REXX/SQL for VM

User's Guide

Software Product Research

TABLE OF CONTENTS

1 Installing REXX/SQL

1.1 Software Prerequisites

Any DB2/VM and z/VM version.

1.2 Pre-installation tasks

A minidisk of 10 3390 cylinders or SFS directory should be available to receive the REXX/SQL material.

1.3 Installing the CMS components

This installation step loads the REXX/SQL modules, execs, and other material from the distribution medium to the disk or directory where the REXX/SQL has been installed (the ***REXX/SQL product disk***).

1.3.1 Prerequisites

1. Ensure that the VM userid performing the installation has write access to the REXX/SQL product disk.
2. Ensure that you have access to the SQL production minidisk in any filemode.
3. If REXX/SQL has been received on tape, ensure that a tape or cartridge device has been attached to the virtual machine with virtual address 181.

1.3.2 Tape Installation

1. Access the product disk for write with CMS filemode A.
2. Issue the command: **TAPE LOAD * * A**
3. When the TAPE LOAD command has completed, you do no longer need the distribution tape.

1.3.3 Installation from other media

If REXX/SQL has been received on a medium other than tape, please follow the installation instructions present in the README file on the medium.

1.3.4 Linking the REXX/SQL Modules

1. With the product disk still accessed as "A" enter **REXSQLPI** at the CMS prompt.
2. On the installation option screen, enter **Y** on the line **CMS Installation (Y/N)**.
3. The REXX/SQL modules are now linked on the product disk.

1.4 Installing the SQL components

SQL installation must be performed in each database where the REXX/SQL facility will be used. The target database must be a DB2/VM database.

1.4.1 Prerequisites

1. You should have write access to the REXX/SQL product disk in filemode A.
2. You should have read access to the SQL production minidisk in any filemode.
3. A DBspace is needed to create the **REXXSQL_PACKAGES** table, which will be used when prepped SQL is performed using REXX/SQL.

1.4.2 Installing

1. Type **REXSQLPI** at the CMS prompt.
2. On the installation option screen, enter **Y** on the line **SQL Installation (Y/N)**.
3. On the next panel, specify the name of the database for installation and the password of user SQLDBA.
4. The REXX/SQL packages are loaded in the database using a SQLDBSU RELOAD PACKAGE command under the SQLDBA userid.
5. If initial installation is being done in the database, the REXX/SQL control table will be created. At this time, the corresponding SQLDBSU command stream will be XEDITed. Specify the storage pool where the DBspace should be created and the NPAGES parameter.
6. The previous panel is shown again, to allow for installation in another database. Press PF3 to terminate the installation procedure.

1.4.3 REXX/SQL DBspace requirements

The **REXXSQL_PACKAGES** table has one row for each prepped SQL statement. The length of these rows depend on the length of the SQL statement itself and on the number and length of the host variables (if any) used by the statement.

2 Using SQL in REXX

REXXSQL is implemented as an external REXX routine. It is called as follows:

“REXXSQL input_argument”

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-VM operator commands
- obtain the DB2 help text for a given help topic

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

Normally, static REXX/SQL can be issued against databases of the DB2/VM-VM family only. REXX/SQL uses the “extended dynamic” facilities of DB2/VM to create and execute packages. This extended dynamic mode is unknown to other DB2 platforms. Following technique allows to bypass this restriction:

- create the REXX/SQL under DB2/VM (eventually using the REXXPACK utility)
- transfer the created package to the non-DB2_VM server
- use REXXSQL EXECUTE statements to execute sections of the transferred package while connected to the non-DB2_VM server

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 25.

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 = “....”
“REXXSQL CONNECT” user “IDENTIFIED BY” password
“REXXSQL SELECT * FROM SYSTEM.SYSCATALOG”
“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

“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

“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

Following 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-VM 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 23.

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:

“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 6. 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 6. 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 26.

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.[1] 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 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 27.

A package can be created

- within a REXX exec using the CREATE PACKAGE and PREPARE statements
- outside a REXX exec using the REXXPACK utility

4.1.1 Initiating package creation

Package creation is initiated using following statement:

“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 <u>without any warning</u>.

4.1.2 Adding statements to a package

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

“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.[2] 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.

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

[2]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 The REXXPACK Package creation utility

As an alternative to the REXXSQL CREATE PACKAGE and REXXSQL PREPARE statements, embedded in the REXX EXEC, the **REXXPACK** utility program can be used to create a package independently of the EXEC that uses the package.

REXXPACK is invoked from the CMS prompt as follows:

REXXPACK creator.packagename

This will create the package “creator.packagename”.

Before invoking REXXPACK, the SQL statements to be stored in the package must be placed in a CMS file named **<packagename> SQL**. This file may reside on any accessed minidisk.

- The SQL file contains all SQL statements to be added. The statement text is written in free format.

- A statement_name must precede the statement text. A : sign must be used to terminate the statement_name. The statement_name should be identical to the name used on the REXXSQL EXECUTE statement when executing the package.

- An asterisk in column 1 denotes a comment line.

- The DB2 userid executing REXXPACK must have the privilege to create the package.

- A CONNECT to the target database must have been performed (using SQLINIT for instance) before invoking REXXPACK.

Example

The CMS file TABINFO SQL contains the following:

```
*
*  Get number of rows in a table
*
GET_ROWCOUNT:
SELECT ROWCOUNT FROM SYSTEM.SYSCATALOG
     WHERE TNAME = :VARCHAR
*
*  Get number of indexes defined on a table
*
GET_INDEXCOUNT:
SELECT COUNT(*) FROM SYSTEM.SYSINDEXES
     WHERE TNAME = :VARCHAR
```

The command **REXXPACK SQLDBA.TABINFO** will store the above SQL statements in the package SQLDBA.TABINFO.

These statements are executed subsequently using the commands:

```
REXXSQL EXECUTE GET_ROWCOUNT IN SQLDBA.TABINFO USING tablename
REXXSQL EXECUTE GET_INDEXCOUNT IN SQLDBA.TABINFO USING tablename
```

4.1.4 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 11.)

- 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.

4.1.5 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.

4.1.6 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 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:

“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 6.

If the executed statement is a SELECT, the selected columns are returned in REXX stems, as described on page 5.

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 = ‘...’
“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 27.

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] [WITH RETURN]"

The WITH RETURN option should be used in REXX/SQL execs executing as a stored procedure. For a description of WITH RETURN, refer to the DB2_VM SQL Reference manual.

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 6.
The selected columns are returned in REXX stems, as described on page 5.

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 28.

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 6.

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 29.

5 Using REXX/SQL in a stored procedure

DB2_VM does not support REXX as a stored procedure language. REXX/SQL provides the **Procedure Prolog** facility to allow a REXX exec to function as a stored procedure.

5.1 Preparatory steps

5.1.1 Stored Procedure Server considerations

The procedure servers that run REXX/SQL execs should have a link to the REXXSQL product disk and to the disk that contains the executed execs and the load modules created by REXXPP.

5.1.2 Creating the stored procedure

A REXX/SQL stored procedure should be created as follows:

```
CREATE PROCEDURE procedurename (procedure_parameters)
        LANGUAGE ASSEMBLE,
        SERVER GROUP xxx,
        EXTERNAL,
        PARAMETER STYLE GENERAL
```

(where “procedurename” equals the name of the REXX EXEC)

5.2 Writing the REXX/SQL exec

5.2.1 Create the SQL statements file

Code all SQL statements contained in the REXX exec into a CMS file with the name **<execname> SQL** and submit the SQL file to the REXXPACK function, which is described on page 9. REXXPACK will generate a DB2/VM package with a name equal to the name of the REXX exec.

5.2.2 Coding the EXEC

Use the **REXXSQL EXECUTE** or the static **REXXSQL OPEN**, **REXXSQL FETCH** and **REXXSQL CLOSE** commands to execute SQL statements in the package generated from the SQL file, created above. Insert additional REXX processing where needed.

To return the processing results to the program that calls the stored procedure, issue the REXX **queue** statement for each procedure parameter defined with the output attribute in the CREATE PROCEDURE statement.

Alternatively, you may use the **WITH RETURN** option on the **REXXSQL OPEN** statement. This allows the calling program to fetch rows using the cursor declared by the stored procedure. For details about WITH RETURN processing, refer to the DB2_VM SQL Reference manual.

Restrictions

Due to DB2_VM imposed restrictions, following SQL statements cannot be used in a REXX/SQL EXEC running as a stored procedure: CONNECT, COMMIT, ROLLBACK, REXXSQL CREATE and REXXSQL LOCATE. All other SQL statements (including dynamic SQL) are allowed.

5.3 Testing the EXEC

An interesting feature of REXX as a stored procedure language is the possibility to test the REXX exec stand-alone, before invoking it from a procedure server.

However, since the exec stacks its return results, these result lines are presented to CMS when the exec returns to CMS and "*Unknown CM/CMS command*" error messages will be issued. To prevent this, a **REXXRUN** exec has been provided with REXX/SQL.

REXXTEST is invoked as follows:

REXXRUN name-of-exec-to-run arguments-for-the-exec

REXXRUN wil invoke the named exec with the specified parameters and display the results of all the result lines that have been stacked.

5.4 Generating the procedure prolog

When DB2_VM calls a stored procedure in the procedure server, an executable load module (CMS filetype MODULE) is called. A load module however does not exist for a REXX stored procedure. Therefore, a load module must be generated before executing the exec in the procedure server.

To do this, issue following command from the CMS prompt:

REXXPP name-of-the-exec

REXXPP will build a small assembler source, assemble and link it. The resulting load module is callable from a procedure server, provided that the latter has access to the minidisk containing the load module.

5.5 Technical information

The load module produced by the REXXPP function, calls the REXX/SQL REXXPPM function.

REXXPPM performs the following operations:

- convert the input arguments from the format specified on the CREATE PROCEDURE into REXX format, that is, into character strings
- call the users REXX exec with the character-string arguments
- for each procedure output argument, a result line is taken from the CMS console stack and converted into the parameter format specified on the CREATE PROCEDURE

5.6 TABINFO Stored Procedure Example

(1) The TABINFO procedure has been defined to DB2 as follows:

```
CREATE PROCEDURE TABINFO (IN CHAR(18), OUT INTEGER)
        LANGUAGE ASSEMBLE,
        SERVER GROUP xxx,
        EXTERNAL,
        PARAMETER STYLE GENERAL
```

(2) The CMS file TABINFO SQL contains the following SQL statements:

```
*
*  Get number of rows in a table
*
GET_ROWCOUNT:
SELECT ROWCOUNT FROM SYSTEM.SYSCATALOG
       WHERE TNAME = :VARCHAR
*
*  Get number of indexes defined on a table
*
GET_INDEXCOUNT:
SELECT COUNT(*) FROM SYSTEM.SYSINDEXES
       WHERE TNAME = :VARCHAR
```

(3) The command **REXXPACK SQLDBA.TABINFO** builds the SQLDBA.TABINFO package containing the two SELECT statements mentioned above.

(4) The TABINFO EXEC accepts a tablename as argument, calls the GET_ROWCOUNT function and returns the result as follows:

```
Arg Table
Table = strip(Table)
Quote = "'"
"REXXSQL EXECUTE GET_ROWCOUNT IN SQLDBA.TABINFO USING" quote||Table||quote
If SQLCODE < 0 then do
   Call REXXSQLM sqlcode sqlerrm
   Exit
If rowcount.0 > 0 then
   Queue rowcount.1                /* Return the result */
```

(5) The TABINFO exec can be tested using a **REXXRUN TABINFO** <tablename>

(6) The procedure prolog is generated with the command **REXXPP TABINFO**.

(7) The TABINFO procedure can now be invoked from a prepped program using the statement:

EXEC SQL CALL TABINFO (:tablename, :rowcount)

6 Calling a stored procedure from REXX/SQL

A REXX/SQL exec calls a stored procedure using the **REXXSQL CALL** statement. The called procedure may be written in REXX/SQL or any other language.

Syntax

REXXSQL CALL procedurename argument(1) argument(2) ... argument(n)

- The number of arguments specified on the CALL should correspond to the number of input arguments defined for the procedure.
- Character arguments need not be enclosed in quotes, except when the argument contains one or more blanks.
- The output arguments returned by the called procedure are available to the exec in the REXX variables **Result_1** to **Result_n**, where n corresponds to the number of output arguments returned by the CALL.

Example

```
Arg Customer_no
REXXSQL CALL CUSTINFO Customer_no
Cust_Name = Result_1
Cust_Address = Result_2
Cust_City = Result_3
```

The parameters of the CUSTINFO stored procedure are assumed to have been defined like:
(IN INTEGER, OUT CHAR(30), OUT CHAR(30), OUT CHAR(30))

7 Issuing DB2/VM-VM operator commands

Issue a DB2 operator command as follows:

“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

```
“REXXSQL CONNECT” user “IDENTIFIED BY” password “TO” database
“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-VM does not allow to pass the FORCE or SHUTDOWN commands using an RDIIN. ARI0064E will be returned if attempted.

8 Obtaining SQL help

8.1 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

```
“REXXSQL CONNECT” user “IDENTIFIED BY” password “TO” database
“REXXSQL SQLHELP” SQLCODE
do i = 1 to _NROWS
    say HELPTEXT.i
end
```

8.2 Using the REXXSQLH procedure

As an alternative for the REXXSQL SQLHELP call, REXX/SQL provides the REXXSQLH EXEC.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].

9 REXX/SQL samples

9.1 Sample dynamic PROC

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

arg db user password

/*   Connect the target database. Exit if connect fails. */
“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”

“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 */
“REXXSQL COMMIT"
exit

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

9.2 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. */
“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”
“REXXSQL OPEN” s

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

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

/* Terminate */
“REXXSQL CLOSE"
“REXXSQL COMMIT"
exit

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

9.3 Sample static PROC

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

arg db user password

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

/* Check if package SAMPLE1 exists, create it if not */
“REXXSQL LOCATE PACKAGE SAMPLE1"
If SQLCODE = 100 then call Create_Package

/* Execute the prepped statement “get_rowcount” */
“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
“REXXSQL COMMIT"
exit

Create_Package:
“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”
“REXXSQL PREPARE GET_ROWCOUNT FROM” s
if SQLCODE <> 0 then     /* process error */
“REXXSQL COMMIT"
return
```

9.4 Sample static PROC using the fetch interface

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

arg db user password

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

/* Check if package SAMPLE1 exists, create it if not */
“REXXSQL LOCATE PACKAGE SAMPLE1"
if SQLCODE = 100 then call Create_Package

“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 */
“REXXSQL FETCH GET_ROWCOUNT IN SAMPLE1”
do while SQLCODE = 0
    tablename = strip(creator)”.”strip(tname)
    say “Table” tablename “has” rowcount “rows”
    “REXXSQL FETCH GET_ROWCOUNT IN SAMPLE1”
end
if SQLCODE < 0 then signal SQL_error
“REXXSQL CLOSE GET_ROWCOUNT IN SAMPLE1”
“REXXSQL COMMIT"
exit

Create_Package:
“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”
“REXXSQL PREPARE GET_ROWCOUNT FROM” s
if SQLCODE <> 0 then     /* process error */
“REXXSQL COMMIT"
return
```

9.5 Static PUT sample

```
/* Using the PUT interface */

arg db user password

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

/* Check if package SAMPLE2 exists, create it if not */
“REXXSQL LOCATE PACKAGE SAMPLE2"
if sqlcode = 100 then call Create_Package

“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’”
    “REXXSQL PUT INSERT1 IN SAMPLE2 USING” i char_value
end
if SQLCODE < 0 then signal SQL_error
“REXXSQL CLOSE INSERT_1 IN SAMPLE2”
“REXXSQL COMMIT"
exit

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