REXX interface to SQL databases

Version 1.3

22 February 1996

Table of Contents

---

1. Introduction

This document defines an interface to provide access to SQL databases for REXX programs. REXX/SQL consists of a number of external REXX functions which provide the necessary capabilities to connect to, query and manipulate data in any SQL database. This document is designed to assist in the implementation of this interface for any SQL-based database system that provides an appropriate 3GL API.

An appendix to this document is included for each implementation of this interface providing implementation-specific features. Where implementations may differ, this is highlighted in the function definitions to assist the user where source code compatibility between different database vendors is required.

---

2. Overview

REXX/SQL consists of REXX external functions that allows a REXX program to communicate with a SQL database.

Actions requested of the database are made by calling these external functions. Information returned to the REXX program as a result of these actions is done principally through the REXX variable pool.

The REXX external functions are:

- **SQLCONNECT** - connect to the SQL database
- **SQLDISCONNECT** - break the connection to the SQL database made by SQLCONNECT
- **SQLDEFAULT***(1)* - switch the default connection to another open connection
- **SQLCOMMAND** - issue a SQL statement to the connected database
- **SQLPREPARE** - allocate a work area for a SQL statement and prepare it for processing
- **SQLDISPOSE** - deallocate a work area for a statement
- **SQLOPEN** - open a cursor for a prepared SELECT statement
- **SQLCLOSE** - close an opened cursor
- **SQLFETCH** - fetch the next row from the open cursor
- **SQLEXECUTE** - execute a prepared statement
- **SQLCOMMIT** - commit the current transaction
- **SQLROLLBACK** - rollback the current transaction
- **SQLDESCRIBE***(1)* - describe expressions from a SELECT statement
- **SQLVARIABLE***(2)* - set or retrieve default run-time values

*(1)*Functions may not be supported in all implementations.
*(2)*Values that can be set can vary between implementations.

Status values set by the REXX external functions are:

- **SQLCA.SQLCODE** - result code of last SQL operation
- **SQLCA.SQLERRM** - text of any error message associated with the above result code
- **SQLCA.SQLTEXT** - text of the last SQL statement
- **SQLCA.ROWCOUNT** - number of rows affected by the last SQL operation
- **SQLCA.FUNCTION** - name of the REXX external function last called
- **SQLCA.INTCODE** - REXX/SQL interface error number
- **SQLCA.INTERRM** - text of last REXX/SQL interface error

3. Functions

This section provides the full syntax and usage of each function that comprises REXX/SQL.

SQLCONNECT([*connection name*], [*username*], [*password*], [*database*], [*host*])

Establishes a connection to the database server. The newly established connection is made the default database connection.

Arguments:

connection name This is an optional name for the connection to be opened. If you need to have multiple connections opened at once, you will need to specify a connection name. For those implementations that do not support multiple connections, this argument is not supported.

username This is the name used to connect to the database.

password This is the password associated with the *username*.

database This is the name that the database to which connection is required is known.

host This is the name of the host on which the *database* resides. The format of this host string depends on the database vendor and operating system.

Some arguments may be mandatory depending on the platform. See the appendices for more details.

Returns:

success: zero
failure: a negative number

SQLDISCONNECT([*connection name*])

Closes a connection with the database server. All open cursors for the database connection are closed. All allocated work areas for the database connection are deallocated.

Arguments:

connection name An optional connection name, as specified in the **SQLCONNECT** function. If no connection name is specified, the default (and only) connection is disconnected.

Returns:

success: zero
failure: a negative number

SQLDEFAULT([*connection name*])

Sets the default database connections to be that which is specified or if no connection name is specified, the current connection name is returned.

Arguments:

connection name An optional connection name specifying the database connection to be made the default connection.

Returns:

with no argument:
the name of the current database connection or an empty string if no database connection is current.
with an argument:
success: zero
failure: a negative number

SQLCOMMAND(*statement name*,*sql statement*[,*bind1*[,*bind2*[,...[,*bindN*]]]])

Executes an SQL statement as a single step. The statement is executed in the default work area for the default database connection. No bind values may be passed for **DDL** statements. Bind values may optionally be passed for **DML** statements.

Arguments:

statement name A name to identify the *sql statement* and used to name the compound variable created when *sql statement* is a SELECT statement. The results of the SELECT statement are returned in compound variables with this name as the stem.

sql statement Any valid **DDL** or **DML** statement. For **DML** statements, the statement may contain placemarkers to which values may be bound. **The format of these placemarkers is implementation dependant.**

bind1...bindN Values supplied to bind to the placemarkers.
The format of bind values is implementation dependant.

Returns:

success: zero
failure: a negative number

When the *sql statement* is a SELECT, all column values are returned as REXX *arrays*. The compound variable name is composed of the statement name followed by a period, followed by the column name specified in the SELECT statement, followed by a number corresponding to the row number. As with all REXX arrays, the number of elements in the array is stored in the zeroth element. If no *statement name* is specified, a default string is used; usually **SQL**. See Section 5. Implementation Notes for information when this is not the case.

If the column selected consists of a constant, or includes a function, a valid REXX variable may not be able to be generated. See the implementation specific sections for details on how each implementation handles this.

After a successful **DML** statement, the variable **SQLCA.ROWCOUNT** is set to the number of rows affected by the statement.

Because the contents of all columns for all rows are returned from a SELECT statement, the statement may return many rows and exhaust available memory. Therefore, the use of the **SQLCOMMAND** function should be restricted to queries that return a small number of rows. For larger queries, use a combination of **SQLPREPARE**, **SQLOPEN**, **SQLFETCH** and **SQLCLOSE**.

Example:

```
rc = sqlcommand(s1,"select ename, empno from emp")
```

If the SELECT statement returns 3 rows then:

- S1.ENAME.0 = 3
- S1.ENAME.1 = "SCOTT"
- S1.ENAME.2 = "SMITH"
- S1.ENAME.3 = "BROWN"
- S1.EMPNO.0 = 3
- S1.EMPNO.1 = "1234"
- S1.EMPNO.2 = "1437"
- S1.EMPNO.3 = "1555"

SQLPREPARE(*statement name*,*sql statement*)

Allocates a work area to a SQL statement and prepares the statement for processing.

If the statement is **DDL** then the statement is executed in this, the preparation step.

If the statement is a **DML** statement then it must be executed by a subsequent call. For INSERT, UPDATE and DELETE commands, the statement must be executed by calling **SQLEXECUTE**. For a SELECT command, the statement must be executed as a cursor. This requires calling **SQLOPEN** followed by multiple calls to **SQLFETCH** and optionally calling **SQLCLOSE**.

Arguments:

statement name A name to identify the *sql statement*.

sql statement Any valid **DDL** or **DML** statement. For **DML** statements, the statement may contain placemarkers to which values may be bound. **The format of these placemarkers is implementation dependant.**

Returns:

success: zero
failure: a negative number

SQLDISPOSE(*statement name*)

Deallocates a work area from a statement and frees all internal resources associated with the statement. If a cursor is open for the nominated statement an implicit close is issued.

Arguments:

statement name A name to identify the *sql statement* to be disposed.

Returns:

success: zero
failure: a negative number

SQLOPEN(*statement name*[,*bind1*[,*bind2*[,...[,*bindN*]]]])

Opens a cursor for the nominated statement. The statement must be a query (a SELECT statement) and must have been prepared prior to opening (with **SQLPREPARE**). Opening the cursor, binds any supplied values to the corresponding placemarkers and then executes the SELECT statement. The first row is made ready to be fetched. If a cursor was already open for the named statement then it will be automatically closed prior to reopening the cursor.

Arguments:

statement name A name to identify the *sql statement*.

bind1...bindN Values supplied to bind to the placemarkers. **The format of bind values is implementation dependant.**

Returns:

success: zero
failure: a negative number

SQLCLOSE(*statement name*)

Ends execution of a cursor. This frees much of the database server resources associated with a cursor. The statement does not have to be reparsed if the cursor is later reopened unless the statement has been disposed (ie by calling **SQLDISPOSE** for the *statement name*).

Arguments:

statement name A name to identify the *sql statement*.

Returns:

success: zero
failure: a negative number

SQLFETCH(*statement name*,[*number rows*])

Fetches the next row (or rows) for the nominated statement. There must be an open cursor for the named statement. If the optional *number rows* is not specified, a *single row* fetch is carried out, otherwise a multi row fetch is carried out.

For *single row* fetches, a compound variable is created for each column name identified in the *sql statement* parsed in the **SQLPREPARE** call, with the stem being *statement name* and the tail corresponding to the column name.

For *multi row* fetches, a REXX *array* is created for each column name in the *sql statement* parsed in the SQLPREPARE call. See **SQLCOMMAND** for a full description of the format of the variables. Variable tails always start with 1.

Arguments:

statement name A name to identify the *sql statement*.

number rows An optional number specifying how many rows are to be fetched.

Returns:

success: a number >= zero. a value of zero indicates no more rows are available to be fetched. for *single row* fetches, a value > zero represents the row number of the row just fetched. for *multi row* fetches, a value > zero indicates the number of rows fetched. Normally this value equals number rows. If this value is less than number rows, no more rows are available to be fetched. This value can never be greater than number rows. The variable **SQLCA.ROWCOUNT** is set to the value returned.
failure: a negative number

SQLEXECUTE(*statement name*[,*bind1*[,*bind2*[,...[,*bindN*]]]])

Executes a prepared statement for non-SELECT **DML** statements (i.e. INSERT, UPDATE and DELETE).

Arguments:

statement name A name to identify the *sql statement*.

bind1...bindN Values supplied to bind to the placemarkers. **The format of bind values is implementation dependant.**

Returns:

success: zero
The variable **SQLCA.ROWCOUNT** is set to the number of rows affected by the **DML** statement executed.
failure: a negative number

SQLCOMMIT()

Commit the current transaction.

Arguments:

none

Returns:

success: zero
failure: a negative number

SQLROLLBACK()

Rollback the current transaction.

Arguments:

none

Returns:

success: zero
failure: a negative number

SQLDESCRIBE(*statement name* [,*stem name*])

Describes the expressions returned by a SELECT statement. The statement should first be prepared (with **SQLPREPARE**) and then described. Creates a compound variable for each column in the select list of the sql statement, with a stem equal to the *statement name*, followed by 'COLUMN' and with at least the following components: NAME, TYPE, SIZE, PRECISION, SCALE, NULLABLE.
See the platform-specific appendix for other variables returned.

Arguments:

statement name A name to identify the *sql statement*.

stem name An optional name specifying the stem name of the REXX variables created.

Returns:

success: a positive number, or zero, indicating the number of expressions in the select list of the SELECT statement
failure: a negative number

Example:

```
rc = sqlprepare(s2,"select ename, empno from emp")
rc = sqldescribe(s2,"AA")
```

results in the following REXX variables being set:

- AA.COLUMN.NAME.1 == "ENAME"
- AA.COLUMN.NAME.2 == "EMPNO"
- AA.COLUMN.TYPE.1 == "VARCHAR2"
- AA.COLUMN.TYPE.2 == "NUMBER"
- AA.COLUMN.SIZE.1 == "20"
- AA.COLUMN.SIZE.2 == "6"
- AA.COLUMN.PRECISION.1 == "20"

- AA.COLUMN.PRECISION.2 == "40"
- AA.COLUMN.SCALE.1 == "0"
- AA.COLUMN.SCALE.2 == "0"
- AA.COLUMN.NULLABLE.1 == "1"
- AA.COLUMN.NULLABLE.2 == "0"

The values returned are implementation dependant.

SQLVARIABLE(*variable name*[,*variable value*])

Set or get the value for the specified variable.

The following variables are available in all implementations:

- **VERSION** *(readonly)* the version of REXXSQL, consisting of:
  - *package name* - usually **REXXSQL**
  - *REXXSQL version* - numerical version; eg. 1.0
  - *REXXSQL date* - REXX standard date format; eg. 10 Jun 1995
  - *OS platform* - current operating system
  - *database platform* - type of the current database

  eg. REXXSQL 1.0 10 Jun 1995 OS/2 ORACLE

- **DEBUG** *(setable)* level of debugging requested.
  - *0* - no debugging information displayed (defualt)
  - *1* - REXX variables displayed as set
  - *2* - function entry/exit information displayed
  - *3* - both level 1 and 2 debugging information displayed

- **ROWLIMIT** *(setable)*
  this is used to limit the number of rows fetched by a SELECT statement passed to **SQLCOMMAND**. A value of zero indicates no limit. The default value is zero.

- **SAVESQL** *(setable)*
  this is used to indicate if the text of the last SQL statement is to be saved. If this variable is set to 1, then **SQLCA.SQLTEXT** will have the value of the last SQL statement; if set to 0 **SQLCA.SQLTEXT** will equal "". The default for this variable is 1.

Arguments:

variable name The name of the variable to be set or retrieved. **The names of variables may be implementation dependant.**

variable value If no *variable value* is specified, the current value of the variable is returned. If a *variable* *value* is specified, the variable assumes the value specified.

Returns:

with *variable value* specified: zero if a valid *variable name* specified and it is able to be set; a negative number if the *variable name* is invalid or the *variable name* is not able to be set.
with *variable value* NOT specified: the current value of the variable or a negative number if the *variable name* is invalid.

SQLLOADFUNCS()

Load all REXX external functions making them available for use.

This function only available in dynamic library implementations.

Arguments:

none

Returns:

success: zero
failure: a negative number

SQLDROPFUNCS()

Terminate REXX/SQL and free up all resources used.

This function only available in dynamic library implementations.

Arguments:

none

Returns:

success: zero
failure: a negative number

---

4. Errors

All functions return a negative number if an error occurred. Zero or positive return values indicate success.

When an error occurs in the REXX/SQL interface, the function returns a negative number corresponding to one of the numbers below and the variable **SQLCA.INTCODE** is set to that number. The variable **SQLCA.INTERRM** is also set to the corresponding message. If a database error occurs, **SQLCA.SQLCODE** and **SQLCA.SQLERRMR** are set to the appropriate values.

Internal Errors:

```
-1  - Database Error
-7  - value is not a valid integer.
-8  - internal error
-9  - no message available for SQLCODE n
-10 - out of memory
-11 - unknown variable variable.
-12 - variable variable is not settable.
-13 - statement statement is not a query.
-14 - num-rows is not a valid integer.
-15 - Conversion/truncation occurred.
-15 - unable to set REXX variable
-18 - extraneous argument - argument
-19 - null ("") variable name.
-20 - connection already open with name connection.
-21 - connection connection is not open.
-22 - no connections open.
-23 - statement name omitted or null
-24 - statement statement does not exist
-25 - no connection is current
-26 - statement has not been opened or executed
-51 - zero length identifier
-52 - garbage in identifier name
-61 - n bind variables passed. m expected
-62 - bind values must be paired for bind by name
-63 - invalid substitution variable name at bind pair n.
-71 - Too many columns specified in SELECT
-75 - no database name supplied
```

---

5. Implementation Notes

To enable multiple database access on those platforms that support the dynamic loading of REXX external functions, implementation-specific function names and status values should be provided as a compile-time option. It is expected that a separately built library be provided with the *standard* function names together with the a library containing the database platform-specific functions and status values.

For example, the OS/2 Oracle implementation provides a dynamic library called **REXXSQL** which contains the *standard* function names like **SQLCONNECT** and *standard* status values like **SQLCA.SQLCODE**. It also provides an implementation-specific dynamic library called **REXXORA** with an equivalent **ORACONNECT** and **ORACA.SQLCODE**. This use of standard and implementation specific names also applies to default statement names and stem variable names. Basically, wherever the string SQL appears in function names or REXX variables names, an implementation specific abbreviation will be used.

This provision of database platform specific external functions will enable access to different vendor databases in the one REXX program.

The following database-specific abbreviations are recommended:

- **ORA** Oracle
- **ING** Ingres
- **DB2** IBM DB2
- **WAT** Watcom
- **SYB** Sybase
- **MIN** Mini SQL (mSQL)

---

6. Using REXX/SQL

SQL statements fall into two broad categories **DDL** and **DML**. **DDL** is Data Definition Language. These are statements like CREATE TABLE, DROP INDEX. DML statements are Data Manipulation Language statements of which there are two forms; queries (SELECT statements) and data modification statements (INSERT, UPDATE and DELETE statements).

To execute any SQL statement the program must first connect to a database server.

Each statement must be executed in a work area or context area.

For **DDL** statements, the underlying steps are:

- allocate a work area
- parse (prepare) the statement (this also executes it if it is **DDL**)
- release any resources

For **DML** data modification statements, the underlying steps are:

- allocate a work area
- parse (prepare) the statement
- bind any required values to the placemarkers (if any)
- execute the statement
- release any resources

For **DML** query statements, the underlying steps are:

- allocate a work area
- parse (prepare) the statement
- bind any required values to the placemarkers (if any)
- execute the statement

- fetch each row until end of selection (or done)
- release any resources

Since there is a reasonable overhead in allocating work areas and in parsing statements these should be minimised. The REXX/SQL interface provides the means of doing this. The **SQLPREPARE** function allocates a work area to a statement and parses the statement. Work areas are deallocated from a statement when the **SQLDISPOSE** call is issued. While a statement is allocated to a work area it remains prepared (that is parsed and optimised). Because statement names are global, preparing a different statement with the same name as an existing statement disposes the existing one. After a statement has been prepared with **SQLPREPARE**, it is bound to a work area and remains bound until the statement is disposed of with **SQLDISPOSE**. The statement can be executed many times by the following means:

- Queries - repeatedly opening and closing the cursor using the functions; **SQLOPEN**, **SQLFETCH** and **SQLCLOSE**. Typically, multiple calls are made to **SQLFETCH** to retrieve all rows selected in the cursor. **SQLCLOSE** is optional.

- Data modification statements - repeatedly calling **SQLEXECUTE**. Each call may supply new bind values. The statement is not reparsed each time.

DDL statements are a special type of statement in that they are executed when they are parsed. Thus, **SQLPREPARE** both parses the **DDL** statement and executes it in one go. For **DDL** statements, the statement must be reparsed each time it requires execution. The work area can be reused by using the same statement name each time, eg.

```
rc = sqlprepare("MY_GRANT","grant select on emp to scott")
rc = sqlprepare("MY_GRANT","grant select, insert, update on dept to scott")
```

In the above example, the first statement is parsed and executed in the work area allocated to the statement **'MY_GRANT'**. The statement is reused to execute the second statement. There is no need to dispose statements before reuse of the same name.

The following table shows the order in which the database functions are to be called for the different types of SQL statements.

DML		**DDL**	
SELECT	INSERT,DELETE etc.	CREATE,DROP etc.	DESCRIBE
SQLPREPARE	SQLPREPARE	SQLPREPARE	SQLPREPARE
SQLOPEN	SQLEXECUTE	SQLDISPOSE	SQLDESCRIBE
SQLFETCH (in loop)	SQLDISPOSE		SQLDISPOSE
SQLCLOSE			

Dynamic Library Implementations

The REXX external functions in the dynamic library need to be loaded by a call to RxFuncAdd() followed by a call to SqlLoadFuncs(). eg.

```
Call RXFuncAdd 'SqlLoadFuncs','REXXSQL','SqlLoadFuncs'
Call SqlLoadFuncs
```

Before exiting from a REXX/SQL program, call the SqlDropFuncs() function. This call does not deregister the external functions, rather it frees up all resources used by the current program.

Appendix A - REXX/SQL for Oracle

This section describes features of REXX/SQL specific to the Oracle implementation.

General:

- All arguments to **SQLCONNECT** are optional.
- Examples of different connections with **SQLCONNECT**:

  The following connects to the database running on the local machine as *SCOTT* with password *TIGER*.

  ```
  rc = sqlconnect(,"scott","tiger")
  ```

  The following connects to the database identifed by the SQL*Net V2 entry; *XYZ.WORLD* as *SCOTT* with password *TIGER* with a connection name of *MYCON*.

  ```
  rc = sqlconnect("MYCON","scott","tiger",,"XYZ.WORLD")
  ```

  The following connects to the database running on the local machine as an externally identified user.

  ```
  rc = sqlconnect()
  ```

- If, when the first call to **SQLCOMMAND** or **SQLPREPARE** is made, the user is not connected to a database, an implicit SqlConnect() is made.

Bind Variables:

REXX/SQL for Oracle can use two forms of placemarkers for bind variables; numbers and names.

Bind by number:

The placemarkers in the *sql statement* are numeric; :1, :2 etc. The arguments passed to the **SQLCOMMAND** and **SQLOPEN** functions for bind values consist of a '#' followed by the bind values. eg.

```
query1 = "select name from emp where id = :1 and deptno = :2"
rc = sqlcommand(q1,query1,"#",345,10)
```

Bind by name:

The placemarkers in the *sql statement* are named; :ID, :DEP. The arguments passed to the **SQLCOMMAND** and **SQLOPEN** functions are pairs of placemarker name and bind variable value.eg.

```
query1 = "select name from emp where id = :ID and deptno = :DEP"
rc = sqlcommand(q1,query1,":ID",345,":DEP",10)
```

Column names:

If a column specification in a SQL statement passed to **SQLCOMMAND** or **SQLPREPARE** contains a function or is a constant, the column specifier *must* be aliased so that a valid REXX variable can be generated for that column.

SQLDESCRIBE variables:

The Oracle implementation does not include any extra variable components.

Appendix B - REXX/SQL for mSQL

This section describes features of REXX/SQL specific to the mSQL implementation.

General:

- The *database name* argument in **SQLCONNECT** is mandatory.
- Examples of different connections with **SQLCONNECT**:

  The following connects to the *TEST* database running on the local machine with a connection name of *MYCON*.

  ```
  rc = sqlconnect("MYCON",,,"TEST")
  ```

The following connects to the *MINERVA* database running on the machine *xyz.my.org*.

```
rc = sqlconnect(,,,"MINERVA","xyz.my.org")
```

The database name argument is mandatory in mSQL.

- As mSQL has no concept of a transaction, the functions, **SQLCOMMIT** and **SQLROLLBACK** don't do anything. They are included for consistency.
- All statements are actually executed by **SQLPREPARE**. This obviates the **SQLEXECUTE** function, but it still should be used for portability.
- The variable **SQLCA.ROWCOUNT** always returns 0 for non **SELECT DML** statements. Thus, there is no way of determining how many rows were deleted, updated, or inserted.

Bind Variables:

mSQL has no provision for bind variables in SQL statements. Hence, any references to bind variables in this document should be ignored.

Column names:

If a column specification in a SQL statement passed to **SQLCOMMAND** or **SQLPREPARE** contains a table alias, eg. a.emp_id, the REXX variables created corresponding to this column DO NOT contain the "a." prefix.

SQLDESCRIBE variables:

The mSQL implementation includes the extra variable component; **PRIMARYKEY**.

---

22 February 1996
Mark Hessling, <M.Hessling@qut.edu.au>
