# Quasar SQL API Help Index

This index lists the help topics available for the Quasar SQL applications programming interface (API). Using the keyboard, tab to select the underlined topic you want to view, then press enter. Using the mouse, point to the underlined topic you want to view, and click the left mouse button. Use the scroll bar to see entries not currently visible in the help window.

To learn how to use help, press F1.

| | |
|---|---|
| **Overview** | This section describes, in general terms, what the Quasar SQL API is and how you might use it to execute SQL statements within your program. Examples are available both in 'C' (EXAMPLEC.*) and 'Visual Basic' (EXAMPLEB.*). |
| **Functions** | This section includes a formal description of all functions which your program can import from the Quasar SQL API dynamic link library (SQL.LIB, SQL.DLL). Prototypes for all functions can be found in the Quasar SQL API include file for 'C' (SQL.H) and the global file for 'Visual Basic' (SQL.GBL). |
| **Constants** | This section includes a formal description of all constants defined in the Quasar SQL API include file for 'C' and the global file for 'Visual Basic'. |
| **Data Structures** | This section includes a formal description of all data structures which your program will use when it communicates with the Quasar SQL API. *Typedefs* for all these structures can be found in the Quasar SQL API include file for 'C'. *Declarations* can be found in the global file for 'Visual Basic'. |
| **Result Codes** | This section gives an explanation of all result codes returned by the Quasar SQL API. *Macro* definitions for all result codes can be found in the Quasar SQL API include file for 'C'. *Global Const* definitions can be found in the global file for 'Visual Basic'. |

# Overview

### What is the Quasar SQL API?

The Quasar SQL API is a powerful tool that enables you, the software developer, to have full access to a modern, highly optimized ANSI standard SQL database manager: the Quasar Database Administrator. Your programs written in any language which can access dynamic link libraries can also access the Quasar SQL API. These languages include 'C', 'C++', 'Visual Basic', 'Pascal' and most assemblers. In fact, any program which can access the Microsoft Windows environment can access the Quasar SQL API. Microsoft Windows itself is a dynamic link library.

The Quasar SQL API is your gateway to modern database technology. With a little more than a dozen functions and only 4 data structures you can perform almost any database task.

### How does the Quasar SQL API work?

"How is this done?" you might ask. Well, while the Quasar Database Administrator is sophisticated, complex and fully optimized for maximum performance: the Quasar SQL API is simple and straight forward. The Quasar SQL API effectively buffers you from all the complex tasks of parsing, query analysis, query optimization, query tree construction, etc.

### How do I link to the Quasar SQL API?

When programming in 'C', you need to do only three things to include all the power of SQL in your application:

1) When you link your object modules, include the SQL.LIB import library along with LIBW.LIB.
2) Include the SQL.H file (e.g. "#include <SQL.H>") in your source code.
3) Put the Quasar Database Administrator (DBA.EXE) and Quasar SQL API (SQL.DLL) someplace where windows can find them (usually in your windows directory).

When programming in 'Visual Basic', you need to do only two things to include all the power of SQL in your application:

1) Be sure to use the SQL.GBL file included with the Quasar SQL API. Note that the 'C' style prefixes used with variable names (i.e. the '**n**' in nResultCode, and the '**sz**' in szErrorMessage) are not used with 'Visual Basic'.
2) Put the Quasar Database Administrator (DBA.EXE) and Quasar SQL API (SQL.DLL) someplace where windows can find them (usually in your windows directory).

### How do I use the Quasar SQL API?

Your program can login to the Quasar Database Administrator (SqlLogin), open a cursor (SqlCursorOpen) and execute a query (SqlExecute). The query itself is passed to the Quasar SQL API as a simple text string. The Quasar SQL API does all the work for you. To get your answers back (from a SELECT statement, for example) you simply fetch them one at a time from the Quasar SQL API. You can fetch the first record (SqlFetchFirst), last record (SqlFetchLast), next record (SqlFetchNext), previous record (SqlFetchPrevious) or even a record at a position you specify (SqlFetchPositioned).

You can have several cursors open at a time so that you can access your data the way you want.

When you're all done, close your cursors (SqlCursorClose) and log out (SqlLogout).

### How can I check out my SQL statements?

You can check out your SQL statements using the Quasar Query Editor (QE.EXE) included with the Quasar SQL API. The Quasar Query Editor allows you to type in the query your program is going to send to the Quasar SQL API and have it execute right there in front of you. You'll get all the results back on the screen where you can verify it's

what you expected. In fact, the Quasar Query Editor uses the Quasar SQL API the same way your program does.

## Functions

All Quasar SQL API functions utilize a common structure to maintain generic information : SQLCONTROL. All functions load descriptive error information into this structure should an error condition arise. The basic status condition is the **nResultCode** field of this data structure. All Quasar SQL API functions set this field to the appropriate result code in the event of an error. Note: ***All Quasar SQL API functions clear this field to 0 at entry***.

| | |
|---|---|
| **SqlCursorClose** | Closes a cursor when it is no longer needed |
| **SqlCursorOpen** | Opens a cursor in preparation for executing a query |
| **SqlDescribeColumn** | Describes an individual column in the result table |
| **SqlDescribeTable** | Describes the overall characteristics of the result table |
| **SqlExecute** | Causes the database administrator to execute a query and potentially create a result table |
| **SqlFetchFirst** | Retrieves the first record from the result table |
| **SqlFetchLast** | Retrieves the last record from the result table |
| **SqlFetchNext** | Retrieves the next record from the table |
| **SqlFetchPrevious** | Retrieves the previous record from the result table |
| **SqlFetchPositioned** | Retrieves a particular record from the result table. The record is identified by its absolute physical position in the table |
| **SqlGetStatus** | Gets status information concerning the last query executed |
| **SqlLogin** | Logs an user onto the database administrator |
| **SqlLogout** | Logs an user off the database administrator |

## Constants

This section includes a formal description of all constants defined in the Quasar SQL API include file for 'C' and the global file for 'Visual Basic'.

Be sure to use the values by name as the numeric values may (and probably will) change in the next release of the Quasar SQL API.

| | |
|---|---|
| **Data Class** | Identifies the general characteristics of a column in the result table |
| **Data Type** | Identifies the particular characteristics of a column in the result table |
| **Data Format** | Instructs the Quasar SQL API as to the format of the returned data in the result table |
| **Automatic Commit** | Instructs the Quasar SQL API whether to commit or roll back any pending transaction at user log out |
| **Name Length** | Defines the maximum length of user, table and column names |

## Data Structures

This section includes a formal description of all data structures which your program will use when it communicates with the Quasar SQL API. Typedefs for all these structures can be found in the Quasar SQL API include file for 'C'. Declarations can be found in the global file for 'Visual Basic'.

| | |
|---|---|
| **SQLCOLUMN** | The data structure which describes a column in the result table which was generated by execution of a query |
| **SQLCONTROL** | The data structure which contains information needed by the Quasar SQL API to perform its functions and which passes process information back to the user |
| **SQLSTATUS** | The data structure with gives the user access to operational information |
| **SQLTABLE** | The data structure which describes the result table which was generated by execution of a query |

# Result Codes

This section gives an explanation of all result codes returned by the Quasar SQL API. *Macro* definitions for all result codes can be found in the Quasar SQL API include file for 'C'. *Global Const* definitions can be found in the global file for 'Visual Basic'.

**Result Codes Part I** Descriptions of result codes (Ambiguous Column - Data File Corrupted)

**Result Codes Part II** Descriptions of result codes (Database Corrupt - Invalid Cursor)

**Result Codes Part III** Descriptions of result codes (Invalid Data - NULL Not Allowed)

**Result Codes Part IV** Descriptions of result codes (Parser Syntax Error - Wrong Version)

# Function: SqlCursorClose

**Syntax** **BOOL FAR PASCAL SqlCursorClose(** *hUser , hCursor , lpSqlControl* **)**

This function closes the cursor specified by the *hCursor* parameter and drops any result table associated with it.

| Parameter | Type/Description |
|---|---|
| *hUser* | **HANDLE** Identifies the user who owns the cursor. The *hUser* parameter must have been created with the SqlLogin function. |
| *hCursor* | **HANDLE** Identifies the cursor to be closed. The *hCursor* parameter must have been created with the SqlCursorOpen function. |
| *lpSqlControl* | **LPSQLCONTROL** Points to an SQL control structure. Any errors that may occur cause the **szErrorDetail**, **szErrorMessage** and **nResultCode** fields of this structure to be loaded with descriptive information. |

**Return Value** The return value is 1 if no errors occurred while closing the cursor, otherwise it is 0.

**Comments** Cursors should be closed before logging off with SqlLogout.

# Function: SqlCursorOpen

**Syntax** **HANDLE FAR PASCAL SqlCursorOpen(** *hUser* , *lpSqlControl* **)**

This function creates a cursor. You must create a cursor after you have logged in (SqlLogin) and before you execute a query (SqlExecute). You can create as many cursors as you need for your application.

| Parameter | Type/Description |
|---|---|
| *hUser* | **HANDLE** Identifies the user who will own the cursor. The *hUser* parameter must have been created with the SqlLogin function. |
| *lpSqlControl* | **LPSQLCONTROL** Points to an SQL Control Structure. Any errors that may occur cause the **szErrorDetail**, **szErrorMessage** and **nResultCode** fields of this structure to be loaded with descriptive information. |

**Return Value** The return value is a cursor handle if no errors occurred while opening the cursor, otherwise it is 0.

**Comments** The cursors created by one user are not available to another. Once a cursor has been created and used in an **SqlExecute**, it may be reused in another **SqlExecute** by the same user. When a cursor is reused in this manner, the Quasar SQL API effectively closes the cursor and re-opens it with the same handle prior to executing the next query. Note that when a cursor is reused, any old result table associated with it is dropped.

Cursors are closed with SqlCursorClose. Cursors should be closed before logging off with SqlLogout.

# Function: SqlDescribeColumn

**Syntax** **BOOL FAR PASCAL SqlDescribeColumn(** *hUser , hCursor , lpSqlControl , nColumnSequenceNumber , lpSqlColumn* **)**

This function describes the characteristics of the specified column (*nColumnSequenceNumber*) in the result table associated with the *hCursor* parameter. The description is loaded into the SQLCOLUMN structure pointed to by the *lpSqlColumn* parameter. To describe a column you must have already logged in (SqlLogin), created a cursor (SqlCursorOpen) and executed a query (SqlExecute) which contained an SQL SELECT statement.

| Parameter | Type/Description |
|---|---|
| *hUser* | **HANDLE** Identifies the user who owns the cursor. The *hUser* parameter must have been created with the SqlLogin function. |
| *hCursor* | **HANDLE** Identifies the cursor which owns the result table in which the column exists. The *hCursor* parameter must have been created with the SqlCursorOpen function. A result table must be associated with the cursor. A result table is created when a query containing an SQL SELECT statement is passed to SqlExecute. |
| *lpSqlControl* | **LPSQLCONTROL** Points to an SQL Control Structure. Any errors that may occur cause the **szErrorDetail**, **szErrorMessage** and **nResultCode** fields of this structure to be loaded with descriptive information. |
| *nColumnSequenceNumber* | **unsigned** Identifies the column to be described. Column numbers range from 1 to the number of columns in the result table. |
| *lpSqlColumn* | **LPSQLCOLUMN** Points to the target SQL Column Description Structure. |

**Return Value** The return value is 1 if no errors occurred, otherwise it is 0.

**Comments** If a result table is not associated with the cursor an error will be generated.

# Function: SqlDescribeTable

**Syntax** **BOOL FAR PASCAL SqlDescribeTable(** *hUser* , *hCursor* , *lpSqlControl* , *lpSqlTable* **)**

This function describes the characteristics of the result table associated with the *hCursor* parameter. The description is loaded into the SQLTABLE structure pointed to by the *lpSqlTable* parameter. To describe a table you must have already logged in (SqlLogin), created a cursor (SqlCursorOpen) and executed a query (SqlExecute) which contained an SQL SELECT statement.

| Parameter | Type/Description |
|---|---|
| *hUser* | **HANDLE** Identifies the user who owns the cursor. The *hUser* parameter must have been created with the SqlLogin function. |
| *hCursor* | **HANDLE** Identifies the cursor which owns the result table. The *hCursor* parameter must have been created with the SqlCursorOpen function. A result table must be associated with the cursor. A result table is created when a query containing an SQL SELECT statement is passed to SqlExecute. |
| *lpSqlControl* | **LPSQLCONTROL** Points to an SQL Control Structure. Any errors that may occur cause the **szErrorDetail**, **szErrorMessage** and **nResultCode** fields of this structure to be loaded with descriptive information. |
| *lpSqlTable* | **LPSQLTABLE** Points to the target SQL Table Description Structure. |

**Return Value** The return value is 1 if no errors occurred, otherwise it is 0.

**Comments** If a result table is not associated with the cursor an error will be generated.

# Function: SqlExecute

**Syntax** **BOOL FAR PASCAL SqlExecute(** *hUser , hCursor , lpSqlControl , lpstrQueryText* **)**

This function executes the query text pointed to by the *lpstrQueryText* parameter as a set of one or more SQL statements. If an SQL SELECT statement is included in the query text, a result table is created and associated with the *hCursor* parameter. To execute a query you must have already logged in (SqlLogin) and created a cursor (SqlCursorOpen).

| Parameter | Type/Description |
|---|---|
| *hUser* | **HANDLE** Identifies the user who owns the cursor. The *hUser* parameter must have been created with the SqlLogin function. |
| *hCursor* | **HANDLE** Identifies the cursor which will own the result table if one is created by the query. The *hCursor* parameter must have been created with the SqlCursorOpen function. |
| *lpSqlControl* | **LPSQLCONTROL** Points to an SQL Control Structure. Any errors that may occur cause the **szErrorDetail**, **szErrorMessage** and **nResultCode** fields of this structure to be loaded with descriptive information. Additionally, syntax errors detected by the query manager cause the **nHiLiteOffset** field to be loaded with the character position of the offending phrase within the query text. In this case, the **nHiLiteLength** field is loaded with the number of characters in the offending phrase. |
| *lpstrQueryText* | **LPSTR** Points to a null terminated string containing the SQL query. The SQL query may contain one or more SQL statements. Among the SQL statements there can be only 0 or 1 SQL SELECT statement. |

**Return Value** The return value is 1 if no errors occurred, otherwise it is 0.

**Comments** Simply calling **SqlExecute** does not commit your query. Queries are committed when you execute an SQL COMMIT WORK statement. Be sure to execute this vital statement whenever you want your program to permanently modify the database.

Each user must close all cursors which have an associated result table before executing an SQL COMMIT WORK statement. This is to insure that your application has completed its access to the result table before permanently committing the transaction. A convenient way to do this is to use the same cursor for the COMMIT WORK as you did for the SELECT: the Quasar SQL API automatically drops any result table that may be associated with a cursor when the cursor is reused.

Without the SQL COMMIT WORK statement, all your transactions will be rolled back when either an error occurs or when you log off (SqlLogout). The only exception to this is if you have the mSqlFlagAutoCommitOnLogout option flag set in the **wFlags** field of the *lpSqlControl* parameter when you log off. Only in this case are all your transactions automatically committed at logout, and then only if no errors have occurred.

# Function: SqlFetchFirst

**Syntax** **BOOL FAR PASCAL SqlFetchFirst(** *hUser , hCursor , lpSqlControl , lpRecordBuffer* **)**

This function reads the first record in the result table associated with the *hCursor* parameter. The data in this record is formatted and loaded into the buffer pointed to by the *lpRecordBuffer* parameter. To fetch this record you must have already logged in (SqlLogin), created a cursor (SqlCursorOpen) and executed a query (SqlExecute) which contained an SQL SELECT statement.

| Parameter | Type/Description |
|---|---|
| *hUser* | **HANDLE** Identifies the user who owns the cursor. The *hUser* parameter must have been created with the SqlLogin function. |
| *hCursor* | **HANDLE** Identifies the cursor which owns the result table in which the record exists. The *hCursor* parameter must have been created with the SqlCursorOpen function. A result table must be associated with the cursor. A result table is created when a query containing an SQL SELECT statement is passed to SqlExecute. |
| *lpSqlControl* | **LPSQLCONTROL** Points to an SQL Control Structure. Any errors that may occur cause the **szErrorDetail**, **szErrorMessage** and **nResultCode** fields of this structure to be loaded with descriptive information. |
| *lpRecordBuffer* | **LPSTR** Points to a buffer provided by the caller to receive the formatted record. This buffer must be at least as large as the **nRecordSize** field of SQLTABLE structure indicates. |

**Return Value** The return value is 1 if no errors occurred and the table had at least one record, otherwise it is 0.

**Comments** The usual procedure for using the **SqlFetchFirst** function is as follows:

1) Set Data Format in the **wFlags** field of the *lpSqlControl* parameter to the option you intend to use. If you have selected the mSqlFlagFormatPadded option, set the **nPadding** field of the *lpSqlControl* parameter to the number of spaces you want between fields.
2) Call SqlExecute.
3) Call SqlDescribeTable to determine the characteristics of the result table. Of particular interest are the **nRecordSize** and **lNumberOfRecords** fields of the resultant SQLTABLE structure.
4) If **SqlTable.lNumberOfRecords** indicates that one or more result table records are available, allocate a buffer sufficiently large (at least as large as **SqlTable.nRecordSize**) to receive the record's data. Set the **lpRecordBuffer** parameter to point to this buffer and set the **nRecordBufferSize** field of the *lpSqlControl* parameter to the size of the buffer.
5) Call **SqlFetchFirst**.

Once these steps are completed, you can make unlimited calls to any of the Quasar SQL API fetch functions without repeating steps (1) through (4). The same *lpRecordBuffer* can be reused for all fetches. Note, however, should you change the Data Format setting in the **wFlags** field of the *lpSqlControl* parameter, you must call SqlDescribeTable to calculate the new **nRecordSize** setting, reallocate the *lpRecordBuffer* and adjust the **nRecordBufferSize** field of the *lpSqlControl* parameter accordingly.

# Function: SqlFetchLast

**Syntax** **BOOL FAR PASCAL SqlFetchLast(** *hUser , hCursor , lpSqlControl , lpRecordBuffer* **)**

This function reads the last record in the result table associated with the *hCursor* parameter. The data in this record is formatted and loaded into the buffer provided by the caller and pointed to by the *lpRecordBuffer* parameter. To fetch this record you must have already logged in (SqlLogin), created a cursor (SqlCursorOpen) and executed a query (SqlExecute) which contained an SQL SELECT statement.

| Parameter | Type/Description |
|---|---|
| *hUser* | **HANDLE** Identifies the user who owns the cursor. The *hUser* parameter must have been created with the SqlLogin function. |
| *hCursor* | **HANDLE** Identifies the cursor which owns the result table in which the record exists. The *hCursor* parameter must have been created with the SqlCursorOpen function. A result table must be associated with the cursor. A result table is created when a query containing an SQL SELECT statement is passed to SqlExecute. |
| *lpSqlControl* | **LPSQLCONTROL** Points to an SQL Control Structure. Any errors that may occur cause the **szErrorDetail**, **szErrorMessage** and **nResultCode** fields of this structure to be loaded with descriptive information. |
| *lpRecordBuffer* | **LPSTR** Points to a buffer provided by the caller to receive the formatted record. This buffer must be at least as large as the **nRecordSize** field of SQLTABLE structure indicates. |

**Return Value** The return value is 1 if no errors occurred and the table had at least one record, otherwise it is 0.

**Comments** The usual procedure for using the **SqlFetchLast** function is as follows:

1) Set Data Format in the **wFlags** field of the *lpSqlControl* parameter to the option you intend to use. If you have selected the mSqlFlagFormatPadded option, set the **nPadding** field of the *lpSqlControl* parameter to the number of spaces you want between fields.
2) Call SqlExecute.
3) Call SqlDescribeTable to determine the characteristics of the result table. Of particular interest are the **nRecordSize** and **lNumberOfRecords** fields of the resultant SQLTABLE structure.
4) If **SqlTable.lNumberOfRecords** indicates that one or more result table records are available, allocate a buffer sufficiently large (at least as large as **SqlTable.nRecordSize**) to receive the record's data. Set the **lpRecordBuffer** parameter to point to this buffer and set the **nRecordBufferSize** field of the *lpSqlControl* parameter to the size of the buffer.
5) Call **SqlFetchLast**.

Once these steps are completed, you can make unlimited calls to any of the Quasar SQL API fetch functions without repeating steps (1) through (4). The same *lpRecordBuffer* can be reused for all fetches. Note, however, should you change the Data Format setting in the **wFlags** field of the *lpSqlControl* parameter, you must call SqlDescribeTable to calculate the new **nRecordSize** setting, reallocate the *lpRecordBuffer* and adjust the **nRecordBufferSize** field of the *lpSqlControl* parameter accordingly.

# Function: SqlFetchNext

**Syntax** **BOOL FAR PASCAL SqlFetchNext(** *hUser , hCursor , lpSqlControl , lpRecordBuffer* **)**

This function reads the next record in the result table associated with the *hCursor* parameter. The data in this record is formatted and loaded into the buffer provided by the caller and pointed to by the *lpRecordBuffer* parameter. To fetch this record you must have already preformed at least one SqlFetchFirst or SqlFetchLast or SqlFetchPositioned function.

| Parameter | Type/Description |
|---|---|
| *hUser* | **HANDLE** Identifies the user who owns the cursor. The *hUser* parameter must have been created with the SqlLogin function. |
| *hCursor* | **HANDLE** Identifies the cursor which owns the result table in which the record exists. The *hCursor* parameter must have been created with the SqlCursorOpen function. A result table must be associated with the cursor. A result table is created when a query containing an SQL SELECT statement is passed to SqlExecute. |
| *lpSqlControl* | **LPSQLCONTROL** Points to an SQL Control Structure. Any errors that may occur cause the **szErrorDetail**, **szErrorMessage** and **nResultCode** fields of this structure to be loaded with descriptive information. |
| *lpRecordBuffer* | **LPSTR** Points to a buffer provided by the caller to receive the formatted record. This buffer must be at least as large as the **nRecordSize** field of SQLTABLE structure indicates. |

**Return Value** The return value is 1 if no errors occurred and the cursor was not already positioned at the last record, otherwise it is 0.

**Comments** The same *lpRecordBuffer* can be reused for all fetches. Note, however, should you change the Data Format setting in the **wFlags** field of the *lpSqlControl* parameter, you must call SqlDescribeTable to calculate the new **nRecordSize** setting, reallocate the *lpRecordBuffer* and adjust the **nRecordBufferSize** field of the *lpSqlControl* parameter accordingly.

# Function: SqlFetchPositioned

**Syntax** **BOOL FAR PASCAL SqlFetchPositioned(** *hUser , hCursor , lpSqlControl , lRecordPosition , lpRecordBuffer* **)**

This function reads a record in the result table associated with the *hCursor* parameter. The *lRecordPosition* parameter indicates which record is to be read. When *lRecordPosition* is '1', the first physical record is read; when *lRecordPosition* is '2', the second physical record is read and so on and so forth. The data in this record is formatted and loaded into the buffer provided by the caller and pointed to by the *lpRecordBuffer* parameter. To fetch this record you must have already logged in (SqlLogin), created a cursor (SqlCursorOpen) and executed a query (SqlExecute) which contained an SQL SELECT statement.

| Parameter | Type/Description |
|---|---|
| *hUser* | **HANDLE** Identifies the user who owns the cursor. The *hUser* parameter must have been created with the SqlLogin function. |
| *hCursor* | **HANDLE** Identifies the cursor which owns the result table in which the record exists. The *hCursor* parameter must have been created with the SqlCursorOpen function. A result table must be associated with the cursor. A result table is created when a query containing an SQL SELECT statement is passed to SqlExecute. |
| *lpSqlControl* | **LPSQLCONTROL** Points to an SQL Control Structure. Any errors that may occur cause the **szErrorDetail**, **szErrorMessage** and **nResultCode** fields of this structure to be loaded with descriptive information. |
| *lRecordPosition* | **unsigned long** Indicates which record is to be read. |
| *lpRecordBuffer* | **LPSTR** Points to a buffer provided by the caller to receive the formatted record. This buffer must be at least as large as the **nRecordSize** field of SQLTABLE structure indicates. |

**Return Value** The return value is 1 if no errors occurred and *lRecordPosition* is greater than zero and less then or equal to the number of records in the table, otherwise it is 0.

**Comments** Physical position is the raw position of the record in the file. Physical order is not the same as the order you specify with the SQL ORDER BY clause.

The usual procedure for using the **SqlFetchPositioned** function is as follows:

1) Set Data Format in the **wFlags** field of the *lpSqlControl* parameter to the option you intend to use. If you have selected the mSqlFlagFormatPadded option, set the **nPadding** field of the *lpSqlControl* parameter to the number of spaces you want between fields.
2) Call SqlExecute.
3) Call SqlDescribeTable to determine the characteristics of the result table. Of particular interest are the **nRecordSize** and **lNumberOfRecords** fields of the resultant SQLTABLE structure.
4) If **SqlTable.lNumberOfRecords** indicates that one or more result table records are available, allocate a buffer sufficiently large (at least as large as **SqlTable.nRecordSize**) to receive the record's data. Set the **lpRecordBuffer** parameter to point to this buffer and set the **nRecordBufferSize** field of the *lpSqlControl* parameter to the size of the buffer.
5) Call **SqlFetchPositioned**.

Once these steps are completed, you can make unlimited calls to any of the Quasar SQL API fetch functions without repeating steps (1) through (4). The same *lpRecordBuffer* can be reused for all fetches. Note, however, should you change the Data Format setting in the **wFlags** field of the *lpSqlControl* parameter, you must call SqlDescribeTable

to calculate the new **nRecordSize** setting, reallocate the *lpRecordBuffer* and adjust the **nRecordBufferSize** field of the *lpSqlControl* parameter accordingly.

# Function: SqlFetchPrevious

**Syntax** **BOOL FAR PASCAL SqlFetchPrevious(** *hUser , hCursor , lpSqlControl , lpRecordBuffer* **)**

This function reads the previous record in the result table associated with the *hCursor* parameter. The data in this record is formatted and loaded into the buffer provided by the caller and pointed to by the *lpRecordBuffer* parameter. To fetch this record you must have already preformed at least one SqlFetchFirst or SqlFetchLast or SqlFetchPositioned function.

| Parameter | Type/Description |
|---|---|
| *hUser* | **HANDLE** Identifies the user who owns the cursor. The *hUser* parameter must have been created with the SqlLogin function. |
| *hCursor* | **HANDLE** Identifies the cursor which owns the result table in which the record exists. The *hCursor* parameter must have been created with the SqlCursorOpen function. A result table must be associated with the cursor. A result table is created when a query containing an SQL SELECT statement is passed to SqlExecute. |
| *lpSqlControl* | **LPSQLCONTROL** Points to an SQL Control Structure. Any errors that may occur cause the **szErrorDetail**, **szErrorMessage** and **nResultCode** fields of this structure to be loaded with descriptive information. |
| *lpRecordBuffer* | **LPSTR** Points to a buffer provided by the caller to receive the formatted record. This buffer must be at least as large as the **nRecordSize** field of SQLTABLE structure indicates. |

**Return Value** The return value is 1 if no errors occurred and the cursor was not already positioned at the first record, otherwise it is 0.

**Comments** The same *lpRecordBuffer* can be reused for all fetches. Note, however, should you change the Data Format setting in the **wFlags** field of the *lpSqlControl* parameter, you must call SqlDescribeTable to calculate the new **nRecordSize** setting, reallocate the *lpRecordBuffer* and adjust the **nRecordBufferSize** field of the *lpSqlControl* parameter accordingly.

# Function: SqlGetStatus

**Syntax** **BOOL FAR PASCAL SqlGetStatus(** *hUser , hCursor , lpSqlControl , lpSqlStatus* **)**

This function returns status information about the most recent call to SqlExecute which used the cursor indicated by the *hCursor* parameter. Status information includes the number of database records deleted, inserted, selected or updated and the amount of time elapsed during query execution.

If the *hUser* and *hCursor* parameters are set to 0, **SqlGetStatus** simply tests for the presence of an active running Quasar Database Administrator. A value of 1 is returned if and only if the database administrator is running.

| Parameter | Type/Description |
|---|---|
| *hUser* | **HANDLE** Identifies the user who owns the cursor. The *hUser* parameter must have been created with the SqlLogin function. |
| *hCursor* | **HANDLE** Identifies the cursor for which the status is to be obtained. The *hCursor* parameter must have been created with the SqlCursorOpen function. |
| *lpSqlControl* | **LPSQLCONTROL** Points to an SQL Control Structure. Any errors that may occur cause the **szErrorDetail**, **szErrorMessage** and **nResultCode** fields of this structure to be loaded with descriptive information. |
| *lpSqlStatus* | **LPSQLSTATUS** Points to the target SQL Status Structure. |

**Return Value** The return value is 1 if no errors occurred, otherwise it is 0.

**Comments** Each cursor's status information is reset every time SqlExecute is called.

# Function: SqlLogin

**Syntax** **HANDLE FAR PASCAL SqlLogin(** *lpSqlControl* , *lpstrUserName* , *lpstrUserPassword* **)**

In order to access the SQL Database Administrator you must first identify yourself as an user with sufficient privilege to access the database. This is accomplished by calling the SqlLogin function with your user name and password. Once logged in, you can reference all your tables without specifying an user name.

To reference another user's table you must prefix the table name with the other user's user name. For example for user "JohnDoe" to select all columns of the system tables he must use:

```
SELECT * FROM SYSTEM.TABLES;
```

For user "SYSTEM" to select the "PartNumber" column of user "JohnDoe"'s table "Inventory" use the form:

```
SELECT JohnDoe.Inventory.PartNumber FROM JohnDoe.Inventory;
```

Or, more simply:

```
SELECT PartNumber FROM JohnDoe.Inventory;
```

| **Parameter** | **Type/Description** |
|---|---|
| *lpSqlControl* | **LPSQLCONTROL** Points to an SQL Control Structure. Any errors that may occur cause the **szErrorDetail**, **szErrorMessage** and **nResultCode** fields of this structure to be loaded with descriptive information. |
| *lpstrUserName* | **LPSTR** Points to a null terminated string containing the user name. |
| *lpstrUserPassword* | **LPSTR** Points to a null terminated string containing the user password. |

**Return Value** The return value is an user handle if no errors occurred while logging in, otherwise it is 0. You must use this user handle in all calls to the Quasar SQL API.

**Comments** Only those users recognized by the Quasar Database Administrator will be allowed to log in. When first installed, the Quasar Database Administrator only recognizes the user "SYSTEM" whose password is "QUASAR". You can create additional users by logging in as "SYSTEM" with password "QUASAR" and inserting records into the SYSTEM.USERS table. This table has three columns: USER_NAME, USER_PASSWORD and REMARK. To add user "JohnDoe" with password "IWantToRegisterNow" execute:

```
INSERT INTO USERS VALUES ('JohnDoe', 'IWantToRegisterNow', NULL);
```

John Doe can now login.

# Function: SqlLogout

**Syntax** **BOOL FAR PASCAL SqlLogout(** *hUser , lpSqlControl* **)**

This function logs the user off the system. If the Automatic Commit option flag is set in the **wFlags** field of the *lpSqlControl* parameter, any transaction still in progress is automatically committed; otherwise, all work performed by the user since logging in or since the last COMMIT WORK query (whichever occurred last) is rolled back.

| Parameter | Type/Description |
|---|---|
| *hUser* | **HANDLE** Identifies the user who wishes to log off. The *hUser* parameter must have been created with the SqlLogin function. |
| *lpSqlControl* | **LPSQLCONTROL** Points to an SQL Control Structure. Any errors that may occur cause the **szErrorDetail**, **szErrorMessage** and **nResultCode** fields of this structure to be loaded with descriptive information. |

**Return Value** The return value is 1 if no errors occurred, otherwise it is 0.

**Comments** While **SqlLogout** automatically closes all the user's cursor before logging the user off, we recommend explicitly closing cursor when they are no longer needed.

## Constants: Data Class

The following constants identify the general characteristics of a column in the result table. The **nDataClass** field of the SQLCOLUMN data structure is set to one of these values as a result of an SqlDescribeColumn function call.

| | |
|---|---|
| **mSqlDataClassAPPROXIMATE** | Indicates that the column is one of the approximate numeric data types |
| **mSqlDataClassBINARY** | Indicates that the column is of the binary data type |
| **mSqlDataClassCHAR** | Indicates that the column is one of the character string data types |
| **mSqlDataClassEXACT** | Indicates that the column is one of the exact numeric data types |
| **mSqlDataClassINVALID** | Indicates that the data type of the column is invalid |

## Constants: Data Type

The following constants identify the particular characteristics of a column in the result table. The **nDataType** field of the SQLCOLUMN data structure is set to one of these values as a result of an SqlDescribeColumn function call.

| | |
|---|---|
| **mSqlDataTypeBINARY** | Indicates that the data type of the column is BINARY: its length is given by the **nPrecision** field of the SQLCOLUMN data structure |
| **mSqlDataTypeCHAR** | Indicates that the data type of the column is CHAR: its length is given by the **nPrecision** field of the SQLCOLUMN data structure |
| **mSqlDataTypeDECIMAL** | Indicates that the data type of the column is DECIMAL: its precision and scale are given by the **nPrecision** and **nScale** fields of the SQLCOLUMN data structure |
| **mSqlDataTypeDOUBLE** | Indicates that the data type of the column is DOUBLE PRECISION: its precision is given by the **nPrecision** field of the SQLCOLUMN data structure |
| **mSqlDataTypeFLOAT** | Indicates that the data type of the column is FLOAT: its precision is given by the **nPrecision** field of the SQLCOLUMN data structure |
| **mSqlDataTypeINTEGER** | Indicates that the data type of the column is INTEGER: its precision is given by the **nPrecision** field of the SQLCOLUMN data structure |
| **mSqlDataTypeNUMERIC** | Indicates that the data type of the column is NUMERIC: its precision and scale are given by the **nPrecision** and **nScale** fields of the SQLCOLUMN data structure |
| **mSqlDataTypeREAL** | Indicates that the data type of the column is REAL: its precision is given by the **nPrecision** field of the SQLCOLUMN data structure |
| **mSqlDataTypeSMALLINT** | Indicates that the data type of the column is SMALLINT: its precision is given by the **nPrecision** field of the SQLCOLUMN data structure |
| **mSqlDataTypeVARCHAR** | Indicates that the data type of the column is VARCHAR: its maximum length is given by the **nPrecision** field of the SQLCOLUMN data structure |
| **mSqlDataTypeINVALID** | Indicates that the data type of the column is invalid |

## Constants: Data Format

When a record is fetched from the result table by means of one of the SqlFetch functions, the Quasar SQL API loads the record into a buffer provided by the user. The record is returned in one of three optional formats. The user selects the format he wants by setting the **wFlags** field of the SQLCONTROL structure to one (and only one) of the following values:

**mSqlFlagFormatPadded**

The record is returned as a single null terminated character string. The data for each column is contained within this string separated by spaces. The number of spaces is set by the user in the **nPadding** field of the SQLCONTROL structure. The length of the string is constant across all records and is available to the user in the **nRecordSize** field of the SQLTABLE data structure. The length of each column's portion of the string is available to the user in the **nFieldWidth** field of the SQLCOLUMN data structure.

**mSqlFlagFormatString**

The record is returned as a set of null terminated character strings, one for each column. The first column's string is loaded into the first bytes of the record buffer. The second column's string immediately follows the first and so on. Each string contains only the minimum number of characters to represent the data. That is, the strings are not padded.

**mSqlFlagFormatStructure**

The record is returned as a packed 'C' data structure. This format is not recommended for 'Visual Basic' users. Fields appear in the 'C' data structure in the same order in which they appear in the result table. Character types are converted to null terminated character strings padded with spaces to bring them to the length of **nFieldWidth** as found in the SQLCOLUMN data structure. Approximate and exact numeric types are converted to their 'double' representation. A record containing three fields: CHAR (10), INTEGER, DECIMAL(7,2) would appear as:

```
struct {
    char szField1[11];
    double dField2;
    double dField3;
}
```

## Constants: Automatic Commit

When a query is executed via an SqlExecute call, it is not committed and will be rolled back if the user should happen to log off.

**mSqlFlagAutoCommitOnLogout** Setting this bit of the **wFlags** field of the SQLCONTROL structure tells the Quasar SQL API to issue a commit command to the database administrator prior to logging the user off. This flag may be set in combination (using the '|' operator) with any one of **Data Format** flags.

## Constants: Name Length

While character string fields may be quite lengthy: names of users, tables and columns are limited.

| | |
|---|---|
| **mSqlMaximumNameLength** | This is the maximum length (not including the null terminator) of names of users, tables and columns. |

# Data Structures: SQLCOLUMN

**Result Table Column Description Record**

The **SQLCOLUMN** data structure describes the characteristics of a specified column of the result table. The **SQLCOLUMN** data structure is loaded as a result of an SqlDescribeColumn function call.

```
typedef struct tagSQLCOLUMN {
    BOOL        bRightJustified;
    char        szColumnName[mSqlMaximumNameLength+1];
    unsigned    nColumnSequenceNumber;
    unsigned    nDataClass;
    unsigned    nDataType;
    unsigned    nFieldWidth;
    unsigned    nPrecision;
    unsigned    nScale;
} SQLCOLUMN;
typedef SQLCOLUMN *PSQLCOLUMN;
typedef SQLCOLUMN far *LPSQLCOLUMN;
```

The **SQLCOLUMN** data structure has the following fields:

| Field | Description |
|---|---|
| **bRightJustified** | Indicates whether the column, by default, is left or right justified. Right justification is indicated by a non-zero value |
| **szColumnName** | Gives the column heading as a null terminated text string. |
| **nColumnSequenceNumber** | Column numbers range from 1 to the number of columns in the result table. |
| **nDataClass** | Identifies the general characteristics of a column in the result table. Its value will be set to one of the Data Class constants. |
| **nDataType** | Identifies the particular characteristics of a column in the result table. Its value will be set to one of the Data Type constants. |
| **nFieldWidth** | The number of character positions guaranteed to hold the value of this column when the record is fetched. If Data Format in the **wFlags** field of the *lpSqlControl* is **mSqlFlagFormatPadded**: **nFieldWidth** is set to the size of the column. If **Data Format** is **mSqlFlagFormatString**: **nFieldWidth** is set to the size of the column plus one for the terminating null. If **Data Format** is **mSqlFlagFormatStructure** and the column is numeric: **nFieldWidth** is set to sizeof(double), if the column is a character string type: **nFieldWidth** is set to the size of the column plus one for the terminating null. |
| **nPrecision** | If a numeric column, **nPrecision** is set to the precision (as defined by ANSI SQL) of the result. If a character field, **nPrecision** is set to the length (as defined by ANSI SQL) of the result. |
| **nScale** | If a numeric column, **nScale** is set to the scale (as defined by ANSI SQL) of the result. If a character field, **nScale** is 0. |

# Data Structures: SQLCONTROL

**Quasar SQL API Interface Control Structure**

The **SQLCONTROL** data structure contains the specifications necessary to provide an interface between the applications programmer and the Quasar SQL API.

```
typedef struct tagSQLCONTROL {
    char            szErrorDetail[65];
    char            szErrorMessage[257];
    unsigned        nHiLiteLength;
    unsigned        nHiLiteOffset;
    unsigned        nPadding;
    unsigned        nRecordBufferSize;
    unsigned        nResultCode;
    unsigned long   lRecordPosition;
    unsigned long   lReserved0
    unsigned long   lReserved1
    unsigned long   lReserved2
    unsigned long   lReserved3
    unsigned long   lReserved4
    WORD            wFlags;
} SQLCONTROL;
typedef SQLCONTROL *PSQLCONTROL;
typedef SQLCONTROL far *LPSQLCONTROL;
```

| Field | Description |
|---|---|
| **szErrorDetail** | Each time an Quasar SQL API function is called, any errors that may occur cause this field to be loaded with a descriptive string: "Error detected at line *n* of *module.c*". This information is useful if you need to call customer service. |
| **szErrorMessage** | Each time an Quasar SQL API function is called, any errors that may occur cause this field to be loaded with a descriptive string: "ERROR *xxx*: *text*." This provides a useful diagnostic while you are debugging your application. "*xxx*" is the ASCII representation (base 10) of the **nResultCode**. |
| **nHiLiteLength** | Each time the SqlExecute is called, any syntax errors within the query cause this field to be set to the length of the offending phrase within the query text. |
| **nHiLiteOffset** | Each time the SqlExecute is called, any syntax errors within the query cause this field to be set to the offset of the offending phrase within the query text. |
| **nPadding** | The caller of the Quasar SQL API sets this field to the number of padding spaces to be inserted between columns when the **wFlags** field of the **SQLCONTROL** structure is set to **mSqlFlagFormatPadded**. |
| **nRecordBufferSize** | The caller of the Quasar SQL API sets this field to the size (in bytes) of the record buffer. This buffer must be at least as large as the value in the **nRecordSize** field of the SQLTABLE data structure. The **SQLTABLE** data structure is available by calling SqlDescribeTable. |
| **nResultCode** | Indicates the basic status condition after any call to an Quasar SQL API function. A value of 0 indicates no error. Other values are listed under Result Codes |
| **lRecordPosition** | Indicates the position within the result table of the most recent record fetched. **lRecordPosition** can range from 1 to the |

| | |
|---|---|
| | number of records in the result table. If **lRecordPosition** is 0 then no records have been fetched. |
| **lReserved0** | Do not tamper with this field. |
| **lReserved1** | Do not tamper with this field. |
| **lReserved2** | Do not tamper with this field. |
| **lReserved3** | Do not tamper with this field. |
| **lReserved4** | Do not tamper with this field. |
| **wFlags** | The caller of the Quasar SQL API sets this field to the desired Data Format to be utilized during record fetch functions. The data format may be optionally combined using the '|' operator with the Automatic Commit flag. |

# Data Structures: SQLSTATUS

**Query Status Structure**

The **SQLSTATUS** data structure describes the overall outcome of the most recent SqlExecute operation for a given cursor.

```
typedef struct tagSQLSTATUS {
    unsigned long    lNumberOfRecordDeletes;
    unsigned long    lNumberOfRecordInserts;
    unsigned long    lNumberOfRecordSelects;
    unsigned long    lNumberOfRecordUpdates;
    unsigned long    lTimeElapsed;
} SQLSTATUS;
typedef SQLSTATUS *PSQLSTATUS;
typedef SQLSTATUS far *LPSQLSTATUS;
```

| Field | Description |
|---|---|
| **lNumberOfRecordDeletes** | Indicates the total number of records deleted by the most recent **SqlExecute**. |
| **lNumberOfRecordInserts** | Indicates the total number of records inserted by the most recent **SqlExecute**. |
| **lNumberOfRecordSelects** | Indicates the total number of records selected by the most recent **SqlExecute**. |
| **lNumberOfRecordUpdates** | Indicates the total number of records updated by the most recent **SqlExecute**. |
| **lTimeElapsed** | Indicates the elapsed time (in milliseconds) during the most recent **SqlExecute**. |

# Data Structures: SQLTABLE

**Result Table Description Record**

The **SQLTABLE** data structure describes the characteristics of the result table. The **SQLTABLE** data structure is loaded as a result of an SqlDescribeTable function call.

```
typedef struct tagSQLTABLE {
    unsigned            nNumberOfColumns;
    unsigned            nRecordSize;
    unsigned long lNumberOfRecords;
} SQLTABLE;
typedef SQLTABLE *PSQLTABLE;
typedef SQLTABLE far *LPSQLTABLE;
```

| Field | Description |
|---|---|
| **nNumberOfColumns** | Indicates the number of columns in the result table. |
| **nRecordSize** | Indicates the minimum size (in bytes) of a buffer large enough to hold a record fetched from the result table. |
| **lNumberOfRecords** | Indicates the number of records in the result table. |

# Result Codes Part I (Ambiguous Column - Data File Corrupted)

| | |
|---|---|
| **mSqlErrorAmbiguousColumn** | When a column name can be found in more than one of several tables given in the "FROM" clause of a query, the database administrator cannot determine which of them you mean. To remove the ambiguity, use the "Tablename.ColumnName" or the "UserName.Tablename.ColumnName" syntax. |
| **mSqlErrorBadEnvironment** | The DatabasePath entry in the "win.ini" file is not properly specified: it must begin with a drive identifier followed by a ':' and a '\'. A typical entry is "c:\database". |
| **mSqlErrorBinaryNotAllowed** | When creating a table, the data type BINARY was specified, the BINARY data type is not allowed in this release. |
| **mSqlErrorCannotActivateFile** | An attempt has been made to access a file which is unrecognized by the b-tree file manager. This is an internal error which should never occur. Please contact customer service. |
| **mSqlErrorCannotCloseFile** | When the b-tree file manager attempted to close either a data file or an index file, _lclose() returned an error. This usually indicates a problem with the DOS directory structure, we recommend you run 'chkdsk'. |
| **mSqlErrorCannotCloseLog** | When the database manager attempted to close either the 'or_log.dat' or the 'or_log.idx' transaction journal file, _lclose() returned an error. This usually indicates a problem with the DOS directory structure, we recommend you run 'chkdsk'. |
| **mSqlErrorCannotCreateDataFile** | When the b-tree file manager attempted to create a data file, _lcreat() returned an error. This usually indicates your setting for 'FILES' in the 'config.sys' file should be increased. |
| **mSqlErrorCannotCreateDumpFile** | When the dump manager attempted to create a dump file, _lcreat() returned an error. This usually indicates your setting for 'FILES' in the 'config.sys' file should be increased. |
| **mSqlErrorCannotCreateFile** | This result code will not occur. |
| **mSqlErrorCannotCreateKeyFile** | When the b-tree file manager attempted to create a key file, _lcreat() returned an error. This usually indicates your setting for 'FILES' in the 'config.sys' file should be increased. |
| **mSqlErrorCannotCreateLog** | When the database manager attempted to create either the 'or_log.dat' or the 'or_log.idx' transaction journal file, _lcreat() returned an error. This usually indicates your setting for 'FILES' in the 'config.sys' file should be increased. |
| **mSqlErrorCannotDeleteFile** | When the b-tree file manager attempted to remove either a data file or a key file, remove() returned an error. This usually indicates someone has modified the file's attribute settings. Please be sure none of the database files are marked as 'read only', 'hidden' or 'system'. |
| **mSqlErrorCannotDropTable** | When the database manager attempted to drop a table, its 'use count' was greater than zero! This is an internal error which should never occur. Please contact customer service. |
| **mSqlErrorCannotOpenDataFile** | When the b-tree file manager attempted to open a data file, _lopen() returned an error. This usually indicates someone has modified the file's attribute settings or deleted the file. Please be sure none of the database files are marked as 'read only', 'hidden' or 'system'. Also, delete files only via SQL. |
| **mSqlErrorCannotOpenFile** | This result code will not occur. |

| | |
|---|---|
| **mSqlErrorCannotOpenKeyFile** | When the b-tree file manager attempted to open a key file, _lopen() returned an error. This usually indicates someone has modified the file's attribute settings or deleted the file. Please be sure none of the database files are marked as 'read only', 'hidden' or 'system'. Also, delete files only via SQL. |
| **mSqlErrorCannotOpenLog** | When the database manager attempted to open either the 'or_log.dat' or the 'or_log.idx' transaction journal file, _lopen() returned an error. This usually indicates your setting for 'FILES' in the 'config.sys' file should be increased. |
| **mSqlErrorCannotRemoveTable** | When the database manager attempted to delete a file during a 'Toolbox Remove' database operation, remove() returned an error. This usually indicates someone has modified the file's attribute settings. Please be sure none of the database files are marked as 'read only', 'hidden' or 'system'. |
| **mSqlErrorCharNotAllowed** | When the bind manager attempted to initialize an AVG() or SUM() set function, it discovered that the argument was a CHAR or VARCHAR column! |
| **mSqlErrorCheckpointDetected** | This result code will not occur. |
| **mSqlErrorDataFileCorrupted** | The b-tree file manager encountered illegal settings in the header portions of a data file. It is likely that your DOS files are corrupted. |

## Result Codes Part II (Database Corrupt - Invalid Cursor)

**mSqlErrorDatabaseCorrupt** The database manager encountered illegal settings in the header portions of the transaction journal file. It is likely that your DOS files are corrupted.

**mSqlErrorDbaNotAvailable** The Quasar SQL API could not establish communication with the Quasar Database Administrator. Is the database administrator running?

**mSqlErrorDivideByZero** During query execution a divide by zero error occurred.

**mSqlErrorDuplicateColumn** During query execution a column is called out more than once in a list where it can appear only once. This error can only occur in the CREATE TABLE statement and the INSERT statement.

**mSqlErrorDuplicateRecord** The b-tree file manager detected an attempt to violate an UNIQUE constraint.

**mSqlErrorExistingDataFile** When the b-tree file manager attempted to create a data file, it discovered the file already existed! Delete files only via SQL.

**mSqlErrorExistingKeyFile** When the b-tree file manager attempted to create a key file, it discovered the file already existed! Delete files only via SQL.

**mSqlErrorFileReadFailed** When the b-tree file manager attempted to read a file, _lread() returned an error.

**mSqlErrorFileSeekFailed** When the b-tree file manager attempted to seek a position in a file, _llseek() returned an error.

**mSqlErrorFileWriteFailed** When the b-tree file manager attempted to write a file, _lwrite() returned an error.

**mSqlErrorGlobalLockFailed** When the query manager attempted to lock a global resource, GlobalLock() or LockResource() returned an error condition.

**mSqlErrorGlobalUnlockFailed** When the query manager attempted to unlock a global resource, GlobalUnlock() or UnlockResource() returned an error condition.

**mSqlErrorImportParseError** This result code will not occur.

**mSqlErrorIncompatibleTypes** An attempt was made within the query to perform an operation between two arguments whose types conflict. An example is adding a text string to a number.

**mSqlErrorIndexExists** An attempt was made within the query to create an index whose name is already used.

**mSqlErrorIndexUndoFailed** When the b-tree file manager attempted to create an unique key value a duplicate was found. The attempt to undo the creation failed. This is an internal error which should never occur. Please contact customer service.

**mSqlErrorInitializationFailed** When the b-tree file manager attempted to initialize its tables, a failure was detected. This is an internal error which should never occur. Please contact customer service.

**mSqlErrorInsufficientBuffer** The record to be returned to the Quasar SQL API user will not fit in the buffer allocated by the user. That is, the value of the **nRecordSize** field of the SQLTABLE structure is larger than that supplied by the user as indicated by the **nRecordBufferSize** field of the SQLCONTROL structure. Please be sure to allocate buffers large enough to accommodate **nRecordSize** bytes.

**mSqlErrorInternalError** This is an internal error which should never occur. Please contact customer service.

**mSqlErrorInvalidColumnSpec** The arguments of an SQL LIKE predicate must be of a character string type.

| | |
|---|---|
| **mSqlErrorInvalidColumnType** | This is an internal error which should never occur. Please contact customer service. |
| **mSqlErrorInvalidCursor** | A call was made to the Quasar SQL API where the **pCursor** field of the SQLCONTROL structure is set to a value which does not match any existing cursor. |

## Result Codes Part III (Invalid Data - NULL Not Allowed)

| | |
|---|---|
| **mSqlErrorInvalidData** | During the execution of a CREATE TABLE query, the scale of a numeric column exceeds its precision. |
| **mSqlErrorInvalidEscapeChar** | The data type of the escape character in a LIKE predicate is not a character string type. |
| **mSqlErrorInvalidPattern** | The pattern of an SQL LIKE predicate must not be NULL. |
| **mSqlErrorInvalidUser** | The SqlLogin function was called with an **lpstrUserName** or **lpstrUserPassword** argument whose length exceeded mSqlMaximumNameLength. |
| **mSqlErrorKeyFileCorrupted** | The b-tree file manager encountered illegal settings in the header portions of a key file. It is likely that your DOS files are corrupted. |
| **mSqlErrorListSizesUnequal** | The number of columns in the column list of an SQL INSERT query does not match the number of values in the value list or the number of fields selected in the subquery. |
| **mSqlErrorMultipleSelects** | Only one SELECT statement may occur within a query. |
| **mSqlErrorNoColumnExists** | The bind manager encountered a column specification which it could not match to a table defined within the current scope. |
| **mSqlErrorNoIndexExists** | A DROP INDEX was attempted when no index exists. |
| **mSqlErrorNoOldRecord** | When the b-tree file manager attempted to carry out an SqlFetchNext or SqlFetchPrevious function, it discovered there was no current record. You must first execute an SqlFetchFirst, SqlFetchLast or SqlFetchPositioned. |
| **mSqlErrorNoTableExists** | During execution of a query a table name was encountered which cannot be found in the database under the currently logged-on user name or under the explicitly declared user name. |
| **mSqlErrorNoUserExists** | During execution of a query an user name was encountered which cannot be found in the database. |
| **mSqlErrorNodeSizeTooSmall** | This is an internal error which should never occur. Please contact customer service. |
| **mSqlErrorNotAggregate** | While the select list of a SELECT statement started with aggregate types (i.e. SUM()), a select list element was later encountered which was a simple type (i.e. a column specification). Aggregate types and simple types cannot be mixed unless the simple type is a grouping column. |
| **mSqlErrorNotGroupingColumn** | A regular column specification was encountered where a grouping column was required. |
| **mSqlErrorNotInGroupedTable** | While the select list of a SELECT statement started with simple types (i.e. a column specification), a select list element was later encountered which was an aggregate type (i.e. SUM()). Simple types and aggregate types cannot be mixed unless the simple type is a grouping column. |
| **mSqlErrorNotSingleRecord** | A subquery returned more than one record. When a subquery is used as an argument to a comparison predicate it must return a single record. |
| **mSqlErrorNullInNotNullCol** | An attempt was made to insert a NULL value in a column which was declared as NOT NULL. |
| **mSqlErrorNullNotAllowed** | An attempt was made to use a NULL value where NULL values are not permitted. |

# Result Codes Part IV (Parser Syntax Error - Wrong Version)

| | |
|---|---|
| **mSqlErrorParserSyntaxError** | There is a syntax error in the SQL statement. |
| **mSqlErrorParserStackOverflow** | This is an internal error which should never occur. Please contact customer service. |
| **mSqlErrorPrecisionConflict** | A mathematical operation was requested which would result in loss of precision. |
| **mSqlErrorResultTableExists** | An attempt was made to commit a transaction (COMMIT WORK) while a result table was still open. Users must close all cursors which have an associated result table before committing work. |
| **mSqlErrorScaleOverflow** | The scale of the result of a mathematical operation overflowed. |
| **mSqlErrorSecurityViolation** | SqlLogin was called with an invalid **lpstrUserName** or **lpstrUserPassword**. |
| **mSqlErrorSetFunctionsNested** | Set functions may not be nested. |
| **mSqlErrorStarNotAllowed** | The SELECT * syntax is illegal when used with the GROUP BY clause. |
| **mSqlErrorSubqueryNotOneColumn** | When a subquery is used as an argument to a predicate it must return a single field in the select list. |
| **mSqlErrorTableConflict** | The target table of an INSERT, DELETE or UPDATE query may not also appear in a 'from' clause of a subquery. |
| **mSqlErrorTableExists** | An attempt was made in a CREATE TABLE query to create a table which already exists. |
| **mSqlErrorTooManyColumnNames** | The list of column names in an INSERT statement exceeds the number of columns in the table. |
| **mSqlErrorTooManyTablesOpen** | An attempt was made to open more tables than the maximum number of tables which may be open at one time. |
| **mSqlErrorTrialSizeExceeded** | The number of events recorded in the log has exceeded the trial size allotment. Further operation will irrecoverably corrupt your data. |
| **mSqlErrorUserNotLoggedIn** | A call was made to the Quasar SQL API where **hUser** does not match an existing user. |
| **mSqlErrorVariableHeaderMark** | The b-tree file manager encountered illegal settings in the header portions of a data file. It is likely that your DOS files are corrupted. |
| **mSqlErrorVariableHeaderSize** | The b-tree file manager attempted to place a record in a position which was not large enough. This is an internal error which should never occur. Please contact customer service. |
| **mSqlErrorVariableHeaderWrong** | The b-tree file manager encountered illegal settings in the header portions of a data file. It is likely that your DOS files are corrupted. |
| **mSqlErrorVariableHeaderZero** | The b-tree file manager encountered illegal settings in the header portions of a data file. It is likely that your DOS files are corrupted. |
| **mSqlErrorWrongVersion** | The b-tree file manager has determined that your data and key files are stamped with a version which is incompatible with the current release. |

b-tree file manager

The Quasar Database Administrator is made of several large function units. One of these is the b-tree file manager. It is responsible for carrying out low level data access and operates directly on the raw DOS files located in the database directory.

bind manager

The Quasar Database Administrator is made of several large function units. One of these is the bind manager. It is responsible for associating the logical constructs in the query with actual users, tables and columns in the database.

cursor

The Quasar SQL API keeps track of the result of a query by means of a *cursor*. The cursor may include status information, the table where the result of a SELECT may be found, the column descriptions for such a table, the current record position for fetch operations, etc.

database manager

The Quasar Database Administrator is made of several large function units. One of these is the database manager. It is responsible for carrying out high level data access and operates on logical tables and columns. The database manager uses the b-tree file manager to access records in the raw DOS files located in the database directory. The database manager is also responsible for policing transactions and the transaction journal (or 'log').

dump manager

The Quasar Database Administrator is made of several large function units. One of these is the dump manager. It is responsible for creating the dump file.

physical position

Physical position is the raw position of the record in the file. Physical order is not the same as the order you specify with the SQL ORDER BY clause.

query

By *query* we mean any SQL statement.

query manager

The Quasar Database Administrator is made of several large function units. One of these is the query manager. It is responsible for parsing the query into its component parts.

record buffer

One of the arguments to the Quasar SQL API fetch functions is a pointer to a record buffer. This is the buffer which receives the formatted contents of a record.

result table

All SELECT queries generate a table containing the chosen records. This is called the *result table*. Its contents are made available to you one at a time through one of the Quasar SQL API fetch functions.

scroll bar

A bar that appears at the right and/or bottom edge of a window whose contents aren't completely visible. Each scroll bar contains two scroll arrows and a scroll box, which allow you to scroll within the window or list box.

transaction journal

A transaction journal is a pair of files ('or_log.dat' and 'or_log.idx') written by the database manager. The transaction journal contains a record for every event which caused a change to the database.