descriptor field. The new offset is not added to the field value plus any earlier offset.

This field is a *deferred field*: It is not used at the time it is set but is used at a later time by the driver when it needs to determine addresses for data buffers.

This field in the ARD can also be set by calling **SQLSetStmtAttr** with the **SQL_ATTR_ROW_BIND_OFFSET_PTR** attribute. This field in the ARD can also be set by calling **SQLSetStmtAttr** with the **SQL_ATTR_PARAM_BIND_OFFSET_PTR** attribute.

For more information, see the description of row-wise binding in [SQLFetchScroll](#) and [SQLBindParameter](#).

SQL_DESC_BIND_TYPE [Application descriptors]

This **SQLINTEGER** header field sets the binding orientation to be used for binding either columns or parameters.

In ARDs, this field specifies the binding orientation when **SQLFetchScroll** or **SQLFetch** is called on the associated statement handle.

To select column-wise binding for columns, this field is set to **SQL_BIND_BY_COLUMN** (the default).

This field in the ARD can also be set by calling **SQLSetStmtAttr** with the **SQL_ATTR_ROW_BIND_TYPE** *Attribute*.

In APDs, this field specifies the binding orientation to be used for dynamic parameters.

To select column-wise binding for parameters, this field is set to **SQL_BIND_BY_COLUMN** (the default).

This field in the APD can also be set by calling **SQLSetStmtAttr** with the **SQL_ATTR_PARAM_BIND_TYPE** *Attribute*.

SQL_DESC_COUNT [All]

This **SQLSMALLINT** header field specifies the 1-based index of the highest-numbered record that contains data. When the driver sets the data structure for the descriptor, it must also set the **SQL_DESC_COUNT** field to show how many records are significant. When an application allocates an instance of this data structure, it does not have to specify how many records to reserve room for. As the application specifies the contents of the records, the driver takes any required action to ensure that the descriptor handle refers to a data structure of the adequate size.

SQL_DESC_COUNT is not a count of all data columns that are bound (if the field is in an ARD) or of all parameters that are bound (if the field is in an APD), but the number of the highest-numbered record. If the highest-numbered column or parameter is unbound, then **SQL_DESC_COUNT** is changed to the number of the next highest-numbered column or parameter. If a column or a parameter with a number that is less than the number of the highest-numbered column is unbound (by calling **SQLBindCol** with the *TargetValuePtr* argument set to a null pointer, or **SQLBindParameter** with the *ParameterValuePtr* argument set to a null pointer), **SQL_DESC_COUNT** is not changed. If additional columns or parameters are bound with numbers greater than the highest-numbered record that contains data, the driver automatically increases the value in the **SQL_DESC_COUNT** field. If all columns are unbound by calling **SQLFreeStmt** with the **SQL_UNBIND** option, the **SQL_DESC_COUNT** fields in the ARD and IRD are set to 0. If **SQLFreeStmt** is called with the **SQL_RESET_PARAMS** option, the **SQL_DESC_COUNT** fields in the APD and IPD are set to 0.

The value in **SQL_DESC_COUNT** can be set explicitly by an application by calling **SQLSetDescField**. If the value in **SQL_DESC_COUNT** is explicitly decreased, all records with numbers greater than the new value in **SQL_DESC_COUNT** are effectively removed. If the value in **SQL_DESC_COUNT** is explicitly set to 0 and the field is in an ARD, all data buffers except a bound bookmark column are released.

The record count in this field of an ARD does not include a bound bookmark column. The only way to unbind a bookmark column is to set the **SQL_DESC_DATA_PTR** field to a null pointer.

SQL_DESC_ROWS_PROCESSED_PTR [Implementation descriptors]

In an IRD, this **SQLINTEGER *** header field points to a buffer containing the number of rows fetched after a call to **SQLFetch** or **SQLFetchScroll**, or the number of rows affected in a bulk operation performed by a call to **SQLBulkOperations** or **SQLSetPos**, including error rows.

In an IPD, this **SQLINTEGER *** header field points to a buffer containing the number of sets of parameters that have been processed, including error sets. No number will be returned if this is a null pointer.

SQL_DESC_ROWS_PROCESSED_PTR is valid only after SQL_SUCCESS or SQL_SUCCESS_WITH_INFO has been returned after a call to **SQLFetch** or **SQLFetchScroll** (for an IRD field) or **SQLExecute**, **SQLExecDirect**, or **SQLParamData** (for an IPD field). If the call that fills in the buffer pointed to by this field does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the contents of the buffer are undefined, unless it returns SQL_NO_DATA, in which case the value in the buffer is set to 0.

This field in the ARD can also be set by calling **SQLSetStmtAttr** with the SQL_ATTR_ROWS_FETCHED_PTR attribute. This field in the APD can also be set by calling **SQLSetStmtAttr** with the SQL_ATTR_PARAMS_PROCESSED_PTR attribute.

The buffer pointed to by this field is allocated by the application. It is a deferred output buffer that is set by the driver. It is set to a null pointer by default.

Record Fields

Each descriptor contains one or more records consisting of fields that define either column data or dynamic parameters, depending on the type of descriptor. Each record is a complete definition of a single column or parameter.

SQL_DESC_AUTO_UNIQUE_VALUE [IRDs]

This read-only SQLINTEGER record field contains SQL_TRUE if the column is an auto-incrementing column, or SQL_FALSE if the column is not an auto-incrementing column. This field is read-only, but the underlying auto-incrementing column is not necessarily read-only.

SQL_DESC_BASE_COLUMN_NAME [IRDs]

This read-only SQLCHAR * record field contains the base column name for the result set column. If a base column name does not exist (as in the case of columns that are expressions), this variable contains an empty string.

SQL_DESC_BASE_TABLE_NAME [IRDs]

This read-only SQLCHAR * record field contains the base table name for the result set column. If a base table name cannot be defined or is not applicable, this variable contains an empty string.

SQL_DESC_CASE_SENSITIVE [Implementation descriptors]

This read-only SQLINTEGER record field contains SQL_TRUE if the column or parameter is treated as case-sensitive for collations and comparisons, or SQL_FALSE if the column is not treated as case-sensitive for collations and comparisons or if it is a noncharacter column.

SQL_DESC_CATALOG_NAME [IRDs]

This read-only SQLCHAR * record field contains the catalog for the base table that contains the column. The return value is driver-dependent if the column is an expression or if the column is part of a view. If the data source does not support catalogs or the catalog cannot be determined, this variable contains an empty string.

SQL_DESC_CONCISE_TYPE [All]

This SQLSMALLINT header field specifies the concise data type for all data types, including the datetime and interval data types.

The values in the `SQL_DESC_CONCISE_TYPE`, `SQL_DESC_TYPE`, and `SQL_DESC_DATETIME_INTERVAL_CODE` fields are interdependent. Each time one of the fields is set, the other must also be set. `SQL_DESC_CONCISE_TYPE` can be set by a call to **SQLBindCol** or **SQLBindParameter**, or **SQLSetDescField**. `SQL_DESC_TYPE` can be set by a call to **SQLSetDescField** or **SQLSetDescRec**.

If `SQL_DESC_CONCISE_TYPE` is set to a concise data type other than an interval or datetime data type, the `SQL_DESC_TYPE` field is set to the same value and the `SQL_DESC_DATETIME_INTERVAL_CODE` field is set to 0.

If `SQL_DESC_CONCISE_TYPE` is set to the concise datetime or interval data type, the `SQL_DESC_TYPE` field is set to the corresponding verbose type (`SQL_DATETIME` or `SQL_INTERVAL`) and the `SQL_DESC_DATETIME_INTERVAL_CODE` field is set to the appropriate subcode.

SQL_DESC_DATA_PTR [Application descriptors and IPDs]

This `SQLPOINTER` record field points to a variable that will contain the parameter value (for APDs) or the column value (for ARDs). This field is a *deferred field*. It is not used at the time it is set but is used at a later time by the driver to retrieve data.

The column specified by the `SQL_DESC_DATA_PTR` field of the ARD is unbound if the *TargetValuePtr* argument in a call to **SQLBindCol** is a null pointer or if the `SQL_DESC_DATA_PTR` field in the ARD is set by a call to **SQLSetDescField** or **SQLSetDescRec** to a null pointer. Other fields are not affected if the `SQL_DESC_DATA_PTR` field is set to a null pointer.

If the call to **SQLFetch** or **SQLFetchScroll** that fills in the buffer pointed to by this field did not return `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`, the contents of the buffer are undefined.

Whenever the `SQL_DESC_DATA_PTR` field of an APD, ARD, or IPD is set, the driver checks that the value in the `SQL_DESC_TYPE` field contains one of the valid ODBC C data types or a driver-specific data type, and that all other fields affecting the data types are consistent. Prompting a consistency check is the only use of the `SQL_DESC_DATA_PTR` field of an IPD. Specifically, if an application sets the `SQL_DESC_DATA_PTR` field of an IPD and later calls **SQLGetDescField** on this field, it is not necessarily returned the value that it had set. For more information, see "Consistency Checks" in [SQLSetDescRec](#).

SQL_DESC_DATETIME_INTERVAL_CODE [All]

This `SQLSMALLINT` record field contains the subcode for the specific datetime or interval data type when the `SQL_DESC_TYPE` field is `SQL_DATETIME` or `SQL_INTERVAL`. This is true for both SQL and C data types. The code consists of the data type name with "CODE" substituted for either "TYPE" or "C_TYPE" (for datetime types), or "CODE" substituted for "INTERVAL" or "C_INTERVAL" (for interval types).

If `SQL_DESC_TYPE` and `SQL_DESC_CONCISE_TYPE` in an application descriptor are set to `SQL_C_DEFAULT` and the descriptor is not associated with a statement handle, the contents of `SQL_DESC_DATETIME_INTERVAL_CODE` are undefined.

This field can be set for the datetime data types listed in the following table.

Datetime types	DATETIME_INTERVAL_CODE
SQL_TYPE_DATE/SQL_C_TY	SQL_CODE_DATE

PE_DATE	
SQL_TYPE_TIME/SQL_C_TYPE_TIME	SQL_CODE_TIME
SQL_TYPE_TIMESTAMP/SQL_C_TYPE_TIMESTAMP	SQL_CODE_TIMESTAMP

This field can be set for the interval data types listed in the following table.

Interval type	DATETIME_INTERVAL_CODE
SQL_INTERVAL_DAY/ SQL_C_INTERVAL_DAY	SQL_CODE_DAY
SQL_INTERVAL_DAY_TO_HOUR / SQL_C_INTERVAL_DAY_TO_HOUR	SQL_CODE_DAY_TO_HOUR
SQL_INTERVAL_DAY_TO_MINUTE/ SQL_C_INTERVAL_DAY_TO_MINUTE	SQL_CODE_DAY_TO_MINUTE
SQL_INTERVAL_DAY_TO_SECOND/ SQL_C_INTERVAL_DAY_TO_SECOND	SQL_CODE_DAY_TO_SECOND
SQL_INTERVAL_HOUR/ SQL_C_INTERVAL_HOUR	SQL_CODE_HOUR
SQL_INTERVAL_HOUR_TO_MINUTE/ SQL_C_INTERVAL_HOUR_TO_MINUTE	SQL_CODE_HOUR_TO_MINUTE
SQL_INTERVAL_HOUR_TO_SECOND/ SQL_C_INTERVAL_HOUR_TO_SECOND	SQL_CODE_HOUR_TO_SECOND
SQL_INTERVAL_MINUTE/ SQL_C_INTERVAL_MINUTE	SQL_CODE_MINUTE
SQL_INTERVAL_MINUTE_TO_SECOND/ SQL_C_INTERVAL_MINUTE_TO_SECOND	SQL_CODE_MINUTE_TO_SECOND
SQL_INTERVAL_MONTH/ SQL_C_INTERVAL_MONTH	SQL_CODE_MONTH
SQL_INTERVAL_SECOND/ SQL_C_INTERVAL_SECOND	SQL_CODE_SECOND
SQL_INTERVAL_YEAR/ SQL_C_INTERVAL_YEAR	SQL_CODE_YEAR
SQL_INTERVAL_YEAR_TO_MONTH/ SQL_C_INTERVAL_YEAR_TO_MONTH	SQL_CODE_YEAR_TO_MONTH

SQL_C_INTERVAL_YEAR_TO_MONTH	
------------------------------	--

For more information about the data intervals and this field, see “Data Type Identifiers and Descriptors” in in Appendix D, “Data Types” of the **SOLID Programmer Guide**.

SQL_DESC_DATETIME_INTERVAL_PRECISION [All]

This SQLINTEGER record field contains the interval leading precision if the SQL_DESC_TYPE field is SQL_INTERVAL. When the SQL_DESC_DATETIME_INTERVAL_CODE field is set to an interval data type, this field is set to the default interval leading precision.

SQL_DESC_DISPLAY_SIZE [IRDs]

This read-only SQLINTEGER record field contains the maximum number of characters required to display the data from the column.

SQL_DESC_FIXED_PREC_SCALE [Implementation descriptors]

This read-only SQLSMALLINT record field is set to SQL_TRUE if the column is an exact numeric column and has a fixed precision and nonzero scale, or to SQL_FALSE if the column is not an exact numeric column with a fixed precision and scale.

SQL_DESC_INDICATOR_PTR [Application descriptors]

In ARDs, this SQLINTEGER * record field points to the indicator variable. This variable contains SQL_NULL_DATA if the column value is a NULL. For APDs, the indicator variable is set to SQL_NULL_DATA to specify NULL dynamic arguments. Otherwise, the variable is zero (unless the values in SQL_DESC_INDICATOR_PTR and SQL_DESC_OCTET_LENGTH_PTR are the same pointer).

If the SQL_DESC_INDICATOR_PTR field in an ARD is a null pointer, the driver is prevented from returning information about whether the column is NULL or not. If the column is NULL and SQL_DESC_INDICATOR_PTR is a null pointer, SQLSTATE 22002 (Indicator variable required but not supplied) is returned when the driver attempts to populate the buffer after a call to **SQLFetch** or **SQLFetchScroll**. If the call to **SQLFetch** or **SQLFetchScroll** did not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the contents of the buffer are undefined.

The SQL_DESC_INDICATOR_PTR field determines whether the field pointed to by SQL_DESC_OCTET_LENGTH_PTR is set. If the data value for a column is NULL, the driver sets the indicator variable to SQL_NULL_DATA. The field pointed to by SQL_DESC_OCTET_LENGTH_PTR is then not set. If a NULL value is not encountered during the fetch, the buffer pointed to by SQL_DESC_INDICATOR_PTR is set to zero and the buffer pointed to by SQL_DESC_OCTET_LENGTH_PTR is set to the length of the data.

If the SQL_DESC_INDICATOR_PTR field in an APD is a null pointer, the application cannot use this descriptor record to specify NULL arguments.

This field is a *deferred field*: It is not used at the time it is set but is used at a later time by the driver to indicate nullability (for ARDs) or to determine nullability (for APDs).

SQL_DESC_LABEL [IRDs]

This read-only SQLCHAR * record field contains the column label or title. If the column does not have a label, this variable contains the column name. If the column is unnamed and unlabeled, this variable contains an empty string.

SQL_DESC_LENGTH [All]

This SQLINTEGER record field is either the maximum or actual character length of a character string or a binary data type. It is the maximum character length for a fixed-length data type or the actual character length for a variable-length data type. Its value always excludes the null-termination character that ends the character string. For values whose type is SQL_TYPE_DATE, SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP, or one of the SQL interval data types, this field has the length in characters of the character string representation of the datetime or interval value.

The value in this field may be different from the value for "length" as defined in ODBC 2.x. For more information, see Appendix D, "Data Types" of the **SOLID Programmer Guide**.

SQL_DESC_LITERAL_PREFIX [IRDs]

This read-only SQLCHAR * record field contains the character or characters that the driver recognizes as a prefix for a literal of this data type. This variable contains an empty string for a data type for which a literal prefix is not applicable.

SQL_DESC_LITERAL_SUFFIX [IRDs]

This read-only SQLCHAR * record field contains the character or characters that the driver recognizes as a suffix for a literal of this data type. This variable contains an empty string for a data type for which a literal suffix is not applicable.

SQL_DESC_LOCAL_TYPE_NAME [Implementation descriptors]

This read-only SQLCHAR * record field contains any localized (native language) name for the data type that may be different from the regular name of the data type. If there is no localized name, an empty string is returned. This field is for display purposes only.

SQL_DESC_NAME [Implementation descriptors]

This SQLCHAR * record field in a row descriptor contains the column alias, if it applies. If the column alias does not apply, the column name is returned. In either case, the driver sets the SQL_DESC_UNNAMED field to SQL_NAMED when it sets the SQL_DESC_NAME field. If there is no column name or a column alias, the driver returns an empty string in the SQL_DESC_NAME field and sets the SQL_DESC_UNNAMED field to SQL_UNNAMED.

An application can set the SQL_DESC_NAME field of an IPD to a parameter name or alias to specify stored procedure parameters by name. (For more information, see the Part I PDF file, "Binding Parameters by Name (Named Parameters)" in Chapter 9, "Executing Statements.") The SQL_DESC_NAME field of an IRD is a read-only field; SQLSTATE HY091 (Invalid descriptor field identifier) will be returned if an application attempts to set it.

In IPDs, this field is undefined if the driver does not support named parameters. If the driver supports named parameters and is capable of describing parameters, the parameter name is returned in this field.

SQL_DESC_NULLABLE [Implementation descriptors]

In IRDs, this read-only SQLSMALLINT record field is SQL_NULLABLE if the column can have NULL values, SQL_NO_NULLS if the column does not have NULL values, or SQL_NULLABLE_UNKNOWN if it is not known whether the column accepts NULL values. This field pertains to the result set column, not the base column.

In IPDs, this field is always set to SQL_NULLABLE because dynamic parameters are always nullable and cannot be set by an application.

SQL_DESC_NUM_PREC_RADIX [All]

This SQLINTEGER field contains a value of 2 if the data type in the SQL_DESC_TYPE field is an approximate numeric data type, because the SQL_DESC_PRECISION field contains the number of bits. This field contains a value of 10 if the data type in the SQL_DESC_TYPE field is an exact numeric data type, because the SQL_DESC_PRECISION field contains the number of decimal digits. This field is set to 0 for all non-numeric data types.

SQL_DESC_OCTET_LENGTH [All]

This SQLINTEGER record field contains the length, in bytes, of a character string or binary data type. For fixed-length character or binary types, this is the actual length in bytes. For variable-length character or binary types, this is the maximum length in bytes. This value always excludes space for the null-termination character for implementation descriptors and always includes space for the null-termination character for application descriptors. For application data, this field contains the size of the buffer. For APDs, this field is defined only for output or input/output parameters.

SQL_DESC_OCTET_LENGTH_PTR [Application descriptors]

This SQLINTEGER * record field points to a variable that will contain the total length in bytes of a dynamic argument (for parameter descriptors) or of a bound column value (for row descriptors).

For an APD, this value is ignored for all arguments except character string and binary; if this field points to SQL_NTS, the dynamic argument must be null-terminated. To indicate that a bound parameter will be a data-at-execution parameter, an application sets this field in the appropriate record of the APD to a variable that, at execute time, will contain the value SQL_DATA_AT_EXEC or the result of the SQL_LEN_DATA_AT_EXEC macro. If there is more than one such field, SQL_DESC_DATA_PTR can be set to a value uniquely identifying the parameter to help the application determine which parameter is being requested.

If the OCTET_LENGTH_PTR field of an ARD is a null pointer, the driver does not return length information for the column. If the SQL_DESC_OCTET_LENGTH_PTR field of an APD is a null pointer, the driver assumes that character strings and binary values are null-terminated. (Binary values should not be null-terminated but should be given a length to avoid truncation.)

If the call to **SQLFetch** or **SQLFetchScroll** that fills in the buffer pointed to by this field did not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the contents of the buffer are undefined. This field is a *deferred field*. It is not used at the time it is set but is used at a later time by the driver to determine or indicate the octet length of the data.

SQL_DESC_PARAMETER_TYPE [IPDs]

This SQLSMALLINT record field is set to SQL_PARAM_INPUT for an input parameter, SQL_PARAM_INPUT_OUTPUT for an input/output parameter, or SQL_PARAM_OUTPUT for an output parameter. It is set to SQL_PARAM_INPUT by default.

For an IPD, the field is set to `SQL_PARAM_INPUT` by default if the IPD is not automatically populated by the driver (the `SQL_ATTR_ENABLE_AUTO_IPD` statement attribute is `SQL_FALSE`). An application should set this field in the IPD for parameters that are not input parameters.

SQL_DESC_PRECISION [All]

This `SQLSMALLINT` record field contains the number of digits for an exact numeric type, the number of bits in the mantissa (binary precision) for an approximate numeric type, or the numbers of digits in the fractional seconds component for the `SQL_TYPE_TIME`, `SQL_TYPE_TIMESTAMP`, or `SQL_INTERVAL_SECOND` data type. This field is undefined for all other data types.

The value in this field may be different from the value for "precision" as defined in ODBC 2.x. For more information, see Appendix D, "Data Types" of the **SOLID Programmer Guide**.

SQL_DESC_ROWVER [Implementation descriptors]

This `SQLSMALLINT` record field indicates whether a column is automatically modified by the DBMS when a row is updated (for example, a column of the type "timestamp" in SQL Server). The value of this record field is set to `SQL_TRUE` if the column is a row versioning column, and to `SQL_FALSE` otherwise. This column attribute is similar to calling **SQLSpecialColumns** with IdentifierType of `SQL_ROWVER` to determine whether a column is automatically updated.

SQL_DESC_SCALE [All]

This `SQLSMALLINT` record field contains the defined scale for decimal and numeric data types. The field is undefined for all other data types.

The value in this field may be different from the value for "scale" as defined in ODBC 2.x. For more information, see Appendix D, "Data Types" of the **SOLID Programmer Guide**.

SQL_DESC_SCHEMA_NAME [IRDs]

This read-only `SQLCHAR *` record field contains the schema name of the base table that contains the column. The return value is driver-dependent if the column is an expression or if the column is part of a view. If the data source does not support schemas or the schema name cannot be determined, this variable contains an empty string.

SQL_DESC_SEARCHABLE [IRDs]

This read-only `SQLSMALLINT` record field is set to one of the following values:

`SQL_PRED_NONE` if the column cannot be used in a **WHERE** clause. (This is the same as the `SQL_UNSEARCHABLE` value in ODBC 2.x.)

`SQL_PRED_CHAR` if the column can be used in a **WHERE** clause but only with the **LIKE** predicate. (This is the same as the `SQL_LIKE_ONLY` value in ODBC 2.x.)

`SQL_PRED_BASIC` if the column can be used in a **WHERE** clause with all the comparison operators except **LIKE**. (This is the same as the `SQL_EXCEPT_LIKE` value in ODBC 2.x.)

`SQL_PRED_SEARCHABLE` if the column can be used in a **WHERE** clause with any comparison operator.

SQL_DESC_TABLE_NAME [IRDs]

This read-only SQLCHAR * record field contains the name of the base table that contains this column. The return value is driver-dependent if the column is an expression or if the column is part of a view.

SQL_DESC_TYPE [All]

This SQLSMALLINT record field specifies the concise SQL or C data type for all data types except datetime and interval data types. For the datetime and interval data types, this field specifies the verbose data type, which is SQL_DATETIME or SQL_INTERVAL.

Whenever this field contains SQL_DATETIME or SQL_INTERVAL, the SQL_DESC_DATETIME_INTERVAL_CODE field must contain the appropriate subcode for the concise type. For datetime data types, SQL_DESC_TYPE contains SQL_DATETIME, and the SQL_DESC_DATETIME_INTERVAL_CODE field contains a subcode for the specific datetime data type. For interval data types, SQL_DESC_TYPE contains SQL_INTERVAL and the SQL_DESC_DATETIME_INTERVAL_CODE field contains a subcode for the specific interval data type.

The values in the SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE fields are interdependent. Each time one of the fields is set, the other must also be set. SQL_DESC_TYPE can be set by a call to **SQLSetDescField** or **SQLSetDescRec**. SQL_DESC_CONCISE_TYPE can be set by a call to **SQLBindCol** or **SQLBindParameter**, or **SQLSetDescField**.

If SQL_DESC_TYPE is set to a concise data type other than an interval or datetime data type, the SQL_DESC_CONCISE_TYPE field is set to the same value and the SQL_DESC_DATETIME_INTERVAL_CODE field is set to 0.

If SQL_DESC_TYPE is set to the verbose datetime or interval data type (SQL_DATETIME or SQL_INTERVAL) and the SQL_DESC_DATETIME_INTERVAL_CODE field is set to the appropriate subcode, the SQL_DESC_CONCISE_TYPE field is set to the corresponding concise type. Trying to set SQL_DESC_TYPE to one of the concise datetime or interval types will return SQLSTATE HY021 (Inconsistent descriptor information).

When the SQL_DESC_TYPE field is set by a call to **SQLBindCol**, **SQLBindParameter**, or **SQLSetDescField**, the following fields are set to the following default values, as shown in the table below. The values of the remaining fields of the same record are undefined.

Value of SQL_DESC_TYPE	Other fields implicitly set
SQL_CHAR, SQL_VARCHAR, SQL_C_CHAR, SQL_C_VARCHAR	SQL_DESC_LENGTH is set to 1. SQL_DESC_PRECISION is set to 0.
SQL_DATETIME	When SQL_DESC_DATETIME_INTERVAL_CODE is set to SQL_CODE_DATE or SQL_CODE_TIME, SQL_DESC_PRECISION is set to 0. When it is set to SQL_DESC_TIMESTAMP, SQL_DESC_PRECISION is set to 6.
SQL_DECIMAL, SQL_NUMERIC, SQL_C_NUMERIC	SQL_DESC_SCALE is set to 0. SQL_DESC_PRECISION is set to the implementation-defined precision for the respective data type.
SQL_FLOAT, SQL_C_FLOAT	SQL_DESC_PRECISION is set to the implementation-defined default precision for SQL_FLOAT.
SQL_INTERVAL	When SQL_DESC_DATETIME_INTERVAL_CODE is set to an interval data type, SQL_DESC_DATETIME_INTERVAL_PRECISION is set to 2 (the default interval leading precision). When

	the interval has a seconds component, SQL_DESC_PRECISION is set to 6 (the default interval seconds precision).
--	----------------------------------------------------------------------------------------------------------------

When an application calls **SQLSetDescField** to set fields of a descriptor rather than calling **SQLSetDescRec**, the application must first declare the data type. When it does, the other fields indicated in the previous table are implicitly set. If any of the values implicitly set are unacceptable, the application can then call **SQLSetDescField** or **SQLSetDescRec** to set the unacceptable value explicitly.

SQL_DESC_TYPE_NAME [Implementation descriptors]

This read-only SQLCHAR * record field contains the data source-dependent type name (for example, "CHAR", "VARCHAR", and so on). If the data type name is unknown, this variable contains an empty string.

SQL_DESC_UNNAMED [Implementation descriptors]

This SQLSMALLINT record field in a row descriptor is set by the driver to either SQL_NAMED or SQL_UNNAMED when it sets the SQL_DESC_NAME field. If the SQL_DESC_NAME field contains a column alias or if the column alias does not apply, the driver sets the SQL_DESC_UNNAMED field to SQL_NAMED. If an application sets the SQL_DESC_NAME field of an IPD to a parameter name or alias, the driver sets the SQL_DESC_UNNAMED field of the IPD to SQL_NAMED. If there is no column name or a column alias, the driver sets the SQL_DESC_UNNAMED field to SQL_UNNAMED.

An application can set the SQL_DESC_UNNAMED field of an IPD to SQL_UNNAMED. A driver returns SQLSTATE HY091 (Invalid descriptor field identifier) if an application attempts to set the SQL_DESC_UNNAMED field of an IPD to SQL_NAMED. The SQL_DESC_UNNAMED field of an IRD is read-only; SQLSTATE HY091 (Invalid descriptor field identifier) will be returned if an application attempts to set it.

SQL_DESC_UNSIGNED [Implementation descriptors]

This read-only SQLSMALLINT record field is set to SQL_TRUE if the column type is unsigned or non-numeric, or SQL_FALSE if the column type is signed.

SQL_DESC_UPDATABLE [IRDs]

This read-only SQLSMALLINT record field is set to one of the following values:

SQL_ATTR_READ_ONLY if the result set column is read-only.

SQL_ATTR_WRITE if the result set column is read-write.

SQL_ATTR_READWRITE_UNKNOWN if it is not known whether the result set column is updatable or not.

SQL_DESC_UPDATABLE describes the updatability of the column in the result set, not the column in the base table. The updatability of the column in the base table on which this result set column is based may be different than the value in this field. 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.

Consistency Checks

A consistency check is performed by the driver automatically whenever an application passes in a value for the SQL_DESC_DATA_PTR field of the ARD, APD, or IPD. If any of the fields is inconsistent with other fields, **SQLSetDescField** will return SQLSTATE HY021 (Inconsistent descriptor information). For more information, see "Consistency Check" in [SQLSetDescRec](#).

Related Functions

For information about	See
Binding a column	SQLBindCol
Binding a parameter	SQLBindParameter
Getting a descriptor field	SQLGetDescField
Getting multiple descriptor fields	SQLSetDescField
Setting multiple descriptor fields	SQLSetDescRec

SQLSetDescRec

Conformance

Version Introduced: ODBC 3.0

Standards Compliance: ISO 92

Summary

The **SQLSetDescRec** function sets multiple descriptor fields that affect the data type and buffer bound to a column or parameter data.

Syntax

SQLRETURN SQLSetDescRec(
SQLHDESC	DescriptorHandle,
SQLSMALLINT	RecNumber,
SQLSMALLINT	Type,
SQLSMALLINT	SubType,
SQLINTEGER	Length,
SQLSMALLINT	Precision,
SQLSMALLINT	Scale,
SQLPOINTER	DataPtr,
SQLINTEGER *	StringLengthPtr,
SQLINTEGER *	IndicatorPtr);

Arguments

DescriptorHandle

[Input]

Descriptor handle. This must not be an IRD handle.

RecNumber

[Input]

Indicates the descriptor record that contains the fields to be set. Descriptor records are numbered from 0, with record number 0 being the bookmark record. This argument must be equal to or greater than 0. If *RecNumber* is greater than the value of SQL_DESC_COUNT, SQL_DESC_COUNT is changed to the value of *RecNumber*.

Type

[Input]

The value to which to set the SQL_DESC_TYPE field for the descriptor record.

SubType

[Input]

For records whose type is SQL_DATETIME or SQL_INTERVAL, this is the value to which to set the SQL_DESC_DATETIME_INTERVAL_CODE field.

Length

[Input]

The value to which to set the SQL_DESC_OCTET_LENGTH field for the descriptor record.

Precision

[Input]

The value to which to set the SQL_DESC_PRECISION field for the descriptor record.

Scale

[Input]

The value to which to set the SQL_DESC_SCALE field for the descriptor record.

DataPtr

[Deferred Input or Output]

The value to which to set the SQL_DESC_DATA_PTR field for the descriptor record. *DataPtr* can be set to a null pointer.

The *DataPtr* argument can be set to a null pointer to set the SQL_DESC_DATA_PTR field to a null pointer. If the handle in the *DescriptorHandle* argument is associated with an ARD, this unbinds the column.

StringLengthPtr

[Deferred Input or Output]

The value to which to set the SQL_DESC_OCTET_LENGTH_PTR field for the descriptor record. *StringLengthPtr* can be set to a null pointer to set the SQL_DESC_OCTET_LENGTH_PTR field to a null pointer.

IndicatorPtr

[Deferred Input or Output]

The value to which to set the `SQL_DESC_INDICATOR_PTR` field for the descriptor record. *IndicatorPtr* can be set to a null pointer to set the `SQL_DESC_INDICATOR_PTR` field to a null pointer.

Returns

`SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

Diagnostics

When **SQLSetDescRec** returns `SQL_ERROR` or `SQL_SUCCESS_WITH_INFO`, an associated `SQLSTATE` value can be obtained by calling **SQLGetDiagRec** with a *HandleType* of `SQL_HANDLE_DESC` and a *Handle* of *DescriptorHandle*. The following table lists the `SQLSTATE` values commonly returned by **SQLSetDescRec** and explains each one in the context of this function; the notation "(DM)" precedes the descriptions of `SQLSTATE`s 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 <code>SQL_SUCCESS_WITH_INFO</code> .)
07009	Invalid descriptor index	<p>The <i>RecNumber</i> argument was set to 0, and the <i>DescriptorHandle</i> referred to an IPD handle.</p> <p>The <i>RecNumber</i> argument was less than 0.</p> <p>The <i>RecNumber</i> argument was greater than the maximum number of columns or parameters that the data source can support, and the <i>DescriptorHandle</i> argument was an APD, IPD, or ARD.</p> <p>The <i>RecNumber</i> argument was equal to 0, and the <i>DescriptorHandle</i> argument referred to an implicitly allocated APD. (This error does not occur with an explicitly allocated application descriptor because it is not known whether an explicitly allocated application descriptor is an APD or ARD until execute time.)</p>
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.
HY000	General error	An error occurred for which there was no specific <code>SQLSTATE</code> and for which no implementation-specific <code>SQLSTATE</code> was defined. The error message returned by SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY010	Function sequence error	(DM) The <i>DescriptorHandle</i> was associated with a <i>StatementHandle</i> for which an asynchronously executing function (not this one) was called and was

		<p>still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> with which the <i>DescriptorHandle</i> was associated and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY016	Cannot modify an implementation row descriptor	The <i>DescriptorHandle</i> argument was associated with an IRD.
HY021	Inconsistent descriptor information	<p>The <i>Type</i> field, or any other field associated with the SQL_DESC_TYPE field in the descriptor, was not valid or consistent.</p> <p>Descriptor information checked during a consistency check was not consistent. (See "Consistency Checks," later in this section.)</p>
HY090	Invalid string or buffer length	(DM) The driver was an ODBC 2.x driver, the descriptor was an ARD, the <i>ColumnNumber</i> argument was set to 0, and the value specified for the argument <i>BufferLength</i> was not equal to 4.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>DescriptorHandle</i> does not support the function.

Comments

An application can call **SQLSetDescRec** to set the following fields for a single column or parameter:

SQL_DESC_TYPE
 SQL_DESC_DATETIME_INTERVAL_CODE (for records whose type is SQL_DATETIME or SQL_INTERVAL)
 SQL_DESC_OCTET_LENGTH
 SQL_DESC_PRECISION
 SQL_DESC_SCALE
 SQL_DESC_DATA_PTR
 SQL_DESC_OCTET_LENGTH_PTR
 SQL_DESC_INDICATOR_PTR

Note If a call to **SQLSetDescRec** fails, the contents of the descriptor record identified by the *RecNumber* argument are undefined.

When binding a column or parameter, **SQLSetDescRec** allows you to change multiple fields affecting the binding without calling **SQLBindCol** or **SQLBindParameter** or making multiple calls to

SQLSetDescField. **SQLSetDescRec** can set fields on a descriptor not currently associated with a statement. Note that **SQLBindParameter** sets more fields than **SQLSetDescRec**, can set fields on both an APD and an IPD in one call, and does not require a descriptor handle.

Note The statement attribute `SQL_ATTR_USE_BOOKMARKS` should always be set before calling **SQLSetDescRec** with a *RecNumber* argument of 0 to set bookmark fields. While this is not mandatory, it is strongly recommended.

Consistency Checks

A consistency check is performed by the driver automatically whenever an application sets the `SQL_DESC_DATA_PTR` field of an APD, ARD, or IPD. If any of the fields is inconsistent with other fields, **SQLSetDescRec** will return `SQLSTATE HY021` (Inconsistent descriptor information).

Whenever an application sets the `SQL_DESC_DATA_PTR` field of an APD, ARD, or IPD, the driver checks that the value of the `SQL_DESC_TYPE` field and the values applicable to that `SQL_DESC_TYPE` field are valid and consistent. This check is always performed when **SQLBindParameter** or **SQLBindCol** is called or when **SQLSetDescRec** is called for an APD, ARD, or IPD. This consistency check includes the following checks on descriptor fields:

The `SQL_DESC_TYPE` field must be one of the valid ODBC C or SQL types or a driver-specific SQL type. The `SQL_DESC_CONCISE_TYPE` field must be one of the valid ODBC C or SQL types or a driver-specific C or SQL type, including the concise datetime and interval types.

If the `SQL_DESC_TYPE` record field is `SQL_DATETIME` or `SQL_INTERVAL`, the `SQL_DESC_DATETIME_INTERVAL_CODE` field must be one of the valid datetime or interval codes. (See the description of the `SQL_DESC_DATETIME_INTERVAL_CODE` field in [SQLSetDescField](#).)

If the `SQL_DESC_TYPE` field indicates a numeric type, the `SQL_DESC_PRECISION` and `SQL_DESC_SCALE` fields are verified to be valid.

If the `SQL_DESC_CONCISE_TYPE` field is a time or timestamp data type, an interval type with a seconds component, or one of the interval data types with a time component, the `SQL_DESC_PRECISION` field is verified to be a valid seconds precision.

If the `SQL_DESC_CONCISE_TYPE` is an interval data type, the `SQL_DESC_DATETIME_INTERVAL_PRECISION` field is verified to be a valid interval leading precision value.

The `SQL_DESC_DATA_PTR` field of an IPD is not normally set; however, an application can do so to force a consistency check of IPD fields. A consistency check cannot be performed on an IRD. The value that the `SQL_DESC_DATA_PTR` field of the IPD is set to is not actually stored and cannot be retrieved by a call to **SQLGetDescField** or **SQLGetDescRec**; the setting is made only to force the consistency check.

Related Functions

For information about	See
Binding a column	SQLBindCol
Binding a parameter	SQLBindParameter
Getting a single descriptor field	SQLGetDescField
Getting multiple descriptor fields	SQLGetDescRec
Setting single descriptor fields	SQLSetDescField

SQLSetEnvAttr

Conformance

Version Introduced: ODBC 3.0
Standards Compliance: ISO 92

Summary

SQLSetEnvAttr sets attributes that govern aspects of environments.

Syntax

SQLRETURN SQLSetEnvAttr(
SQLHENV	EnvironmentHandle,
SQLINTEGER	Attribute,
SQLPOINTER	ValuePtr,
SQLINTEGER	StringLength);

Arguments

EnvironmentHandle

[Input]
Environment handle.

Attribute

[Input]
Attribute to set, listed in "Comments."

ValuePtr

[Input]
Pointer to the value to be associated with *Attribute*. Depending on the value of *Attribute*, *ValuePtr* will be a 32-bit integer value or point to a null-terminated character string.

StringLength

[Input] If *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **ValuePtr*. If *ValuePtr* is an integer, *StringLength* is ignored.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLSetEnvAttr** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value can be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_ENV and a *Handle* of *EnvironmentHandle*. The following table lists the SQLSTATE

values commonly returned by **SQLSetEnvAttr** 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. If a driver does not support an environment attribute, the error can be returned only during connect time.

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 value specified in <i>ValuePtr</i> and substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY009	Invalid use of null pointer	The <i>Attribute</i> argument identified an environment attribute that required a string value, and the <i>ValuePtr</i> argument was a null pointer.
HY010	Function sequence error	(DM) A connection handle has been allocated on <i>EnvironmentHandle</i> .
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY024	Invalid attribute value	Given the specified <i>Attribute</i> value, an invalid value was specified in <i>ValuePtr</i> .
HY090	Invalid string or buffer length	The <i>StringLength</i> argument was less than 0 but was not SQL_NTS.
HY092	Invalid attribute/option identifier	(DM) The value specified for the argument <i>Attribute</i> was not valid for the version of ODBC supported by the driver.
HYC00	Optional feature not implemented	The value specified for the argument <i>Attribute</i> was a valid ODBC environment attribute for the version of ODBC supported by the driver, but was not supported by the driver. (DM) The <i>Attribute</i> argument was SQL_ATTR_OUTPUT_NTS, and <i>ValuePtr</i> was SQL_FALSE.

Comments

An application can call **SQLSetEnvAttr** only if no connection handle is allocated on the environment. All environment attributes successfully set by the application for the environment persist until **SQLFreeHandle** is called on the environment. More than one environment handle can be allocated simultaneously in ODBC 3.x.

The format of information set through *ValuePtr* depends on the specified *Attribute*. **SQLSetEnvAttr** will accept attribute 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 attribute's description.

There are no driver-specific environment attributes.

Connection attributes cannot be set by a call to **SQLSetEnvAttr**. Attempting to do so will return SQLSTATE HY092 (Invalid attribute/option identifier).

Attribute	<i>ValuePtr</i> contents
SQL_ATTR_CONNECTION_POOLING (ODBC 3.0)	<p>A 32-bit SQLINTEGER value that enables or disables connection pooling at the environment level. The following values are used:</p> <p>SQL_CP_OFF = Connection pooling is turned off. This is the default.</p> <p>SQL_CP_ONE_PER_DRIVER = A single connection pool is supported for each driver. Every connection in a pool is associated with one driver.</p> <p>SQL_CP_ONE_PER_HENV = A single connection pool is supported for each environment. Every connection in a pool is associated with one environment.</p> <p>Connection pooling is enabled by calling SQLSetEnvAttr to set the SQL_ATTR_CONNECTION_POOLING attribute to SQL_CP_ONE_PER_DRIVER or SQL_CP_ONE_PER_HENV. This call must be made before the application allocates the shared environment for which connection pooling is to be enabled. The environment handle in the call to SQLSetEnvAttr is set to null, which makes SQL_ATTR_CONNECTION_POOLING a process-level attribute. After connection pooling is enabled, the application then allocates an implicit shared environment by calling SQLAllocHandle with the <i>InputHandle</i> argument set to SQL_HANDLE_ENV.</p> <p>After connection pooling has been enabled and a shared environment has been selected for an application, SQL_ATTR_CONNECTION_POOLING cannot be reset for that environment, because SQLSetEnvAttr is called with a null environment handle when setting this attribute. If this attribute is set while connection pooling is already enabled on a shared environment, the attribute affects only shared environments that are allocated subsequently.</p> <p>For more information, see the Part I PDF file, "ODBC Connection Pooling" in Chapter 6, "Connecting to a Data Source or Driver."</p>
SQL_ATTR_CP_MATCH (ODBC 3.0)	<p>A 32-bit SQLINTEGER value that determines how a connection is chosen from a connection pool. When SQLConnect or SQLDriverConnect is called, the Driver Manager determines which connection is reused from the pool. The Driver Manager attempts to match the connection options in the call and the connection attributes set by the application to the keywords and connection attributes of the connections in the pool. The value of this attribute determines the</p>

	<p>level of precision of the matching criteria.</p> <p>The following values are used to set the value of this attribute:</p> <p>SQL_CP_STRICT_MATCH = Only connections that exactly match the connection options in the call and the connection attributes set by the application are reused. This is the default.</p> <p>SQL_CP_RELAXED_MATCH = Connections with matching connection string keywords can be used. Keywords must match, but not all connection attributes must match.</p> <p>For more information on how the Driver Manager performs the match in connecting to a pooled connection, see SQLConnect. For more information on connection pooling, see the Part I PDF file, "ODBC Connection Pooling" in Chapter 6, "Connecting to a Data Source or Driver."</p>
SQL_ATTR_ODBC_VERSION (ODBC 3.0)	<p>A 32-bit integer that determines whether certain functionality exhibits ODBC 2.x behavior or ODBC 3.x behavior. The following values are used to set the value of this attribute:</p> <p>SQL_OV_ODBC3 = The Driver Manager and driver exhibit the following ODBC 3.x behavior:</p> <p>The driver returns and expects ODBC 3.x codes for date, time, and timestamp.</p> <p>The driver returns ODBC 3.x SQLSTATE codes when SQLError, SQLGetDiagField, or SQLGetDiagRec is called.</p> <p>The <i>CatalogName</i> argument in a call to SQLTables accepts a search pattern.</p> <p>SQL_OV_ODBC2 = The Driver Manager and driver exhibit the following ODBC 2.x behavior. This is especially useful for an ODBC 2.x application working with an ODBC 3.x driver.</p> <p>The driver returns and expects ODBC 2.x codes for date, time, and timestamp.</p> <p>The driver returns ODBC 2.x SQLSTATE codes when SQLError, SQLGetDiagField, or SQLGetDiagRec is called.</p> <p>The <i>CatalogName</i> argument in a call to SQLTables does not accept a search pattern.</p> <p>An application must set this environment attribute before calling any function that has an SQLHENV argument, or the call will return SQLSTATE HY010 (Function sequence error).</p> <p>For more information, see the Part I PDF file, "Declaring the Application's ODBC Version," in Chapter 6, and "Connecting to a Data Source or Driver" and "Behavioral Changes" in Chapter 17, "Programming Considerations."</p>
SQL_ATTR_OUTPUT_NTS (ODBC 3.0)	<p>A 32-bit integer that determines how the driver returns string data. If SQL_TRUE, the driver returns string data null-terminated. If SQL_FALSE, the driver does not return string data null-terminated.</p> <p>This attribute defaults to SQL_TRUE. A call to SQLSetEnvAttr to</p>

	set it to SQL_TRUE returns SQL_SUCCESS. A call to SQLSetEnvAttr to set it to SQL_FALSE returns SQL_ERROR and SQLSTATE HYC00 (Optional feature not implemented).
--	------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Related Functions

For information about	See
Allocating a handle	SQLAllocHandle
Returning the setting of an environment attribute	SQLGetEnvAttr

SQLSetParam

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: Deprecated

Summary

In ODBC 2.0, the ODBC 1.0 function **SQLSetParam** has been replaced by **SQLBindParameter**. For more information, see [SQLBindParameter](#).

Note For more information about what the Driver Manager maps this function to when an ODBC 2.x application is working with an ODBC 3.x driver, see “Mapping Deprecated Functions” (Appendix G, "Driver Guidelines for Backward Compatibility") on the Microsoft Web site (ODBC Programmer’s Reference).

SQLSetPos

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: ODBC

Summary

SQLSetPos sets the cursor position in a rowset and allows an application to refresh data in the rowset or to update or delete data in the result set.

Syntax

SQLRETURN SQLSetPos (
SQLHSTMT	StatementHandle,
SQLUSMALLINT	RowNumber,
SQLUSMALLINT	Operation,
SQLUSMALLINT	LockType);

Arguments

StatementHandle

[Input]

Statement handle.

RowNumber

[Input]

Position of the row in the rowset on which to perform the operation specified with the *Operation* argument. If *RowNumber* is 0, the operation applies to every row in the rowset.

For additional information, see "Comments."

Operation

[Input]

Operation to perform:

SQL_POSITION

SQL_REFRESH

SQL_UPDATE

SQL_DELETE

Note The SQL_ADD value for the *Operation* argument has been deprecated for ODBC 3.x. ODBC 3.x drivers will need to support SQL_ADD for backward compatibility. This functionality has been replaced by a call to **SQLBulkOperations** with an *Operation* of SQL_ADD. When an ODBC 3.x application works with an ODBC 2.x driver, the Driver Manager maps a call to **SQLBulkOperations** with an *Operation* of SQL_ADD to **SQLSetPos** with an *Operation* of SQL_ADD.

For more information, see "Comments."

LockType

[Input]

Specifies how to lock the row after performing the operation specified in the *Operation* 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 **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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.

For all those SQLSTATEs that can return SQL_SUCCESS_WITH_INFO or SQL_ERROR (except 01xxx SQLSTATEs), SQL_SUCCESS_WITH_INFO is returned if an error occurs on one or more, but not all, rows of a multirow operation, and SQL_ERROR is returned if an error occurs on a single-row operation.

SQLSTATE	Error	Description
01000	General warning	Driver-specific informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01001	Cursor operation conflict	<p>The <i>Operation</i> argument was SQL_DELETE or SQL_UPDATE, and no rows or more than one row were deleted or updated. (For more information about updates to more than one row, see the description of the SQL_ATTR_SIMULATE_CURSOR <i>Attribute</i> in SQLSetStmtAttr.) (Function returns SQL_SUCCESS_WITH_INFO.)</p> <p>The <i>Operation</i> argument was SQL_DELETE or SQL_UPDATE, and the operation failed because of optimistic concurrency. (Function returns SQL_SUCCESS_WITH_INFO.)</p>
01004	String data right truncation	The <i>Operation</i> argument was SQL_REFRESH, and string or binary data returned for a column or columns with a data type of SQL_C_CHAR or SQL_C_BINARY resulted in the truncation of nonblank character or non-NULL binary data.
01S01	Error in row	<p>The <i>RowNumber</i> argument was 0, and an error occurred in one or more rows while performing the operation specified with the <i>Operation</i> argument.</p> <p>(SQL_SUCCESS_WITH_INFO is returned if an error occurs on one or more, but not all, rows of a multirow operation, and SQL_ERROR is returned if an error occurs on a single-row operation.)</p> <p>(This SQLSTATE is returned only when SQLSetPos is called after SQLExtendedFetch, if the driver is an ODBC 2.x driver and the cursor library is not used.)</p>
01S07	Fractional truncation	<p>The <i>Operation</i> argument was SQL_REFRESH, the data type of the application buffer was not SQL_C_CHAR or SQL_C_BINARY, and the data returned to application buffers for one or more columns was truncated. For numeric data types, the fractional part of the number was truncated. For time, timestamp, and interval data types containing a time component, the fractional portion of the time was truncated.</p> <p>(Function returns SQL_SUCCESS_WITH_INFO.)</p>

07006	Restricted data type attribute violation	The data value of a column in the result set could not be converted to the data type specified by <i>TargetType</i> in the call to SQLBindCol .
07009	Invalid descriptor index	The argument <i>Operation</i> was SQL_REFRESH or SQL_UPDATE, and a column was bound with a column number greater than the number of columns in the result set.
21S02	Degree of derived table does not match column list	The argument <i>Operation</i> was SQL_UPDATE, and no columns were updatable because all columns were either unbound, read-only, or the value in the bound length/indicator buffer was SQL_COLUMN_IGNORE.
22001	String data, right truncation	The <i>Operation</i> argument was SQL_UPDATE, and the assignment of a character or binary value to a column resulted in the truncation of nonblank (for characters) or non-null (for binary) characters or bytes.
22003	Numeric value out of range	<p>The argument <i>Operation</i> was SQL_UPDATE, and the assignment of a numeric value to a column in the result set caused the whole (as opposed to fractional) part of the number to be truncated.</p> <p>The argument <i>Operation</i> was SQL_REFRESH, and returning the numeric value for one or more bound columns would have caused a loss of significant digits.</p>
22007	Invalid datetime format	<p>The argument <i>Operation</i> was SQL_UPDATE, and the assignment of a date or timestamp value to a column in the result set caused the year, month, or day field to be out of range.</p> <p>The argument <i>Operation</i> was SQL_REFRESH, and returning the date or timestamp value for one or more bound columns would have caused the year, month, or day field to be out of range.</p>
22008	Date/time field overflow	<p>The <i>Operation</i> argument was SQL_UPDATE, and the performance of datetime arithmetic on data being sent to a column in the result set resulted in a datetime field (the year, month, day, hour, minute, or second field) of the result being outside the permissible range of values for the field, or being invalid based on the Gregorian calendar's natural rules for datetimes.</p> <p>The <i>Operation</i> argument was SQL_REFRESH, and the performance of datetime arithmetic on data being retrieved from the result set resulted in a datetime field (the year, month, day, hour, minute, or second field) of the result being outside the permissible range of values for the field, or being invalid based on the Gregorian calendar's natural rules for datetimes.</p>
22015	Interval field overflow	The <i>Operation</i> argument was SQL_UPDATE, and the assignment of an exact numeric or interval C type to an

		<p>interval SQL data type caused a loss of significant digits.</p> <p>The <i>Operation</i> argument was SQL_UPDATE; when assigning to an interval SQL type, there was no representation of the value of the C type in the interval SQL type.</p> <p>The <i>Operation</i> argument was SQL_REFRESH, and assigning from an exact numeric or interval SQL type to an interval C type caused a loss of significant digits in the leading field.</p> <p>The <i>Operation</i> argument was SQL_REFRESH; when assigning to an interval C type, there was no representation of the value of the SQL type in the interval C type.</p>
22018	Invalid character value for cast specification	<p>The <i>Operation</i> argument was SQL_REFRESH; the C type was an exact or approximate numeric, a datetime, or an interval data type; the SQL type of the column was a character data type; and the value in the column was not a valid literal of the bound C type.</p> <p>The argument <i>Operation</i> was SQL_UPDATE; the SQL type was an exact or approximate numeric, a datetime, or an interval data type; the C type was SQL_C_CHAR; and the value in the column was not a valid literal of the bound SQL type.</p>
23000	Integrity constraint violation	The argument <i>Operation</i> was SQL_DELETE or SQL_UPDATE, and an integrity constraint was violated.
24000	Invalid cursor state	<p>The <i>StatementHandle</i> was in an executed state, but no result set was associated with the <i>StatementHandle</i>.</p> <p>(DM) A cursor was open on the <i>StatementHandle</i>, but SQLFetch or SQLFetchScroll had not been called.</p> <p>A cursor was open on the <i>StatementHandle</i>, and SQLFetch or SQLFetchScroll had been called, but the cursor was positioned before the start of the result set or after the end of the result set.</p> <p>The argument <i>Operation</i> 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.</p>
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.

42000	Syntax error or access violation	<p>The driver was unable to lock the row as needed to perform the operation requested in the argument <i>Operation</i>.</p> <p>The driver was unable to lock the row as requested in the argument <i>LockType</i>.</p>
44000	WITH CHECK OPTION violation	<p>The <i>Operation</i> argument was SQL_UPDATE, and the update was performed on a viewed table or a table derived from the viewed table which was created by specifying WITH CHECK OPTION, such that one or more rows affected by the update will no longer be present in the viewed table.</p>
HY000	General error	<p>An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by SQLGetDiagRec in the *<i>MessageText</i> buffer describes the error and its cause.</p>
HY001	Memory allocation error	<p>The driver was unable to allocate memory required to support execution or completion of the function.</p>
HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>, and then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY010	Function sequence error	<p>(DM) The specified <i>StatementHandle</i> was not in an executed state. The function was called without first calling SQLExecDirect, SQLExecute, or a catalog function.</p> <p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p> <p>(DM) The driver was an ODBC 2.x driver, and SQLSetPos was called for a <i>StatementHandle</i> after SQLFetch was called.</p>
HY011	Attribute cannot be set now	<p>(DM) The driver was an ODBC 2.x driver; the SQL_ATTR_ROW_STATUS_PTR statement attribute was set; then SQLSetPos was called before</p>

		SQLFetch , SQLFetchScroll , or SQLExtendedFetch was called.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	<p>The <i>Operation</i> argument was SQL_UPDATE, a data value was a null pointer, and the column length value was not 0, SQL_DATA_AT_EXEC, SQL_COLUMN_IGNORE, SQL_NULL_DATA, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.</p> <p>The <i>Operation</i> argument was SQL_UPDATE; a data value was not a null pointer; the C data type was SQL_C_BINARY or SQL_C_CHAR; and the column length value was less than 0 but not equal to SQL_DATA_AT_EXEC, SQL_COLUMN_IGNORE, SQL_NTS, or SQL_NULL_DATA, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET.</p> <p>The value in a length/indicator buffer was SQL_DATA_AT_EXEC; the SQL type was either SQL_LONGVARCHAR, SQL_LONGVARBINARY, or a long data source-specific data type; and the SQL_NEED_LONG_DATA_LEN information type in SQLGetInfo was "Y".</p>
HY092	Invalid attribute identifier	<p>(DM) The value specified for the <i>Operation</i> argument was invalid.</p> <p>(DM) The value specified for the <i>LockType</i> argument was invalid.</p> <p>The <i>Operation</i> argument was SQL_UPDATE or SQL_DELETE, and the SQL_ATTR_CONCURRENCY statement attribute was SQL_ATTR_CONCUR_READ_ONLY.</p>
HY107	Row value out of range	The value specified for the argument <i>RowNumber</i> was greater than the number of rows in the rowset.
HY109	Invalid cursor position	<p>The cursor associated with the <i>StatementHandle</i> was defined as forward-only, so the cursor could not be positioned within the rowset. See the description for the SQL_ATTR_CURSOR_TYPE attribute in SQLSetStmtAttr.</p> <p>The <i>Operation</i> argument was SQL_UPDATE, SQL_DELETE, or SQL_REFRESH, and the row identified by the <i>RowNumber</i> argument had been deleted or had not been fetched.</p> <p>(DM) The <i>RowNumber</i> argument was 0, and the <i>Operation</i> argument was SQL_POSITION.</p>

		SQLSetPos was called after SQLBulkOperations was called and before SQLFetchScroll or SQLFetch was called.
HYC00	Optional feature not implemented	The driver or data source does not support the operation requested in the <i>Operation</i> argument or the <i>LockType</i> argument.
HYT00	Timeout expired	The query timeout period expired before the data source returned the result set. The timeout period is set through SQLSetStmtAttr with an <i>Attribute</i> of SQL_ATTR_QUERY_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

Caution For information on the statement states that **SQLSetPos** can be called in and what it needs to do for compatibility with ODBC 2.x applications, see the "Block Cursors, Scrollable Cursors, and Backward Compatibility" (Appendix G, "Driver Guidelines for Backward Compatibility") contained on the Microsoft Web site (ODBC Programmer's Reference).

RowNumber Argument

The *RowNumber* argument specifies the number of the row in the rowset on which to perform the operation specified by the *Operation* argument. If *RowNumber* is 0, the operation applies to every row in the rowset. *RowNumber* must be a value from 0 to the number of rows in the rowset.

Note In the C language, arrays are 0-based and the *RowNumber* 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 a *RowNumber* of 5.

All operations position the cursor on the row specified by *RowNumber*. 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 *RowNumber* is 2 for a call to **SQLSetPos** with an *Operation* of SQL_DELETE, the cursor is positioned on the second row of the rowset and that row is deleted. The entry in the implementation row status array (pointed to by the SQL_ATTR_ROW_STATUS_PTR statement attribute) for the second row is changed to SQL_ROW_DELETED.

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**.

Operation Argument

The *Operation* argument supports the following operations. To determine which options are supported by a data source, an application calls **SQLGetInfo** with the SQL_DYNAMIC_CURSOR_ATTRIBUTES1, SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1, SQL_KEYSET_CURSOR_ATTRIBUTES1, or SQL_STATIC_CURSOR_ATTRIBUTES1 information type (depending on the type of the cursor).

<i>Operation</i> argument	Operation
SQL_POSITION	<p>The driver positions the cursor on the row specified by <i>RowNumber</i>.</p> <p>The contents of the row status array pointed to by the SQL_ATTR_ROW_OPERATION_PTR statement attribute are ignored for the SQL_POSITION <i>Operation</i>.</p>
SQL_REFRESH	<p>The driver positions the cursor on the row specified by <i>RowNumber</i> 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 SQLBindCol.</p> <p>SQLSetPos with an <i>Operation</i> of SQL_REFRESH updates the status and content of the rows within the current fetched rowset. This includes refreshing the bookmarks. Because the data in the buffers is refreshed but not refetched, the membership in the rowset is fixed. This is different from the refresh performed by a call to SQLFetchScroll with a <i>FetchOrientation</i> of SQL_FETCH_RELATIVE and a <i>RowNumber</i> equal to 0, which refetches the rowset from the result set so that it can show added data and remove deleted data if those operations are supported by the driver and the cursor.</p> <p>A successful refresh with SQLSetPos will not change a row status of SQL_ROW_DELETED. Deleted rows within the rowset will continue to be marked as deleted until the next fetch. The rows will disappear at the next fetch if the cursor supports packing (in which a subsequent SQLFetch or SQLFetchScroll does not return deleted rows).</p> <p>Added rows do not appear when a refresh with SQLSetPos is performed. This behavior is different from SQLFetchScroll with a <i>FetchType</i> of SQL_FETCH_RELATIVE and a <i>RowNumber</i> equal to 0, which also refreshes the current rowset but will show added records or pack deleted records if these operations are supported by the cursor.</p> <p>A successful refresh with SQLSetPos will change a row status of SQL_ROW_ADDED to SQL_ROW_SUCCESS (if the row status array exists).</p> <p>A successful refresh with SQLSetPos will change a row status of SQL_ROW_UPDATED to the row's new status (if the row status array exists).</p> <p>If an error occurs in a SQLSetPos operation on a row, the row status is set to SQL_ROW_ERROR (if the row status array exists).</p> <p>For a cursor opened with an SQL_ATTR_CONCURRENCY statement attribute of SQL_CONCUR_ROWVER or SQL_CONCUR_VALUES, a refresh with SQLSetPos might update the optimistic concurrency values used by the data source to detect that the row has changed. If this occurs, the row versions or values used to ensure cursor concurrency are updated whenever the rowset buffers are refreshed from the server. This occurs for each row that is refreshed.</p>

	<p>The contents of the row status array pointed to by the <code>SQL_ATTR_ROW_OPERATION_PTR</code> statement attribute are ignored for the <code>SQL_REFRESH Operation</code>.</p>
<code>SQL_UPDATE</code>	<p>The driver positions the cursor on the row specified by <i>RowNumber</i> and updates the underlying row of data with the values in the rowset buffers (the <i>TargetValuePtr</i> argument in SQLBindCol). It retrieves the lengths of the data from the length/indicator buffers (the <i>StrLen_or_IndPtr</i> argument in SQLBindCol). If the length of any column is <code>SQL_COLUMN_IGNORE</code>, the column is not updated. After updating the row, the driver changes the corresponding element of the row status array to <code>SQL_ROW_UPDATED</code> or <code>SQL_ROW_SUCCESS_WITH_INFO</code> (if the row status array exists).</p> <p>It is driver-defined what the behavior is if SQLSetPos with an <i>Operation</i> argument of <code>SQL_UPDATE</code> is called on a cursor that contains duplicate columns. The driver can return a driver-defined <code>SQLSTATE</code>, can update the first column that appears in the result set, or perform other driver-defined behavior.</p> <p>The row operation array pointed to by the <code>SQL_ATTR_ROW_OPERATION_PTR</code> statement attribute can be used to indicate that a row in the current rowset should be ignored during a bulk update. For more information, see "Status and Operation Arrays" later in this function reference.</p>
<code>SQL_DELETE</code>	<p>The driver positions the cursor on the row specified by <i>RowNumber</i> and deletes the underlying row of data. It changes the corresponding element of the row status array to <code>SQL_ROW_DELETED</code>. After the row has been deleted, the following are not valid for the row: positioned update and delete statements, calls to SQLGetData, and calls to SQLSetPos with <i>Operation</i> set to anything except <code>SQL_POSITION</code>. For drivers that support packing, the row is deleted from the cursor when new data is retrieved from the data source.</p> <p>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.</p> <p>The row operation array pointed to by the <code>SQL_ATTR_ROW_OPERATION_PTR</code> statement attribute can be used to indicate that a row in the current rowset should be ignored during a bulk delete. For more information, see "Status and Operation Arrays" later in this function reference.</p>

***LockType* Argument**

The *LockType* argument provides a way for applications to control concurrency. In most cases, data sources that support concurrency levels and transactions will support only the `SQL_LOCK_NO_CHANGE` value of the *LockType* argument. The *LockType* argument is generally used only for file-based support.

The *LockType* argument specifies the lock state of the row after **SQLSetPos** has been executed. If the driver is unable to lock the row either to perform the requested operation or to satisfy the *LockType* argument, it returns `SQL_ERROR` and `SQLSTATE 42000` (Syntax error or access violation).

Although the *LockType* argument is specified for a single statement, the lock accords the same privileges to all statements on the connection. In particular, a lock that is acquired by one statement on a connection can be unlocked by a different statement on the same connection.

A row locked through **SQLSetPos** remains locked until the application calls **SQLSetPos** for the row with *LockType* set to SQL_LOCK_UNLOCK, or until the application calls **SQLFreeHandle** for the statement or **SQLFreeStmt** with the SQL_CLOSE option. For a driver that supports transactions, a row locked through **SQLSetPos** is unlocked when the application calls **SQLEndTran** to commit or roll back a transaction on the connection (if a cursor is closed when a transaction is committed or rolled back, as indicated by the SQL_CURSOR_COMMIT_BEHAVIOR and SQL_CURSOR_ROLLBACK_BEHAVIOR information types returned by **SQLGetInfo**).

The *LockType* argument supports the following types of locks. To determine which locks are supported by a data source, an application calls **SQLGetInfo** with the SQL_DYNAMIC_CURSOR_ATTRIBUTES1, SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1, SQL_KEYSET_CURSOR_ATTRIBUTES1, or SQL_STATIC_CURSOR_ATTRIBUTES1 information type (depending on the type of the cursor).

<i>LockType</i> 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 <i>LockType</i> 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.
SQL_LOCK_EXCLUSIVE	The driver or data source locks the row exclusively. A statement on a different connection or in a different application cannot be used to acquire any locks on the row.
SQL_LOCK_UNLOCK	The driver or data source unlocks the row.

If a driver supports SQL_LOCK_EXCLUSIVE but does not support SQL_LOCK_UNLOCK, a row that is locked will remain locked until one of the function calls described in the previous paragraph occurs.

If a driver supports SQL_LOCK_EXCLUSIVE but does not support SQL_LOCK_UNLOCK, a row that is locked will remain locked until the application calls **SQLFreeHandle** for the statement or **SQLFreeStmt** with the SQL_CLOSE option. If the driver supports transactions and closes the cursor upon committing or rolling back the transaction, the application calls **SQLEndTran**.

For the update and delete operations in **SQLSetPos**, the application uses the *LockType* argument as follows:

To guarantee that a row does not change after it is retrieved, an application calls **SQLSetPos** with *Operation* set to SQL_REFRESH and *LockType* set to SQL_LOCK_EXCLUSIVE.

If the application sets *LockType* to SQL_LOCK_NO_CHANGE, the driver guarantees that an update or delete operation will succeed only if the application specified SQL_CONCUR_LOCK for the SQL_ATTR_CONCURRENCY statement attribute.

If the application specifies SQL_CONCUR_ROWVER or SQL_CONCUR_VALUES for the SQL_ATTR_CONCURRENCY statement attribute, 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_ATTR_CONCURRENCY statement attribute, the driver rejects any update or delete operation.

For more information about the SQL_ATTR_CONCURRENCY statement attribute, see [SQLSetStmtAttr](#).

Status and Operation Arrays

The following status and operation arrays are used when calling **SQLSetPos**:

The row status array (as pointed to by the `SQL_DESC_ARRAY_STATUS_PTR` field in the IRD and the `SQL_ATTR_ROW_STATUS_ARRAY` statement attribute) contains status values for each row of data in the rowset. The driver sets the status values in this array after a call to **SQLFetch**, **SQLFetchScroll**, **SQLBulkOperations**, or **SQLSetPos**. This array is pointed to by the `SQL_ATTR_ROW_STATUS_PTR` statement attribute.

The row operation array (as pointed to by the `SQL_DESC_ARRAY_STATUS_PTR` field in the ARD and the `SQL_ATTR_ROW_OPERATION_ARRAY` statement attribute) contains a value for each row in the rowset that indicates whether a call to **SQLSetPos** for a bulk operation is ignored or performed. Each element in the array is set to either `SQL_ROW_PROCEED` (the default) or `SQL_ROW_IGNORE`. This array is pointed to by the `SQL_ATTR_ROW_OPERATION_PTR` statement attribute.

The number of elements in the status and operation arrays must equal the number of rows in the rowset (as defined by the `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute).

For information about the row status array, see [SQLFetch](#). For information about the row operation array, see "Ignoring a Row in a Bulk Operation," later in this section.

Using SQLSetPos

Before an application calls **SQLSetPos**, it must perform the following sequence of steps:

If the application will call **SQLSetPos** with *Operation* set to `SQL_UPDATE`, call **SQLBindCol** (or **SQLSetDescRec**) for each column to specify its data type and bind buffers for the column's data and length.

If the application will call **SQLSetPos** with *Operation* set to `SQL_DELETE` or `SQL_UPDATE`, call **SQLColAttribute** to make sure that the columns to be deleted or updated are updatable.

Call **SQLExecDirect**, **SQLExecute**, or a catalog function to create a result set.

Call **SQLFetch** or **SQLFetchScroll** to retrieve the data.

For more information about using **SQLSetPos**, see the Part I PDF file, "Updating Data with SQLSetPos" in Chapter 12, "Updating Data."

Deleting Data Using SQLSetPos

To delete data with **SQLSetPos**, an application calls **SQLSetPos** with *RowNumber* set to the number of the row to delete and *Operation* set to `SQL_DELETE`.

After the data has been deleted, the driver changes the value in the implementation row status array for the appropriate row to `SQL_ROW_DELETED` (or `SQL_ROW_ERROR`).

Updating Data Using SQLSetPos

An application can pass the value for a column either in the bound data 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 data with **SQLSetPos**, an application:

Places values in the data and length/indicator buffers bound with **SQLBindCol**:

For normal columns, the application places the new column value in the **TargetValuePtr* buffer and the length of that value in the **StrLen_or_IndPtr* buffer. If the row should not be updated, the application places `SQL_ROW_IGNORE` in that row's element of the row operation array.

For data-at-execution columns, the application places an application-defined value, such as the column number, in the **TargetValuePtr* buffer. The value can be used later to identify the column.

The application places the result of the `SQL_LEN_DATA_AT_EXEC(length)` macro in the `*StrLen_or_IndPtr` buffer. If the SQL data type of the column is `SQL_LONGVARIABLE`, `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 non-negative value and is ignored.

Calls **SQLSetPos** with the *Operation* argument set to `SQL_UPDATE` to update 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` and proceeds to step 3.

Calls **SQLParamData** to retrieve the address of the `*TargetValuePtr` buffer for the first data-at-execution column to be processed. **SQLParamData** returns `SQL_NEED_DATA`. The application retrieves the application-defined value from the `*TargetValuePtr` 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** or by setting descriptors with **SQLSetDescRec**. The value returned by **SQLParamData** is a 32-bit value passed to **SQLBindParameter** in the *ParameterValuePtr* argument.

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

Calls **SQLPutData** one or more times to send data for the column. More than one call is needed if all the data values cannot be returned in the `*TargetValuePtr` buffer specified in **SQLPutData**; 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.

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 `TargetValuePtr` 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**.

If data has been updated, the driver changes the value in the implementation row status array for the appropriate row to `SQL_ROW_UPDATED`.

If the operation is canceled or an error occurs in **SQLParamData** or **SQLPutData**, after **SQLSetPos** returns `SQL_NEED_DATA` and before data is sent for all data-at-execution columns, the application can call only **SQLCancel**, **SQLGetDiagField**, **SQLGetDiagRec**, **SQLGetFunctions**, **SQLParamData**, or **SQLPutData** for the statement or the connection associated with the statement. If it calls any other function for the statement or the connection associated with the statement, the function returns `SQL_ERROR` and `SQLSTATE HY010` (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.

When the SELECT-list of the query specification associated with the cursor contains more than one reference to the same column, whether an error is generated or the driver ignores the duplicated references and performs the requested operations is driver-defined.

Performing Bulk Operations

If the *RowNumber* argument is 0, the driver performs the operation specified in the *Operation* argument for every row in the rowset that has a value of `SQL_ROW_PROCEED` in its field in the row operation array pointed to by `SQL_ATTR_ROW_OPERATION_PTR` statement attribute. This is a valid value of the *RowNumber* argument for an *Operation* argument of `SQL_DELETE`, `SQL_REFRESH`, or `SQL_UPDATE`, but not `SQL_POSITION`. **SQLSetPos** with an *Operation* of `SQL_POSITION` and a *RowNumber* equal to 0 will return `SQLSTATE HY109` (Invalid cursor position).

If an error occurs that pertains to the entire rowset, such as `SQLSTATE HYT00` (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 for the row in the row status array pointed to by the `SQL_ATTR_ROW_STATUS_PTR` statement attribute to `SQL_ROW_ERROR`.

Posts one or more additional `SQLSTATE`s for the error in the error queue and sets the `SQL_DIAG_ROW_NUMBER` field in the diagnostic data structure.

After it has processed the error or warning, if the driver completes the operation for the remaining rows in the rowset, it returns `SQL_SUCCESS_WITH_INFO`. Thus, for each row that returned an error, the error queue contains zero or more additional `SQLSTATE`s. If the driver stops the operation after it has processed the error or warning, it returns `SQL_ERROR`.

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.

If *RowNumber* is equal to 0 and *Operation* is `SQL_UPDATE`, `SQL_REFRESH`, or `SQL_DELETE`, the number of rows that **SQLSetPos** operates on is pointed to by the `SQL_ATTR_ROWS_FETCHED_PTR` statement attribute.

If *RowNumber* is equal to 0 and *Operation* is `SQL_DELETE`, `SQL_REFRESH`, or `SQL_UPDATE`, the current row after the operation is the same as the current row before the operation.

Ignoring a Row in a Bulk Operation

The row operation array can be used to indicate that a row in the current rowset should be ignored during a bulk operation using **SQLSetPos**. To direct the driver to ignore one or more rows during a bulk operation, an application should perform the following steps:

Call **SQLSetStmtAttr** to set the `SQL_ATTR_ROW_OPERATION_PTR` statement attribute to point to an array of `SQLUSMALLINT`s. This field can also be set by calling **SQLSetDescField** to set the `SQL_DESC_ARRAY_STATUS_PTR` header field of the ARD, which requires that an application obtains the descriptor handle.

Set each element of the row operation array to one of two values:

SQL_ROW_IGNORE, to indicate that the row is excluded for the bulk operation.

SQL_ROW_PROCEED, to indicate that the row is included in the bulk operation. (This is the default value.)

Call **SQLSetPos** to perform the bulk operation.

The following rules apply to the row operation array:

SQL_ROW_IGNORE and SQL_ROW_PROCEED affect only bulk operations using **SQLSetPos** with an *Operation* of SQL_DELETE or SQL_UPDATE. They do not affect calls to **SQLSetPos** with an *Operation* of SQL_REFRESH or SQL_POSITION.

The pointer is set to null by default.

If the pointer is null, all rows are updated as if all elements were set to SQL_ROW_PROCEED.

Setting an element to SQL_ROW_PROCEED does not guarantee that the operation will occur on that particular row. For example, if a certain row in the rowset has the status SQL_ROW_ERROR, the driver might not be able to update that row regardless of whether the application specified

SQL_ROW_PROCEED. An application must always check the row status array to see whether the operation was successful.

SQL_ROW_PROCEED is defined as 0 in the header file. An application can initialize the row operation array to 0 in order to process all rows.

If element number "n" in the row operation array is set to SQL_ROW_IGNORE and **SQLSetPos** is called to perform a bulk update or delete operation, the nth row in the rowset remains unchanged after the call to **SQLSetPos**.

An application should automatically set a read-only column to SQL_ROW_IGNORE.

Ignoring a Column in a Bulk Operation

To avoid unnecessary processing diagnostics generated by attempted updates to one or more read-only columns, an application can set the value in the bound length/indicator buffer to SQL_COLUMN_IGNORE. For more information, see [SQLBindCol](#).

Code Example

In the following example, an application allows a user to browse the ORDERS table and update order status. 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 it and allows the user to select and update the status of an order. 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 STATUS_LEN 6

SQLCHAR    szStatus[ROWS][STATUS_LEN], szReply[3];
SQLINTEGERcbStatus[ROWS], cbOrderID;
SQLUSMALLINT  rgfRowStatus[ROWS];
SQLUINTEGER   sOrderID, crow = ROWS, irow;
SQLHSTMT  hstmtS, hstmtU;

SQLSetStmtAttr(hstmtS, SQL_ATTR_CONCURRENCY, (SQLPOINTER) SQL_CONCUR_ROWVER,
0);
SQLSetStmtAttr(hstmtS, SQL_ATTR_CURSOR_TYPE, (SQLPOINTER)
SQL_CURSOR_KEYSET_DRIVEN, 0);
SQLSetStmtAttr(hstmtS, SQL_ATTR_ROW_ARRAY_SIZE, (SQLPOINTER) ROWS, 0);
SQLSetStmtAttr(hstmtS, SQL_ATTR_ROW_STATUS_PTR, (SQLPOINTER) rgfRowStatus, 0);
SQLSetCursorName(hstmtS, "C1", SQL_NTS);
SQLExecDirect(hstmtS, "SELECT ORDERID, STATUS FROM ORDERS ", SQL_NTS);
```



```

SQLBindCol(hstmtS, 1, SQL_C_ULONG, &sOrderID, 0, &cbOrderID);
SQLBindCol(hstmtS, 2, SQL_C_CHAR, szStatus, STATUS_LEN, &cbStatus);

while ((retcode == SQLFetchScroll(hstmtS, SQL_FETCH_NEXT, 0)) != SQL_ERROR) {
    if (retcode == SQL_NO_DATA_FOUND)
        break;
    for (irow = 0; irow < crow; irow++) {
        if (rgfRowStatus[irow] != SQL_ROW_DELETED)
            printf("%2d %5d %*s\n", irow+1, sOrderID, NAME_LEN-1, szStatus[irow]);
        while (TRUE) {
            printf("\nRow number to update?");
            gets(szReply);
            irow = atoi(szReply);
            if (irow > 0 && irow <= crow) {
                printf("\nNew status?");
                gets(szStatus[irow-1]);
                SQLSetPos(hstmtS, irow, SQL_POSITION, SQL_LOCK_NO_CHANGE);
                SQLPrepare(hstmtU,
"UPDATE ORDERS SET STATUS=? WHERE CURRENT OF C1", SQL_NTS);
                SQLBindParameter(hstmtU, 1, SQL_PARAM_INPUT,
                SQL_C_CHAR, SQL_CHAR,
                STATUS_LEN, 0, szStatus[irow], 0, NULL);
                SQLExecute(hstmtU);
            } else if (irow == 0) {
                break;
            }
        }
    }
}

```

For more examples, see the Part 1 PDF file, “Positioned Update and Delete Statements” and “Updating Rows in the Rowset with SQLSetPos” in Chapter 12, “Updating Data.”

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Performing bulk operations that do not relate to the block cursor position	SQLBulkOperations
Canceling statement processing	SQLCancel
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Getting a single field of a descriptor	SQLGetDescField
Getting multiple fields of a descriptor	SQLGetDescRec
Setting a single field of a descriptor	SQLSetDescField
Setting multiple fields of a descriptor	SQLSetDescRec
Setting a statement attribute	SQLSetStmtAttr

SQLSetScrollOptions

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: Deprecated

Summary

In ODBC 3.x, the ODBC 2.0 function **SQLSetScrollOptions** has been replaced by calls to **SQLGetInfo** and **SQLSetStmtAttr**.

Note For more information about what the Driver Manager maps this function to when an ODBC 2.x application is working with an ODBC 3.x driver, see “Mapping Deprecated Functions (Appendix G, “Driver Guidelines for Backward Compatibility”) contained on the Microsoft Web site (ODBC Programming Reference).

Note When the Driver Manager maps **SQLSetScrollOptions** for an application working with an ODBC 3.x driver that does not support **SQLSetScrollOptions**, the Driver Manager sets the **SQL_ROWSET_SIZE** statement option, not the **SQL_ATTR_ROW_ARRAY_SIZE** statement attribute, to the *RowsetSize* argument in **SQLSetScrollOption**. As a result, **SQLSetScrollOptions** cannot be used by an application when fetching multiple rows by a call to **SQLFetch** or **SQLFetchScroll**. It can be used only when fetching multiple rows by a call to **SQLExtendedFetch**.

SQLSetStmtAttr

Conformance

Version Introduced: ODBC 3.0
Standards Compliance: ISO 92

Summary

SQLSetStmtAttr sets attributes related to a statement.

Note For more information about what the Driver Manager maps this function to when an ODBC 3.x application is working with an ODBC 2.x driver, see the Part 1 PDF file, “Mapping Replacement Functions for Backward Compatibility of Applications” in Chapter 17, “Programming Considerations.”

Syntax

SQLRETURN SQLSetStmtAttr(
SQLHSTMT	StatementHandle,
SQLINTEGER	Attribute,
SQLPOINTER	ValuePtr,
SQLINTEGER	StringLength);

Arguments

StatementHandle

[Input]
Statement handle.

Attribute

[Input]

Option to set, listed in "Comments."

ValuePtr

[Input]

Pointer to the value to be associated with *Attribute*. Depending on the value of *Attribute*, *ValuePtr* will be a 32-bit unsigned integer value or a pointer to a null-terminated character string, a binary buffer, or a driver-defined value. If the *Attribute* argument is a driver-specific value, *ValuePtr* may be a signed integer.

StringLength

[Input]

If *Attribute* is an ODBC-defined attribute and *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **ValuePtr*. If *Attribute* is an ODBC-defined attribute and *ValuePtr* is an integer, *StringLength* is ignored.

If *Attribute* is a driver-defined attribute, the application indicates the nature of the attribute to the Driver Manager by setting the *StringLength* argument. *StringLength* can have the following values:

If *ValuePtr* is a pointer to a character string, then *StringLength* is the length of the string or SQL_NTS.

If *ValuePtr* is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(*length*) macro in *StringLength*. This places a negative value in *StringLength*. If *ValuePtr* is a pointer to a value other than a character string or a binary string, then *StringLength* should have the value SQL_IS_POINTER.

If *ValuePtr* contains a fixed-length value, then *StringLength* is either SQL_IS_INTEGER or SQL_IS_UINTEGER, as appropriate.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

Diagnostics

When **SQLSetStmtAttr** returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. The following table lists the SQLSTATE values commonly returned by **SQLSetStmtAttr** 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 value specified in <i>ValuePtr</i> , or the value specified in <i>ValuePtr</i> was invalid because of implementation working conditions, so the driver substituted a similar value. (SQLGetStmtAttr can be called to determine the temporarily substituted value.) The substitute value is valid for the <i>StatementHandle</i> until the cursor is closed, at which point the statement attribute reverts to its previous value. The statement attributes that can be

		<p>changed are:</p> <p>SQL_ATTR_CONCURRENCY SQL_ATTR_CURSOR_TYPE SQL_ATTR_KEYSET_SIZE SQL_ATTR_MAX_LENGTH SQL_ATTR_MAX_ROWS SQL_ATTR_QUERY_TIMEOUT SQL_ATTR_ROW_ARRAY_SIZE SQL_ATTR_SIMULATE_CURSOR</p> <p>(Function returns SQL_SUCCESS_WITH_INFO.)</p>
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 <i>Attribute</i> was SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_SIMULATE_CURSOR, or SQL_ATTR_USE_BOOKMARKS, and the cursor was open.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY009	Invalid use of null pointer	The <i>Attribute</i> argument identified a statement attribute that required a string attribute, and the <i>ValuePtr</i> argument was a null pointer.
HY010	Function sequence error	<p>(DM) An asynchronously executing function was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY011	Attribute cannot be set now	The <i>Attribute</i> was SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_SIMULATE_CURSOR, or SQL_ATTR_USE_BOOKMARKS, and the statement was prepared.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY017	Invalid use of an automatically allocated	(DM) The <i>Attribute</i> argument was SQL_ATTR_IMP_ROW_DESC or

	descriptor handle	SQL_ATTR_IMP_PARAM_DESC. (DM) The <i>Attribute</i> argument was SQL_ATTR_APP_ROW_DESC or SQL_ATTR_APP_PARAM_DESC, and the value in <i>ValuePtr</i> was an implicitly allocated descriptor handle other than the handle originally allocated for the ARD or APD.
HY024	Invalid attribute value	Given the specified <i>Attribute</i> value, an invalid value was specified in <i>ValuePtr</i> . (The Driver Manager returns this SQLSTATE only for connection and statement attributes that accept a discrete set of values, such as SQL_ATTR_ACCESS_MODE or SQL_ATTR_ASYNC_ENABLE. For all other connection and statement attributes, the driver must verify the value specified in <i>ValuePtr</i> .) The <i>Attribute</i> argument was SQL_ATTR_APP_ROW_DESC or SQL_ATTR_APP_PARAM_DESC, and <i>ValuePtr</i> was an explicitly allocated descriptor handle that is not on the same connection as the <i>StatementHandle</i> argument.
HY090	Invalid string or buffer length	(DM) * <i>ValuePtr</i> is a character string, and the <i>StringLength</i> argument was less than 0 but was not SQL_NTS.
HY092	Invalid attribute/option identifier	(DM) The value specified for the argument <i>Attribute</i> was not valid for the version of ODBC supported by the driver. (DM) The value specified for the argument <i>Attribute</i> was a read-only attribute.
HYC00	Optional feature not implemented	The value specified for the argument <i>Attribute</i> was a valid ODBC statement attribute for the version of ODBC supported by the driver but was not supported by the driver. The <i>Attribute</i> argument was SQL_ATTR_ASYNC_ENABLE, and a call to SQLGetInfo with an <i>InfoType</i> of SQL_ASYNC_MODE returns SQL_AM_CONNECTION. The <i>Attribute</i> argument was SQL_ATTR_ENABLE_AUTO_IPD, and the value of the connection attribute SQL_ATTR_AUTO_IPD was SQL_FALSE.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support	(DM) The driver associated with the <i>StatementHandle</i>

	this function	does not support the function.
--	---------------	--------------------------------

Comments

Statement attributes for a statement remain in effect until they are changed by another call to **SQLSetStmtAttr** or until the statement is dropped by calling **SQLFreeHandle**. Calling **SQLFreeStmt** with the SQL_CLOSE, SQL_UNBIND, or SQL_RESET_PARAMS option does not reset statement attributes.

Some statement attributes support substitution of a similar value if the data source does not support the value specified in *ValuePtr*. In such cases, the driver returns SQL_SUCCESS_WITH_INFO and SQLSTATE 01S02 (Option value changed). For example, if *Attribute* is SQL_ATTR_CONCURRENCY and *ValuePtr* is SQL_CONCUR_ROWVER, and if the data source does not support this, the driver substitutes SQL_CONCUR_VALUES and returns SQL_SUCCESS_WITH_INFO. To determine the substituted value, an application calls **SQLGetStmtAttr**.

The format of information set with *ValuePtr* depends on the specified *Attribute*. **SQLSetStmtAttr** accepts attribute information in one of two different formats: a character string or a 32-bit integer value. The format of each is noted in the attribute's description. This format applies to the information returned for each attribute in **SQLGetStmtAttr**. Character strings pointed to by the *ValuePtr* argument of **SQLSetStmtAttr** have a length of *StringLength*.

Note The ability to set statement attributes at the connection level by calling **SQLSetConnectAttr** has been deprecated in ODBC 3.x. ODBC 3.x applications should never set statement attributes at the connection level. ODBC 3.x statement attributes cannot be set at the connection level, with the exception of the SQL_ATTR_METADATA_ID and SQL_ATTR_ASYNC_ENABLE attributes, which are both connection attributes and statement attributes, and can be set at either the connection level or the statement level.

ODBC 3.x drivers need only support this functionality if they should work with ODBC 2.x applications that set ODBC 2.x statement options at the connection level. For more information, see "Setting Statement Options on the Connection Level" under "SQLSetConnectOptionMapping" (Appendix G, "Driver Guidelines for Backward Compatibility") contained on the Microsoft Web site (ODBC Programming Reference).

Statement Attributes That Set Descriptor Fields

Many statement attributes correspond to a header field of a descriptor. Setting these attributes actually results in the setting of the descriptor fields. Setting fields by a call to **SQLSetStmtAttr** rather than to **SQLSetDescField** has the advantage that a descriptor handle does not have to be obtained for the function call.

Caution Calling **SQLSetStmtAttr** for one statement can affect other statements. This occurs when the APD or ARD associated with the statement is explicitly allocated and is also associated with other statements. Because **SQLSetStmtAttr** modifies the APD or ARD, the modifications apply to all statements with which this descriptor is associated. If this is not the required behavior, the application should dissociate this descriptor from the other statements (by calling **SQLSetStmtAttr** to set the SQL_ATTR_APP_ROW_DESC or SQL_ATTR_APP_PARAM_DESC field to a different descriptor handle) before calling **SQLSetStmtAttr** again.

When a descriptor field is set as a result of the corresponding statement attribute being set, the field is set only for the applicable descriptors that are currently associated with the statement identified by the

StatementHandle argument, and the attribute setting does not affect any descriptors that may be associated with that statement in the future. When a descriptor field that is also a statement attribute is set by a call to **SQLSetDescField**, the corresponding statement attribute is set. If an explicitly allocated descriptor is dissociated from a statement, a statement attribute that corresponds to a header field will revert to the value of the field in the implicitly allocated descriptor.

When a statement is allocated (see [SQLAllocHandle](#)), four descriptor handles are automatically allocated and associated with the statement. Explicitly allocated descriptor handles can be associated with the statement by calling **SQLAllocHandle** with an *fHandleType* of `SQL_HANDLE_DESC` to allocate a descriptor handle and then calling **SQLSetStmtAttr** to associate the descriptor handle with the statement.

The statement attributes in the following table correspond to descriptor header fields.

Statement attribute	Header field	Desc.
SQL_ATTR_PARAM_BIND_OFFSET_PTR	SQL_DESC_BIND_OFFSET_PTR	APD
SQL_ATTR_PARAM_BIND_TYPE	SQL_DESC_BIND_TYPE	APD
SQL_ATTR_PARAM_OPERATION_PTR	SQL_DESC_ARRAY_STATUS_PTR	APD
SQL_ATTR_PARAM_STATUS_PTR	SQL_DESC_ARRAY_STATUS_PTR	IPD
SQL_ATTR_PARAMS_PROCESSED_PTR	SQL_DESC_ROWS_PROCESSED_PTR	IPD
SQL_ATTR_PARAMSET_SIZE	SQL_DESC_ARRAY_SIZE	APD
SQL_ATTR_ROW_ARRAY_SIZE	SQL_DESC_ARRAY_SIZE	ARD
SQL_ATTR_ROW_BIND_OFFSET_PTR	SQL_DESC_BIND_OFFSET_PTR	ARD
SQL_ATTR_ROW_BIND_TYPE	SQL_DESC_BIND_TYPE	ARD
SQL_ATTR_ROW_OPERATION_PTR	SQL_DESC_ARRAY_STATUS_PTR	ARD
SQL_ATTR_ROW_STATUS_PTR	SQL_DESC_ARRAY_STATUS_PTR	IRD
SQL_ATTR_ROWS_FETCHED_PTR	SQL_DESC_ROWS_PROCESSED_PTR	IRD

Statement Attributes

The currently defined attributes and the version of ODBC in which they were introduced are shown in the following table; it is expected that more attributes will be defined by drivers to take advantage of different data sources. A range of attributes is reserved by ODBC; driver developers must reserve values for their own driver-specific use from X/Open. For more information, see the Part 1 PDF file, "Driver-Specific Data Types, Descriptor Types, Information Types, Diagnostic Types, and Attributes" in Chapter 17, "Programming Considerations."

Attribute	<i>ValuePtr</i> contents
SQL_ATTR_APP_PARAM_DESC (ODBC 3.0)	The handle to the APD for subsequent calls to SQLExecute and SQLExecDirect on the statement handle. The initial value of this attribute is the descriptor implicitly allocated when the statement

	<p>was initially allocated. If the value of this attribute is set to <code>SQL_NULL_DESC</code> or the handle originally allocated for the descriptor, an explicitly allocated APD handle that was previously associated with the statement handle is dissociated from it and the statement handle reverts to the implicitly allocated APD handle.</p> <p>This attribute cannot be set to a descriptor handle that was implicitly allocated for another statement or to another descriptor handle that was implicitly set on the same statement; implicitly allocated descriptor handles cannot be associated with more than one statement or descriptor handle.</p>
SQL_ATTR_APP_ROW_DESC (ODBC 3.0)	<p>The handle to the ARD for subsequent fetches on the statement handle. The initial value of this attribute is the descriptor implicitly allocated when the statement was initially allocated. If the value of this attribute is set to <code>SQL_NULL_DESC</code> or the handle originally allocated for the descriptor, an explicitly allocated ARD handle that was previously associated with the statement handle is dissociated from it and the statement handle reverts to the implicitly allocated ARD handle.</p> <p>This attribute cannot be set to a descriptor handle that was implicitly allocated for another statement or to another descriptor handle that was implicitly set on the same statement; implicitly allocated descriptor handles cannot be associated with more than one statement or descriptor handle.</p>
SQL_ATTR_ASYNC_ENABLE (ODBC 1.0)	<p>An SQLINTEGER value that specifies whether a function called with the specified statement is executed asynchronously:</p> <p><code>SQL_ASYNC_ENABLE_OFF</code> = Off (the default) <code>SQL_ASYNC_ENABLE_ON</code> = On</p> <p>Once a function has been called asynchronously, only the original function, SQLCancel, SQLGetDiagField, or SQLGetDiagRec can be called on the statement, and only the original function, SQLAllocHandle (with a <i>HandleType</i> of <code>SQL_HANDLE_STMT</code>), SQLGetDiagField, SQLGetDiagRec, or SQLGetFunctions can be called on the connection associated with the statement, until the original function returns a code other than <code>SQL_STILL_EXECUTING</code>. Any other function called on the statement or the connection associated with the statement returns <code>SQL_ERROR</code> with an <code>SQLSTATE</code> of <code>HY010</code> (Function sequence error). Functions can be called on other statements. For more information, see the Part 1 PDF file, "Asynchronous Execution" in Chapter 9, "Executing Statements."</p> <p>For drivers with statement level asynchronous execution support, the statement attribute <code>SQL_ATTR_ASYNC_ENABLE</code> may be set. Its initial value is the same as the value of the connection level attribute with the same name at the time the statement handle was allocated.</p> <p>For drivers with connection-level, asynchronous-execution</p>

	<p>support, the statement attribute <code>SQL_ATTR_ASYNC_ENABLE</code> is read-only. Its value is the same as the value of the connection level attribute with the same name at the time the statement handle was allocated. Calling SQLSetStmtAttr to set <code>SQL_ATTR_ASYNC_ENABLE</code> when the <code>SQL_ASYNC_MODE</code> <i>InfoType</i> returns <code>SQL_AM_CONNECTION</code> returns <code>SQLSTATE HYC00</code> (Optional feature not implemented). (See SQLSetConnectAttr for more information.)</p> <p>As a standard practice, applications should execute functions asynchronously only on single-thread operating systems. On multithread operating systems, applications should execute functions on separate threads rather than executing them asynchronously on the same thread. No functionality is lost if drivers that operate only on multithread operating systems do not need to support asynchronous execution.</p> <p>The following functions can be executed asynchronously:</p> <ul style="list-style-type: none"> SQLBulkOperations SQLColAttribute SQLColumnPrivileges SQLColumns SQLCopyDesc SQLDescribeCol SQLDescribeParam SQLExecDirect SQLExecute SQLFetch SQLFetchScroll SQLForeignKeys SQLGetData SQLGetDescField [1] SQLGetDescRec [1] SQLGetDiagField SQLGetDiagRec SQLGetTypeInfo SQLMoreResults SQLNumParams SQLNumResultCols SQLParamData SQLPrepare SQLPrimaryKeys SQLProcedureColumns SQLProcedures SQLPutData SQLSetPos SQLSpecialColumns SQLStatistics SQLTablePrivileges SQLTables
--	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<p>SQL_ATTR_CONCURRENCY (ODBC 2.0)</p>	<p>An SQLINTEGER value that specifies the cursor concurrency:</p> <p>SQL_CONCUR_READ_ONLY = Cursor is read-only. No updates are allowed.</p> <p>SQL_CONCUR_LOCK = Cursor uses the lowest level of locking sufficient to ensure that the row can be updated.</p> <p>SQL_CONCUR_ROWVER = Cursor uses optimistic concurrency control, comparing row versions such as SQLBase ROWID or Sybase TIMESTAMP.</p> <p>SQL_CONCUR_VALUES = Cursor uses optimistic concurrency control, comparing values.</p> <p>The default value for SQL_ATTR_CONCURRENCY is SQL_CONCUR_READ_ONLY.</p> <p>This attribute cannot be specified for an open cursor. For more information, see the Part I PDF file, "Concurrency Types," in Chapter 14, "Transactions."</p> <p>If the SQL_ATTR_CURSOR_TYPE <i>Attribute</i> is changed to a type that does not support the current value of SQL_ATTR_CONCURRENCY, the value of SQL_ATTR_CONCURRENCY will be changed at execution time, and a warning issued when SQLExecDirect or SQLPrepare is called.</p> <p>If the driver supports the SELECT FOR UPDATE statement and such a statement is executed while the value of SQL_ATTR_CONCURRENCY is set to SQL_CONCUR_READ_ONLY, an error will be returned. If the value of SQL_ATTR_CONCURRENCY is changed to a value that the driver supports for some value of SQL_ATTR_CURSOR_TYPE but not for the current value of SQL_ATTR_CURSOR_TYPE, the value of SQL_ATTR_CURSOR_TYPE will be changed at execution time and SQLSTATE 01S02 (Option value changed) is issued when SQLExecDirect or SQLPrepare is called.</p> <p>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. The validity of the substituted value is not checked until execution time.</p> <p>For more information about the relationship between SQL_ATTR_CONCURRENCY and the other cursor attributes,</p>
--------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	see the Part I PDF file, “Cursor Characteristics and Cursor Types” in Chapter 11, “Retrieving Results (Advanced).”
SQL_ATTR_CURSOR_SCROLLABLE (ODBC 3.0)	<p>An SQLINTEGER value that specifies the level of support that the application requires. Setting this attribute affects subsequent calls to SQLExecDirect and SQLExecute.</p> <p>SQL_NONSCROLLABLE = Scrollable cursors are not required on the statement handle. If the application calls SQLFetchScroll on this handle, the only valid value of <i>FetchOrientation</i> is SQL_FETCH_NEXT. This is the default.</p> <p>SQL_SCROLLABLE = Scrollable cursors are required on the statement handle. When calling SQLFetchScroll, the application may specify any valid value of <i>FetchOrientation</i>, achieving cursor positioning in modes other than the sequential mode.</p> <p>For more information about scrollable cursors, see the Part I PDF file, “Scrollable Cursors” in Chapter 11, “Retrieving Results (Advanced).” For more information about the relationship between SQL_ATTR_CURSOR_SCROLLABLE and the other cursor attributes, see the Part I PDF file, “Cursor Characteristics and Cursor Types” in Chapter 11, “Retrieving Results (Advanced).”</p>
SQL_ATTR_CURSOR_SENSITIVITY (ODBC 3.0)	<p>An SQLINTEGER value that specifies whether cursors on the statement handle make visible the changes made to a result set by another cursor. Setting this attribute affects subsequent calls to SQLExecDirect and SQLExecute. An application can read back the value of this attribute to obtain its initial state or its state as most recently set by the application.</p> <p>SQL_UNSPECIFIED = It is unspecified what the cursor type is and whether cursors on the statement handle make visible the changes made to a result set by another cursor. Cursors on the statement handle may make visible none, some, or all such changes. This is the default.</p> <p>SQL_INSENSITIVE = All cursors on the statement handle show the result set without reflecting any changes made to it by any other cursor. Insensitive cursors are read-only. This corresponds to a static cursor, which has a concurrency that is read-only.</p> <p>SQL_SENSITIVE = All cursors on the statement handle make visible all changes made to a result set by another cursor.</p> <p>For more information about the relationship between SQL_ATTR_CURSOR_SENSITIVITY and the other cursor attributes, see the Part I PDF file, “Cursor Characteristics and Cursor Types” in Chapter 11, “Retrieving Results (Advanced).”</p>
SQL_ATTR_CURSOR_TYPE (ODBC 2.0)	<p>An SQLINTEGER value that specifies the cursor type:</p> <p>SQL_CURSOR_FORWARD_ONLY = The cursor only scrolls</p>

	<p>forward.</p> <p>SQL_CURSOR_STATIC = The data in the result set is static.</p> <p>SQL_CURSOR_KEYSET_DRIVEN = The driver saves and uses the keys for the number of rows specified in the SQL_ATTR_KEYSET_SIZE statement attribute.</p> <p>SQL_CURSOR_DYNAMIC = The driver saves and uses only the keys for the rows in the rowset.</p> <p>The default value is SQL_CURSOR_FORWARD_ONLY. This attribute cannot be specified after the SQL statement has been prepared.</p> <p>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.</p> <p>For more information about scrollable cursor types, see the Part I PDF file, "Scrollable Cursor Types" in Chapter 11, "Retrieving Results (Advanced)." For more information about the relationship between SQL_ATTR_CURSOR_TYPE and the other cursor attributes, see the Part I PDF file, "Cursor Characteristics and Cursor Type" in Chapter 11, "Retrieving Results (Advanced)."</p>
SQL_ATTR_ENABLE_AUTO_IPD (ODBC 3.0)	<p>An SQLINTEGER value that specifies whether automatic population of the IPD is performed:</p> <p>SQL_TRUE = Turns on automatic population of the IPD after a call to SQLPrepare.</p> <p>SQL_FALSE = Turns off automatic population of the IPD after a call to SQLPrepare. (An application can still obtain IPD field information by calling SQLDescribeParam, if supported.)</p> <p>The default value of the statement attribute SQL_ATTR_ENABLE_AUTO_IPD is equal to the value of the connection attribute SQL_ATTR_AUTO_IPD. If the connection attribute SQL_ATTR_AUTO_IPD is SQL_FALSE, the statement attribute SQL_ATTR_ENABLE_AUTO_IPD cannot be set to SQL_TRUE.</p> <p>For more information, see the Part I PDF file, "Automatic Population of the IPD" in Chapter 13, "Descriptors."</p>
SQL_ATTR_FETCH_BOOKMARK_PTR (ODBC 3.0)	<p>A pointer that points to a binary bookmark value. When SQLFetchScroll is called with <i>fFetchOrientation</i> equal to SQL_FETCH_BOOKMARK, the driver picks up the bookmark value from this field. This field defaults to a null pointer.</p> <p>The value pointed to by this field is not used for delete by</p>

	bookmark, update by bookmark, or fetch by bookmark operations in SQLBulkOperations , which use bookmarks cached in rowset buffers.
SQL_ATTR_IMP_PARAM_DESC (ODBC 3.0)	<p>The handle to the IPD. The value of this attribute is the descriptor allocated when the statement was initially allocated. The application cannot set this attribute.</p> <p>This attribute can be retrieved by a call to SQLGetStmtAttr but not set by a call to SQLSetStmtAttr.</p>
SQL_ATTR_IMP_ROW_DESC (ODBC 3.0)	<p>The handle to the IRD. The value of this attribute is the descriptor allocated when the statement was initially allocated. The application cannot set this attribute.</p> <p>This attribute can be retrieved by a call to SQLGetStmtAttr but not set by a call to SQLSetStmtAttr.</p>
SQL_ATTR_KEYSET_SIZE (ODBC 2.0)	<p>An SQLINTEGER 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. For more information about keyset-driven cursors, see the Part I PDF file, "Keyset-Driven Cursors" in Chapter 11, "Retrieving Results (Advanced)."</p> <p>If the specified size exceeds the maximum keyset size, the driver substitutes that size and returns SQLSTATE 01S02 (Option value changed).</p> <p>SQLFetch or SQLFetchScroll returns an error if the keyset size is greater than 0 and less than the rowset size.</p>
SQL_ATTR_MAX_LENGTH (ODBC 1.0)	<p>An SQLINTEGER value that specifies the maximum amount of data that the driver returns from a character or binary column. If <i>ValuePtr</i> is less than the length of the available data, SQLFetch or SQLGetData truncates the data and returns SQL_SUCCESS. If <i>ValuePtr</i> is 0 (the default), the driver attempts to return all available data.</p> <p>If the specified length is less than the minimum amount of data that the data source can return 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).</p> <p>The value of this attribute can be set on an open cursor; however, the setting might not take effect immediately, in which case the driver will return SQLSTATE 01S02 (Option value changed) and reset the attribute to its original value.</p> <p>This attribute is intended to reduce network traffic and should be supported only when the data source (as opposed to the driver) in a multiple-tier driver can implement it. This mechanism should</p>

	<p>not be used by applications to truncate data; to truncate data received, an application should specify the maximum buffer length in the <i>BufferLength</i> argument in SQLBindCol or SQLGetData.</p>
SQL_ATTR_MAX_ROWS (ODBC 1.0)	<p>An SQLINTEGER value corresponding to the maximum number of rows to return to the application for a SELECT statement. If *<i>ValuePtr</i> equals 0 (the default), the driver returns all rows.</p> <p>This attribute is intended to reduce network traffic. Conceptually, it is applied when the result set is created and limits the result set to the first <i>ValuePtr</i> rows. If the number of rows in the result set is greater than <i>ValuePtr</i>, the result set is truncated.</p> <p>SQL_ATTR_MAX_ROWS applies to all result sets on the <i>Statement</i>, including those returned by catalog functions. SQL_ATTR_MAX_ROWS establishes a maximum for the value of the cursor row count.</p> <p>A driver should not emulate SQL_ATTR_MAX_ROWS behavior for SQLFetch or SQLFetchScroll (if result set size limitations cannot be implemented at the data source) if it cannot guarantee that SQL_ATTR_MAX_ROWS will be implemented properly.</p> <p>It is driver-defined whether SQL_ATTR_MAX_ROWS applies to statements other than SELECT statements (such as catalog functions).</p> <p>The value of this attribute can be set on an open cursor; however, the setting might not take effect immediately, in which case the driver will return SQLSTATE 01S02 (Option value changed) and reset the attribute to its original value.</p>
SQL_ATTR_METADATA_ID (ODBC 3.0)	<p>An SQLINTEGER value that determines how the string arguments of catalog functions are treated.</p> <p>If SQL_TRUE, the string argument of catalog functions are treated as identifiers. The case is not significant. For nondelimited strings, the driver removes any trailing spaces and the string is folded to uppercase. For delimited strings, the driver removes any leading or trailing spaces and takes whatever is between the delimiters literally. If one of these arguments is set to a null pointer, the function returns SQL_ERROR and SQLSTATE HY009 (Invalid use of null pointer).</p> <p>If SQL_FALSE, the string arguments of catalog functions are not treated as identifiers. The case is significant. They can either contain a string search pattern or not, depending on the argument.</p> <p>The default value is SQL_FALSE.</p> <p>The <i>TableType</i> argument of SQLTables, which takes a list of values, is not affected by this attribute.</p>

	<p>SQL_ATTR_METADATA_ID can also be set on the connection level. (It and SQL_ATTR_ASYNC_ENABLE are the only statement attributes that are also connection attributes.)</p> <p>For more information, see the Part I PDF file “Arguments in Catalog Functions” in Chapter 7, "Catalog Functions."</p>
SQL_ATTR_NOSCAN (ODBC 1.0)	<p>An SQLINTEGER value that indicates whether the driver should scan SQL strings for escape sequences:</p> <p>SQL_NOSCAN_OFF = The driver scans SQL strings for escape sequences (the default).</p> <p>SQL_NOSCAN_ON = The driver does not scan SQL strings for escape sequences. Instead, the driver sends the statement directly to the data source.</p> <p>For more information, see the Part I PDF file, “Escape Sequences in ODBC” in Chapter 8, "SQL Statements."</p>
SQL_ATTR_PARAM_BIND_OFFSET_PTR (ODBC 3.0)	<p>An SQLINTEGER * value that points to an offset added to pointers to change binding of dynamic parameters. If this field is non-null, the driver dereferences the pointer, adds the dereferenced value to each of the deferred fields in the descriptor record (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR), and uses the new pointer values when binding. It is set to null by default.</p> <p>The bind offset is always added directly to the SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR fields. If the offset is changed to a different value, the new value is still added directly to the value in the descriptor field. The new offset is not added to the field value plus any earlier offsets.</p>
SQL_ATTR_PARAM_BIND_OFFSET_PTR (ODBC 3.0) <i>(continued)</i>	<p>For more information, see the Part I PDF file, “Parameter Binding Offsets” in Chapter 9, "Executing Statements."</p> <p>Setting this statement attribute sets the SQL_DESC_BIND_OFFSET_PTR field in the APD header.</p>
SQL_ATTR_PARAM_BIND_TYPE (ODBC 3.0)	<p>An SQLINTEGER value that indicates the binding orientation to be used for dynamic parameters.</p> <p>This field is set to SQL_PARAM_BIND_BY_COLUMN (the default) to select column-wise binding.</p> <p>To select row-wise binding, this field is set to the length of the structure or an instance of a buffer that will be bound to a set of dynamic parameters. This length must include space for all of the bound parameters and any padding of the structure or buffer to ensure that when the address of a bound parameter is incremented with the specified length, the result will point to the beginning of the same parameter in the next set of parameters. When using the</p>

	<p><i>sizeof</i> operator in ANSI C, this behavior is guaranteed.</p> <p>For more information, see the Part I PDF file, “Binding Arrays of Parameters” in Chapter 9, "Executing Statements."</p> <p>Setting this statement attribute sets the SQL_DESC_BIND_TYPE field in the APD header.</p>
SQL_ATTR_PARAM_OPERATION_PTR (ODBC 3.0)	<p>An SQLUSMALLINT * value that points to an array of SQLUSMALLINT values used to ignore a parameter during execution of an SQL statement. Each value is set to either SQL_PARAM_PROCEED (for the parameter to be executed) or SQL_PARAM_IGNORE (for the parameter to be ignored).</p> <p>A set of parameters can be ignored during processing by setting the status value in the array pointed to by SQL_DESC_ARRAY_STATUS_PTR in the APD to SQL_PARAM_IGNORE. A set of parameters is processed if its status value is set to SQL_PARAM_PROCEED or if no elements in the array are set.</p> <p>This statement attribute can be set to a null pointer, in which case the driver does not return parameter status values. This attribute can be set at any time, but the new value is not used until the next time SQLExecDirect or SQLExecute is called.</p> <p>For more information, see the Part I PDF file, “Using Arrays of Parameters” in Chapter 9, "Executing Statements."</p> <p>Setting this statement attribute sets the SQL_DESC_ARRAY_STATUS_PTR field in the APD header.</p>
SQL_ATTR_PARAM_STATUS_PTR (ODBC 3.0)	<p>An SQLUSMALLINT * value that points to an array of SQLUSMALLINT values containing status information for each row of parameter values after a call to SQLExecute or SQLExecDirect. This field is required only if PARAMSET_SIZE is greater than 1.</p> <p>The status values can contain the following values:</p> <p>SQL_PARAM_SUCCESS: The SQL statement was successfully executed for this set of parameters.</p> <p>SQL_PARAM_SUCCESS_WITH_INFO: The SQL statement was successfully executed for this set of parameters; however, warning information is available in the diagnostics data structure.</p> <p>SQL_PARAM_ERROR: There was an error in processing this set of parameters. Additional error information is available in the diagnostics data structure.</p> <p>SQL_PARAM_UNUSED: This parameter set was unused, possibly due to the fact that some previous parameter set caused an error that aborted further processing, or because</p>

	<p>SQL_PARAM_IGNORE was set for that set of parameters in the array specified by the SQL_ATTR_PARAM_OPERATION_PTR.</p> <p>SQL_PARAM_DIAG_UNAVAILABLE: The driver treats arrays of parameters as a monolithic unit and so does not generate this level of error information.</p> <p>This statement attribute can be set to a null pointer, in which case the driver does not return parameter status values. This attribute can be set at any time, but the new value is not used until the next time SQLExecute or SQLExecDirect is called. Note that setting this attribute can affect the output parameter behavior implemented by the driver.</p> <p>For more information, see the Part I PDF file, “Using Arrays of Parameters” in Chapter 9, "Executing Statements."</p> <p>Setting this statement attribute sets the SQL_DESC_ARRAY_STATUS_PTR field in the IPD header.</p>
SQL_ATTR_PARAMS_PROCESSED_PTR (ODBC 3.0)	<p>An SQLINTEGER * record field that points to a buffer in which to return the number of sets of parameters that have been processed, including error sets. No number will be returned if this is a null pointer.</p> <p>Setting this statement attribute sets the SQL_DESC_ROWS_PROCESSED_PTR field in the IPD header.</p> <p>If the call to SQLExecDirect or SQLExecute that fills in the buffer pointed to by this attribute does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the contents of the buffer are undefined.</p> <p>For more information, Part I PDF file, “Using Arrays of Parameters” in Chapter 9, "Executing Statements."</p>
SQL_ATTR_PARAMSET_SIZE (ODBC 3.0)	<p>An SQLINTEGER value that specifies the number of values for each parameter. If SQL_ATTR_PARAMSET_SIZE is greater than 1, SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR of the APD point to arrays. The cardinality of each array is equal to the value of this field.</p> <p>For more information, Part I PDF file, “Using Arrays of Parameters” in Chapter 9, "Executing Statements."</p> <p>Setting this statement attribute sets the SQL_DESC_ARRAY_SIZE field in the APD header.</p>
SQL_ATTR_QUERY_TIMEOUT (ODBC 1.0)	<p>An SQLINTEGER value corresponding to the number of seconds to wait for an SQL statement to execute before returning to the application. If <i>ValuePtr</i> is equal to 0 (default), there is no timeout.</p> <p>If the specified timeout exceeds the maximum timeout in the data</p>

	<p>source or is smaller than the minimum timeout, SQLSetStmtAttr substitutes that value and returns SQLSTATE 01S02 (Option value changed).</p> <p>Note that the application need not call SQLCloseCursor to reuse the statement if a SELECT statement timed out.</p> <p>The query timeout set in this statement attribute is valid in both synchronous and asynchronous modes.</p>
SQL_ATTR_RETRIEVE_DATA (ODBC 2.0)	<p>An SQLINTEGER value:</p> <p>SQL_RD_ON = SQLFetchScroll and, in ODBC 3.x, SQLFetch retrieve data after it positions the cursor to the specified location. This is the default.</p> <p>SQL_RD_OFF = SQLFetchScroll and, in ODBC 3.x, SQLFetch do not retrieve data after it positions the cursor.</p> <p>By setting SQL_RETRIEVE_DATA to SQL_RD_OFF, an application can verify that a row exists or retrieve a bookmark for the row without incurring the overhead of retrieving rows. For more information, see the Part I PDF file, "Scrolling and Fetching Rows" in Chapter 11, "Retrieving Results (Advanced)."</p> <p>The value of this attribute can be set on an open cursor; however, the setting might not take effect immediately, in which case the driver will return SQLSTATE 01S02 (Option value changed) and reset the attribute to its original value.</p>
SQL_ATTR_ROW_ARRAY_SIZE (ODBC 3.0)	<p>An SQLINTEGER value that specifies the number of rows returned by each call to SQLFetch or SQLFetchScroll. It is also the number of rows in a bookmark array used in a bulk bookmark operation in SQLBulkOperations. The default value is 1.</p> <p>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).</p> <p>For more information, see the Part I PDF file, "Rowset Size" in Chapter 11, "Retrieving Results (Advanced)."</p> <p>Setting this statement attribute sets the SQL_DESC_ARRAY_SIZE field in the ARD header.</p>
SQL_ATTR_ROW_BIND_OFFSET_PTR (ODBC 3.0)	<p>An SQLINTEGER * value that points to an offset added to pointers to change binding of column data. If this field is non-null, the driver dereferences the pointer, adds the dereferenced value to each of the deferred fields in the descriptor record (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR), and uses the new pointer values when binding. It is set to null by default.</p> <p>Setting this statement attribute sets the SQL_DESC_BIND_OFFSET_PTR field in the ARD header.</p>

<p>SQL_ATTR_ROW_BIND_TYPE (ODBC 1.0)</p>	<p>An SQLINTEGER value that sets the binding orientation to be used when SQLFetch or SQLFetchScroll is called on the associated statement. Column-wise binding is selected by setting the value to SQL_BIND_BY_COLUMN. Row-wise binding is selected by setting the value to the length of a structure or an instance of a buffer into which result columns will be bound.</p> <p>If a length is specified, it 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.</p> <p>Column-wise binding is the default binding orientation for SQLFetch and SQLFetchScroll.</p> <p>For more information, see the Part I PDF file, “Binding Columns for Use with Block Cursors” in Chapter 11, “Retrieving Results (Advanced).”</p> <p>Setting this statement attribute sets the SQL_DESC_BIND_TYPE field in the ARD header.</p>
<p>SQL_ATTR_ROW_NUMBER (ODBC 2.0)</p>	<p>An SQLINTEGER value that is 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.</p> <p>This attribute can be retrieved by a call to SQLGetStmtAttr but not set by a call to SQLSetStmtAttr.</p>
<p>SQL_ATTR_ROW_OPERATION_PTR (ODBC 3.0)</p>	<p>An SQLUSMALLINT * value that points to an array of SQLUSMALLINT values used to ignore a row during a bulk operation using SQLSetPos. Each value is set to either SQL_ROW_PROCEED (for the row to be included in the bulk operation) or SQL_ROW_IGNORE (for the row to be excluded from the bulk operation). (Rows cannot be ignored by using this array during calls to SQLBulkOperations.)</p> <p>This statement attribute can be set to a null pointer, in which case the driver does not return row status values. This attribute can be set at any time, but the new value is not used until the next time SQLSetPos is called.</p> <p>For more information, see the Part I PDF file, “Updating Rows in the Rowset with SQLSetPos” and “Deleting Rows in the Rowset with SQLSetPos” in Chapter 12, “Updating Data.”</p> <p>Setting this statement attribute sets the SQL_DESC_ARRAY_STATUS_PTR field in the ARD.</p>
<p>SQL_ATTR_ROW_STATUS_PTR (ODBC 3.0)</p>	<p>An SQLUSMALLINT * value that points to an array of SQLUSMALLINT values containing row status values after a call to SQLFetch or SQLFetchScroll. The array has as many elements as there are rows in the rowset.</p>

	<p>This statement attribute can be set to a null pointer, in which case the driver does not return row status values. This attribute can be set at any time, but the new value is not used until the next time SQLBulkOperations, SQLFetch, SQLFetchScroll, or SQLSetPos is called.</p> <p>For more information, see the Part I PDF file, “Number of Rows Fetched and Status” in Chapter 11, "Retrieving Results (Advanced)."</p> <p>Setting this statement attribute sets the SQL_DESC_ARRAY_STATUS_PTR field in the IRD header.</p> <p>This attribute is mapped by an ODBC 2.x driver to the <i>rgbRowStatus</i> array in a call to SQLExtendedFetch.</p>
SQL_ATTR_ROWS_FETCHED_PTR (ODBC 3.0)	<p>An SQLINTEGER * value that points to a buffer in which to return the number of rows fetched after a call to SQLFetch or SQLFetchScroll; the number of rows affected by a bulk operation performed by a call to SQLSetPos with an <i>Operation</i> argument of SQL_REFRESH; or the number of rows affected by a bulk operation performed by SQLBulkOperations. This number includes error rows.</p> <p>For more information, see the Part I PDF file, “Number of Rows Fetched and Status” in Chapter 11, "Retrieving Results (Advanced)."</p> <p>Setting this statement attribute sets the SQL_DESC_ROWS_PROCESSED_PTR field in the IRD header.</p> <p>If the call to SQLFetch or SQLFetchScroll that fills in the buffer pointed to by this attribute does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the contents of the buffer are undefined.</p>
SQL_ATTR_SIMULATE_CURSOR (ODBC 2.0)	<p>An SQLINTEGER value that specifies whether drivers that simulate positioned update and delete statements guarantee that such statements affect only one single row.</p> <p>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 make up a unique key, such a statement can affect more than one row.</p> <p>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 make up a unique key, the driver is not required to do so. This may reduce execution time.</p> <p>SQL_SC_NON_UNIQUE = The driver does not guarantee that simulated positioned update or delete statements will affect only</p>

	<p>one row; it is the application's responsibility to do so. If a statement affects more than one row, SQLExecute, SQLExecDirect, or SQLSetPos returns SQLSTATE 01001 (Cursor operation conflict).</p> <p>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, SQLExecDirect, or SQLSetPos returns SQLSTATE 01001 (Cursor operation conflict).</p> <p>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.</p> <p>If the data source provides native SQL support for positioned update and delete statements and the driver does not simulate cursors, SQL_SUCCESS is returned when SQL_SC_UNIQUE is requested for SQL_SIMULATE_CURSOR. SQL_SUCCESS_WITH_INFO is returned if SQL_SC_TRY_UNIQUE or SQL_SC_NON_UNIQUE is requested. If the data source provides the SQL_SC_TRY_UNIQUE level of support and the driver does not, SQL_SUCCESS is returned for SQL_SC_TRY_UNIQUE and SQL_SUCCESS_WITH_INFO is returned for SQL_SC_NON_UNIQUE.</p> <p>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.</p> <p>For more information, see the Part I PDF file, "Simulating Positioned Update and Delete Statements" in Chapter 12, "Updating Data."</p>
SQL_ATTR_USE_BOOKMARKS (ODBC 2.0)	<p>An SQLINTEGER value that specifies whether an application will use bookmarks with a cursor:</p> <p>SQL_UB_OFF = Off (the default)</p> <p>SQL_UB_VARIABLE = An application will use bookmarks with a cursor, and the driver will provide variable-length bookmarks if they are supported. SQL_UB_FIXED is deprecated in ODBC 3.x. ODBC 3.x applications should always use variable-length bookmarks, even when working with ODBC 2.x drivers (which supported only 4-byte, fixed-length bookmarks). This is because a</p>

	<p>fixed-length bookmark is just a special case of a variable-length bookmark. When working with an ODBC 2.x driver, the Driver Manager maps SQL_UB_VARIABLE to SQL_UB_FIXED.</p> <p>To use bookmarks with a cursor, the application must specify this attribute with the SQL_UB_VARIABLE value before opening the cursor.</p>
--	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

[1] These functions can be called asynchronously only if the descriptor is an implementation descriptor, not an application descriptor.

See the Part I PDF file, "Column-Wise Binding" and "Row-Wise Binding" in Chapter 11, "Retrieving Results (Advanced)."

Related Functions

For information about	See
Canceling statement processing	SQLCancel
Returning the setting of a connection attribute	SQLGetConnectAttr
Returning the setting of a statement attribute	SQLGetStmtAttr
Setting a connection attribute	SQLSetConnectAttr
Setting a single field of the descriptor	SQLSetDescField

SQLSetStmtOption

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: Deprecated

Summary

In ODBC 3.x, the ODBC 2.0 function **SQLSetStmtOption** has been replaced by **SQLSetStmtAttr**. For more information, see [SQLGetStmtAttr](#).

Note For more information about what the Driver Manager maps this function to when an ODBC 2.x application is working with an ODBC 3.x driver, see "Mapping Deprecated Functions (Appendix G, "Driver Guidelines for Backward Compatibility") contained on the Microsoft Web site (ODBC Programming Reference).

SQLSpecialColumns

Conformance

Version Introduced: ODBC 1.0
Standards Compliance: X/Open

Summary

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

SQLRETURN SQLSpecialColumns(
SQLHSTMT	StatementHandle,
SQLSMALLINT	IdentifierType,
SQLCHAR *	CatalogName,
SQLSMALLINT	NameLength1,
SQLCHAR *	SchemaName,
SQLSMALLINT	NameLength2,
SQLCHAR *	TableName,
SQLSMALLINT	NameLength3,
SQLSMALLINT	Scope,
SQLSMALLINT	Nullable);

Arguments

StatementHandle

[Input]

Statement handle.

IdentifierType

[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 pseudo-column 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).

CatalogName

[Input]

Catalog name for the table. If a driver supports catalogs 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 catalogs. *CatalogName* cannot contain a string search pattern.

If the **SQL_ATTR_METADATA_ID** statement attribute is set to **SQL_TRUE**, *CatalogName* is treated as an identifier and its case is not significant. If it is **SQL_FALSE**, *CatalogName* is an ordinary argument; it is

treated literally, and its case is significant. For more information, see Part I PDF file, “Arguments in Catalog Functions” in Chapter 7, “Catalog Functions.”

NameLength1

[Input]

Length of **CatalogName*.

SchemaName

[Input]

Schema name for the table. If a driver supports schemas 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 schemas. *SchemaName* cannot contain a string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *SchemaName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *SchemaName* is an ordinary argument; it is treated literally, and its case is significant.

NameLength2

[Input]

Length of **SchemaName*.

TableName

[Input]

Table name. This argument cannot be a null pointer. *TableName* cannot contain a string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *TableName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *TableName* is an ordinary argument; it is treated literally, and its case is significant.

NameLength3

[Input]

Length of **TableName*.

Scope

[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).

Nullable

[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. Some drivers cannot support **SQL_NO_NULLS**, and these drivers will return an empty result set if **SQL_NO_NULLS** was specified. Applications should be prepared for this case and request **SQL_NO_NULLS** only if it is absolutely required.

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 **SQLGetDiagRec** with a *HandleType* of **SQL_HANDLE_STMT** and a *Handle* of *StatementHandle*. 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	A cursor was open on the <i>StatementHandle</i> , and SQLFetch or SQLFetchScroll had been called. This error is returned by the Driver Manager if SQLFetch or SQLFetchScroll has not returned SQL_NO_DATA and is returned by the driver if SQLFetch or SQLFetchScroll has returned SQL_NO_DATA . A cursor was open on the <i>StatementHandle</i> , but SQLFetch or SQLFetchScroll had not been called.
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.

HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>. Then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY009	Invalid use of null pointer	<p>The <i>TableName</i> argument was a null pointer.</p> <p>(DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE, the <i>CatalogName</i> argument was a null pointer, and the SQL_CATALOG_NAME <i>InfoType</i> returns that catalog names are supported.</p> <p>(DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE, and the <i>SchemaName</i> argument was a null pointer.</p>
HY010	Function sequence error	<p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	<p>(DM) The value of one of the length arguments was less than 0 but not equal to SQL_NTS.</p> <p>The value of one of the length arguments exceeded the maximum length value for the corresponding name. The maximum length of each name can be obtained by calling SQLGetInfo with the <i>InfoType</i> values: SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, or SQL_MAX_TABLE_NAME_LEN.</p>
HY097	Column type out of range	(DM) An invalid <i>IdentifierType</i> value was specified.
HY098	Scope type out of range	(DM) An invalid <i>Scope</i> value was specified.
HY099	Nullable type out of range	(DM) An invalid <i>Nullable</i> value was specified.
HYC00	Optional feature not implemented	<p>A catalog was specified, and the driver or data source does not support catalogs.</p> <p>A schema was specified, and the driver or data source</p>

		<p>does not support schemas.</p> <p>The combination of the current settings of the SQL_ATTR_CONCURRENCY and SQL_ATTR_CURSOR_TYPE statement attributes was not supported by the driver or data source.</p> <p>The SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_VARIABLE, and the SQL_ATTR_CURSOR_TYPE statement attribute was set to a cursor type for which the driver does not support bookmarks.</p>
HYT00	Timeout expired	The query timeout period expired before the data source returned the requested result set. The timeout period is set through SQLSetStmtAttr , SQL_ATTR_QUERY_TIMEOUT .
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

When the *IdentifierType* 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. **SQLColumns**, which is used to return a variety of information on the columns of a table, does not necessarily return the columns that uniquely identify each row, or columns that are automatically updated when any value in the row is updated by a transaction. For example, **SQLColumns** might not return the Oracle pseudo-column ROWID. This is why **SQLSpecialColumns** is used to return these columns. For more information, see the Part I PDF file, “Uses of Catalog Data” in Chapter 7, “Catalog Functions.”

Note For more information about the general use, arguments, and returned data of ODBC catalog functions, see the Part I PDF file, Chapter 7, “Catalog Functions.”

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 **SQLFetchScroll** on the statement returns **SQL_NO_DATA**.

If the *IdentifierType*, *Scope*, or *Nullable* arguments specify characteristics that are not supported by the data source, **SQLSpecialColumns** returns an empty result set.

If the **SQL_ATTR_METADATA_ID** statement attribute is set to **SQL_TRUE**, the *CatalogName*, *SchemaName*, and *TableName* arguments are treated as identifiers, so they cannot be set to a null pointer in certain situations. (For more information, see the Part I PDF file, “Arguments in Catalog Functions” in Chapter 7, “Catalog Functions.”)

SQLSpecialColumns returns the results as a standard result set, ordered by **SCOPE**.

The following columns have been renamed for ODBC 3.x. The column name changes do not affect backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
PRECISION	COLUMN_SIZE
LENGTH	BUFFER_LENGTH
SCALE	DECIMAL_DIGITS

To determine the actual length of the COLUMN_NAME column, an application can call **SQLGetInfo** with the SQL_MAX_COLUMN_NAME_LEN option.

The following table lists the columns in the result set. Additional columns beyond column 8 (PSEUDO_COLUMN) can be defined by the driver. An application should gain access to driver-specific columns by counting down from the end of the result set rather than specifying an explicit ordinal position. For more information, see the Part I PDF file, "Data Returned by Catalog Functions" in Chapter 7, "Catalog Functions."

Column name	Column number	Data type	Comments
SCOPE (ODBC 1.0)	1	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 <i>IdentifierType</i> is SQL_ROWVER. For a description of each value, see the description of <i>Scope</i> in "Syntax," earlier in this section.
COLUMN_NAME (ODBC 1.0)	2	Varchar not NULL	Column name. The driver returns an empty string for a column that does not have a name.
DATA_TYPE (ODBC 1.0)	3	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" in Appendix D, "Data Types" of the SOLID Programmer Guide . For information about driver-specific SQL data types, see the driver's documentation.
TYPE_NAME (ODBC 1.0)	4	Varchar not NULL	Data source-dependent data type name; for example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINARY", or "CHAR () FOR BIT DATA".
COLUMN_SIZE (ODBC 1.0)	5	Integer	The size of the column on the data source. For more information concerning column size, see Appendix D, "Data Types" of the SOLID Programmer Guide .
BUFFER_LENGTH (ODBC 1.0)	6	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 COLUMN_SIZE column for character or binary data. For more information, Appendix D, "Data Types" of the SOLID Programmer Guide .
DECIMAL_DIGITS (ODBC 1.0)	7	Smallint	The decimal digits of the column on the data source. NULL is returned for data types where decimal digits are not applicable. For more information concerning decimal digits, see Appendix D, "Data Types" of the SOLID Programmer Guide .
PSEUDO_COLUMN (ODBC 2.0)	8	Smallint	Indicates whether the column is a pseudo-column, such as Oracle ROWID: SQL_PC_UNKNOWN SQL_PC_NOT_PSEUDO SQL_PC_PSEUDO Note For maximum interoperability, pseudo-columns should not be quoted with the identifier quote character returned by SQLGetInfo .

After 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, 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 forward and back 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 whether 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

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Returning the columns in a table or tables	SQLColumns
Fetching a single row or a block of data in a forward-only direction	SQLFetch
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Returning the columns of a primary key	SQLPrimaryKeys

SQLStatistics

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ISO 92

Summary

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

SQLRETURN SQLStatistics(
SQLHSTMT	StatementHandle,
SQLCHAR *	CatalogName,
SQLSMALLINT	NameLength1,
SQLCHAR *	SchemaName,
SQLSMALLINT	NameLength2,
SQLCHAR *	TableName,
SQLSMALLINT	NameLength3,
SQLUSMALLINT	Unique,
SQLUSMALLINT	Reserved);

Arguments

StatementHandle

[Input]

Statement handle.

CatalogName

[Input]

Catalog name. If a driver supports catalogs 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 catalogs. *CatalogName* cannot contain a string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *CatalogName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *CatalogName* is an ordinary argument; it is treated literally, and its case is significant. For more information, see the Part I PDF file, "Arguments in Catalog Functions" in Chapter 7, "Catalog Functions."

NameLength1

[Input]

Length of **CatalogName*.

SchemaName

[Input]

Schema name. If a driver supports schemas 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 schemas. *SchemaName* cannot contain a string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *SchemaName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *SchemaName* is an ordinary argument; it is treated literally, and its case is significant.

NameLength2

[Input]

Length of **SchemaName*.

TableName

[Input]

Table name. This argument cannot be a null pointer. *SchemaName* cannot contain a string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *TableName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *TableName* is an ordinary argument; it is treated literally, and its case is significant.

NameLength3

[Input]

Length of **TableName*.

Unique

[Input]

Type of index: SQL_INDEX_UNIQUE or SQL_INDEX_ALL.

Reserved

[Input]

Indicates the importance of the CARDINALITY and PAGES columns in the result set. The following options affect the return of the CARDINALITY and PAGES columns only; index information is returned even if CARDINALITY and PAGES are not returned.

SQL_ENSURE requests that the driver unconditionally retrieve the statistics. (Drivers that conform only to the X/Open standard and do not support ODBC extensions will not be able to support SQL_ENSURE.)

SQL_QUICK requests that the driver retrieve the CARDINALITY and PAGES only if they are readily available from the server. In this case, the driver does not ensure that the values are current. (Applications that are written to the X/Open standard will always get SQL_QUICK behavior from ODBC 3.x-compliant drivers.)

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 **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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	A cursor was open on the <i>StatementHandle</i> , and SQLFetch or SQLFetchScroll had been called. This error is returned by the Driver Manager if SQLFetch or SQLFetchScroll has not returned SQL_NO_DATA and is returned by the driver if SQLFetch or SQLFetchScroll has returned SQL_NO_DATA. A cursor was open on the <i>StatementHandle</i> , but SQLFetch or SQLFetchScroll had not been called.
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	Asynchronous processing was enabled for the

		<p><i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>, and then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.</p>
HY009	Invalid use of null pointer	<p>The <i>TableName</i> argument was a null pointer.</p> <p>(DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE, the <i>CatalogName</i> argument was a null pointer, and the SQL_CATALOG_NAME <i>InfoType</i> returns that catalog names are supported.</p> <p>(DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE, and the <i>SchemaName</i> argument was a null pointer.</p>
HY010	Function sequence error	<p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	<p>(DM) The value of one of the name length arguments was less than 0 but not equal to SQL_NTS.</p> <p>The value of one of the name length arguments exceeded the maximum length value for the corresponding name.</p>
HY100	Uniqueness option type out of range	(DM) An invalid <i>Unique</i> value was specified.
HY101	Accuracy option type out of range	(DM) An invalid <i>Reserved</i> value was specified.
HYC00	Optional feature not implemented	<p>A catalog was specified, and the driver or data source does not support catalogs.</p> <p>A schema was specified, and the driver or data source does not support schemas.</p> <p>The combination of the current settings of the SQL_ATTR_CONCURRENCY and</p>

		SQL_ATTR_CURSOR_TYPE statement attributes was not supported by the driver or data source. The SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_VARIABLE, and the SQL_ATTR_CURSOR_TYPE statement attribute was set to a cursor type for which the driver does not support bookmarks.
HYT00	Timeout expired	The query timeout period expired before the data source returned the requested result set. The timeout period is set through SQLSetStmtAttr , SQL_ATTR_QUERY_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

SQLStatistics returns information about a single table as a standard result set, ordered by NON_UNIQUE, TYPE, INDEX_QUALIFIER, INDEX_NAME, and ORDINAL_POSITION. The result set combines statistics information (in the CARDINALITY and PAGES columns of the result set) for the table with information about each index. For information about how this information might be used, see the Part I PDF file, "Uses of Catalog Data" in Chapter 7, "Catalog Functions."

To determine the actual lengths of the TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and COLUMN_NAME columns, an application can call **SQLGetInfo** with the SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN options.

Note For more information about the general use, arguments, and returned data of ODBC catalog functions, see the Part I PDF file, Chapter 7, "Catalog Functions."

The following columns have been renamed for ODBC 3.x. The column name changes do not affect backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM
SEQ_IN_INDEX	ORDINAL_POSITION
COLLATION	ASC_OR_DESC

The following table lists the columns in the result set. Additional columns beyond column 13 (FILTER_CONDITION) can be defined by the driver. An application should gain access to driver-specific columns by counting down from the end of the result set rather than specifying an explicit ordinal position.

For more information, see the Part I PDF file, “Data Returned by Catalog Functions” in Chapter 7, “Catalog Functions.”

Column name	Column number	Data type	Comments
TABLE_CAT (ODBC 1.0)	1	Varchar	Catalog name of the table to which the statistic or index applies; NULL if not applicable to the data source. If a driver supports catalogs 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 catalogs.
TABLE_SCHEM (ODBC 1.0)	2	Varchar	Schema name of the table to which the statistic or index applies; NULL if not applicable to the data source. If a driver supports schemas 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 schemas.
TABLE_NAME (ODBC 1.0)	3	Varchar not NULL	Table name of the table to which the statistic or index applies.
NON_UNIQUE (ODBC 1.0)	4	Smallint	Indicates whether the index prohibits duplicate values: SQL_TRUE if the index values can be nonunique. SQL_FALSE if the index values must be unique. NULL is returned if TYPE is SQL_TABLE_STAT.
INDEX_QUALIFIER (ODBC 1.0)	5	Varchar	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_SCHEM should be used to qualify the index name.
INDEX_NAME (ODBC 1.0)	6	Varchar	Index name; NULL is returned if TYPE is SQL_TABLE_STAT.
TYPE (ODBC 1.0)	7	Smallint not NULL	Type of information being returned: SQL_TABLE_STAT indicates a statistic for the table (in the CARDINALITY or PAGES column).

			<p>SQL_INDEX_BTREE indicates a B-Tree index.</p> <p>SQL_INDEX_CLUSTERED indicates a clustered index.</p> <p>SQL_INDEX_CONTENT indicates a content index.</p> <p>SQL_INDEX_HASHED indicates a hashed index.</p> <p>SQL_INDEX_OTHER indicates another type of index.</p>
ORDINAL_POSITION (ODBC 1.0)	8	Smallint	Column sequence number in index (starting with 1); NULL is returned if TYPE is SQL_TABLE_STAT.
COLUMN_NAME (ODBC 1.0)	9	Varchar	Column name. 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. NULL is returned if TYPE is SQL_TABLE_STAT.
ASC_OR_DESC (ODBC 1.0)	10	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 (ODBC 1.0)	11	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 (ODBC 1.0)	12	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 (ODBC 2.0)	13	Varchar	<p>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.</p> <p>NULL if the index is not a filtered index, it cannot be determined whether the index is a</p>

			filtered index, or TYPE is SQL_TABLE_STAT.
--	--	--	--------------------------------------------

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, ORDINAL_POSITION, COLUMN_NAME, and ASC_OR_DESC 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

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Fetching a single row or a block of data in a forward-only direction.	SQLFetch
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Returning the columns of foreign keys	SQLForeignKeys
Returning the columns of a primary key	SQLPrimaryKeys

SQLTablePrivileges

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: ODBC

Summary

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 statement.

Syntax

SQLRETURN SQLTablePrivileges(
SQLHSTMT	StatementHandle,
SQLCHAR *	CatalogName,
SQLSMALLINT	NameLength1,
SQLCHAR *	SchemaName,
SQLSMALLINT	NameLength2,
SQLCHAR *	TableName,
SQLSMALLINT	NameLength3);

Arguments

StatementHandle

[Input]

Statement handle.

CatalogName

[Input]

Table catalog. If a driver supports catalogs 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 catalogs. *CatalogName* cannot contain a string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *CatalogName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *CatalogName* is an ordinary argument; it is treated literally, and its case is significant. For more information, see the Part I PDF file, "Arguments in Catalog Functions" in Chapter 7, "Catalog Functions."

NameLength1

[Input]

Length of **CatalogName*.

SchemaName

[Input]

String search pattern for schema names. If a driver supports schemas 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 schemas.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *SchemaName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *SchemaName* is a pattern value argument; it is treated literally, and its case is significant.

NameLength2

[Input]

Length of **SchemaName*.

TableName

[Input]

String search pattern for table names.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *TableName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *TableName* is a pattern value argument; it is treated literally, and its case is significant.

NameLength3

[Input]

Length of **TableName*.

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 **SQLGetDiagRec** with a *HandleType* of SQL_HANDLE_STMT and a *Handle* of *StatementHandle*. 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	<p>A cursor was open on the <i>StatementHandle</i>, and SQLFetch or SQLFetchScroll had been called. This error is returned by the Driver Manager if SQLFetch or SQLFetchScroll has not returned SQL_NO_DATA and is returned by the driver if SQLFetch or SQLFetchScroll has returned SQL_NO_DATA.</p> <p>A cursor was open on the <i>StatementHandle</i>, but SQLFetch or SQLFetchScroll had not been called.</p>
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	<p>Asynchronous processing was enabled for the <i>StatementHandle</i>. The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i>. Then the function was called again on the <i>StatementHandle</i>.</p> <p>The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a</p>

		multithread application.
HY009	Invalid use of null pointer	<p>(DM) The <code>SQL_ATTR_METADATA_ID</code> statement attribute was set to <code>SQL_TRUE</code>, the <i>CatalogName</i> argument was a null pointer, and the <code>SQL_CATALOG_NAME InfoType</code> returns that catalog names are supported.</p> <p>(DM) The <code>SQL_ATTR_METADATA_ID</code> statement attribute was set to <code>SQL_TRUE</code>, and the <i>SchemaName</i> or <i>TableName</i> argument was a null pointer.</p>
HY010	Function sequence error	<p>(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called.</p> <p>(DM) SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos was called for the <i>StatementHandle</i> and returned <code>SQL_NEED_DATA</code>. This function was called before data was sent for all data-at-execution parameters or columns.</p>
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	Invalid string or buffer length	<p>(DM) The value of one of the name length arguments was less than 0 but not equal to <code>SQL_NTS</code>.</p> <p>The value of one of the name length arguments exceeded the maximum length value for the corresponding qualifier or name.</p>
HYC00	Optional feature not implemented	<p>A catalog was specified, and the driver or data source does not support catalogs.</p> <p>A schema was specified, and the driver or data source does not support schemas.</p> <p>A string search pattern was specified for the table schema, table name, or column name, and the data source does not support search patterns for one or more of those arguments.</p> <p>The combination of the current settings of the <code>SQL_ATTR_CONCURRENCY</code> and <code>SQL_ATTR_CURSOR_TYPE</code> statement attributes was not supported by the driver or data source.</p> <p>The <code>SQL_ATTR_USE_BOOKMARKS</code> statement attribute was set to <code>SQL_UB_VARIABLE</code>, and the <code>SQL_ATTR_CURSOR_TYPE</code> statement attribute was set to a cursor type for which the driver does not support bookmarks.</p>

HYT00	Timeout expired	The query timeout period expired before the data source returned the result set. The timeout period is set through SQLSetStmtAttr , SQL_ATTR_QUERY_TIMEOUT .
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , SQL_ATTR_CONNECTION_TIMEOUT .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

The *SchemaName* and *TableName* arguments accept search patterns. For more information about valid search patterns, see the Part I PDF file, "Pattern Value Arguments" in Chapter 7, "Catalog Functions."

SQLTablePrivileges returns the results as a standard result set, ordered by **TABLE_CAT**, **TABLE_SCHEM**, **TABLE_NAME**, **PRIVILEGE**, and **GRANTEE**.

To determine the actual lengths of the **TABLE_CAT**, **TABLE_SCHEM**, and **TABLE_NAME** columns, an application can call **SQLGetInfo** with the **SQL_MAX_CATALOG_NAME_LEN**, **SQL_MAX_SCHEMA_NAME_LEN**, and **SQL_MAX_TABLE_NAME_LEN** options.

Note For more information about the general use, arguments, and returned data of ODBC catalog functions, see the Part I PDF file, Chapter 7, "Catalog Functions."

The following columns have been renamed for ODBC 3.x. The column name changes do not affect backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM

The following table lists the columns in the result set. Additional columns beyond column 7 (**IS_GRANTABLE**) can be defined by the driver. An application should gain access to driver-specific columns by counting down from the end of the result set rather than specifying an explicit ordinal position. For more information, see the Part I PDF file, "Data Returned By Catalog Functions" in Chapter 7, "Catalog Functions."

Column name	Column number	Data type	Comments
TABLE_CAT (ODBC 1.0)	1	Varchar	Catalog name; NULL if not applicable to the data source. If a driver supports catalogs 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 catalogs.
TABLE_SCHEM (ODBC 1.0)	2	Varchar	Schema name; NULL if not applicable to the data source. If a driver supports schemas

			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 schemas.
TABLE_NAME (ODBC 1.0)	3	Varchar not NULL	Table name.
GRANTOR (ODBC 1.0)	4	Varchar	Name of the user who granted the privilege; NULL if not applicable to the data source. For all rows in which the value in the GRANTEE column is the owner of the object, the GRANTOR column will be "_SYSTEM".
GRANTEE (ODBC 1.0)	5	Varchar not NULL	Name of the user to whom the privilege was granted.
PRIVILEGE (ODBC 1.0)	6	Varchar not NULL	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 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 (ODBC 1.0)	7	Varchar	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.

			A privilege is either grantable or not grantable but not both. The result set returned by SQLColumnPrivileges will never contain two rows for which all columns except the IS_GRANTABLE column contain the same value.
--	--	--	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Code Example

For a code example of a similar function, see [SQLColumns](#).

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Returning privileges for a column or columns	SQLColumnPrivileges
Returning the columns in a table or tables	SQLColumns
Fetching a single row or a block of data in a forward-only direction	SQLFetch
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Returning table statistics and indexes	SQLStatistics
Returning a list of tables in a data source	SQLTables

SQLTables

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: X/Open

Summary

SQLTables returns the list of table, catalog, or schema names, and table types, stored in a specific data source. The driver returns the information as a result set.

Syntax

SQLRETURN SQLTables (
SQLHSTMT	StatementHandle,
SQLCHAR *	CatalogName,
SQLSMALLINT	NameLength1,
SQLCHAR *	SchemaName,
SQLSMALLINT	NameLength2,
SQLCHAR *	TableName,

SQLSMALLINT	NameLength3,
SQLCHAR *	TableType,
SQLSMALLINT	NameLength4);

Arguments

StatementHandle

[Input]

Statement handle for retrieved results.

CatalogName

[Input]

Catalog name. The *CatalogName* argument accepts search patterns if the SQL_ODBC_VERSION environment attribute is SQL_OV_ODBC3; it does not accept search patterns if SQL_OV_ODBC2 is set. If a driver supports catalogs 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 catalogs.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *CatalogName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *CatalogName* is a pattern value argument; it is treated literally, and its case is significant. For more information, see the Part I PDF file, "Arguments in Catalog Functions" in Chapter 7, "Catalog Functions."

NameLength1

[Input]

Length of **CatalogName*.

SchemaName

[Input]

String search pattern for schema names. If a driver supports schemas 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 schemas.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, *SchemaName* is treated as an identifier and its case is not significant. If it is SQL_FALSE, *SchemaName* is a pattern value argument; it is treated literally, and its case is significant.

NameLength2

[Input]

Length of **SchemaName*.

TableName

[Input]

String search pattern for table names.

If the `SQL_ATTR_METADATA_ID` statement attribute is set to `SQL_TRUE`, *TableName* is treated as an identifier and its case is not significant. If it is `SQL_FALSE`, *TableName* is a pattern value argument; it is treated literally, and its case is significant.

NameLength3

[Input]

Length of **TableName*.

TableType

[Input]

List of table types to match.

Note that the `SQL_ATTR_METADATA_ID` statement attribute has no effect upon the *TableType* argument. *TableType* is a value list argument, no matter what the setting of `SQL_ATTR_METADATA_ID`.

NameLength4

[Input]

Length of **TableType*.

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 **SQLGetDiagRec** with a *HandleType* of `SQL_HANDLE_STMT` and a *Handle* of *StatementHandle*. 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 `SQLSTATE`s 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 <code>SQL_SUCCESS_WITH_INFO</code> .)
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	A cursor was open on the <i>StatementHandle</i> , and SQLFetch or SQLFetchScroll had been called. This error is returned by the Driver Manager if SQLFetch or SQLFetchScroll has not returned <code>SQL_NO_DATA</code> and is returned by the driver if SQLFetch or SQLFetchScroll has returned <code>SQL_NO_DATA</code> . A cursor was open on the <i>StatementHandle</i> , but SQLFetch or SQLFetchScroll had not been called.
40001	Serialization failure	The transaction was rolled back due to a resource

		deadlock with another transaction.
40003	Statement completion unknown	The associated connection failed during the execution of this function, and the state of the transaction cannot be determined.
HY000	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 SQLGetDiagRec in the <i>*MessageText</i> buffer describes the error and its cause.
HY001	Memory allocation error	The driver was unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled	Asynchronous processing was enabled for the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> . Then the function was called again on the <i>StatementHandle</i> . The function was called, and before it completed execution, SQLCancel was called on the <i>StatementHandle</i> from a different thread in a multithread application.
HY009	Invalid use of null pointer	(DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE, the <i>CatalogName</i> argument was a null pointer, and the SQL_CATALOG_NAME <i>InfoType</i> returns that catalog names are supported. (DM) The SQL_ATTR_METADATA_ID statement attribute was set to SQL_TRUE, and the <i>SchemaName</i> or <i>TableName</i> argument was a null pointer.
HY010	Function sequence error	(DM) An asynchronously executing function (not this one) was called for the <i>StatementHandle</i> and was still executing when this function was called. (DM) SQLExecute , SQLExecDirect , SQLBulkOperations , or SQLSetPos was called for the <i>StatementHandle</i> and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY090	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 name length arguments exceeded the maximum length value for the corresponding name.
HYC00	Optional feature not	A catalog was specified, and the driver or data source

	implemented	<p>does not support catalogs.</p> <p>A schema was specified, and the driver or data source does not support schemas.</p> <p>A string search pattern was specified for the catalog name, table schema, or table name, and the data source does not support search patterns for one or more of those arguments.</p> <p>The combination of the current settings of the <code>SQL_ATTR_CONCURRENCY</code> and <code>SQL_ATTR_CURSOR_TYPE</code> statement attributes was not supported by the driver or data source.</p> <p>The <code>SQL_ATTR_USE_BOOKMARKS</code> statement attribute was set to <code>SQL_UB_VARIABLE</code>, and the <code>SQL_ATTR_CURSOR_TYPE</code> statement attribute was set to a cursor type for which the driver does not support bookmarks.</p>
HYT00	Timeout expired	The query timeout period expired before the data source returned the requested result set. The timeout period is set through SQLSetStmtAttr , <code>SQL_ATTR_QUERY_TIMEOUT</code> .
HYT01	Connection timeout expired	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through SQLSetConnectAttr , <code>SQL_ATTR_CONNECTION_TIMEOUT</code> .
IM001	Driver does not support this function	(DM) The driver associated with the <i>StatementHandle</i> does not support the function.

Comments

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

Call **SQLGetInfo** and check the `SQL_ACCESSIBLE_TABLES` information type.

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 *SchemaName* and *TableName* arguments accept search patterns. The *CatalogName* argument accepts search patterns if the `SQL_ODBC_VERSION` environment attribute is `SQL_OV_ODBC3`; it does not accept search patterns if `SQL_OV_ODBC2` is set. If `SQL_OV_ODBC3` is set, an ODBC 3.x driver will require that wildcard characters in the *CatalogName* argument be escaped to be treated literally. For more information about valid search patterns, see the Part I PDF file, “Pattern Value Arguments” in Chapter 7, “Catalog Functions.”

Note For more information about the general use, arguments, and returned data of ODBC catalog functions, see the Part I PDF file, Chapter 7, “Catalog Functions.”

To support enumeration of catalogs, schemas, and table types, the following special semantics are defined for the *CatalogName*, *SchemaName*, *TableName*, and *TableType* arguments of **SQLTables**:

If *CatalogName* is `SQL_ALL_CATALOGS` and *SchemaName* and *TableName* are empty strings, the result set contains a list of valid catalogs for the data source. (All columns except the `TABLE_CAT` column contain NULLs.)

If *SchemaName* is `SQL_ALL_SCHEMAS` and *CatalogName* and *TableName* are empty strings, the result set contains a list of valid schemas for the data source. (All columns except the `TABLE_SCHEM` column contain NULLs.)

If *TableType* is `SQL_ALL_TABLE_TYPES` and *CatalogName*, *SchemaName*, and *TableName* are empty strings, the result set contains a list of valid table types for the data source. (All columns except the `TABLE_TYPE` column contain NULLs.)

If *TableType* 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 quotation marks (') or unquoted—for example, 'TABLE', 'VIEW' or TABLE, VIEW. An application should always specify the table type in uppercase; the driver should convert the table type to whatever case is needed by the data source. 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_CAT`, `TABLE_SCHEM`, and `TABLE_NAME`. For information about how this information might be used, see the Part I PDF file, “Uses of Catalog Data” in Chapter 7, “Catalog Functions.”

To determine the actual lengths of the `TABLE_CAT`, `TABLE_SCHEM`, and `TABLE_NAME` columns, an application can call **SQLGetInfo** with the `SQL_MAX_CATALOG_NAME_LEN`, `SQL_MAX_SCHEMA_NAME_LEN`, and `SQL_MAX_TABLE_NAME_LEN` information types.

The following columns have been renamed for ODBC 3.x. The column name changes do not affect backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM

The following table lists the columns in the result set. Additional columns beyond column 5 (REMARKS) can be defined by the driver. An application should gain access to driver-specific columns by counting down from the end of the result set rather than specifying an explicit ordinal position. For more information, see the Part I PDF file, “Data Returned By Catalog Functions” in Chapter 7, “Catalog Functions.”

Column name	Column number	Data type	Comments
TABLE_CAT (ODBC 1.0)	1	Varchar	Catalog name; NULL if not applicable to the data source. If a driver supports catalogs 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 catalogs.
TABLE_SCHEM (ODBC 1.0)	2	Varchar	Schema name; NULL if not applicable to the data source. If a driver supports schemas 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 schemas.
TABLE_NAME (ODBC 1.0)	3	Varchar	Table name.
TABLE_TYPE (ODBC 1.0)	4	Varchar	Table type name; one of the following: "TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL TEMPORARY", "LOCAL TEMPORARY", "ALIAS", "SYNONYM", or a data source-specific type name. The meanings of "ALIAS" and "SYNONYM" are driver-specific.
REMARKS (ODBC 1.0)	5	Varchar	A description of the table.

Code Example

For a code example of a similar function, see [SQLColumns](#).

Related Functions

For information about	See
Binding a buffer to a column in a result set	SQLBindCol
Canceling statement processing	SQLCancel
Returning privileges for a column or columns	SQLColumnPrivileges
Returning the columns in a table or tables	SQLColumns
Fetching a single row or a block of data in a forward-only direction	SQLFetch
Fetching a block of data or scrolling through a result set	SQLFetchScroll
Returning table statistics and indexes	SQLStatistics
Returning privileges for a table or tables	SQLTablePrivileges

SQLTransact

Conformance

Version Introduced: ODBC 1.0

Standards Compliance: Deprecated

Summary

In ODBC 3.x, the ODBC 2.x function **SQLTransact** has been replaced by **SQLEndTran**. For more information, see [SQLEndTran](#).

Note For more information about what the Driver Manager maps this function to when an ODBC 2.x application is working with an ODBC 3.x driver, see “Mapping Deprecated Functions” (Appendix G,

"Driver Guidelines for Backward Compatibility) contained on the Microsoft Web site (ODBC Programmer's Reference).