REXX/SQL
for
VSE

User's Guide

Software
Product
Research

REXX/SQL
Version 1

TABLE OF CONTENTS

1 Installing REXX/SQL

1.1 Software Prerequisites

- Any DB2/VM-VSE version
- Any VSE/ESA version providing REXX/VSE

1.2 Pre-installation tasks

If you did install another SPR product previously, REXX/SQL should be stored in the same library and sublibrary as the other SPR product and you can skip the remainder of this paragraph.

Define the VSE library where REXX/SQL will be catalogued, by submitting the following jobstream:

```
// EXEC LIBR
 DEFINE SUBLIB=xxxxx.xxxx
/*
```

Notes

- The REXX/SQL material requires less than 1000 library blocks.
- If you have other SPR products installed, install REXX/SQL in that library. All SPR products must reside in the same library.

1.3 Update LIBDEF and Standard Labels

- Add the REXX/SQL library to the PHASE and OBJ SEARCH LIBDEF in your LIBDEF.PROC.
- Submit the updated LIBDEFs to the system before continuing installation. The installation procedures expect that the REXX/SQL library is in the current chain.
- Ensure that the standard labels have a DLBL for the DB2 files SQLBIND, SQLGLOB and BINDWKF, as suggested by the DB2 installation guide. Submit the updated label definitions before continuing installation.

1.4 Installing REXX/SQL

The REXX/SQL software is delivered as a PC file in ZIP format.

Place the ZIP file in a dedicated directory (e.g. REXXSQL) and unzip the file. Following files should now be present in the REXXSQL directory:

INSTALL.BAT	installation procedure for Windows
SEND2RDR.BAT	installation procedure for Windows
REXXSQL.PCF	REXX/SQL software
RXSQLSCF.PCF	software key
REXXSQL0.VSEJOB	setparm SPRLIB
REXXSQL1.VSEJOB	job to linkedit REXX/SQL
REXXSQL2.VSEJOB	job to install REXX/SQL in a database
REXXSQL.BKS	bookshelf for IBM Library Reader
REXXSQL.BOO	REXX/SQL User's Guide in IBM Library Reader format

1.4.1 Preliminary Note

The INSTALL.BAT and SEND2RDR.BAT installation procedures use the "SEND.EXE" to upload the PC files to the POWER reader queue. Ensure that your 3270 emulator supports SEND (some emulators don't) and that the EXE is on your active path.

If you cannot use the SEND.EXE, use the upload facilities of your emulator to perform the equivalent of the SEND commands contained in the BAT files, that is:

for the INSTALL.BAT:

```
SEND RXSQLSCF.PCF (FILE=RDR BINARY LRECL=80 NOUC
SEND REXXSQL.PCF (FILE=RDR BINARY LRECL=80 NOUC
```

for the SEND2RDR.BAT:

```
SEND <filename> (FILE=RDR
```

1.4.2 Issue SETPARM SPRLIB

Skip this step when running a VSE/ESA version lower than 2.4. The job submits a SETPARM SYSTEM statement that is not accepted in older VSE versions.

- Start ICCF PC file transfer (fast path 386) in 3270 emulator session A.
- Edit the file REXXSQL0.VSEJOB and insert the name of the target library on the SETPARM SPRLIB statement. The SETPARM statement remains active until the next VSE IPL. If you installed another SPR product previously into the SPRLIB, insert the name of that library on the SETPARM statement.
- Drag and drop the file REXXSQL0.VSEJOB on the SEND2RDR.BAT. This will send and start the job in class 0 and DISP D. The output listing is in DISP H.
- Check the output listing for errors.

1.4.3 Uploading the REXX/SQL software

- Start ICCF PC file transfer (fast path 386) in 3270 emulator session A.
- Execute the INSTALL.BAT by clicking. This will upload 2 jobs to the POWER/VSE reader queue with DISP D and class 0. The jobs catalog all REXX/SQL components into the VSE library chosen as REXX/SQL residence.
- Both jobs contain a // PAUSE statement. This allows to enter a // SETPARM SPRLIB='...'. If running VSE/ESA 2.4 or higher, ignore the PAUSE statement, as the SETPARM has been submitted by the REXXSQL0.VSEJOB, described above. If running a version older than 2.4, issue the SETPARM statement.
- After job completion, check the DISP=H listing for any errors.

1.4.4 Linking the REXX/SQL phases

- Start ICCF PC file transfer (fast path 386) in 3270 emulator session A.
- In the file REXXSQL1.VSEJOB, assign the SETPARM symbol SPRLIB, specifying the library where REXX/SQL has been uploaded. This is required for VSE versions lower than 2.4. If running 2.4 or higher, you can delete the SETPARM statement, as it has been submitted by the REXXSQL0.VSEJOB, described above.
- Drag and drop the file REXXSQL1.VSEJOB on the SEND2RDR.BAT. This will send and start the job in class 0 and DISP D. The output listing is in DISP H.
- Check the output listing for errors.

1.4.5 Installing REXX/SQL in a database

This installation step should be executed in all databases where REXX/SQL will be used. Following functions will be performed in the target database:

- Load the REXX/SQL packages
- Grant EXECUTE on those packages to PUBLIC
- Create the REXX/SQL table REXXSQL_PACKAGES[1]
- Grant SELECT on this table to PUBLIC

Edit the PC file REXXSQL2.VSEJOB and assign the following SETPARM symbols

CAT
the VSAM catalog for a SAM ESDS workfile (default is VSESPUC)
DB
the name of the target database
DBAPASS
the password of user SQLDBA in that database
POOL
the storage pool number for the REXX/SQL dbspace
PAGES
the number of pages in the REXX/SQL dbspace; for an estimate of the number of pages required, see the note at the end of this paragraph

Submit the REXXSQL2.VSEJOB as follows:

- Start ICCF PC file transfer (fast path 386) in 3270 emulator session A.
- Drag and drop the file REXXSQL2.VSEJOB on the SEND2RDR.BAT. This will send and start the job in class 0 and DISP D. The output listing is in DISP H.
- Check the output listing for errors.

Note

The number of pages required for REXXSQL_PACKAGES depends on the number of static REXX/SQL packages that will be in use and on the number of SQL statements in each static package.

For each static SQL statement, a row is entered into REXXSQL_PACKAGES. The length of the fixed part of the row is 60 bytes. The row also contains the input SQLDA for each statement. An SQLDA entry of 44 bytes is required for each hostvariable appearing:
- in a WHERE clause
- in an INSERT VALUES clause
- in an UPDATE SET clause

[1] REXX/SQL uses this table for static REXX/SQL procedures, to keep the relationship between the REXX/SQL statement name and the corresponding DB2 package section number.

2 Using SQL in REXX/VSE

REXX is an extremely versatile language. It offers efficient programming structures, powerful functions and extensive mathematical capabilities. Commands to host environments can be freely intermixed with REXX statements. This makes the language particularly suitable for command procedures, application prototyping or TCP/IP connectivity.

The I/O capabilities of REXX/VSE provide access to VSE libraries, sequential files and the POWER/VSE queues. However, unlike REXX/VM, REXX/VSE procedures cannot access data in DB2 tables.

REXX/SQL is a product developed by Software Product Research to provide an SQL interface for REXX/VSE procedures.

REXXSQL is implemented as an external REXX function. It is called as

- a function
  r = REXXSQL(input_argument) passes the REXXSQL returncode in the user specified result variable

- a routine
  CALL REXXSQL input_argument passes the REXXSQL returncode in the REXX variable RESULT.

The REXXSQL input argument is a character string, a REXX variable or expression. It contains or refers to the SQL statement to be executed. After submitting the statement to DB2, REXX/SQL returns the completion status and the execution results to the invoking procedure as REXX variables or stems.

REXXSQL accepts all DML statements (SELECT, UPDATE, DELETE, INSERT), all DDL statements (CREATE, DROP, GRANT etc) and the statements CONNECT, COMMIT and ROLLBACK.

REXX/SQL provides the following SQL interfaces:

- execute SQL statements in dynamic mode
- execute SQL statements in static (prepped) mode
- issue DB2/VM-VSE operator commands
- obtain the DB2 help text for a given help topic

Dynamic REXX/SQL can be issued from the REXX/VSE environment against any DB2 database that can be connected using the private or the DRDA protocol.

Static REXX/SQL can be issued against databases of the DB2/VM-VSE family only. REXX/SQL uses the “extended dynamic” facilities of DB2/VM-VSE to create and execute packages. This extended dynamic mode is unknown to other DB2 platforms.

3 Executing dynamic SQL statements

The dynamic REXX/SQL mode is the simplest interface to DB2. Input to the call is a character string or a REXX expression that contains the SQL statement to be executed.

If a SELECT is submitted, the fetched columns are returned in REXX stems. For all statements, execution status is returned as REXX variables.

A sample dynamic REXXSQL procedure is shown on page 21.

3.1 Function input argument

The only input argument on the REXXSQL function call is the SQL statement to be executed. The argument is passed as a string or a REXX expression. The length of the input argument should not exceed 8192 characters.

Following statements can be passed to REXXSQL:

- CONNECT [<userid> IDENTIFIED BY <password>] [TO <database>]'
- COMMIT
- COMMIT RELEASE
- ROLLBACK
- SELECT ....
- UPDATE ....
- INSERT ....
- DELETE ....
- any DDL statement

Example

```
user = "SQLBDA"
password = "...."
r = REXXSQL("CONNECT" user "IDENTIFIED BY" password)
r = REXXSQL("SELECT * FROM SYSTEM.SYSCATALOG")
r = REXXSQL("COMMIT")
```

3.2 REXX/SQL processing

- The CONNECT, COMMIT and ROLLBACK statements are executed directly using a section within the REXXSQL package.
- SQL statements other than SELECT are executed using an SQL EXECUTE IMMEDIATE.
- SELECT statements are executed using an SQL PREPARE / OPEN / FETCH / CLOSE sequence on a dynamic cursor.
- If an SQLCODE occurs during processing, REXX/SQL will automatically issue a ROLLBACK statement.
- There is no auto-commit function in REXX/SQL. COMMIT statements must be submitted explicitly by the user.

3.3 Output column stems

If a SELECT statement has been issued, all selected columns are returned in REXX column stems. These stems have the same name as the corresponding column. The number of lines in the stem is found in the REXX/SQL variable **_NROWS** (and in **stem.0**).

Example

r=REXXSQL('SELECT TNAME FROM SYSTEM.SYSCATALOG')

The above statement will setup _NROWS with the number of rows selected. The selected TNAME's are returned in TNAME.1, TNAME.2, ... thru TNAME.(_NROWS)

If expressions are coded in the SELECT column list, the name of the column stem will be **EXPR_n**, where **n** is a sequence number assigned by REXXSQL for each expression in the SELECT list.

Example

r=REXXSQL('SELECT col1, (col2+col3), (col4+col5) FROM table')

Will setup the stems

- **COL1.**
- **EXPR_1.** (results of col2+col3)
- **EXPR_2.** (results of col4+col5)

The above stems have _NROWS lines. The stem lines are addressed as:
EXPR_1.1, EXPR_2.1, EXPR1.2, EXPR_2.2 and so on.

3.4 Status output variables

After executing an SQL statement, the DB2 SQLCODE is returned as the function or routine returncode.

Following additional REXX variables are available on return from the REXXSQL call:

_COST
When processing a SELECT, a searched UPDATE or a searched DELETE, _COST contains the query cost estimate. (The same value is also in the variable SQLERRD4.)

_NROWS
With a zero SQLCODE, _NROWS contains the number of rows returned by a SELECT or the number of rows processed by an INSERT, a DELETE or an UPDATE.

SQLCODE
The DB2 execution status. Zero if successful completion. The SQLCODEs are described in the DB2/VM-VSE manuals and in the DB2 tables SYSTEXT1 and SYSTEXT2. The REXX/SQL **SQLHELP** function can be used to extract the SYSTEXT2 rows for a given SQLCODE. The SQLHELP interface is described on page 19.

SQLERRM
If a non-zero SQLCODE has been returned, SQLERRM may contain tokens that further describe the error. REXX/SQL replaces the token separators (x'FF') with blanks.

SQLERRD1
If a non-zero SQLCODE has been returned, SQLERRD1 contains the Relational Data System (RDS) error code.

SQLERRD2
If a non-zero SQLCODE has been returned, SQLERRD1 contains the Database Storage System (DBSS) error code.

SQLERRD3
If a zero SQLCODE is returned, SQLERRD3 contains the number of rows affected by INSERT, UPDATE and DELETE. The same value is returned in the REXXSQL variable _NROWS, which also returns the number of rows returned by a SELECT.

SQLERRD4
When processing a SELECT or a searched UPDATE or DELETE, SQLERRD4 contains the query cost estimate. The same value is returned in the REXX/SQL variable _COST.

SQLERRD5
Number of dependent rows affected by a successful DELETE.

SQLNAMES.
This REXX stem contains the names of the columns returned by a SELECT.
SQLNAMES.0 contains the number of columns fetched.
SQLNAMES.1 to SQLNAMES(SQLNAMES.0) contain the table column names.

3.5 Using the dynamic FETCH interface

A SELECT statement that returns a large number of rows may need considerable amounts of storage for the column stems. To avoid storage problems, a FETCH interface has been designed to select one table row at a time. The interface is opened with a REXXSQL('OPEN') call. Each REXXSQL('FETCH') call transfers a single table row. A REXXSQL('CLOSE') call terminates the fetch sequence.

Open the dynamic fetch

The fetch interface is opened with a REXXSQL("OPEN" statement_text), for example:

```
r = REXXSQL("OPEN SELECT TNAME,ROWCOUNT FROM SYSTEM.SYSCATALOG")
```

The open call does not transfer data. It does assign the output status variables, described on page 8. If needed, the variable _COST (the execution cost estimate) should be retrieved at this time. It is no longer available once FETCH calls have been made.

Perform dynamic fetch

The **REXXSQL("FETCH")** call returns one row on the open cursor. The fetched row columns are stored in column variables, not in column stems, as only one column is passed. The REXXSQL variable _NROWS always has the value 1.

The fetch call also returns the output status variables, described on page 8. SQLCODE 100 will be returned when all rows have been passed.

Close the dynamic fetch

A **REXXSQL("CLOSE")** call terminates the fetch sequence. A COMMIT call will implicitly close an open fetch cursor.

An example of the dynamic fetch interface can be found on page 22.

Note

The dynamic fetch interface does not provide for multiple cursors that are open simultaneously. (The static fetch cursor does allow it). However, while fetching a cursor, it is allowed to issue statements that do not use dynamic fetch.

4 Executing prepped SQL statements

While the dynamic interface is easy to use, it incurs the overhead of DB2 “prepare” processing.[2] Since this overhead is not trivial, SQL statements that are executed often can be prepped and executed in the ”static” mode.

4.1 Prepping SQL statements

When a REXX/VSE procedure wants to execute SQL statements in static mode, it must create a DB2 package first. This package may contain multiple SQL statements (package sections). After the package has been created, selected statements can be called from the package for execution. For an example of REXXSQL prep, see page 23.

4.1.1 Initiating package creation

Package creation is initiated using following statement:

r = REXXSQL(“CREATE PACKAGE” [creator.]packagename)

If “creator” is omitted, the currently connected DB2 userid becomes the creator.
If a package with the same name already exists, it will be replaced with the new package without any warning.

4.1.2 Adding statements to a package

An SQL statement is added to a package using following statement:

r = REXXSQL(“PREPARE statement_name FROM statement_text”)

statement_name
Assigns a symbolic name to the added statement. This name will be used when requesting execution of that particular package statement.[3] The name is also used internally by REXX/SQL as a SELECT cursor, if needed. Therefore, the statement name should not exceed 18 characters and conform to the SQL naming conventions. It should be unique with the new package.

statement_text
The text of the statement to be added to the package.

[2]When our **SQL/Monitoring Facility** product has been installed, its **AutoPrep** facility is able to automatically and transparently transform dynamic statements into static SQL.

[3]DB2 identifies package statements by means of package **section numbers**. REXX/SQL uses a DB2 table to maintain the relationship between a statement_name and the section number assigned to it by DB2. When executing a named statement, REXX/SQL retrieves the section number from the table and executes that section.

4.1.3 Defining hostvariables

If the prepped statement contains variables, the following applies:

- REXX/SQL determines the data type and length for the SELECT **output** hostvariables automatically, using SQL DESCRIBE. The user needs not to be concerned about this.

- The user must specify the **input** variables for INSERT and UPDATE statements and the variables occurring in WHERE predicates.

- These input variables can be passed either as a **parameter marker** or as a **hostvariable**.

    - A parameter marker is designated by a question mark, for example:
      INSERT INTO <table> VALUES(?, ?)

    - A hostvariable definition starts with a semicolon, followed by the data type and length of the hostvariable, for example:
      INSERT INTO <table> VALUES (:INTEGER , :CHAR(8)).
      (For a list of allowed data types, see page 12.)

- Since parameter markers (?) do not specify the format of the hostvariable at prep time, an implicit definition will take place during execution, depending on the contents of the variables at run-time. REXX/SQL will make the following assumptions:

    - Data enclosed in quotes are submitted with the CHARACTER data type and the actual length of the character string.
    - Numerical data without a decimal point are passed as INTEGERs.
    - Numerical data containing a decimal point are submitted as DECIMAL, with the precision and the scale of the actual value.

- Best DB2 performance is achieved when the data type and length of each hostvariable is known at prep time. Therefore, parameter markers should be used with caution.

4.1.4 Defining hostvariable data types

Hostvariables in the statement text should be defined with one of the following data types:

:INTeger
At execution time, REXX/SQL will present the corresponding input value to DB2 as a 4-byte integer value.

:SmallINT
At execution time, REXX/SQL will present the corresponding input value to DB2 as a 2-byte small integer value.

:CHARacter (length)
At execution time, REXX/SQL will present the corresponding input value to DB2 in a character type field of the specified length. If the input is shorter than “length”, right padding with blanks will be done.

:VARCHAR [(length)]
At execution time, REXX/SQL will present the corresponding input value to DB2 in a varchar type field. The specified length indicates the maximum length. At execution, the effective length of the input string will be passed to DB2. If no maximum length is specified, a default of 254 is assumed.

:DECimal (precision, scale)
At execution time, REXX/SQL will present the corresponding input value to DB2 as a decimal field with the specified precision and scale. If the input value has different precision or scale, the value will be adjusted before being passed to DB2. For example: if the hostvar has been defined as DEC(5,2), an input value of 15 will be submitted as 015.00.

:DATE
At execution time, REXX/SQL will present the input value in a 10-byte character field.

:TIME
At execution time, REXX/SQL will present the input value in an 8-byte character field.

:TIMESTAMP
At execution time, REXX/SQL will present the input value in an 26-byte character field.

Notes

- The upper-case characters in the above data type definitions represent an abbreviation of the keyword. For example, :INTEGER and :INT are equivalent specifications.

- Any number of blanks may appear between the data type and the parentheses that enclose the length specification.

- Hostvariables following a LIKE clause should be defined as VARCHAR, not as CHAR. This is a DB2 requirement. The restriction does not apply to parameter markers.

- Floating point columns should be assigned from decimal hostvariables.

4.1.5 Terminating package creation

When all statements have been added to the package, a **REXXSQL("COMMIT")** must be issued to actually create the package. This is a DB2/VM-VSE requirement.

When the package has been created, only its creator has the EXECUTE authority. GRANT statements must be used to propagate the EXECUTE privilege to other users.

4.2 Executing prepped SQL statements

REXX procedures may execute any statement in any package that was created using REXX/SQL, provided the necessary EXECUTE privileges have been granted. Users of a prepped statement must know the creator and name of the package and the REXXSQL statement_name of the SQL statement they want to execute.

A statement that was prepped previously by means of a REXXSQL PREPARE call, is executed using the following call:

r = REXXSQL("EXECUTE" statement_name "IN" [creator.] package "USING" data_list)

statement_name
Specify the statement_name that was used at PREPARE time.

[creator.]package
Specify the package creator and name that was used in the REXXSQL CREATE PACKAGE call. The creator name may be omitted, in which case it defaults to the DB2 userid currently connected.

data_list
Provides values for each hostvariable or parameter marker in the prepared statement text.
- The data_list items must be specified in the same order as the hostvariables or parameter markers occurring in the prepared statement.
- A character value must be enclosed in single (‘) or double (“) quotes.
- A blank is used as separator in a list of values.
- To assign a null value, code NULL (without quotes).

After execution, a number of status variables is available, as described on page 8.
If the executed statement is a SELECT, the selected columns are returned in REXX stems, as described on page 7.

Example

If the UPD_CUSTNAME statement has been prepared as:
UPDATE CUSTOMERS SET CUSTNAME = :VARCHAR WHERE CUSTNO = :INTEGER

the following call will update the name of customer 100:
New_name = ‘...’
r = REXXSQL(“EXECUTE UPD_CUSTNAME IN CUSTPACK USING New_name 100”)

Note

- REXX programs can execute statements from different packages within the same LUW.
- For an example of static REXXSQL, see page 23.

4.3 Locate package function

The **"REXXSQL LOCATE [creator.]packagename"** call can be used to determine whether a package exists and to automatically generate it when it does not.

The call returns SQLCODE 0 if the package exists and SQLCODE 100 if it does not.

4.4 Using the static FETCH interface

Like the dynamic interface, the static interface fetches single rows.

The static fetch interface is initiated by the following call:

REXXSQL("OPEN statement_name IN [creator.]package [USING data_list]")

Each table row is fetched using:

REXXSQL("FETCH statement_name IN [creator.]package")

After FETCH, a number of status variables is available, as described on page 8.
The selected columns are returned in REXX stems, as described on page 7.

To terminate the fetch sequence, issue:

REXXSQL("CLOSE statement_name IN [creator.]package")

statement_name
Specifies the statement_name that was used at PREPARE time.

[creator.]package
Specifies the package creator and name that was used in the REXXSQL CREATE PACKAGE call. The creator name may be omitted, in which case it defaults to the DB2 userid currently connected.

data_list
Provides values for each hostvariable or parameter marker in the prepared statement text.
- The data_list items must be specified in the same order as the hostvariables or parameter markers occurring in the prepared statement.
- A character value must be enclosed in single (') or double (") quotes.
- A blank is used as separator in a list of values.

Since each fetch sequence is identified by a statement name (which corresponds to an open cursor), multiple FETCH sequences on different cursors can be open concurrently.

An example of the static fetch interface can be found on page 24.

4.5 Using the static PUT interface

Use the PUT interface for blocked INSERTs. When multiple rows must be inserted into the same table, the PUT interface will provide better performance than the INSERT statement.

The PUT statement is prepared (following a CREATE PACKAGE) by the following call:

REXXSQL("PREPARE statement_name FROM statement_text")

where statement_text is an INSERT
- using parameter markers e.g. "INSERT INTO table VALUES (?,?,...)"
- using hostvariables e.g. "INSERT INTO table VALUES (:INT, :CHAR(8),...)"

The PUT interface is initiated by the following call:

REXXSQL("OPEN statement_name IN [creator.]package")

Each table row is inserted using:

REXXSQL("PUT statement_name IN [creator.]package USING data_list")

After PUT, a number of status variables is available, as described on page 8.

To terminate the PUT sequence, issue:

REXXSQL("CLOSE statement_name IN [creator.]package")

statement_name
Specifies the statement_name that was used at PREPARE time.

[creator.]package
Specifies the package creator and name that was used in the REXXSQL CREATE PACKAGE call. The creator name may be omitted, in which case it defaults to the DB2 userid currently connected.

data_list
Provides the values for each hostvariable or parameter marker in the prepared statement text.
- The data_list items must be specified in the same order as the hostvariables or parameter markers occurring in the prepared statement.
- A character value must be enclosed in single (') or double (") quotes.
- A blank is used as separator in a list of values.
- To assign a null value, code NULL (without quotes).

An example of the PUT interface can be found on page 25.

5 Issuing DB2/VM-VSE operator commands

Issue a DB2 operator command as follows:

r = REXXSQL(“DB2COM” command_text)

Where command_text holds the operator command to be executed.

A REXXSQL CONNECT must be issued before the DB2COM call.

The output of the operator command is returned in the REXX stem **_REPLY** and the number of reply lines is in the variable **_NROWS**. **SQLCODE** will be set if the command cannot be forwarded (because no connect was done for example).

Example

```
r = REXXSQL(“CONNECT ... IDENTIFIED BY ... TO ...”)
r = REXXSQL(“DB2COM SHOW LOG”)
do i = 1 to _NROWS
    say strip (_REPLY.i)
end
```

Notes

- A DB2 agent structure is required to execute the operator command.
- The command is passed using an RDIIN type 155 / 170 mailbox.
- DB2/VM-VSE does not allow to pass the FORCE or SHUTDOWN commands using an RDIIN. ARI0064E will be returned if attempted.

6 Obtaining SQL help

6.1 Using the REXX/SQL SQLHELP function

The REXX/SQL SQLHELP function returns the DB2 SYSTEXT2 rows for a given topic. The topic can be an SQLCODE or any other topic that appears in SYSTEXT2.

The SQLHELP output is returned in the REXX stem **_HELPTEXT** and the number of help lines is in the variable **_NROWS**.

A REXXSQL CONNECT must be issued before the SQLHELP call.

Example

```
r = REXXSQL("CONNECT" user "IDENTIFIED BY" password "TO" database)
r = REXXSQL("SQLHELP" SQLCODE)
do i = 1 to _NROWS
    say HELPTEXT.i
end
```

6.2 Using the REXXSQLH procedure

As an alternative for the REXXSQL("SQLHELP") call, REXX/SQL provides the REXXSQLH.PROC in the library where REXX/SQL has been installed. It prints (using REXX SAY) following items:

- the SQLCODE passed
- the SQLERRM if passed
- the SQL helptext for the SQLCODE passed

The procedure is invoked using the REXX statement

CALL REXXSQLH SQLCODE [sqlerrm].

Note that a LIBDEF PROC,SEARCH=<REXX/SQL_library> is required in the JCL that invokes your own REXX procedure.

7 REXX/SQL samples

7.1 Sample dynamic PROC

```
/* List DB2 tables by rowcount */

arg db user password

/*   Connect the target database. Exit if connect fails. */
r = REXXSQL("CONNECT" user "IDENTIFIED BY" password "TO" db)
if sqlcode < 0 then exit 8

/* Prepare and execute a SELECT on SYSCATALOG. */
s =     "SELECT CREATOR,TNAME,ROWCOUNT FROM SYSTEM.SYSCATALOG" ,
        "WHERE ROWCOUNT >= 100 ORDER BY ROWCOUNT DESC"

r = REXXSQL(s)

/* Check for SQL errors. */
if SQLCODE <> 0 then signal SQL_error

/* Show the result stems CREATOR, TNAME and ROWCOUNT */
do i = 1 to _NROWS
    tablename = strip(creator.i)"."strip(tname.i)
    say "Table" tablename "has" rowcount.i "rows"
end

/* Terminate */
r=REXXSQL("COMMIT")
exit

/* Show SQLCODE, SQLERRM and related help text */
SQL_error:
call REXXSQLH sqlcode sqlerrm
exit 8
```

7.2 Sample dynamic PROC using the fetch interface

```
/* List DB2 tables by rowcount */

arg db user password nrows

/*   Connect the target database. Exit if connect fails. */
r = REXXSQL("CONNECT" user "IDENTIFIED BY" password "TO" db)
if sqlcode < 0 then exit 8

/* Prepare and execute a SELECT on SYSCATALOG. */
s =     "SELECT CREATOR,TNAME,ROWCOUNT FROM SYSTEM.SYSCATALOG" ,
        "WHERE ROWCOUNT >=" nrows "ORDER BY ROWCOUNT DESC"
r = REXXSQL("OPEN" s)

/* Check for SQL errors. */
if SQLCODE <> 0 then signal SQL_error

/* Show the result stems CREATOR, TNAME and ROWCOUNT */
r = REXXSQL("FETCH")
do while sqlcode = 0
    tablename = strip(creator)"."strip(tname)
    say "Table" tablename "has" rowcount "rows"
    r = REXXSQL("FETCH")
end
if SQLCODE < 0 then signal SQL_error

/* Terminate */
r=REXXSQL("CLOSE")
r=REXXSQL("COMMIT")
exit

/* Show SQLCODE, SQLERRM and related help text */
SQL_error:
call REXXSQLH sqlcode sqlerrm
exit 8
```

7.3 Sample static PROC

```
/* List DB2 tables by rowcount */

arg db user password

r = REXXSQL(“CONNECT” user “IDENTIFIED BY” password “TO” db)
if SQLCODE < 0 then exit 8

/* Check if package SAMPLE1 exists, create it if not */
if 0 <> REXXSQL("LOCATE PACKAGE SAMPLE1") then call Create_Package

/* Execute the prepped statement “get_rowcount” */
r = REXXSQL("EXECUTE GET_ROWCOUNT IN SAMPLE1 USING ‘%’ 100 ”)
if SQLCODE <> 0 then exit sqlcode
/* Show results */
do i = 1 to _NROWS
    tablename = strip(creator.i)”.”strip(tname.i)
    say “Table” tablename “has” rowcount.i “rows”
end
r = REXXSQL("COMMIT")
exit

Create_Package:
r = REXXSQL("CREATE PACKAGE SAMPLE1")
if SQLCODE <> 0 then     /* process error */
s =     “SELECT CREATOR,TNAME,ROWCOUNT FROM SYSTEM.SYSCATALOG” ,
        “WHERE TNAME LIKE :VARCHAR AND ROWCOUNT >= :INTEGER” ,
        “ORDER BY ROWCOUNT DESC”
r = REXXSQL("PREPARE GET_ROWCOUNT FROM” s)
if SQLCODE <> 0 then     /* process error */
r = REXXSQL("COMMIT")
return
```

7.4 Sample static PROC using the fetch interface

```
/* List DB2 tables by rowcount */

arg db user password

r = REXXSQL(“CONNECT” user “IDENTIFIED BY” password “TO” db)
if SQLCODE < 0 then exit 8

/* Check if package SAMPLE1 exists, create it if not */
if 0 <> REXXSQL("LOCATE PACKAGE SAMPLE1") then call Create_Package

r = REXXSQL(“OPEN GET_ROWCOUNT IN SAMPLE1 USING ‘%’ 100")

/* Check for SQL errors. */
if SQLCODE <> 0 then signal SQL_error

/* Show the result stems CREATOR, TNAME and ROWCOUNT */
r = REXXSQL(“FETCH GET_ROWCOUNT IN SAMPLE1”)
do while sqlcode = 0
    tablename = strip(creator)”.”strip(tname)
    say “Table” tablename “has” rowcount “rows”
    r = REXXSQL(“FETCH GET_ROWCOUNT IN SAMPLE1”)
end
if SQLCODE < 0 then signal SQL_error
r = REXXSQL(“CLOSE GET_ROWCOUNT IN SAMPLE1”)
r = REXXSQL("COMMIT")
exit

Create_Package:
r = REXXSQL("CREATE PACKAGE SAMPLE1")
if SQLCODE <> 0 then     /* process error */
s =     “SELECT CREATOR,TNAME,ROWCOUNT FROM SYSTEM.SYSCATALOG” ,
        “WHERE TNAME LIKE :VARCHAR AND ROWCOUNT >= :INTEGER” ,
        “ORDER BY ROWCOUNT DESC”
r = REXXSQL("PREPARE GET_ROWCOUNT FROM” s)
if SQLCODE <> 0 then     /* process error */
r = REXXSQL("COMMIT")
return
```

7.5 Static PUT sample

```
/* Using the PUT interface */

arg db user password

r = REXXSQL("CONNECT" user "IDENTIFIED BY" password "TO" db)
if SQLCODE < 0 then exit 8

/* Check if package SAMPLE2 exists, create it if not */
if 0 <> REXXSQL("LOCATE PACKAGE SAMPLE2") then call Create_Package

r = REXXSQL("OPEN INSERT_1 IN SAMPLE2")

/* Check for SQL errors. */
if SQLCODE <> 0 then signal SQL_error

/* Insert 100 rows into table TEST */
do i = 1 to 100 while sqlcode = 0
    char_value = "'DATA"i""
    r = REXXSQL("PUT INSERT1 IN SAMPLE2 USING i char_value")
end
if SQLCODE < 0 then signal SQL_error
r = REXXSQL("CLOSE INSERT_1 IN SAMPLE2")
r = REXXSQL("COMMIT")
exit

Create_Package:
r = REXXSQL("CREATE PACKAGE SAMPLE2")
if SQLCODE <> 0 then     /* process error */
insert_statement = 'INSERT INTO TEST (C1,C2) VALUES ( :INTEGER , :CHAR(8) )'
r = REXXSQL("PREPARE INSERT_1 FROM" insert_statement)
if SQLCODE <> 0 then     /* process error */
r = REXXSQL("COMMIT")
return
```

8 Index