18th International Rexx Symposium
29 April – 3 May, 2007
Sheraton Tampa Riverwalk – Tampa, FL USA
Sponsored By The

Rexx/SQL Procedural To Object Oriented
Presented By

Lee Peedin, VP Research & Development
Safe Data, Inc. Wallace, NC
lee@safeddatausa.com
Rexx/SQL Procedural To Object Oriented – Part 1

1. What SQL Is and What SQL Is Not
2. What Rexx/SQL Is
3. The Rexx/SQL Functions
4. Straight From The “Horse’s Mouth”
5. The SQL Communications Area (sqlca.)
6. Setting Up For MySQL - ODBC
7. Procedural Coding For Startup & Connect
8. Procedural Coding For Select, Show Statements
9. Procedural Coding For Insert, Update, Delete, Create, Drop, Alter, etc. Statements
10. A Sample Program Using Procedural Coding
Rexx/SQL Procedural To Object Oriented – Part 2

1. Creating An Object Oriented Wrapper Class
2. Object Oriented Coding For Startup & Connect
3. Object Oriented Coding For Select, Show Statements
4. Object Oriented Coding For Insert, Update, Delete, Create, Drop, Alter, etc. Statements
5. A Sample Program Using the Wrapper Class
6. Let The SQL DBMS Server Do The Work!
“SQL” IS NOT a database management system (DBMS).

DBMS are products such as MySQL, Oracle, DB2, mSQL, etc.

SQL IS a “structured query language” that provides a means to access data stored in a DBMS.

```sql
select somecolumnndata from atable where atest
update atable set somecolumnndata = somenewvalue
```
A DBMS will have a basic structure

- Database(s)
- Table(s)
- Rows
- Columns

- safedata (database)
- employees (table – each row has a column for uid, name, address, birthdate, etc.)
- assets (table – each row has a column for uid, description, purchasedate, etc.)
What Rexx/SQL Is

- **Rexx/SQL** provides Rexx programmers with a consistent, simple, yet powerful interface to DBMS that support SQL.
- Multiple connections to different databases from different vendors can be made in the one Rexx program.
- Multiple statements can be open on each database connection at the same time.
- Databases supported by **Rexx/SQL** include Oracle, mSQL, DB2, SyBase, MySQL, Solid Server and SQLite.
- **Rexx/SQL** also supports access to ODBC datasources such as Excel and Access.
Rexx/SQL Procedural To Object Oriented – Part 1

The Rexx/SQL Functions

- SQLLoadFuncs
- SQLConnect
- SQLPrepare
- SQLOpen
- SQLExecute
- SQLFetch
- SQLDispose
- SQLClose
- SQLDisconnect

SQLDropFuncs
- SQLCommit
- SQLRollback
- SQLVariable

- SQLDescribe
- SQLGetinfo
- SQLDatasources
- SQLTables
- SQLColumns
- SQLDefault
- SQLGetdata

SQLCommand
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.

There following are reasons why you might need to consider using the individual Rexx/SQL functions rather than SQLCOMMAND:

1. When you need to execute the same query multiple times with different values of columns in the WHERE clause. See *Other DML Statements* below for more details.

2. When the number of rows expected to be returned is very large. SQLCOMMAND fetches every row from the query into stems for each column. If you are returning a large number of rows this can take quite a long time and use quite a lot of memory for the column contents. Calling SQLFETCH for each row, or fetching a small number of rows, say 100, in each call to SQLFETCH will reduce memory usage. It won't however reduce the time it takes; it will increase it if you eventually return every row.

3. When you don't require the contents of every row in the query. In this case you may have a query that returns many rows, but you are only interested in the first row. Rather than have SQLCOMMAND fetch every row, you can simply call SQLFETCH once to get the contents of the first row of data.”
<table>
<thead>
<tr>
<th>Select, Show</th>
<th>Insert, Update, Delete, Create, Drop, Alter, etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLPrepare</td>
<td>SQLPrepare</td>
</tr>
<tr>
<td>SQLOpen</td>
<td>SQLExecute</td>
</tr>
<tr>
<td>SQLFetch (in loop)</td>
<td>SQLCommit</td>
</tr>
<tr>
<td>SQLClose</td>
<td>SQLDispose</td>
</tr>
<tr>
<td>SQLDispose</td>
<td></td>
</tr>
</tbody>
</table>
The SQL Communications Area

Every Rexx/SQL function call, creates a stem called “sqlca.” This stem contains the following information:

- **SQLCA.SQLCODE**: result code of last SQL operation
- **SQLCA.SQLErrM**: text of any error message associated with the above result code
- **SQLCA.SQLSTATE**: a detailed status string (N/A on some ports)
- **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

And again from the documentation:

Page 54: SQLCA. stem is read-only; don't change values or DROP the variables. You have been warned!

And from the experience of Lee:

Do NOT use any of the leaf names (rowcount) as variables in your program unless you drop them just before retrieving the value of sqlca.leaf. You have been warned! 😊
Rexx/SQL Procedural To Object Oriented – Part 1
Setting Up From MySQL - ODBC


Once downloaded and installed: Start > Control Panel > Administrative Tools > Data Sources (ODBC) > System DSN
if rxfuncquery('SQLLoadFuncs') then call rxfuncadd 'SQLLoadFuncs', 'rexxsql'
if rxfuncquery('sqlconnect') then call sqlloadfuncs

dsn = 'leegreen'
database = 'rexxla'
con_str = dsn';database='database

if sqlconnect('c1',,,, 'dsn='con_str) < 0 then call mySQLError .false
tablename = 'asampletable2'
ss = "select * from" tablename "where etype is null"
if sqlprepare('q1',ss) < 0 then call mySQLError .false
if sqlopen('q1') < 0 then call mySQLError .false
do forever
   rv = sqlfetch('q1')
   if rv < 0 then call mySQLError .false
   if rv = 0 then leave
   -- data for 1 row returned as q1.column_name
end
count = sqlca.rowcount
if sqlclose('q1') < 0 then call mySQLError .false
if sqldispose('q1') < 0 then call mySQLError .false
ss = “update” tablename “set etype = 1 where etype is null”
if sqlprepare('e1',ss) < 0 then call mySQLException .false
if sqlexecute('e1') < 0 then call mySQLException .false
if sqlcommit() \= 0 then call mySQLException .false
if sqldispose('e1') < 0 then call mySQLException .false
A Sample Program Using Procedural Coding

demo_procedural.rex

(Pass out demo_procedural.rex code)

In the example code, you will note that the data returned by the SQL “select” statements is converted to an “array of directories”. To allow Rexx/SQL to work with Regina, etc., all Mark had was “a hammer”.

In this code and the code that follows in part 2, we’ll make a directory from each returned row and make each directory an index of an array.

A word about SQLyog

1. Creating An Object Oriented Wrapper Class
2. Object Oriented Coding For Startup & Connect
3. Object Oriented Coding For Select, Show Statements
4. Object Oriented Coding For Insert, Update, Delete, Create, Drop, Alter, etc. Statements
5. A Sample Program Using the Wrapper Class
6. Let The SQL DBMS Server Do The Work!

(Pass out RxSQLWrapper.cls code)
We are about to create an ooRexx class file.

- Can be used by all your ooRexx programs that need access to Rexx/SQL.
- Will work on any platform supported by ooRexx & Rexx/SQL.
- Can be used with any Rexx/SQL supported DBMS (may need to modify the “connect” method).
- Will have less than 150 lines of code (including blank lines).
- Can be built upon to include other Rexx/SQL BIFs that your DBMS may not provide internally.

So What Will This Class Do For Us?
Creating An Object Oriented Wrapper Class

### ss = “update users set clue = 1 where pigs_fly is not null”

<table>
<thead>
<tr>
<th>Without Wrapper</th>
<th>With Wrapper</th>
</tr>
</thead>
<tbody>
<tr>
<td>if sqlPrepare('u1',ss) &lt; 0 then…</td>
<td>if object_name~execute(ss,.true) = 0 then…</td>
</tr>
<tr>
<td>if sqlExecute('u1') &lt; 0 then…</td>
<td>update_counter = object_name~rowcount</td>
</tr>
<tr>
<td>if sqlCommit('u1') = 0 then…</td>
<td></td>
</tr>
<tr>
<td>if sqlDispose('u1') &lt; 0 then…</td>
<td></td>
</tr>
<tr>
<td>update_counter = sqlca.rowcount</td>
<td></td>
</tr>
</tbody>
</table>

5 lines - 2 lines

60% reduction in code
Creating An Object Oriented Wrapper Class

```plaintext
ss = “select * from users where clue > 0”

Without Wrapper

if sqlPrepare(‘q1’, ss) < 0 then…
if sqlOpen(‘q1’) < 0 then…
rows = .array~new()
do forever
    rv = sqlfetch(‘q1’)
    if rv < 0 then… --Error!
    if rv = 0 then leave
next_rows = rows~items + 1
rows[next_rows] = .directory~new
do x over q1.
    parse lower var x xx
    rows[next_rows][xx] = q1.x
end
end
if sqlClose(‘q1’) < 0 then…
if sqlDispose(‘q1’) < 0 then…

With Wrapper

if object_name~query(ss) \= 0 then…
rows = object_name~rows
```

16 lines - 2 lines
87.5% reduction in code
Creating An Object Oriented Wrapper Class

/* RxSQLWrapper.cls */
::class rxsqlwrapper public

::method rowcount attribute  -- Used to hold the rowcount
::method rows attribute      -- An array of directories of select/show results
::method details attribute   -- A directory of the contents of sqlca.
::method last_insert_id attribute  -- Used to hold the last insert id of insert statements
::method vardir attribute    -- Values of Rexx/SQL Variables
::method init

    expose connection varlist

    use arg connection

    if rxfuncquery('SQLLoadFuncs') then call rxfuncadd 'SQLLoadFuncs', 'rexxsql'

    if rxfuncquery('sqlconnect') then call sqlloadfuncs

    varset = .set~of (VERSION, DEBUG, ROWLIMIT, LONGLIMIT, SAVESQL, AUTOCOMMIT, IGNORETRUNCATE, ,
                      NULLSTRINGOUT, NULLSTRINGIN, STANDARDPLACEMARKERS, SUPPORTSPLACEMARKERS, ,
                      SUPPORTSDMLROWCOUNT, SUPPORTSTHREADS, ALL) --ALL added to list – not a normal sqlvariable

    return
::method connect
  expose connection sqlca.
  use arg dsn
  rv = sqlconnect(connection,,,'dsn='dsn)
  self~populatedetails()
return rv
::method disconnect

  expose connection sqlca.
  
  rv = sqldisconnect(connection)
  
  self~populatedetails()

return rv
::method query

expose connection rowcount rows last_insert_id sqlca.

use arg statement

drop rowcount

if sqldefault(connection) < 0 then;self~populatedetails();return -1;end

if sqlprepare('q1',statement) < 0 then;self~populatedetails();return -1;end

if sqlopen('q1') < 0 then;self~populatedetails();return -1;end

rows = .array~new()

next_rows = 0  -- in case none are fetched
do forever
  rv = sqlfetch('q1')
  if rv < 0 then do;self~populatedetails();return -1;end
  if rv = 0 then leave
  next_row = rows~items+1
  rows[next_row] = .directory~new
  do x over q1.
    parse lower var x xx
    rows[next_row][xx] = q1.x
  end
end
rowcount = next_row
if sqlclose('q1') < 0 then do;self~populatedetails();return -1;end
if sqldispose('q1') < 0 then do;self~populatedetails();return -1;end
self~populatedetails()
return 0
Rexx/SQL Procedural To Object Oriented – Part 2
An Object Oriented Wrapper Class - execute

::method execute

  expose connection rowcount rows last_insert_id sqlca.

  use arg statement,autocommit

  drop rowcount
  if arg(2,'o') then autocommit = .false
  if \autocommit~datatype('o') then raise syntax 34      -- Only available in 3.1.2

  if sqldefault(connection) < 0 then;self~populatedetails();return -1;end
  if sqlprepare('e1',statement) < 0 then;self~populatedetails();return -1;end
  if sqlexecute('e1') < 0 then;self~populatedetails();return -1;end

  rowcount = sqlca.rowcount
  if statement~word(1)~translate = 'INSERT' then do
    drop last_insert_id
    rv = sqlcommand('qlid','select last_insert_id() as lid')
    last_insert_id = qlid.lid.1
  end
  if autocommit then self~commit()
  if sqldispose('e1') < 0 then;self~populatedetails();return -1;end

  self~populatedetails()

  return 0
::method commit
   expose connection sqlca.
   if sqldefault(connection) < 0 then do;self~populatedetails();return -1;end
   rv = sqlcommit()
   self~populatedetails()
return 0
::method rollback

    expose connection sqlca.
    
    if sqldefault(connection) < 0 then do;
        self~populatedetails();
        return -1;
    end

    rv = sqlrollback()
    self~populatedetails()

    return 0
An Object Oriented Wrapper Class - variable

::method variable
  expose connection rowcount rows last_insert_id sqlca. vardir varset
  use arg var_name, var_value
  if sqldefault(connection) < 0 then do; self~populatedetails(); return -1; end
  save_case = var_name -- return back same case as supplied
  var_name = var_name~translate
  drop vardir
  if \varset~hasindex(var_name) then
    raise syntax 40.26 array('The Variable', 1, arg(1))
  if arg(2,'e') & var_name = 'ALL' then raise syntax 93.902 array(1)
  if arg(2,'e') then do
    rv = sqlvariable(var_name, var_value)
    if rv \= 0 then do; self~populatedetails(); return -1; end
    self~populatedetails()
    return 0
  end
vardir = .directory~new()

if var_name = 'ALL' then do i over varset
  rv = sqlvariable(varset[i])
  vardir[varset[i]] = rv
end

else do
  rv = sqlvariable(var_name)
  if rv \= 0 then do; self~populatedetails(); return -1; end
  vardir[save_case] = rv
end

rv = 0

self~populatedetails()

return 0
::method populatedetails
  expose details sqlca.

  details = .directory~new()

  do x over sqlca.
    if x~pos('.') > 0 then do
      parse lower var x y '.' i
      if i \= 0 then do
        if \details~hasindex(y) then details[y] = .array~new()
        if details[y]~defaultname() = 'an Array' then details[y][i] = sqlca.x
      end
    end

  end

  else do
    parse lower var x y
    details[y] = sqlca.x

  end

end
This demo does exactly the same thing as the previous demo. There are corresponding blank lines in both programs; however, this demo, using our “wrapper class” resulted in a 36.6% reduction in lines of code.
Let The SQL DBMS Server Do The Work

let_server_do_work.rex

(Pass out let_server_do_work.rex code)
No matter which SQL DBMS you choose to use, study its documentation. It’ll save you many lines of code and a lot of “execution” time! – Rexx does the work below:

```rexx
ss = "select enum,rptdate1,stime,etime,refno1 from tablename where rptdate1 >= 20070301 order by"
     "enum,rptdate1,stime"
if osql~query(ss) \= 0 then call mySQLError .false
rows = osql~rows
ostream = .stream~new(csvfile)
ostream~open("WRITE REPLACE")
do i = 1 to rows~items
   dline = '"rows[i]['enum']"','||­
            '"rows[i]['rptdate1']"','||­
            '"rows[i]['stime']"','||­
            '"rows[i]['etime']"','||­
            '"rows[i]['refno1']"'
ostream~lineout(dline)
end
ostream~close
```
No matter which SQL DBMS you choose to use, study its documentation. It’ll save you many lines of code and a lot of “execution” time! SQL Server now does the work.

```plaintext
ss = "select enum,rptdate1,ifnull(stime,""),ifnull(etime,""),refno1 into outfile ""csvfile"" fields terminated","n by ""dquotes"" lines terminated by "\r\n" from" tablenamename "where rptdate1 >= 20070301",
     "order by enum,rptdate1,stime"
if osql~execute(ss) != 0 then call mySQLLError .false
```