SLAC-TN-90-2
February 1990
(TN)

USING REXX TO COORDINATE RELATIONAL COMMANDS*

Frank Rothacker

Stanford Linear Accelerator Center
Stanford University, Stanford, CA 94309

Relational technology has drastically reduced, but not eliminated, the need for a procedural language. The main advantage of relational commands is that they can manipulate entire sets of records in complex ways, without the need to loop through files. What remains is coordinating the execution of relational commands. This paper discusses how Rexx may help to fill that void.

The problem

SQL*PLUS was designed to be an interactive interface to the Oracle database. With SQL*PLUS the user can issue a wide variety of commands and immediately see the results on the terminal screen.

An important advantage of SQL*PLUS is that it is complete---all Oracle commands can be issued from SQL*PLUS. Another advantage of SQL*PLUS is that it is interactive---the result of each command is displayed on the screen, without the need to compile a program.

* Work supported by Department of Energy contract DE–AC03–76SF00515.

A serious limitation of SQL*PLUS is that the task of issuing Oracle commands cannot be automated. Commands must be manually entered through the keyboard. The START command can execute a file of SQL*PLUS commands, but conditional logic is not possible. The new language PL/SQL has been proposed by Oracle Corporation as a solution to this problem, but PL/SQL blocks cannot even issue SQL*PLUS commands, DDL commands, or DCL commands, and the SELECT command must contain the INTO option.

For the above reasons, all procedural programming must be done in an old-fashioned 3GL like Cobol, Fortran, etc. This has many limitations. The programs must be compiled. Instead of displaying output from database commands on the screen, it is placed into host variables, making it difficult to watch what is happening. Few 3GL compilers provide for interactive tracing of program execution, making object programs difficult to debug. Most 3GL programming languages have poor access to operating system commands, making it difficult to write programs that use database information to issue operating system commands, and difficult to store operating system information in the database. Most 3GL's were designed at a time when keypunch machines were used for data entry, punched cards were in fixed format, and programs had little need for parsing input. These languages still have little facility for string parsing and manipulation.

Given the tools we presently have to work with, how can we write a simple program to create and execute an Oracle SELECT command and display the output on the terminal? The answer is, we can't!

Proposed solution

In the VM environment, Rexx is ideally suited for communicating with the operating system. In addition Rexx requires no compiling, has interactive tracing, a rich function library, powerful string manipulation features, and is easy to learn. Rexx control structures are uniquely powerful, having features not found in most other programming languages. Most VM programmers already know Rexx and use it at least some of the time. In fact, many VM new

applications use Rexx as the primary programming language, because Rexx is so easy.

Why not use Rexx as a procedural language for SQL*PLUS? Using Rexx with SQL*PLUS requires three things:

1. The user must be able to call Rexx from SQL*PLUS.
2. Rexx must be able to send commands to SQL*PLUS for execution.
3. Rexx must be able to access the output from SQL*PLUS.

Most of these three capabilities almost exist. Let's go over each of them.

Calling Rexx

Rexx can be called by using the SQL*PLUS command HOST EXEC. A better way would be for Oracle Corporation to enhance SQL*PLUS so that it works a bit more like VM. In the VM environment, the operating system first searches disks for EXEC files before executing commands. If an EXEC is found, it is executed, instead of the native VM command. The Rexx EXEC can then modify the user's command, and if necessary, issue the native version of the command.

Sending commands to SQL*PLUS

After being called from the interactive SQL*PLUS environment, Rexx must be able to send commands back to SQL*PLUS for execution. Using the present facilities in SQL*PLUS, this can be done either by stacking SQL*PLUS commands, or by writing them to a file and stacking an SQL*PLUS START command. In many cases this works fine. As soon as Rexx finishes execution, SQL*PLUS reads and executes the stacked commands.

The problem with this approach is that SQL*PLUS will not execute the stacked commands until the Rexx EXEC terminates execution. The only way for the Rexx program to regain control is to stack a call to itself each time it

finishes execution. Exiting Rexx destroys all local variables and control information. The Rexx programmer must preserve this information by writing it to disk or storing it in global variables. Such programs are difficult to write and tedious to debug.

A much more straightforward approach is to use the existing SUBCOM entry point within SQL*PLUS. For example, to learn what columns are in a table, the Rexx EXEC could contain the command:

ADDRESS SQLPLUS DESCRIBE <table name>

With spooling turned on, the Rexx program can then use EXECIO to read the disk, and learn how the table was structured. The information from the DESCRIBE command could then be used by Rexx to format a SELECT command.

This works because the present SQL*PLUS program supplied by Oracle Corporation always issues a VM operating system command called SUBCOM. SUBCOM is an IBM 370 assembly language command. Programs like Oracle use SUBCOM to define internal entry points to the VM operation system. Once such an entry point has been defined, other programs can call it using an SVC 202, in exactly the same way they call programs residing on disk. Rexx programmers use the command "ADDRESS SQLPLUS" to access the internal entry point defined by SQL*PLUS. In this way, the above DESCRIBE command is passed to the internal entry point within SQL*PLUS. SQL*PLUS then executes the DESCRIBE and returns control to Rexx. In this way, control jumps back and forth between SQL*PLUS and Rexx, without the need to keep reloading or reinitializing either Rexx or SQL*PLUS.

Even though they might not realize it, VM programmers always use the SUBCOM operating system feature whenever they write an Xedit macro. By default, all Rexx programs with file type XEDIT automatically "ADDRESS XEDIT". Such programs can directly issue XEDIT commands, and regain control after Xedit executes the command.

Unfortunately, the present SQLPLUS entry point is not fully implemented. It only works with a few SQL*PLUS commands, and no SQL*PLUS DML, DDL, or DCL commands. Although we can issue a DESCRIBE command and use the result to create a SELECT, we cannot execute the SELECT until we exit Rexx.

Everyone would benefit greatly if Oracle Corporation would finish the implementation of the SQLPLUS entry point, so it also works with SQL commands.

Accessing SQL*PLUS output

SQL*PLUS was designed to be used interactively and the output of all commands usually goes to the screen. Fortunately, SQL*PLUS has SPOOL and TERMINAL commands, which can redirect the output to a disk file. Rexx can then read the file and act accordingly.

It works, but not well. Reading and writing a requires extra IO operations. The size of the spool file keeps growing, because each new output is appended to the end of the file. And SQL*PLUS complains when Rexx tries to erase the spool file.

The SQL*PLUS terminal setting is insensitive to errors that occur while it is in effect. Commands may fail without any indication that something is wrong. To see the error messages, the user must either turn typing back on, or examine the contents of the SQL*PLUS spool file (if one exists). The lack of error messages is especially frustrating when debugging a VM service machine, because nothing shows up in the console log.

The SPOOL and TERMINAL commands are global to the SQL*PLUS session. Once set, they remain in effect until changed or until the user exits SQL*PLUS. In order for a program to change the settings of SPOOL or TERMINAL, without changing the settings made by another program or by the user, the following steps are needed:

1. Determine the settings of SPOOL and TERMINAL.
2. Change them.
3. Issue the SQL*PLUS commands to create the spool files.
4. Restore the settings back to the way they were in Step 1.

The above is actually impossible. The spool setting can't be determined by a program without first turning on spooling, which also destroys the original setting!

In contrast, the way Rexx obtains information from the operating is usually very straightforward. The results of most commands can be placed in the program stack. Many VM commands also have a disk option, causing the results to be written to disk. A choice between "replace" and "append" gives the option of erasing the file each time, or adding onto the end of the file. The output of VM commands having syntax errors is not stacked or written to disk, but displayed on the screen, making it easy tell what went wrong. In addition to stacking, Rexx has many functions that return operating system information.

A major difference between VM and SQL*PLUS is that output options are given as part of each command, rather than through global settings. In an interactive environment, global settings are desirable because the user does not have to keep entering the options with each command. But in a program environment, global settings must always be preserved and restored. Otherwise modular programming is impossible. One program can't call another if global settings are not respected and restored.

The Oracle Corporation could greatly enhance the power of SQL*PLUS by providing output options for commands. The options could be recognized by a left parenthesis following the command, using the same syntax as VM options. An alternative, which is less likely to cause syntax conflicts, is to define a new command to act as prefix option for existing SQL*PLUS commands. For example, suppose the new command were called OPTIONS, then a SELECT command would be issued as:

```
OPTIONS( ... ) SELECT ... ;
```

The parenthesis would contain the output options for the SELECT command that followed. A possible set of keywords for the output option might be:

QUIET	No output is produced, except for syntax errors.
LIFO	Stacking last in, first out, unless syntax error.
FIFO	Stacking first in, first out, " " "
DISK fn.ft.fm	Output to disk, " " "
REP	Replace existing disk file, " " "
APPEND	Add to end of existing file, " " "

The stack option should prefix stacked lines with an asterisk so that the operating system will never try to execute them, even if a defective program leaves the lines in the stack.

It seems that adding output options to SQL*PLUS is far simpler than directing SQL*PLUS output to host variables. Unlike 3GL's, Rexx programs do not need a cursor. The console stack serves that purpose. VM programmers already know how to use the console stack. Why should they need to learn the concepts associated with DECLARE, OPEN, and FETCH? Having the choice between screen output and stacked output is far more useful than having all output directed to host variables.

Training advantages

Most programmers learn Oracle by starting with SQL*PLUS. A good Rexx interface would provide a natural transition from interactive use to program development. After watching what commands do on the screen, it is easy to think about EXEC's to automate the process. There is no sudden break to learn a whole new system of imbedded SQL, cursor management, host variables, etc. Not only would training costs be less, but programmers would become more productive as they wrote tools to automate their use of SQL*PLUS.

The competition

RXSQL is an IBM product giving Rexx programmers access to the relational database SQL/DS. Unlike the Rexx interface to Xedit, RXSQL does not use SUBCOM, but is implemented as a command. Instead of simply typing SQL commands in a Rexx exec, the programmer must prefix each one with 'RXSQL'. Rather than stacking the results of queries, RXSQL requires complicated cursor management, including learning entirely new commands called CREATE, PREPARE, DECLARE, OPEN, FETCH, and CLOSE. The picture is further complicated by the distinction between dynamic and extended dynamic SQL. IBM has provided some EXEC's to simulate multirow selects; however, headings and subtotals are not supported.

Nomad2 is a DBMS and 4GL from Must Software International. It includes a procedural language that can issue all database commands, test the results, and execute conditional logic. When an interactive user issues an undefined Nomad2 command, Nomad2 automatically searches for a procedural file on all accessed disks. In that way, native Nomad2 commands can be supplemented by user written commands. Nomad2 provides both single row (cursor style), as well as multirow table access commands. Output can optionally be directed to program variables, the screen, or to a file. When the programmer directs output to a file, he/she can either allow error messages from failed commands to appear on the screen for easy debugging, or provide for programmatic error handling. However, the Nomad2 procedural language lacks local variables and local subroutines, making modular programming difficult. It has poor string manipulation abilities. These defects can sometimes be overcome by calling a Rexx EXEC; however, Nomad2 lacks a SUBCOM entry point, making it very difficult for Rexx to pass commands back to Nomad2.

Conclusion

Rexx is a powerful and increasingly popular programming language. Although designed for VM it has been ported to MVS, VAX, and PS/2 computers. With a relatively small effort, this extensive procedural language

could be added to SQL*PLUS, providing a unique combination of database and procedural power. Oracle would have a product unmatched by competition.

To make this happen, Oracle Corporation needs to make some relatively minor modifications to SQL*PLUS. In order of priority, the most important ones are:

1. Allow the existing SUBCOM entry point to execute SQL commands.
2. Provide output stacking options for all commands.
3. Allow native SQL*PLUS commands to be replaced by Rexx programs of a certain file type, like SQLREXX.

The author gratefully acknowledges contributions made by George Crane, Geoffrey Girvin, and Sidney Orr.