IBM Software Group

DB2 Information Management Technical Conference :

Session Z12 REXX and DB2

Jim Ruddy IBM DB2 for z/OS Development
jaruddy@us.ibm.com

DB2 Information Management Software

e business on demand software

Student Notes

Jim Ruddy of DB2 for z/OS Development from the IBM Silicon Valley Laboratory will present the REXX language support for DB2 for z/OS Version 8 and DB2 for z/OS and OS/390 Version 7.

Overview

- Why use REXX?
- REXX Interface to DB2 for z/OS
- REXX Stored Procedures
- Sample to "DRAW"
  - SQL statements
  - LOAD statements
- Sample REXX Stored Procedure
- Sample REXX Calling Program

Student Notes

The presentation will describe the REXX language interface to DB2 for z/OS, the support for REXX language stored procedures, a sample ISPF edit macro program similar to the QMF DRAW command, a sample REXX language stored procedure, and a sample REXX language program which calls a stored procedure.

To order the REXX Language Support feature for DB2 for z/OS and OS/390 Version 7:

- **3480 cartridge, order feature #5963**
- **4mm cartridge, order 6011.**
- **6250 tape, order 5962**

The REXX Language Support feature for DB2 for Version 8 is included as part of the base product

Why REXX?

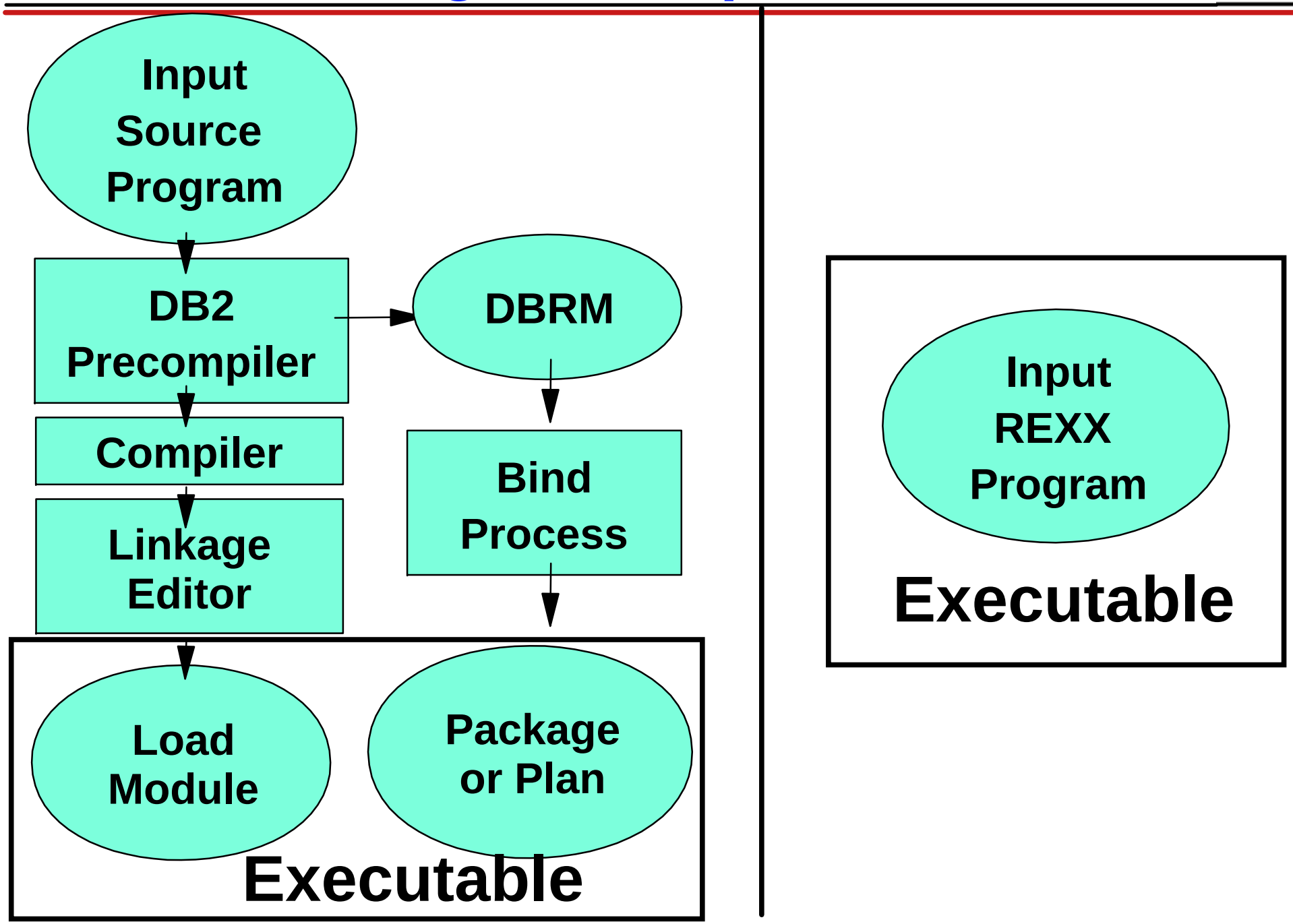
- Readability
- Natural data typing
- Emphasis on symbolic manipulation
- Nothing to declare
- Dealing with reality
- Interactive debugging

- One-step program preparation

Student Notes

REXX was designed with these principles in mind plus, when used with DB2, program preparation is greatly simplified over other languages.

Program Preparation



Student Notes

Typical preparation for a DB2 application program requires the use of the DB2 precompiler, the language compiler, the linkage editor, and the DB2 BIND command.

A REXX application program can be executed immediately after it is written.

REXX Interface to DB2 for z/OS

- Uses dynamic SQL
  - PREPARE STATEMENT
  - EXECUTE STATEMENT
- or
  - PREPARE STATEMENT
  - DECLARE CURSOR
  - DESCRIBE CURSOR
  - OPEN CURSOR
  - FETCH, FETCH, FETCH,... CURSOR
  - CLOSE CURSOR
- or
  - EXECUTE IMMEDIATE

Student Notes

All of the standard SQL related to cursors are supported. Cursors must be declared before they (or their associated statements) can be used.

REXX Interface Cursors

- 100 predefined statements and cursors
  - Statements S1..S100
  - DECLARE C1..C50 CURSOR WITH RETURN FOR S1..S50
  - DECLARE C51..C100 CURSOR WITH RETURN WITH HOLD FOR S51..S100
- 100 cursors for result sets
  - ALLOCATE C101..C200 CURSOR FOR RESULT SET

Student Notes

There are 100 predefined statements (S1 through S100) and cursors (C1 through C100). All cursors are defined "WITH RETURN" to allow them to be used to return result sets from stored procedures and cursor C51 through C100 are defined "WITH HOLD" to hold position across a commit.

Cursors C101 through C200 are available for ALLOCATE CURSOR for manipulating stored procedure result sets.

SQL Supported from REXX (cont.)

- DESCRIBE CURSOR / INPUT / PROCEDURE / TABLE / STATEMENT
- CALL
- ALLOCATE CURSOR
- ASSOCIATE LOCATOR
- SET
- CONNECT and RELEASE
- COMMIT and ROLLBACK
- All DDL
  - (implicit EXECUTE IMMEDIATE)

Student Notes

You can also DESCRIBE cursors, input, procedures, and tables. We will see a use for DESCRIBE TABLE later.

Calling stored procedures is supported as well as the SQL need for the handling of returned result sets. You can set special registers, connect and release to and from remote sites, commit and rollback, and plus execute all the SQL which can be dynamically executed.

DB2 for z/OS and OS/390 V7 Support

- SAVEPOINT
- ROLLBACK TO SAVEPOINT
- RELEASE TO SAVEPOINT
- Scrollable Cursors
  - SENSITIVE or INSENSITIVE
  - NEXT, PRIOR, FIRST, LAST, CURRENT, BEFORE, AFTER, ABSOLUTE n, RELATIVE n
  - ATTRIBUTES clause on PREPARE
  - or specified on FETCH

Student Notes

V7 SQL supports SAVEPOINTs and the ability to ROLLBACK to SAVEPOINTs or RELEASE TO SAVEPOINTs is handled natively. Also in V7 is support for scrollable cursors, both SENSITIVE and INSENSITIVE. You will be able to FETCH NEXT (the default), PRIOR, FIRST, LAST, CURRENT, BEFORE, AFTER, ABSOLUTE n, or RELATIVE n

DB2 for z/OS V8 Support

- SQL statements greater that 32K
- 28 times reduction in elapsed time
- 30% reduction in CPU time

Student Notes

V8 supports SQL statements up to 2 megabytes.

There are long awaited performance improvements in V8.

We have measured a 28 times reduction in elapsed time and about 30% reduction in CPU time

SQL Not Supported from REXX

- BEGIN/END DECLARE SECTION
- INCLUDE
- SELECT INTO
- WHENEVER

Student Notes

This SQL just doesn't make sense in a REXX program.

REXX Command Environment

- "SUBCOM DSNREXX"
  - /* HOST CMD ENV ALREADY AVAILABLE? */
- IF RC THEN
- RXSUBCOM('ADD','DSNREXX','DSNREXX')
  - /* ADD HOST CMD ENV */
- ADDRESS DSNREXX
  - change the destination of commands to DSNREXX
- ...
- RXSUBCOM('DELETE','DSNREXX','DSNREXX')
  - /* REMOVE CMD ENV */

Student Notes

First you must make sure the DSNREXX command environment exists. The REXX SUBCOM tests if the specified command environment exists. If it does not, the DSNREXX command environment is added with the RXSUBCOM call to ADD the DSNREXX command environment and specify the DSNREXX load module is to be called whenever a REXX statement is ADDRESSed to DSNREXX.

Connect to DB2 for z/OS

- add DSNREXX command environment
- **ADDRESS DSNREXX CONNECT ssid**
  - **Call Attach CONNECT, OPEN**
  - **not for stored procedures**
  - **NOT the same as SQL CONNECT**
- **...**
- **ADDRESS DSNREXX DISCONNECT**
  - **Call Attach CLOSE, DISCONNECT**
  - **not for stored procedures**
  - delete DSNREXX command environment

Student Notes

One way to access DB2 from REXX is by ADDRESSing the DSNREXX environment. The three commands supported in this environment are "CONNECT", "DISCONNECT", and "EXECSQL". "CONNECT" and "DISCONNECT" used when executing in TSO or batch and create the Call Attach connection and thread to the specified DB2 and then close the thread and connection. "EXECSQL" is for the dynamic processing of SQL.

If "CONNECT" is used, all access is through Call Attach. Otherwise, all access will use RRS Attach (as required by stored procedures). Use of Call Attach allows ISPF Edit Macros to be used from SPUFI as we will see in one of our examples.

USS Connect to DB2 for z/OS

- add DSNREXX command environment
- **Call SQLDBS "ATTACH TO " ssid**
  - **Call Attach CONNECT, OPEN**
  - **not for stored procedures**
  - **...**
- **Call SQLDBS "DETACH"**
  - **Call Attach CLOSE, DISCONNECT**
  - **not for stored procedures**
- delete DSNREXX command environment

Student Notes

Under Unix System Services (or when porting a REXX program from UNIX, Windows, or OS/2) the way to access DB2 from REXX is by calling SQLDBS or SQLEXEC. Only a subset of SQLDBS functions are support - "ATTACH TO ssid" and "DETACH" to create the Call Attach connection and thread to the specified DB2 and then close the thread and connection. CALL SQLEXEC is for the dynamic processing of SQL.

If "ATTACH TO ssid" is used, all access is through Call Attach. Otherwise, all access will use RRS Attach (as required by stored procedures).

Cursor Fetch Example

```
"EXECSQL DECLARE C1 CURSOR FOR S1"
STMT = "SELECT PHONENO",
       " FROM EMP",
       " WHERE LASTNAME = ?"
/* NULL test has to be "IS NULL" */
"EXECSQL PREPARE S1 FROM :STMT"
"EXECSQL OPEN C1 USING :LASTNAME"
DO WHILE SQLCODE \= 0
  "EXECSQL FETCH C1 INTO :PHONE"
  SAY "PHONE NUMBER FOR" LASTNAME "IS"
PHONE
END
"EXECSQL CLOSE C1"
```

Student Notes

Here we see a simple example of declaring, preparing, opening, fetching, and closing a cursor. Very straightforward, right?

But this is a bad example because the SQLCODE is not being checked after each SQL statement.

Update Example

```
STMT = "UPDATE EMP",
       " SET MIDINIT = ?",
       " WHERE EMPNO = '000200'"
"EXECSQL PREPARE S1 FROM :STMT"
IF SQLCODE < 0 THEN
  Call SQLError
INITIAL = 'M'
"EXECSQL EXECUTE S1 USING :INITIAL"
IF SQLCODE < 0 THEN
  Call SQLError
```

Student Notes

Here is an even simpler example showing how to perform updates from REXX.

This is a better example because the SQLCODE is being checked for errors.

SQLCA

- No "INCLUDE SQLCA"
- SQLCODE - The primary SQL return code.
- SQLERRMC - Error and warning message tokens.
- SQLERRP - Product code and, if there is an error, the name of the module that returned the error.
- SQLERRD.n - Six variables (n is a number between 1 and 6) containing diagnostic information.
- SQLWARN.n - Eleven variables (n is a number between 0 and 10) containing warning flags.
- SQLSTATE - The alternate SQL return code.

Student Notes

Of course, sometimes not all goes as you expected and you get an SQL error. The contents for the SQL Communication Area (SQLCA) are returned in these REXX variables. The contents of these are just as you would expect from your former favorite programming language such as C or COBOL. Or maybe you like PL/I or Assembler.

When using the SQLDBS or SQLEXEC interfaces, the SQLCA variables are prefixed with "SQLCA." for compatibility with DB2 for UNIX, and Windows.

SQLDA

- The following statements require an SQLDA:
  - EXECUTE...USING DESCRIPTOR descriptor-name
  - FETCH...USING DESCRIPTOR descriptor-name
  - OPEN...USING DESCRIPTOR descriptor-name
  - CALL...USING DESCRIPTOR descriptor-name
  - DESCRIBE statement-name INTO descriptor-name
  - DESCRIBE INPUT statement-name INTO descriptor-name
  - DESCRIBE CURSOR host-variable INTO descriptor-name
  - DESCRIBE PROCEDURE host-variable INTO descriptor-name
  - DESCRIBE TABLE host-variable INTO descriptor-name

Student Notes

Some of the SQL statements require an SQLDA. EXECUTE, OPEN, FETCH, and CALL with the USING DESCRIPTOR clause require the SQLDA variables as input and must be set up with the pertinent information before the SQL statement. DESCRIBE will return information in the SQLDA variables.

SQLDA Variables

- No "INCLUDE SQLDA"
- stem.SQLD -
  - Number of variable elements that the SQLDA actually contains.
- stem.n.SQLNAME
  - column name.
- stem.n.SQLTYPE
  - data type
- stem.n.SQLLEN
  - length of non decimal data.
- stem.n.SQLLEN.SQLPRECISION
  - precision of a decimal number.
- stem.n.SQLLEN.SQLSCALE
  - scale of a decimal number.

Student Notes

The convention used is that you provide the REXX variable stem name as the program variable name in the SQL statement. "stem".SQLD provide the number of variable elements that the SQLDA actually contains. Each of the elements of the SQLDA are prefixed with "stem.n.". For example, if the stem name was specified as ":MYSQLDA" in the SQL statement, the first SQLNAME variable would be MYSQLDA.1.SQLNAME.

Note that SQLLEN is further qualified by SQLPRECISION and SQLSCALE if the value in SQLTYPE specifies a decimal data type.

SQLDA Variables (cont.)

- stem.n.SQLCCSID
  - CCSID of the data.
- stem.n.SQLLOCATOR
  - result set locator value for a DESCRIBE PROCEDURE.
- stem.n.SQLDATA
  - input or output value.
  - value is converted to the attributes specified in SQLTYPE, SQLLEN, SQLLEN.SQLPRECISION, and SQLLEN.SQLSCALE.
- stem.n.SQLIND
  - negative number indicates output data value is null.

Student Notes

Instead of supplying a pointer to the data or indicator variable in SQLDATA and SQLIND like you would in other programming languages, you supply the actual value of the data or indicator value. Isn't REXX great?

REXX Stored Procedures

- CREATE PROCEDURE ... LANGUAGE REXX
- One output parameter only
- REXX Called as "Subroutine"
  - Parameters separated by commas
  - All parameters passed as external text strings
- Runs in a TSO environment
  - Can use ALLOCATE, FREE, EXECIO, etc.
- WLM Managed Only
- Use any REXX/DB2 Interface that uses RRS Attach

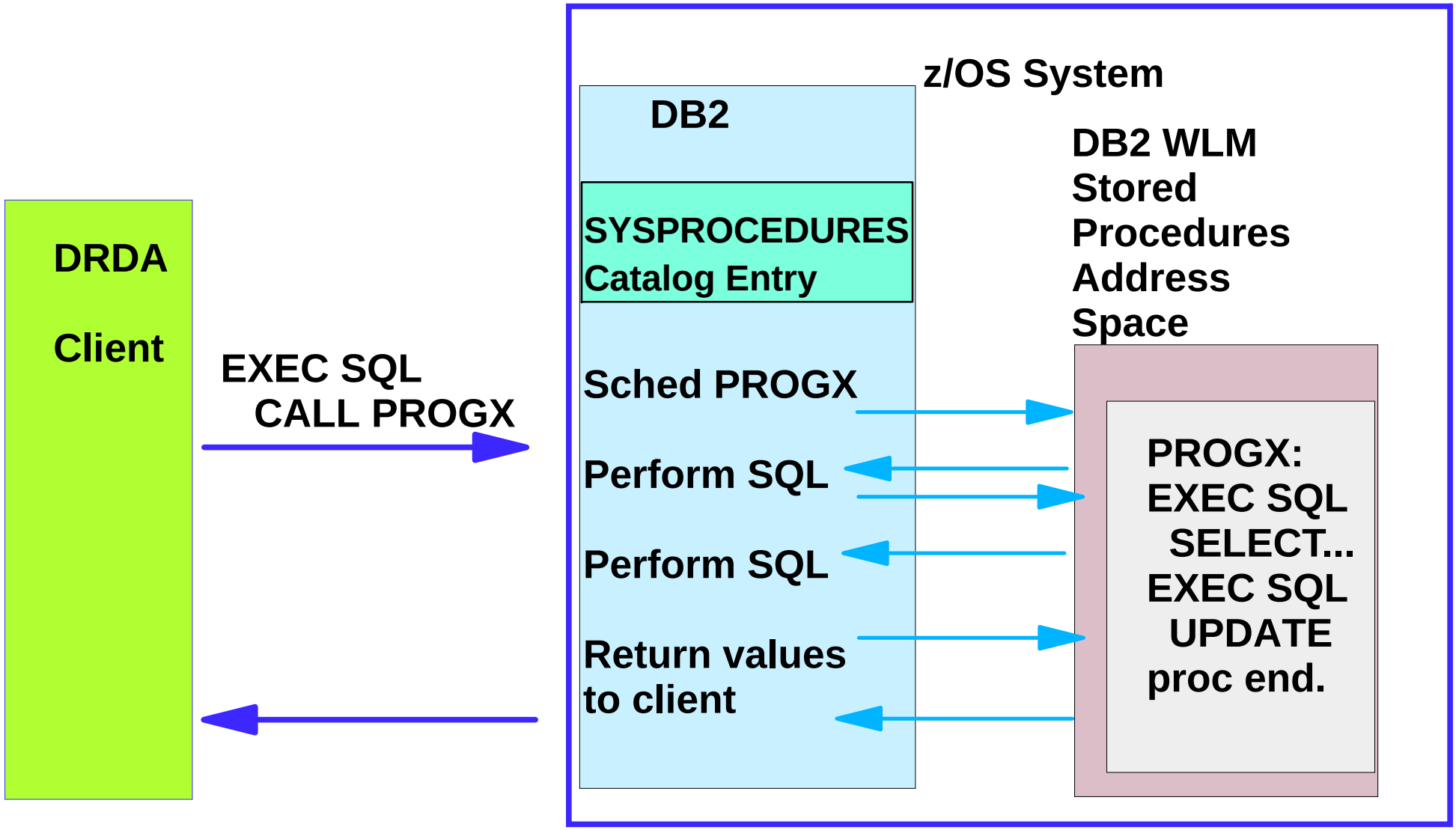
Student Notes

Let's move on to the stored procedure REXX language support. Because REXX programs don't have typical parameter lists, we can only support a single output variable - the value specified on the REXX "EXIT" statement. When the REXX program is invoked as a stored procedure, the calling parameters are passed separated by commas just as when you call a REXX subroutine.

REXX stored procedures are only supported in WLM managed stored procedure address spaces - this is where all of our stored procedure enhancements are coming so it is time to these enabled if you haven't already.

If you already have a REXX to DB2 product that you like, you can use it as well as long as it uses RRS attach.

Stored Procedures on z/OS



Fewer network send/receive operations
SQL has same CPU pathlength as local DB2 SQL
Locks held for shorter duration

Student Notes

By using a stored procedure the client application issues a single network send/receive operation to run the stored procedure. The stored procedure can then issue multiple SQL statements.

The number of network send/receive operations is reduced, which improves the elapsed time and CPU time consumed by the application.

The SQL issued by the stored procedure uses a local DB2 interface (Call Attach or RRS Attach), so there is no added "distributed" overhead on the SQL statements issued by the stored procedure.

For applications that issue several SQL statements, the savings can be significant. The CPU and elapsed time reductions become more dramatic as the number of SQL statements increases.

WLM for REXX Stored Procedures

```
  Application-Environment  Notes  Options  Help
------------------------------------------------------------------------

IWMAP6J              Modify an Application Environment

Command ===> ____________________________________________________________

Application Environment Name . : DSNREXX

Description  . . . . . . . . . . WLM Environment for REXX
Subsystem Type . . . . . . . . . DB2

Procedure Name . . . . . . . . . DSNREXX

Start Parameters . . . . . . . . DB2SSN=DSN,NUMTCB=1,APPLENV=DSNREXX
                                 ____________________________________________

                                 ____________________________________________


Limit on starting server address spaces for a subsystem instance:
1   1.  No limit

    2.  Single address space per system

    3.  Single address space per sysplex
```

Student Notes

This is one way you might specify the WLM environment definition for REXX language stored procedures.

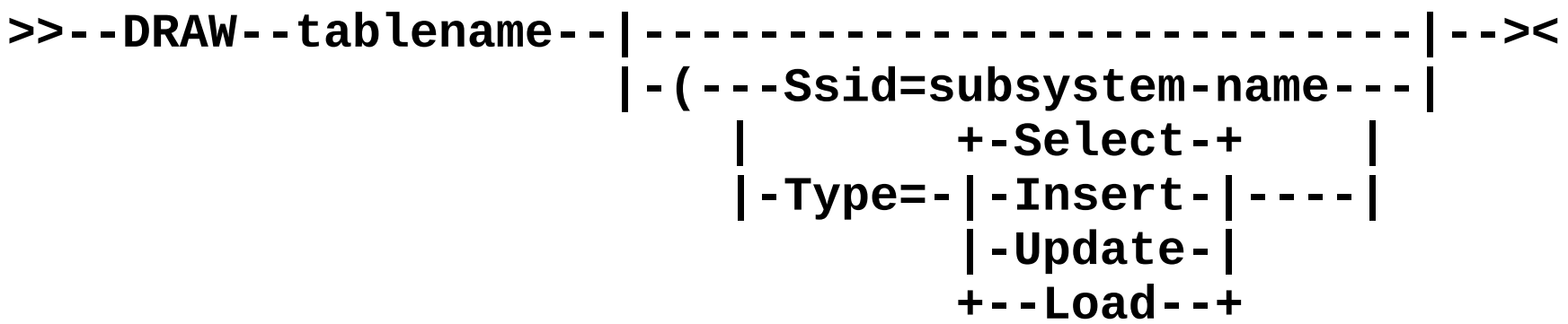
JCL for REXX Stored Procedures

```
//*****************************************************************
//*       THIS PROC IS USED TO START THE WLM-ESTABLISHED SPAS      *
//*       ADDRESS SPACE FOR THE WLMENV1 APPLICATION ENVIRONMENT.   *
//* D WLM,APPLENV=DSNREXX                                          *
//* VARY WLM,APPLENV=DSNREXX,REFRESH                               *
//* VARY WLM,APPLENV=DSNREXX,QUIESCE                               *
//* VARY WLM,APPLENV=DSNREXX,RESUME                                *
//*****************************************************************
//DSNREXX  PROC  DB2SSN=DSN,NUMTCB=1,APPLENV=DSNREXX
//DSNREXX  EXEC  PGM=DSNX9WLM,TIME=1440,
//            PARM='&DB2SSN,&NUMTCB,&APPLENV',
//            REGION=0M
//STEPLIB  DD    DSN=DSN!!0.SDSNLOAD,DISP=SHR
//         DD    DSN=CEE.SCEERUN,DISP=SHR
//SYSEXEC  DD    DISP=SHR,DSN=DSN!!0.SDSNCLIST  <== REXX EXEC HOME
//CEEDUMP  DD    SYSOUT=A
//SYSABEND DD    SYSOUT=A
//SYSTSPRT DD    SYSOUT=A
//SYSPRINT DD    SYSOUT=A
```

Student Notes

This is a suggested JCL procedure for REXX language stored procedures.

Sample: "DRAW" SQL or LOAD statements



- Ssid - subsystem-name specified the name of a DB2 subsystem.
- Select - Composes a basic query for selecting data from the columns of a table or view.
- Insert - Composes a basic query to insert data into the columns of a table or view.
- Update - Composes a basic query to change the data in a table or view.
- Load - Composes a load statement to load the data in a table.

Student Notes

How many times have you wished you could "draw" an SQL statement in SPUFI like you can in QMF? You would even settle for a simple version just to avoid running a SELECT * FROM SYSIBM.SYSCOLUMNS ... wouldn't you? How many times have you run DSNTIAUL just to get a LOAD statement? This little sample REXX program will generate SELECT, INSERT, UPDATE, and LOAD statements.

"DRAW Pseudocode

```
:
Address DSNREXX "CONNECT" SSID
If (SQLcode \= 0) Then . . .
Address DSNREXX "EXECSQL DESCRIBE TABLE :TABLE
INTO :SQLDA"
If (SQLcode \= 0) Then . . .
Address DSNREXX "DISCONNECT"
If (SQLcode \= 0) Then . . .
Select
  When (Left(Type,1) = "S") Then Call DrawSelect
  When (Left(Type,1) = "I") Then Call DrawInsert
  When (Left(Type,1) = "U") Then Call DrawUpdate
  When (Left(Type,1) = "L") Then Call DrawLoad
  Otherwise EXIT (20)
End
:
```

Student Notes

The DRAW program is really simple, DESCRIBE the specified table, and then, depending on the statement type, go generate the SQL or LOAD statement from the SQLDA information. I bet you are already thinking of some other tools that you would be able to write very quickly in REXX.

"DRAW" SELECT Example

- DRAW DSN8810.EMP

```
SELECT "EMPNO", "FIRSTNME", "MIDINIT",
       "LASTNAME", "WORKDEPT",
       "PHONENO", "HIREDATE", "JOB",
       "EDLEVEL", "SEX", "BIRTHDATE",
       "SALARY", "BONUS", "COMM"
FROM DSN8810.EMP
```

Student Notes

Here is an example of drawing a SELECT of the EMP sample table.

"DRAW" LOAD Example

- DRAW DSN8810.EMP (S=DSNA T=LOAD

```
LOAD DATA INDDN SYSREC INTO TABLE DSN8810.EMP
 ( "EMPNO"     POSITION(   1) CHAR(6)
 , "FIRSTNME" POSITION(   8) VARCHAR
 , "MIDINIT"   POSITION(  21) CHAR(1)
 , "LASTNAME" POSITION(  23) VARCHAR
 , "WORKDEPT" POSITION(  39) CHAR(3)
                   NULLIF(   39)='?'
:
 , "COMM"      POSITION(  96) DECIMAL EXTERNAL(9,2)
                   NULLIF(   96)='?' )
```

Student Notes

Here is a (cut down) example of drawing a LOAD utility statement for a specific DB2 system. (It is cut down because it would fit on the foil without making the font unreadble).

Stored procedure example

```
/* REXX */
Parse Upper Arg CMD
If Left(CMD,2) = """ & Right(CMD,2) = """ Then
  CMD = Substr(CMD,2,Length(CMD)-2)
COMMAND  = Substr("COMMAND",1,18," ")
IFCA     = Substr(D2C(LENGTH(IFCA),2)||'0000'X||'IFCA',1,180,'00'X)
RTRNAREASIZE = 262144
RTRNAREA = D2C(RTRNAREASIZE+4,4)Left(' ',RTRNAREASIZE,' ')
OUTPUT   = D2C(Length(CMD)+4,2)||'0000'X||CMD

Address LINKPGM "DSNWLIR COMMAND IFCA RTRNAREA OUTPUT"

RTRN     = Substr(IFCA,12+1,4);REAS = Substr(IFCA,16+1,4)
TOTLEN   = C2D(Substr(IFCA,20+1,4))
"SUBCOM DSNREXX"            /* HOST CMD ENV AVAILABLE?*/
If RC Then                  /* ADD HOST CMD ENV*/
  Call RXSUBCOM('ADD','DSNREXX','DSNREXX')
Address DSNREXX
```

Student Notes

Over the next couple of foils is a example of a stored procedure which issues a DB2 command through the instrumentation interface, puts the results in a global temporary table and returns the contents as a result set. In this foil we see the set up of the parameters and call to DSNWLIR - the instrumentation API for RRS Attach.

Stored procedure example (cont.)

```
SQLSTMT='INSERT INTO SYSIBM.SYSPRINT(SEQNO,TEXT)
VALUES(?,?)'
EXECSQL "DECLARE C1 CURSOR FOR S1"
If SQLCODE <> 0 Then Call SQLCA
EXECSQL "PREPARE S1 FROM :SQLSTMT"
If SQLCODE <> 0 Then Call SQLCA
OFFSET = 4+1
Do Seqno = 1 By 1 While ( OFFSET <  TOTLEN )
  LEN = C2D(Substr(RTRNAREA,OFFSET,2))
  TEXT = Substr(RTRNAREA,OFFSET+4,LEN-4-1)
  EXECSQL "EXECUTE S1 USING :SEQNO,:TEXT"
  If SQLCODE <> 0 Then Call SQLCA
  OFFSET = OFFSET + LEN
End
```

Student Notes

In this foil, we are extracting each command output line from the IFI return area buffer and inserting the line into the global temporary table.

Stored procedure example (cont.)

```
SQLSTMT=,
 'SELECT SEQNO,TEXT FROM SYSIBM.SYSPRINT ORDER BY SEQNO'
EXECSQL "DECLARE C2 CURSOR FOR S2"
If SQLCODE <> 0 Then Call SQLCA
EXECSQL "PREPARE S2 FROM :SQLSTMT"
If SQLCODE <> 0 Then Call SQLCA
EXECSQL "OPEN C2"
If SQLCODE <> 0 Then Call SQLCA
Call RXSUBCOM('DELETE','DSNREXX','DSNTREXX') /* REMOVE CMD ENV*/
Exit "RTRN="RTRN "REAS="REAS
SQLCA: Exit 'SQLERRM ='SQLERRMC';' 'SQLERRP ='SQLERRP';' ,
      || 'SQLERRD='SQLERRD.1','SQLERRD.2','SQLERRD.3',',
      || 'SQLERRD.4','SQLERRD.5','SQLERRD.6';',
      || 'SQLWARN ='SQLWARN.0 ||SQLWARN.1||SQLWARN.2,
               ||SQLWARN.3||SQLWARN.4,||SQLWARN.5,
               ||SQLWARN.6||SQLWARN.7||SQLWARN.8,
               ||SQLWARN.9||SQLWARN.10';',
      || 'SQLSTATE='SQLSTATE';'
```

Student Notes

Finally, in this foil we open a cursor for the result set. We also see the code for returning SQLCA information if an error had occurred.

Sample REXX Calling Program

```
/* REXX */
Parse Arg SSID COMMAND
Address TSO "SUBCOM DSNREXX"
If Rc Then Call RXSUBCOM('ADD','DSNREXX','DSNREXX')
Address DSNREXX
CONNECT SSID
If SQLCODE <> 0 Then Call SQLCA
EXECSQL "CALL COMMAND (:COMMAND, :RESULT)"
If SQLCODE < 0 Then CALL SQLCA
EXECSQL "ASSOCIATE LOCATOR (:RESULT) WITH PROCEDURE :PROC"
If SQLCODE <> 0 Then Call SQLCA
EXECSQL "ALLOCATE C101 CURSOR FOR RESULT SET :RESULT"
If SQLCODE <> 0 Then Call SQLCA
Do Until(SQLCODE <> 0)
  EXECSQL "FETCH C101 INTO :SEQNO, :TEXT"
  If SQLCODE = 0 Then Say TEXT
End
If SQLCODE <> 0 & SQLCODE <> 100 Then Call SQL
```

Student Notes

Now that we have a stored procedure, let's show a REXX example of how to call it and do something with the result set. We call the Command stored procedure we just covered, associate a locator with the stored procedure result set, allocate a cursor for the result set, fetch the results, and display them.

Sample REXX Calling Program (cont.)

```
EXECSQL "CLOSE C101"
If SQLCODE <> 0 Then Call SQLCA
EXECSQL "COMMIT"
If SQLCODE <> 0 Then Call SQLCA
DISCONNECT
If SQLCODE <> 0 Then Call SQLCA
Call RXSUBCOM('DELETE','DSNREXX','DSNREXX')
Return
SQLCA: Exit 'SQLERRM ='SQLERRMC';' 'SQLERRP ='SQLERRP';' ,
     || 'SQLERRD='SQLERRD.1','SQLERRD.2','SQLERRD.3',',
     || 'SQLERRD.4','SQLERRD.5','SQLERRD.6';',
     || 'SQLWARN ='SQLWARN.0 ||SQLWARN.1||SQLWARN.2,
              ||SQLWARN.3||SQLWARN.4,||SQLWARN.5,
              ||SQLWARN.6||SQLWARN.7||SQLWARN.8,
              ||SQLWARN.9||SQLWARN.10';',
     || 'SQLSTATE='SQLSTATE';'
```

Student Notes

Here we have the clean up of the cursor and code to handle bad return codes.
An alternative is to use DSNTIAR to format the message

```
 if right(sqlerrmc,1) <> 'ff'x then
   sqlerrmc = sqlerrmc || 'ff'x
 /* rebuild a 'real' sqlca */
 message_llen = 79
 message_lines = 10
 realsqlca = 'SQLCA   '
 upper realsqlca
 realsqlca = realsqlca                          ,
         ||x2c(d2x(136,8))                     ,
         ||x2c(d2x(sqlcode,8))                   ,
         ||x2c(d2x(length(sqlerrmc),4))          ,
         ||left(sqlerrmc,70)                   ,
         ||left(sqlerrp,8)                    ,
         ||x2c(d2x(sqlerrd.1,8))||x2c(d2x(sqlerrd.2,8)) ,
         ||x2c(d2x(sqlerrd.3,8))||x2c(d2x(sqlerrd.4,8)) ,
         ||x2c(d2x(sqlerrd.5,8))||x2c(d2x(sqlerrd.6,8)) ,
         ||left(sqlwarn.0,1)||left(sqlwarn.1,1)    ,
         ||left(sqlwarn.2,1)||left(sqlwarn.3,1)    ,
         ||left(sqlwarn.4,1)||left(sqlwarn.5,1)    ,
         ||left(sqlwarn.6,1)||left(sqlwarn.7,1)    ,
         ||left(sqlwarn.8,1)||left(sqlwarn.9,1)    ,
         ||left(sqlwarn.a,1)                   ,
         ||left(sqlstate,5)
 message_area = x2c(d2x(message_llen*message_lines,4))||,
             copies(' ',message_llen*message_lines)
 message_lrecl = x2c(d2x(message_llen,8))
 address linkpgm "DSNTIAR realsqlca message_area message_lrecl"
 message_line_start = 3
 do messagei = 1 to message_lines
   message_line = substr(message_area,message_line_start,,
                         message_llen)
   message_line_start = message_line_start + message_llen
   if message_line <> '' then
     say message_line
 end
```

Making DB2 Load Modules Available to TSO/Batch Users

2.6.9.1 Making DB2 Load Modules Available to TSO and Batch Users

If you included prefix.SDSNEXIT and prefix.SDSNLOAD in your LNKLSTxx, you can skip this step.

If you have not included prefix.SDSNEXIT and prefix.SDSNLOAD in your LNKLSTxx, you must add STEPLIB statements to your logon procedures and JCL for jobs to ensure that you access the DB2 Version 5 load modules.

If prefix.SDSNEXIT is not in your LINKxx, then add it to your STEPLIB and JOBLIB concatenations before prefix.SDSNLOAD. Refer to "Choosing Link List options" in topic 2.4.3.2.2 for information on link lists.

Student Notes

In case your system programmer missed this section of the install manual - your REXX execs will have problems getting to DB2 if this wasn't done

Summary

- Simple, easy to use language to write system programmer or DBA tools
- Useful for rapid prototyping of
  - simple applications
  - stored procedures
- Useful for stored procedures
  - Need customer tailoring
  - Need to do file allocations
- Stored procedure enablement open to any REXX to DB2 interface vendor
  - Just need to use RRS Attach

Isn't this GREAT???

References

- The NetRexx Language
  - Mike Cowlishaw, ISBN 0-13-806332-X, 197pp, Prentice Hall
- REXX Langauge
  - http://www2.hursley.ibm.com/rexx/
- DB2 for z/OS
  - http://www.ibm.com/software/db2os390
- DB2 for z/OS and OS/390 : Squeezing the Most Out of Dynamic SQL
  - SG24-6418

Student Notes

To learn more about REXX and the IBM DB2 REXX Language Support, check out the information in these publications