Using REXX as a Database Tool

Mark Hessling
Griffith University

Using REXX as a Database Tool

Mark Hessling

Griffith University
Brisbane, Australia

Introduction

Having been involved in Database Administration for the last 5 years, and having a long relationship with REXX (over 10 years) it was inevitable that the two should come together eventually.

GUROO History

During 1993, I found I needed a scripting tool to manipulate some data in Oracle tables as part of my Grad. Dip. course. I decided that it would be quicker to write an interface from REXX to an Oracle database, and then write the programs I required in REXX than it was to write the same programs in the tools supplied by Oracle. GUROO (Griffith University Rexx Oracle Overseer) was the result.

As I already had the basic framework, courtesy of the SAA REXX API in Regina, and the interface to Regina in THE, filling in the remainder of the tool was relatively simple.

The prime design consideration in GUROO was simplicity. I had a rough idea of the interface to existing REXX-SQL tools like the Database Manager in OS/2 Extended Edition. These interfaces seem too complicated. Compare the same program written using the REXX interface to Database Manager and the GUROO example. See Examples 1 and 2.

Example 1

```
/*-------------------------------------------------------------*/
/* Display the names of all tables owned by the default user.*/
/* DBM version.                                                */
/*-------------------------------------------------------------*/
/*-------------------------------------------------------------*/
/* Load the DBM dynamic link libraries...                      */
/*-------------------------------------------------------------*/
If rxfuncquery('SQLDBS') \= 0 Then
   rcy = rxfuncadd('SQLDBS','SQLAR','SQLDBS');
If rxfuncquery('SQLEXEC') \= 0 Then
   rcy = rxfuncadd('SQLEXEC','SQLAR','SQLEXEC');
/*-------------------------------------------------------------*/
/* Connect to the SAMPLE database...                           */
/*-------------------------------------------------------------*/
Call sqlexec 'CONNECT TO sample IN SHARE MODE';
If ( SQLCA.SQLCODE \= 0) Then
   Do
     Say 'CONNECT TO Error: SQLCODE =' SQLCA.SQLCODE;
     Exit
   End
/*-------------------------------------------------------------*/
/* Prepare and declare the cursor for the SQL statement...     */
/*-------------------------------------------------------------*/
st = "SELECT name FROM sysibm.systables WHERE name <> ?";
Call sqlexec 'PREPARE s1 FROM :st';
Call sqlexec 'DECLARE c1 CURSOR FOR s1';
If ( SQLCA.SQLCODE \= 0) Then
   Say 'Error preparing statement: SQLCODE =' SQLCA.SQLCODE;
Else
/*-------------------------------------------------------------*/
/* Open the cursor associated with the SQL statement...        */
/*-------------------------------------------------------------*/
   Do
     parm_var = "STAFF";
     Call sqlexec 'OPEN c1 USING :parm_var';
/*-------------------------------------------------------------*/
/* Fetch and display each row selected...                      */
/*-------------------------------------------------------------*/
     Do While ( SQLCA.SQLCODE = 0 )
        Call sqlexec 'FETCH c1 INTO :table_name';
        If (SQLCA.SQLCODE = 0) Then
           Say 'Table = ' table_name;
        End
/*-------------------------------------------------------------*/
/* Close the cursor and end the transaction...                 */
/*-------------------------------------------------------------*/
       Call sqlexec 'CLOSE c1';
       Call sqlexec 'COMMIT';
   End
/*-------------------------------------------------------------*/
/* Disconnect from the database...                             */
/*-------------------------------------------------------------*/
Call sqlexec 'CONNECT RESET';
Return
```

Example 2

```
/*-------------------------------------------------------------*/
/* Display the names of all tables owned by the default user.*/
/* GUROO version.                                              */
/*-------------------------------------------------------------*/
/*-------------------------------------------------------------*/
/* Connect to the SAMPLE database...                           */
/*-------------------------------------------------------------*/
if sql_connect('sample') < 0 Then
   Do
     Say sql_error_text()
     Exit
   End
/*-------------------------------------------------------------*/
/* Execute the select statement and return data...             */
/*-------------------------------------------------------------*/
st = "SELECT name FROM sysibm.systables WHERE name <> ?"
parm_var = 'STAFF'
if sql_command(q1,st,parm_var) < 0 Then
   Do
     Say sql_error_text()
     Exit
   End
/*-------------------------------------------------------------*/
/* Display each row selected...                                */
/*-------------------------------------------------------------*/
Do i = 1 To q1.name.0
   Say 'Table = ' q1.name.i
End
/*-------------------------------------------------------------*/
/* End the transaction...                                      */
/*-------------------------------------------------------------*/
if sql_command(q1,"COMMIT") < 0 Then
   Do
     Say sql_error_text()
     Exit
   End
/*-------------------------------------------------------------*/
/* Disconnect from the database...                             */
/*-------------------------------------------------------------*/
if sql_disconnect() < 0 Then
   Do
     Say sql_error_text()
     Exit
   End
Return
```

What Is GUROO

In its current original form, GUROO is a standalone program, written using Oracle's Pro*C and linked with Regina. GUROO is really 7 external functions:

- sql_connect()
  connect to an Oracle database
- sql_disconnect()
  disconnect from an Oracle database
- sql_command()
  execute an SQL command (select, update etc)
- sql_open_cursor()
  open a cursor
- sql_close_cursor
  close a cursor
- sql_fetch_row()
  fetch a row from a cursor
- sql_error_text()
  return the text of the last GUROO or Oracle error

For more details on the syntax of these functions, see the attachment.

Current Status

GUROO is currently in use solely as an internal tool within the Information Systems section of Griffith University. Many of the DBA tools are written using GUROO and the programming staff have also begun to use GUROO in situations where the Oracle supplied tools are inappropriate. In one instance, GUROO has replaced one function which was originally written using various combinations of C shell, awk, SQL*ReportWriter, SQL*Plus and SQL*Loader. The GUROO program is quicker, smaller and much easier to understand.

Despite being interpreted, the performance of GUROO programs is on par with other Oracle tools.

Currently, GUROO is not available for distribution outside of the Information Systems section of Griffith University.

Future Directions

As a result of Griffith University's reluctance to allow distribution of GUROO, I and a colleague of mine have begun an independent development of a similar tool; REXX/SQL. The structure and operation of REXX/SQL will be fundamentally the same as GUROO, but will also include most of the low-level functions like PARSE and EXECUTE that exists in current tools. This will give users the option of a simple interface or one which they are more familiar (for users of RXSQL, DB2 etc).

REXX/SQL will have the ability to make multiple connections to the same or different databases from the same vendor or different vendors. The ultimate goal for REXX/SQL will be to allow a programmer to access data from any combination of SQL databases as though all data were stored in the one database.

This can really only be achieved by writing each database access functions as dynamically linked libraries that can be loaded at run-time. Example 3 illustrates this goal.

Example 3

```
/*-------------------------------------------------------------*/
/* Display the name and payment details for all employees.     */
/* Employee information is stored in a DB2 database,           */
/* financial information stored in an Oracle database.         */
/* REXX/SQL multi-database example.                            */
/*-------------------------------------------------------------*/
/*-------------------------------------------------------------*/
/* Load the Oracle and DB2 external function libraries...      */
/*-------------------------------------------------------------*/
Call rxfuncadd 'LoadOracleFuncs','REXXSQL','LoadOracleFuncs'
Call LoadOracleFuncs
Call rxfuncadd 'LoadDB2Funcs','REXXSQL','LoadDB2Funcs'
Call LoadDB2Funcs
/*-------------------------------------------------------------*/
/* Connect to local Oracle database...                         */
/*-------------------------------------------------------------*/
If oracle_connect('/') < 0 Then
   Do
     Say oracle_error_text()
     Exit
   End
/*-------------------------------------------------------------*/
/* Connect to local DB2 database PERSONNEL..                   */
/*-------------------------------------------------------------*/
If DB2_connect('PERSONNEL') < 0 Then
   Do
     Say DB2_error_text()
     Exit
   End
/*-------------------------------------------------------------*/
/* Declare queries to be performed...                          */
/*-------------------------------------------------------------*/
query1 = 'select name,empno,from emp order by name'
query2 = 'select amt,paydate from gl_trans where accno = :ACCNO'
/*-------------------------------------------------------------*/
/* Execute the first query on the DB2 database and return all*/
/* rows ...                                                    */
/*-------------------------------------------------------------*/
if DB2_command(q1,query1) < 0 Then
   Do
     Say DB2_error_text()
     Exit
   End
/*-------------------------------------------------------------*/
/* For each employee record, obtain the payment details from */
/* the Oracle database and display them...                     */
/*-------------------------------------------------------------*/
Do i = 1 To q1.name.0
   if oracle_command(q2,query2,'ACCNO',q1.empno.i) < 0 Then
      Do
        Say oracle_error_text()
        Exit
      End
   Say ''
   Say Right(q1.empno.i,8) Left(q1.name.i,35)
   Do j = 1 To q2.amt.0
      Say Copies(' ',15) Left(q2.paydate.j,12) Right(q2.amt.j,12)
   End
End
/*-------------------------------------------------------------*/
/* Disconnect from the Oracle database...                      */
/*-------------------------------------------------------------*/
If oracle_disconnect() < 0 Then
   Do
     Say oracle_error_text()
     Exit
   End
/*-------------------------------------------------------------*/
/* Disconnect from the DB2 database...                         */
/*-------------------------------------------------------------*/
If DB2_disconnect() < 0 Then
   Do
     Say DB2_error_text()
     Exit
   End
Return
```

ATTACHMENT A

Synopsis

sql_connect(username/password[,remote_database_string])

Description

This function connects you to an oracle database. You supply the function with the username/password and optionally the remote database connect string.

Arguments

username/password	- username/password
remote_database_string	- connect string for remote database

Return Values

0	- successful connection
Negative number	- Oracle error number

Example

To connect as your OPS$ Oracle login you would use the following call:

```
rcode = sql_connect('/')
```

To connect to the Oracle account SCOTT with TIGER as password:

```
rcode = sql_connect('scott/tiger')
```

To connect to the Oracle account SCOTT with TIGER as password on the remote host overthere; Oracle SID of X, using SQL*Net TCP/IP:

```
rcode = sql_connect('scott/tiger','T:overthere:X')
```

Synopsis

sql_disconnect()

Description

This function disconnects you from an oracle database and commits any outstanding transaction. By default, whenever the GUROO program exits, you are disconnected from the database.

Arguments

None

Return Values

0	- successful connection
Negative number	- Oracle error number

Example

```
rcode = sql_disconnect()
```

Synopsis

sql_command(statement_name,sql_command[,bind_variable_name,bind_variable_value...])

Description

This function enables you to execute any Oracle SQL*Plus command including DML and DDL statements. Typically you would execute commands like *select* or *update* using this function. Note that the command does not end in a semi-colon. If you do append a semi-colon to the end of the command, GUROO will remove it.

When the SQL command issued is a *select* statement, GUROO returns all column values in arrays. The stem variable name is composed of the statement name followed by a period followed by the column name specified in the select statement. As with all REXX arrays the number of elements in the array is stored in the variable with an index of 0. When the value of a column is NULL, an extra REXX variable is created. This variable has the same structure as the REXX variable containing the column value, but with 'NULL' before the index value. For example; the REXX variable created for a select statement containing the column 'COL_NAME' and a statement name of 'Q1' will be Q1.COL_NAME.1 (for the first row). If the value of that column is NULL, the REXX variable created is Q1.COL_NAME.NULL.1. To determine if a column is NULL, use the following test:

```
If q1.col_name.null.i = 'NULL' Then ... (column is NULL)
```

Because the contents of all columns for all rows are returned from a 'select' statement, the select command may return many rows and exhaust memory. Therefore the use of sql_command() should be restricted to queries that will return a small number of rows. For larger queries use a combination of sql_open_cursor() and multiple sql_fetch_row() calls.

When bind variables are used, a pair of arguments is used for each unique bind variable name. The first argument is the name of the bind variable as specified in the SQL statement, the second is the value that bind variable is to take.

Arguments

statement_name	- a string of up to 30 characters to identify the SQL command. This is used as the first part of the stem variable name containing column values.
sql_command	- any valid SQL*Plus command.
bind_variable	- optional bind variables values as specified in the SQL command.

Return Values

Positive number	- successful operation; the number of rows affected by the SQL command.
Negative number	- Oracle error number

Example

To select the names of all tables owned by the current user, issue the following call:

```
rcode = sql_command('Q1','select table_name from user_tables')
```

Assuming the user owns the tables EMP, DEPT, and CUSTOMER rcode will be set to 3 and the following REXX variables will be set:

```
Q1.TABLE_NAME.0 = 3
Q1.TABLE_NAME.1 = EMP
Q1.TABLE_NAME.2 = DEPT
Q1.TABLE_NAME.3 = CUSTOMER
```

To select the names of all tables and the tablespace in which they reside owned by the user, SCOTT, and use a bind variable, issue the following call:

```
query.1 = 'select table_name,tablespace_name'
query.2 = 'from all_tables where owner = :OWNER'
query = query.1 query.2
rcode = sql_command('Q1',query,'OWNER','SCOTT')
```

Assuming that user SCOTT owns the tables:

EMP	in TEMP_SPACE tablespace
DEPT	in USER_SPACE tablespace
CUSTOMER	in TEMP_SPACE tablespace
PRICE	in USER_SPACE tablespace

rcode will be set to 4 and the following REXX variables will be set:

```
Q1.TABLE_NAME.0 = 4
Q1.TABLE_NAME.1 = EMP
Q1.TABLE_NAME.2 = DEPT
Q1.TABLE_NAME.3 = CUSTOMER
Q1.TABLE_NAME.4 = PRICE

Q1.TABLESPACE_NAME.0 = 4
Q1.TABLESPACE_NAME.1 = TEMP_SPACE
Q1.TABLESPACE_NAME.2 = USER_SPACE
Q1.TABLESPACE_NAME.3 = TEMP_SPACE
Q1.TABLESPACE_NAME.4 = USER_SPACE
```

To delete rows from the EMP table where DEPTNO = 10, issue the following call:

```
rcode = sql_command('Q1','delete from emp where deptno = 10')
```

Assuming there were 5 rows in EMP for DEPTNO 10,rcode will be set to 5.

To delete rows from the EMP table where DEPTNO = 10, issue the following call:

```
rcode = sql_command('Q1','delete from emp where deptno = 10')
```

Assuming there is no table called EMP, then rcode is set to -947; the Oracle error number.

Synopsis

sql_open_cursor(statement_name,sql_command[,bind_variable_name,bind_variable_value...])

Description

This function passes a *select* statement to be parsed, prepared and executed. The rows that the SQL command retrieves are then made ready for repeated calls by sql_fetch_row(). An explicit cursor is associated with the statement name passed.

This function takes the same arguments as sql_command().

Arguments

statement_name	- a string of up to 30 characters to identify the SQL command. This is used as the first part of the stem variable name containing column values and as the argument to sql_fetch_row() so it knows from which cursor to fetch rows.
sql_command	- any valid SQL*Plus *select* command.
bind_variable	- optional bind variables values as specified in the SQL command.

Return Values

0	- successful connection
Negative number	- Oracle error number

Example

To prepare for returning the names of tables owned by the current user:

```
rcode = sql_open_cursor('Q1','select table_name from user_tables')
```

Assuming the user has select permission on the user_tables object, rcode is set to 0.

See sql_fetch_row() for further examples.

Synopsis

sql_fetch_row(statement_name)

Description

This function retrieves the next row in the previously opened cursor and sets REXX variables for each column specified in the sql_command passed to the sql_open_cursor() function.

The format of the REXX variables set is statement_name followed by a period followed by the column name. If the value of a column is NULL, an extra REXX variable is created. See the format and usage in the description for sql_command.

Arguments

statement_name	- a string of up to 30 characters to identify the SQL command. This is used as the first part of the stem variable name containing columns values and as the argument to sql_open_cursor().

Return Values

0	- successful connection
Negative number	- end of cursor

Example

To return the names of tables owned by the current user using an explicit cursor:

```
rcode = sql_open_cursor('Q1','select table_name from user_tables')
If rcode < 0 Then
Do
   Say sql_error_text()
   Exit 1
End
Do Forever
   rcode = sql_fetch_row('Q1')
   If rcode < 0 Then Leave
   Say q1.table_name
End
rcode = sql_close_cursor('Q1')
```

Assuming the user owns the tables, EMP, DEPT, and CUSTOMER the output from this code will be:

```
EMP
DEPT
CUSTOMER
```

Synopsis

sql_close_cursor(statement_name)

Description

This function closes the cursor associated with statement_name and frees up resources held by that cursor.

Arguments

statement_name	- a string of up to 30 characters used to identify which cursor is to be closed.

Return Values

0	- successful connection
Negative number	- Oracle error number

Example

To close an already opened cursor:

```
rcode = sql_close_cursor('Q1')
```

See sql_fetch_row() for a further example.

Synopsis

sql_error_text()

Description

This function returns the text of the last error encountered from the most recent GUROO external function. The error may relate to an Oracle error or to an error within GUROO itself.

If the most recent GUROO external function was successful, then the value returned is 'Last operation successful'.

Arguments

None

Return Values

Text of the result of the most recent GUROO external function.

Example

To display the text of the result of the most recent operation:

```
Say sql_error_text()
```

See sql_fetch_row() for a further example.