Session: 17829

DB2 and the Joy of Rexx

Ron Brown
Ronek Consulting

Wednesday 7th October 2009
17.00 – 18.00

Platform: DB2 for z/OS and LUW

IDUG
The Worldwide DB2 User Community

Abstract:

This presentation aims to cover the use of Rexx and OO-Rexx languages to access DB2 on z/OS and LUW platforms. It will give a brief description of the languages, their standard and non-standard interfaces to DB2, explain some advanced methods of coding for DB2 access, suggest solutions to some common problems, and provide a set of useful code fragment samples.

Speaker Biography:

For over 20 years Ron Brown has worked with DB2 in many different large companies in roles ranging from systems programming to database administration, and he has been writing Rexx programs on VM, MVS and Windows. Hence he has written many DB2-Rexx programs, many of which are contained in technical articles in Xephon magazines and IT websites.

DB2 and the Joy of Rexx

- Agenda
  - What is Rexx?
  - Rexx – DB2 interfaces
  - Advanced SQL invocation/debugging on z/OS
  - SQL invocation on LUW
  - Sample Rexx-DB2 code

1. A brief history of Rexx and OO-Rexx languages on VM, z/OS and LUW, including the differences from other languages and the advantages of using Rexx.

2. Standard interfaces from Rexx to DB2 for SQL, commands, IFI, utilities and Stored Procedures on z/OS and LUW.

3. Advanced invocation and debugging of SQL on z/OS, including third-party interfaces, DSNREXX considerations and invocation of DSNTIAR.

4. Invocation and debugging of SQL on LUW, including SQLEXEC considerations and code for transporting cross platforms.

5. Samples of Rexx-DB2 code including: Unicode considerations, RUNSQL, DBRM interpretation, accessing remote DB2 objects, DB2 message processing, SPUFI invocation and DB2 command

DB2 and the Joy of Rexx

- History of Rexx
  - 1979 – 1982: creator Mike Cowlishaw on VM
  - 1988: Ported to TSO & all other IBM platforms
  - 1996: ANSI standard TRL-2 for “classic” Rexx
  - 1997: Object Rexx for Windows & OS/2, then ported to AIX, Linux & SUN Solaris
  - 2004: Open Object Rexx replaced Object Rexx

1. Mike Colishaw created Rexx from 20th March 1979 to mid 1982 as a replacement for EXEC and EXEC2 on VM.

2. Rexx was ported to all other IBM platforms, available with TSO/E Version 2 in late 1988 and early 1989 as an alternative to CLIST when MVS/ESA was replacing MVS/XA.

3. ANSI standard, called “TRL-2”, was agreed in 1996 for all the various forms of “procedural-” or “classic-” Rexx.

4. IBM released Object Rexx for Windows and OS/2 in 1997, then it was ported to AIX, Linux & SUN Solaris.

5. Object Rexx was withdrawn by IBM in October 2004, and it became Open Object Rexx on a Common Public License, with some of the original Object Rexx authors still developing it. The Rexx Language Association is now maintaining Object Rexx and NetRexx; they have a website: *http://www.rexxla.org/rexxla/about.html*

6. Also see *http://www.rexxlinfo.org* formore information about and links to download 9 more Rexx interpreter/compilers which run on various non-IBM operating systems.

DB2 and the Joy of Rexx

- Features of Rexx
  - English-like language with few format rules
  - Rich in Built-in Functions and Methods
  - Typeless variables and compound variables
  - Powerful string handling and decimal arithmetic
  - Clear error messages & powerful debugging
  - Interpreted or compiled
  - Portability

• Rexx is good for Rapid Application Development because it is very easy to learn and use due to its English-like language with instructions like: DO … END, INTERPRET, SAY, PARSE, IF …. THEN … ELSE, and many with language constructs derived from PL/I and PASCAL. It has fewer formatting rules than most languages and is case-insensitive.

•An extensive set of built-in functions (and methods for Object Rexx) are provided and the use can easily write their own additional ones (in various languages).

-All Rexx variables are stored as variable-length character strings. Compound (or “stem”) variables provide arrays and/or associative arrays. Variables can be scoped local or global and can be passed to other programs or put into a Rexx “stack”.

-Strings can be manipulated by various built-in functions and by the PARSE statement. OO-Rexx can also use Regular Expressions like other languages such as perl, grep, awk and sed.

-Arithmetic calculations are done in decimal.

-TRACE provides easy debugging

-On MVS it is usually interpreted, because the Rexx Compiler is an extra-cost option. On LUW it is usually compiled.

-Rexx code can run on any platform, though any calls it makes to platform-specific products (eg. TSO or Windows commands) can only work on their own platform. OO-Rexx using object methods can be easily portable (as can any object-oriented language).

DB2 and the Joy of Rexx

- DB2 Interfaces
  - Commands
  - IFI
  - Utilities
  - File access
  - Storage access
  - SQL
  - Stored Procedures

-On z/OS it is normal to use the DSN program to run commands from Rexx programs, which are run under TSO.

-IFI can be accessed from Rexx like other languages. It can be used to issue DB2 commands, obtain monitor trace records, or even write trace data.

-DB2 utilities can be invoked via the DSNUTILS Stored Procedure from Rexx on z/OS

-Rexx can be used to read files from DB2 (eg. DB2 message logs, DBRM libraries) for various purposes

-Some DB2 control blocks can be read directly from storage (be cautious about doing this!)

-Dynamic SQL is supported (DCL, DDL & DML), but not static SQL.

-DSNREXX and SQLEXEC support the calling of Stored Procedures via the "CALL" statement. Stored procedures can be written in Rexx.

We will show some examples of these interfaces being used, but mostly concentrate on SQL (particularly on z/OS).

DB2 and the Joy of Rexx

- DB2 Commands
  - **On LUW, call the Administrative APIs or the DB2 Command Line Processor or ADMIN_CMD procedure**
    - Call SQLDBS 'CATALOG GLOBAL DATABASE /.../cell1/subsys/database/DB3 AS dbtest USING DIRECTORY DCE WITH "Sample Database"'
  - **On z/OS (TSO or batch) call DSN program**

```
Queue command_text
Queue “END”
Queue “”
x = Outtrap(outline.)
Address TSO "DSN SYSTEM("ssid")"
x = Outtrap("OFF")
/* say "The number of lines trapped is" outline.0 */
Do i = 1 to outline.0
   Say outline.i
   End
```


LUW

The SQLDB2 interface supports calling APIs from REXX. The SQLDB2 interface was created to provide support in REXX for new or previously unsupported APIs that do not have any output other than the SQLCA. Invoking a command through the SQLDB2 interface is syntactically the same as invoking the command through the command line processor (CLP), except that the token **call db2** is replaced by **CALL SQLDB2**.

Using the CALL SQLDB2 from REXX has the following advantages over calling the CLP directly:

- The compound REXX variable SQLCA is set
- By default, all CLP output messages are turned off.

ADMIN_CMD procedure is shown in a later slide for invoking Utilities

DB2 and the Joy of Rexx

- DB2 Commands
  - **In z/OS Rexx Stored Procedure**

```
Queue "DSNE"
Queue command_text
Queue "END"
Queue ""
x = Outtrap(outline.)
Address ATTCHMVS "DSNESM71"
x = Outtrap("OFF")
Do i = 1 to outline.0
   Say outline.i
   End
```

  - **On z/OS call IFI program DSNWLI2**

Similarly to **DSN** in TSO, you can use **DSNESM71** to run commands in Stored Procedures. However, note that the use of program DSNESM71 is **not documented** in the DB2 manuals and therefore its use is not fully supported by IBM. For an example of Stored Procedure code using DSNESM71, see the standard DB2 system exec library member called:
db2prefix.SDSNCLIST(DSNADMCS)

IFI program **DSNWLI2** can be used from Rexx in TSO, batch or Stored Procedures.

DB2 and the Joy of Rexx

- **IFI**
- **Issuing DB2 commands**

```
IFCA = '00B4'X||'  IFCA    '||COPIES('00'X,24)'    ' ||'0000'X||COPIES('
',34)||'0000'X,
          ||COPIES(' ',18)||'0000'X||COPIES(' ',82)
CMD = "-STA TRACE(MON) CLASS(1)"
COMMAND = SUBSTR("COMMAND",1,18)
RTRNAREASIZE = 512
RTRNAREA = D2C(RTRNAREASIZE+4,4)LEFT(' ',RTRNAREASIZE,' ')
OUTPUT = D2C(LENGTH(CMD)+4,2)||'0000'X||CMD
BUFFER = '00000000'X||'WBUF'||'0000000000000000'X
ADDRESS LINKPGM "DSNWLI2 COMMAND IFCA RTRNAREA OUTPUT
BUFFER"
RETCD = C2D(SUBSTR(IFCA,13,4))
REASCD = D2X(C2D(SUBSTR(IFCA,17,4)))
```


Program DSNWLI2 can be used to invoke DB2 commands and like the example shown.

Reference: DB2 version 8.1 Administration Guide, Appendix E: Programming for the Instrumentation Facility Interface

DB2 version 9.1 Performance Monitoring & Tuning Guide, Appendix B: Programming for the Instrumentation Facility Interface

DB2 and the Joy of Rexx

- **IFI**
- **Reading DB2 Trace records**
  - /* Issue READS for IFCID225 */
  - TOTLEN = C2D(SUBSTR(IFCA,20+1,4))
  - READS = SUBSTR('READS',1,8)
  - RTRNAREA = '00001004'X||COPIES(' ',4096)
  - IFCID = '0006'X||' '||'0081'X
  - WQUAL = '00A8'X||'
    WQAL'||COPIES('00'X,88)'002B'X||COPIES('00',42)'CIR'||'0001'X||'N ‘,
  - ||COPIES('00'X,8)' '||'0001'X||' '
  - IFCIDAREA = '0006000000E1'X
  - ADDRESS LINKPGM "DSNWLI2 READS IFCA RTRNAREA IFCIDAREA"
  - RETCD = C2D(SUBSTR(IFCA,13,4))
  - REASCD = D2X(C2D(SUBSTR(IFCA,17,4)))

Program DSNWLI2 can be used to **read** (synchronously or asynchronously) or **write** trace records, like the example shown.

DB2 and the Joy of Rexx

- DB2 Utilities
- **On LUW**
  - calling the Administrative APIs
    - Call SQLDBS “EXPORT :stmt TO datafile OF filetype MESSAGES msgfile
  - calling the ADMIN_CMD stored procedure
    - Call SQLEXEC “CALL SYSPROC.ADMIN_CMD(‘RUNSTATS ON TABLE RON.MYTABLE WITH DISTRIBUTION AND INDEXES ALL’)

DB2 for LUW has utilities, like EXPORT, RUNSTATS or RECOVER which can be easily invoked from an OO-Rexx exec by calling the appropriate Administrative API.

Beginning with DB2 version 8, fix pack 9, you can use the ADMIN_CMD stored procedure too.

DB2 and the Joy of Rexx

- DB2 Utilities
- **On z/OS**
  - stand-alone utilities or DSNTIAUL can be directly invoked
  - DSNUTILB cannot be invoked from TSO (batch or online)
  - DSNUTILS Stored Procedure can be used to invoke utilities

However, it is not so easy for DB2 on z/OS.

a) Stand-alone utilities can be invoked directly from Rexx (eg DSNJU003 Print Log Map), by allocationg input and output files then calling the program. DSNTIAUL sample unload program can also be invoked.

b) All the online utilities, which use program DSNUTILB, cannot be invoked in TSO and therefore not in Rexx. DSNUTILB uses storage key 7 and TSO uses storage key 8.

c) DSNUTILS is a Stored Procedure which can be called via DSNREXX / SQLEXEC to invoke online utilities

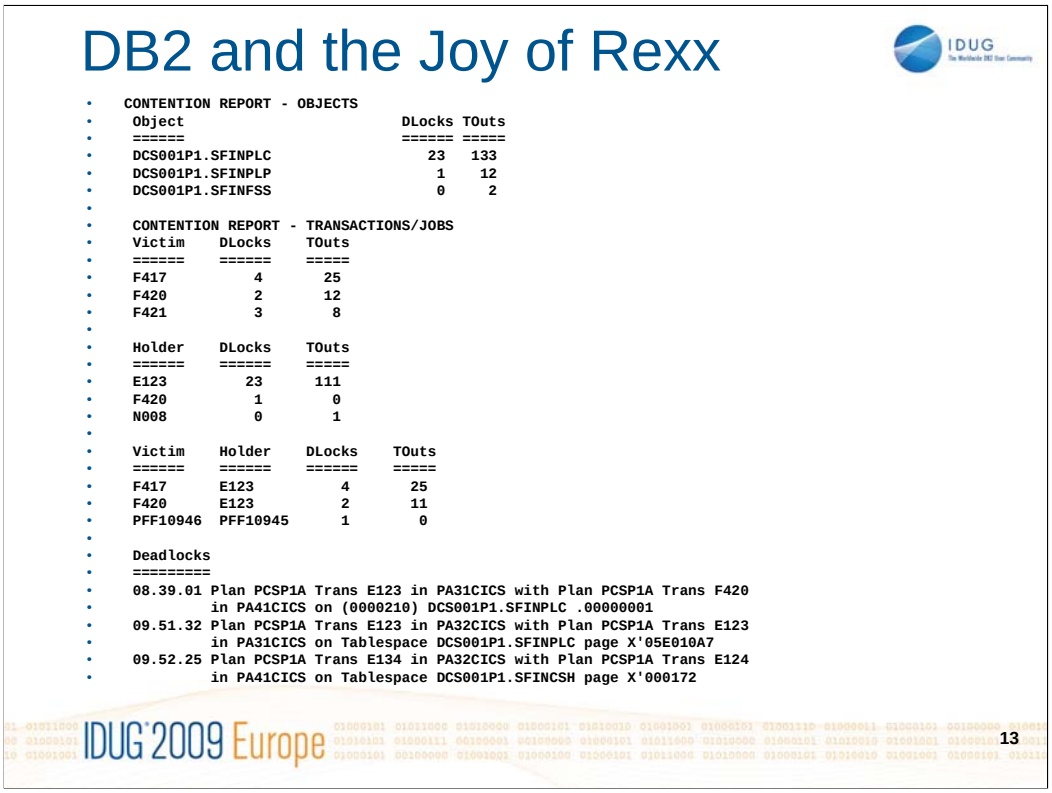
DB2 and the Joy of Rexx

- File Access
- Rexx can be used to read many different types of files which hold DB2 data. Some examples:
  - a) DB2 journal files
  - b) DB2MSTR message output
  - c) DBRMLIB library members
- Some sample output is on the next slide from a program in DB2 Update, November 2004

There are many possible uses of reading DB2 related files, but we will show just one example.

On the next slide is some output from a Rexx program which is reading the DB2MSTR address space collecting DEADLOCK and TIMEOUT messages to collate and summarise:

DB2 and the Joy of Rexx

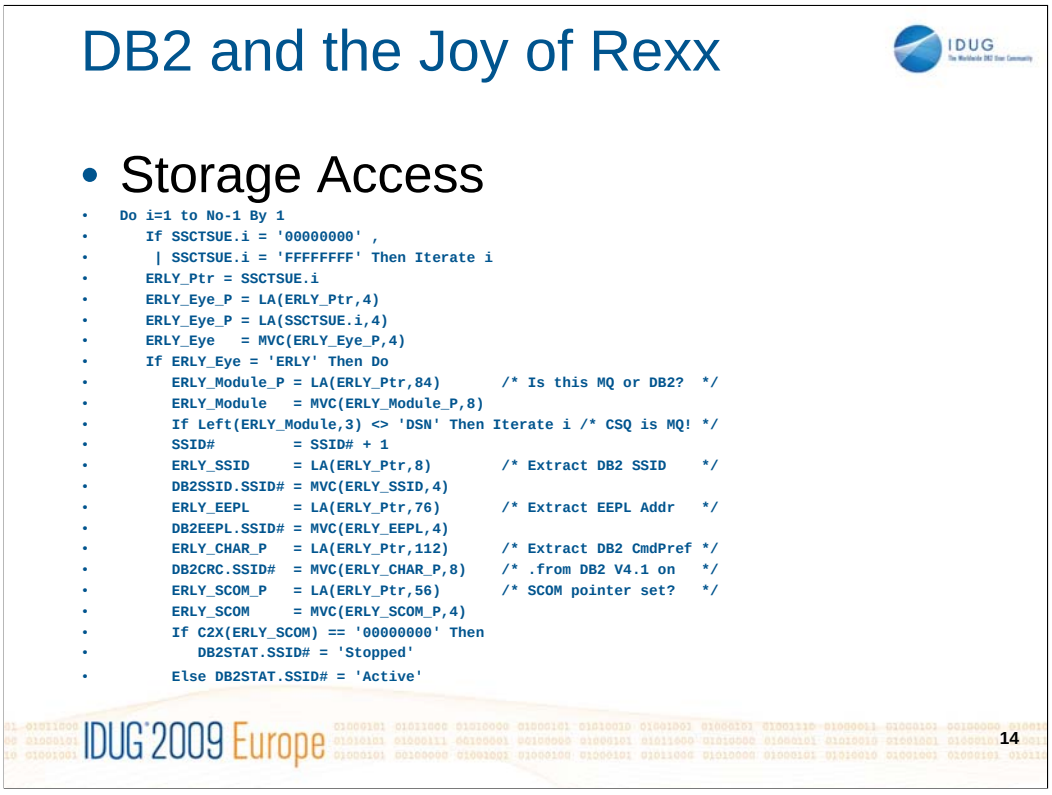


```
CONTENTION REPORT - OBJECTS
 Object                          DLocks TOuts
 ======                          ====== =====
 DCS001P1.SFINPLC                   23   133
 DCS001P1.SFINPLP                    1    12
 DCS001P1.SFINFSS                    0     2

 CONTENTION REPORT - TRANSACTIONS/JOBS
 Victim    DLocks    TOuts
 ======    ======    =====
 F417           4       25
 F420           2       12
 F421           3        8

 Holder    DLocks    TOuts
 ======    ======    =====
 E123          23      111
 F420           1        0
 N008           0        1

 Victim    Holder    DLocks    TOuts
 ======    ======    ======    =====
 F417      E123           4       25
 F420      E123           2       11
 PFF10946  PFF10945       1        0

 Deadlocks
 =========
 08.39.01 Plan PCSP1A Trans E123 in PA31CICS with Plan PCSP1A Trans F420
          in PA41CICS on (0000210) DCS001P1.SFINPLC .00000001
 09.51.32 Plan PCSP1A Trans E123 in PA32CICS with Plan PCSP1A Trans E123
          in PA31CICS on Tablespace DCS001P1.SFINPLC page X'05E010A7
 09.52.25 Plan PCSP1A Trans E134 in PA32CICS with Plan PCSP1A Trans E124
          in PA41CICS on Tablespace DCS001P1.SFINCSH page X'000172
```


This example shows a small part of a contention report,

This comes from Xephon **DB2 Update** magazine no. 145, November 2004, “Application Contention Report from DB2 for z/OS”.

It can be found at: *http://www.cbttape.org/xephon/xephond/db20411.pdf*



On z/OS it is feasible to use Rexx code to access some DB2 control block information by directly reading from storage, using Rexx built-in functions.

This program produced output like:

```
SSID   A/S Name      CmdPref  Status
----   --------      --------  -------
DB2M   DB2MMSTR    -DB2M    Active
DB1M  DB1MMSTR     -DB1M    Stopped
DB1T   DB1TMSTR     -DB1T    Active
```

The above example is extracting information from z/OS subsystem name table control block (SSCT) then extracting information from the related DB2 ERLY control blocks.

This code extract shows the use of user-written functions **LA** and **MVC** to follow pointers and get data from storage. The author wrote them to look similar to ASSEMBLER instructions, rather than just using the built-in Rexx functions **X2D**, **D2X** and **STORAGE**. That is not the simplest way to code the above, but it demonstrates what can be done in Rexx.

Here are those user-written functions:

```
/*-----------------------------------------------------------------*/
/*    ***  Simulate assembler LA instruction (LOAD ADDRESS) ***      */
/* Dese     : Copies LENGTH bytes of storage from ADDR+OFFSET to    */
```

DB2 and the Joy of Rexx

- SQL Interfaces
  - DSNREXX
  - SQLEXEC
  - Self-written interfaces
  - Third-party Rexx-DB2 products

-IBM provide "**DSNREXX**" as the standard interface for Rexx on z/OS. It was available as a separate offering, then it became an optional extra during DB2 V5.1 for OS/390; until May 2000, when it was finally included in the "RML" refreshed base code of DB2 V6.1 for OS/390. It uses similar syntax to preparing and executing dynamic SQL as in languages like PL/I, COBOL or C.

-

-"**CALL SQLEXEC**" is used on LUW and can also be used on z/OS to be platform independent, it is very similar to DSNREXX. It does format its SQLCA output differently from DSNREXX, (which is detailed in later slides).

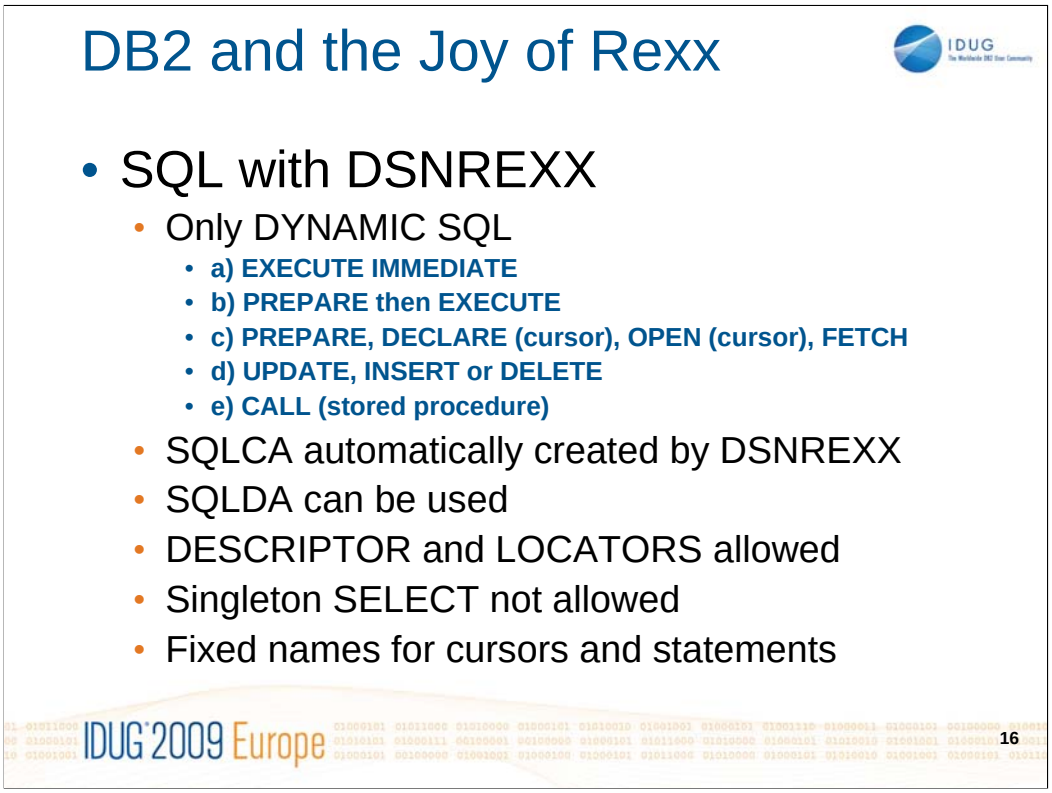
-There are a few self-written DB2-SQL interfaces in circulation, some of them write multi-row result sets into rexx compound (or "stem") variables

-A few Rexx-DB2 products are on sale, including:

-a) REXXTOOLS/MVS http://www.open-softech.com/rexmvs.htm

-b) RLX/SQL www.relarc.com

-c) CA MAX/REXX http://supportconnectw.ca.com/public/camax/infodocs/MAX_REXX_PB.pdf



-IBM's supplied interface for DB2 uses dynamic SQL very similar to PL/I, COBOL or C programs.

-SQLCA automatically created by DSNREXX, INCLUDE SQLCA statement is not valid in Rexx program.

-If SQLDA is specified in SQL, DSNREXX will create it automatically with the SQLDA name as the stem name of an array

-Use "EXECSQL EXECUTE IMMEDIATE sql_text" or "EXECSQL EXECUTE Snn USING ……"

-There is no singleton SELECT, a cursor is always required.

-DSNREXX has required names for statements ("Snn") and cursors ("Cnn")

> **-c1 to c100** Cursor names for DECLARE CURSOR, OPEN, CLOSE, and FETCH statements.
>
> -By default, c1 to c100 are defined with the WITH RETURN clause, and **c51 to c100** are defined with the WITH HOLD clause. You can use the ATTRIBUTES clause of the PREPARE statement to override these attributes or add additional attributes. For example, you might want to add attributes to make your cursor scrollable.
>
> **- c101 to c200** Cursor names for ALLOCATE, DESCRIBE, FETCH, and CLOSE statements that are used to retrieve result sets in a program that calls a stored procedure. These are only for z/OS.
>
> **-s1 to s100** Prepared statement names for DECLARE STATEMENT, PREPARE, DESCRIBE, and EXECUTE statements.

References:

ITSO - DB2 Server for OS/390 Version 5 Recent Enhancements – Reference

DB2 and the Joy of Rexx

- Stored Procedures
  - Rexx execs can call Stored Procedures
    - `PROCOUT = Left(' ',32000)      /* DSNWZP output */`
    - `Address DSNREXX "EXECSQL CALL SYSPROC.DSNWZP (:PROCOUT)"`
  - Rexx Stored Procedures on z/OS
    - JCL Procedure must have NUMTCB=1
    - //SYSEXEC DD DSN=rexx_library,DISP=SHR
    - Isolation depends on COLLID specified in CREATE PROCEDURE
    - CONNECT / DISCONNECT not required
    - Can use TSO commands
  - Rexx Stored Procedures not allowed on AIX

Rexx execs can call Stored Procedures.

z/OS Rexx stored procedures run in an address space under program DSNX9WLM, started under WLM control when required

NUMTCB=1 means that only one exec runs at a time.

The exec code is interpreted from the library concatenation in the SYSEXEC DD statement.

Isolation depends on COLLID: DSNREXRR (RR), DSNREXRS (RS), DSNREXCS (CS), DSNREXUR (UR)

If the exec is going to use DSNREXX it does not need to do CONNECT or DISCONNECT.

Execs can address the TSO environment the same as when running in a real TSO (online or batch).

Rexx execs on AIX can call Stored Procedures in any valid language (eg. C/C++, COBOL), but Rexx is not one of those valid languages.

Rexx Stored Procedures are allowed on other LUW platforms (eg. Windows).

DB2 and the Joy of Rexx

- Stored Procedures
- Handling a result set

```
ADDRESS DSNREXX  /* all EXECSQL statements to DSNREXX */
"EXECSQL CALL :PROC"
IF SQLCODE < 0 THEN SIGNAL SQL_ERROR
"EXECSQL ASSOCIATE LOCATOR (:LOC1) WITH PROCEDURE :PROC"
IF SQLCODE < 0 THEN SIGNAL SQL_ERROR
"EXECSQL ALLOCATE C110 CURSOR FOR RESULT SET :LOC1"
IF SQLCODE < 0 THEN SIGNAL SQL_ERROR
DO UNTIL(SQLCODE <> 0)
   "EXECSQL FETCH C110 INTO :PARM1, :PARM2, :PARM3"
   IF SQLCODE < 0 THEN SIGNAL SQL_ERROR
   . . . .
END
. . . .
"EXECSQL CLOSE C110“
. . . . .
```


Rexx execs on z/OS can call Stored Procedures, like in this example to get a result set.

SQLEXEC on LUW does not support ASSOCIATE LOCATOR.

DB2 and the Joy of Rexx

- SQL invocation on z/OS
  - DSNREXX by another name?
    - `Address TSO "SUBCOM DB2"`
    - `If rc Then`
    - `   rc = RXSUBCOM('ADD','DB2','DSNREXX')`
    - `Address DB2 "EXECSQL" sql_statement`
  - DSNREXX verses SQLEXEC
    - `Call SQLEXEC sql_statement`

      a) performance is the same

      b) SQLEXEC can be used unchanged on LUW

      c) different connecting to DB2

      d) SQLCA output structure is different

It is possible to initialise DSNREXX and call it something else if you want to. In the example shown "DB2" is used instead of "DSNREXX" as the name of the host command environment.

The performance using DSNREXX appears to be the same as when using SQLEXEC.

There are differences in the code for starting a thread with DB2 and the SQLCA variables are different, but only the SQLEXEC form will run on LUW.

DB2 and the Joy of Rexx

- Accessing remote DB2 objects
  - DSNREXX & SQLEXEC can use 3 part object names (eg. "*location.schema.table*")
  - Sometimes a WAIT is required, like:
    - Address DSNREXX "CONNECT DB2M"
    - Address TSO "Call *(WAITER) '500'" /* wait 500 ms */
    - sqltxt = "SELECT * FROM" loc".SYSIBM.LUNAMES"
    - Address DSNREXX "DECLARE C1 CURSOR FOR S1"
    - Address DSNREXX "PREPARE S1 INTO :SQLDA1 FROM :SQLTXT"
    - Address DSNREXX "OPEN C1"
    - Address DSNREXX "FETCH C1 INTO DESCRIPTOR :SQLDA1"

3 part names can also be used with SQL running on LUW.

You may not need to ever do a WAIT when accessing objects on a remote DB2, but if you experience problems – try this type of solution.

Note: this example is calling a non-standard external program called **WAITER** to do the wait. This was method found to be necessary in a Rexx application which usually worked, but occasionally failed (even though the CONNECT had run with SQLCODE = 0). After inserting this wait, it always worked correctly. A similar effect was achieved in another case by displaying an ISPF panel for the user to respond, after the CONNECT, but before the first SQL that was accessing a remote DB2.

DB2 and the Joy of Rexx

- SQL Errors
  - Return Code
    - `Call SQLEXEC sql_statement`
    - `Say rc`

      +1 Warning (SQLCODE > 0)
      0 OK (SQLCODE = 0)
      -1 Error (SQLCODE < 0)
  - SQL Code / SQL State
    - `Address DSNREXX "EXECSQL" sql_statement`
    - `Say 'SQLCODE =' sqlcode`
    - `Say 'SQLSTATE =' sqlstate`

After an SQL statement is executed the normal Rexx return code (rc) is set.

It is more common to check the SQL Code or SQL State.

DB2 and the Joy of Rexx

- SQL debugging on z/OS
  - **SQLCA variables returned by DSNREXX**

```
Say 'Sqlcode =' SQLCODE '  ',
                ' Sqlstate =' SQLSTATE '   Sqlerrp =' SQLERRP
Say 'Tokens =' Translate(SQLERRMC,',','FF'x)
Say 'Sqlerrd = 'SQLERRD.1','SQLERRD.2','SQLERRD.3',',
               ||SQLERRD.4','SQLERRD.5','SQLERRD.6
Say 'Sqlwarn = 'SQLWARN.0','SQLWARN.1','SQLWARN.2','SQLWARN.3',',
               ||SQLWARN.4','SQLWARN.5','SQLWARN.6','SQLWARN.7',',
               ||SQLWARN.8','SQLWARN.9','SQLWARN.10
```

  - Sqlcode = -904 Sqlstate = 57011 Sqlerrp = DSNLVCLM
  - Tokens = 00D31024,00001004,NABNETDB2D0001.P682861.DSNREXX
  - Sqlerrd = 9,0,0,-1,0,0
  - Sqlwarn = , , , , , , , , , ,

Displaying the SQLCA variables which are returned after every SQL statement processed by DSNREXX

Reference: DB2 version 8.1 for z/OS - SQL Reference, Appendix D SQL Communication Area (SQLCA), The REXX SQLCA

DB2 version 9.1 for z/OS - SQL Reference, Appendix E SQL Communication Area (SQLCA), The REXX SQLCA

DB2 and the Joy of Rexx

- SQL debugging on z/OS
  - **Direct DSNTIAR invocation**

```
/* Build normal SQLCA for DSNTIAR from Rexx variables */
   SQLCA = 'SQLCA   '||D2C(136,4),           /* 136 = length of SQLCA */
           ||D2C(SQLCODE,4),
           ||D2C(70,2)||Left(SQLERRMC,70),          /* message tokens */
           ||Left(SQLERRP,8),   /* (or set to 'DSN     ' to suppress) */
           ||D2C(SQLERRD.1,4)||D2C(SQLERRD.2,4)||D2C(SQLERRD.3,4),
           ||D2C(SQLERRD.4,4)||D2C(SQLERRD.5,4)||D2C(SQLERRD.6,4),
           ||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.10,1)||Left(SQLSTATE,5)
    msglen = 80          /* Message line length ( 72 to 240 allowed) */
    msglen12 = msglen * 12                /* up to 12 lines of msgs */
    sqlerrmsg = D2C(msglen12,2)||Copies(' ',msglen12)
    sqlerrlen = D2C(msglen,4)
   /* run DSNTIAR passing SQLCA to it & getting message text returned */
   Address LINKPGM "DSNTIAR SQLCA sqlerrmsg sqlerrlen"
    errmsg. = ''
    Parse Var SQLERRMSG 2 errmsg.1 81  82 errmsg.2 161 162 errmsg.3 241,
                        242 errmsg.4 321 322 errmsg.5 401 402 errmsg.6 481,
                        482 errmsg.7 561 562 errmsg.8 641 642 errmsg.9 721
    Do m = 1 to 9
       If errmsg.m <> '' Then Say errmsg.m
       End
```


The same **SQLCA** variables can be processed by **DSNTIAR** program to produce standardised output, including descriptive message text.

This sample code takes the standard variables from DSNREXX and reformats them into the SQLCA variable for DSNTIAR to process. The resulting output from DSNTIAR in the SQLERRMSG variable is then broken into multiple lines for displaying to the user. The next slide shows how it might look.

DB2 and the Joy of Rexx

- SQL debugging on z/OS
  - The direct DSNTIAR invocation might result in error messages like:

```
DSNT408I SQLCODE = -904, ERROR:  UNSUCCESSFUL EXECUTION CAUSED BY AN
         UNAVAILABLE RESOURCE. REASON 00D31024, TYPE OF RESOURCE 00001004,
AND
         RESOURCE NAME NABNETDB2D0001.P682861.DSNREXX
DSNT418I SQLSTATE   = 57011 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNLVCLM SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = 9  0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'00000009'  X'00000000'  X'00000000'  X'FFFFFFFF'
         X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION
```

  - compared to

```
Sqlcode = -904     Sqlstate = 57011    Sqlerrp = DSNLVCLM
Tokens = 00D31024,00001004,NABNETDB2D0001.P682861.DSNREXX
Sqlerrd = 9,0,0,-1,0,0
Sqlwarn =  , , , , , , , , , ,.
```


This comparison shows the difference between DSNTIAR messages and the SQLCA variables returned by DSNREXX or SQLEXEC

DB2 and the Joy of Rexx

- SQL and Abends
  - Code an explicit ROLLBACK for abends if SQL is doing UPDATE, INSERT, CREATE etc.
  - To protect against errors:

```
signal on syntax name error          /* turn on syntax trap       */
signal on halt name error            /* trap this only on LUW     */
. . . .
ERROR:
rexx = sysvar(sysicmd)
condition = condition('C')
If condition = 'HALT' Then Call SQLDBS "INTERRUPT"  /* only for LUW */
source = strip(sourceline(sigl),"B")
Call SQLEXEC "ROLLBACK"   /* make sure we roll back */
say copies('*',79)
say left('* 'condition' CONDITION ON LINE 'sigl' OF REXX 'rexx,78)'*'
say left('* 'source,78)'*'
say left('* RETURN CODE 'rc,78)'*'
say left('* 'errortext(rc),78)'*'
say copies('*',79)
exit 100
```

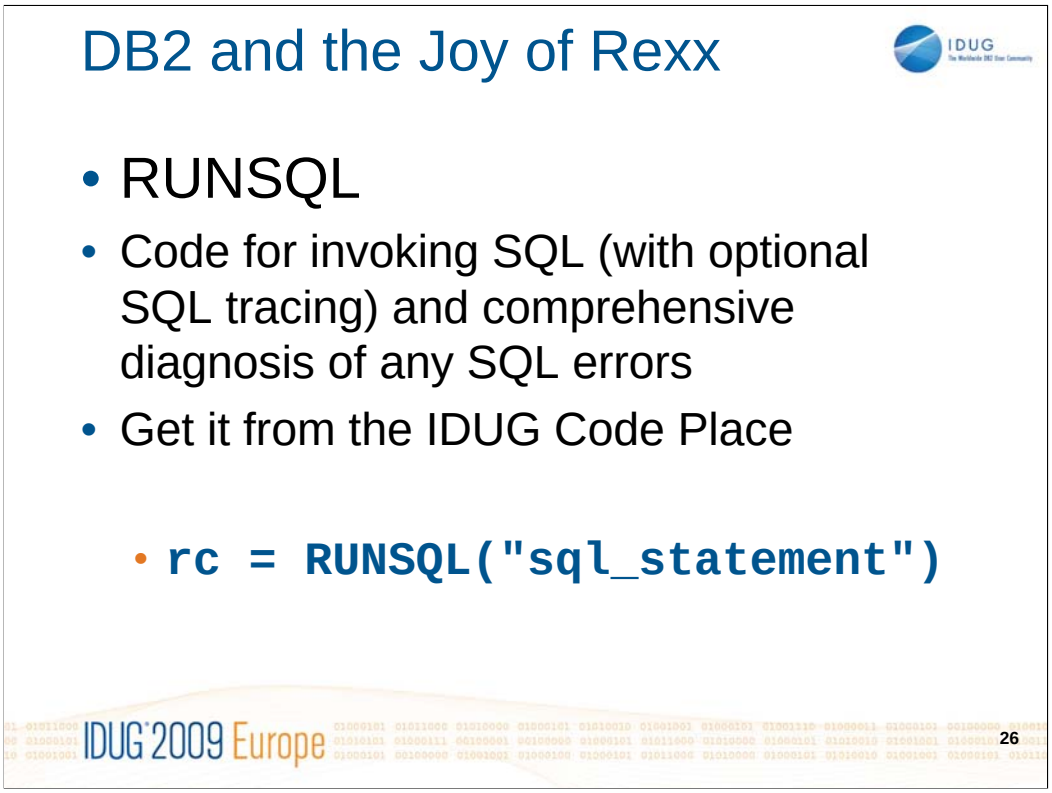

When a Rexx-DB2 program suffers an abend and terminates there is an implicit COMMIT and termination of the thread.

That includes abends because of Rexx syntax errors which occur at run time.

To avoid potential problems with any program which is doing any type of DB2 update (eg. INSERT, UPDATE or DELETE, CREATE, GRANT) it is safest to include an abend routine which includes an explicit ROLLBACK.

Rexx execs are syntax checked before they start, but that does not guarantee there will be no syntax error found at run time. That also applies to compiled Rexx execs.

The above sample code shows how to do a ROLLBACK in case of an error.



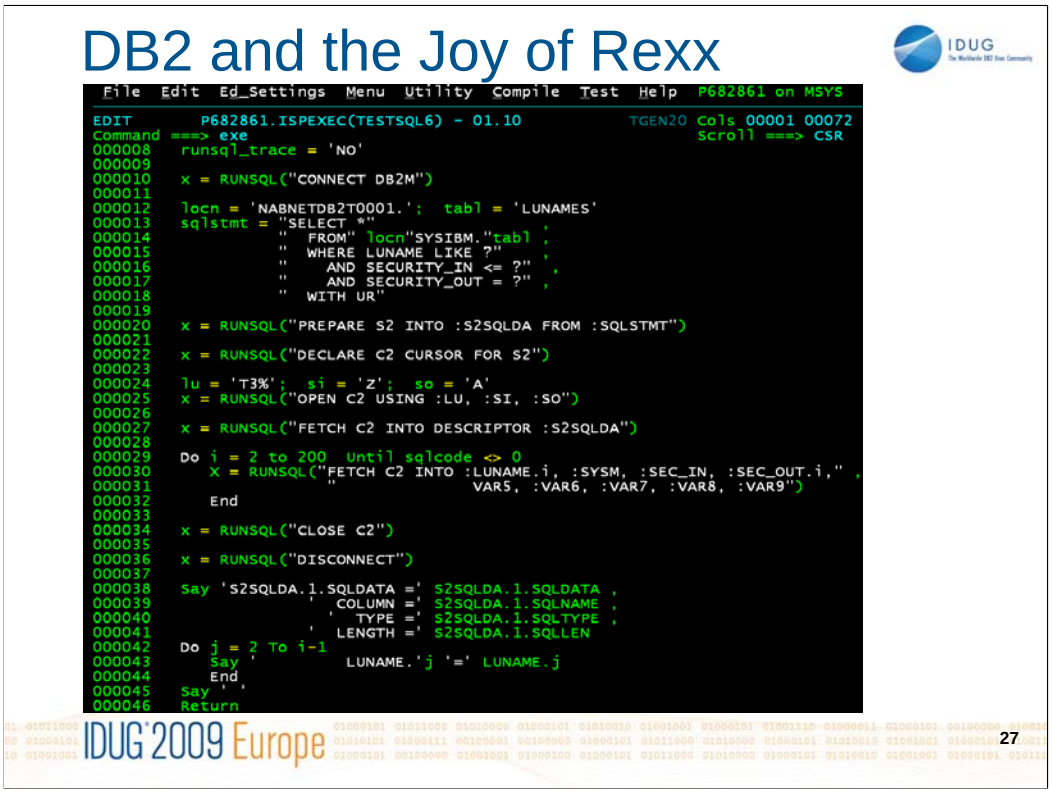
The IDUG Code Place text for RUNSQL has quite detailed documentation about the functionality. Here is an extract from the description of it:

This is designed to make problem diagnosis much easier for SQL run using DSNREXX. It provides comprehensive ERROR information in the case of a bad SQLCODE, and it can also provide a useful TRACE facility to check particular SQL statements as they run, or to write warning messages whenever +ve SQLCODEs occur. Those features are particularly useful for Rexx programs with complex SQL code, or with variable substitution or host variables in their SQL.

It can also provide standarised SQL error handling - which you can utilise to make it much quicker to write new new Rexx/SQL programs, simpler to code and read, but providing thorough SQL diagnostics.

If it is run online the output goes to the terminal, but if it runs in batch (or as a stored procedure) it defaults to writing to a DDNAME which is the same as the name of the exec.

IMPLEMENTATION:



This simple program shows SQL processed using RUNSQL. It is dynamically creating a cursor and fetching rows from it, just as is normal for DSNREXX. The main difference is that all error handling (or tracing) will be looked after by RUNSQL.

Line 8: this tells RUNSQL to NOT trace the SQL statements. (We will later run this again with runsql_trace = 'YES' to trace them all.)

Line 13: specifying an SQL statement with 2 host variables (locn, tabl) being substituted directly into it, and 3 place holders (?) for host variables. Note that this SELECT will read

from a remote table specified by a 3-part name.

Line 20: S2 is prepared and an SQLDA created into S2SQLDA compound variable.

Line 25: cursor C2 is opened using 3 host variables, whose values are specified on the previous line. They correspond to the 3 place-holders in SQLSTMT.

Line 27: first FETCH stores a row in the S2SQLDA variable.

Line 30: subsequent fetches store values into the host variables specified in the statement, and they must correspond to all columns of the table (because SELECT * used). Note

there are 2 compound output variables (LUNAME.i and SEC_OUT.i). Fetching into simple variables is often used if they are then inserted into an ISPF table after each fetch,

whereas compound variables can be a useful way of storing a whole result set.

OOPS !!

The program abended. This output was automatically generated by RUNSQL. It shows many details which can assist diagnosing SQL errors.

a) It shows the source of the program and the line number which failed, followed by the date and time.

b) It shows the text of the failing statement

c) DSNTIAR formatted output shows the SQLCODE, error message text, SQLSTATE

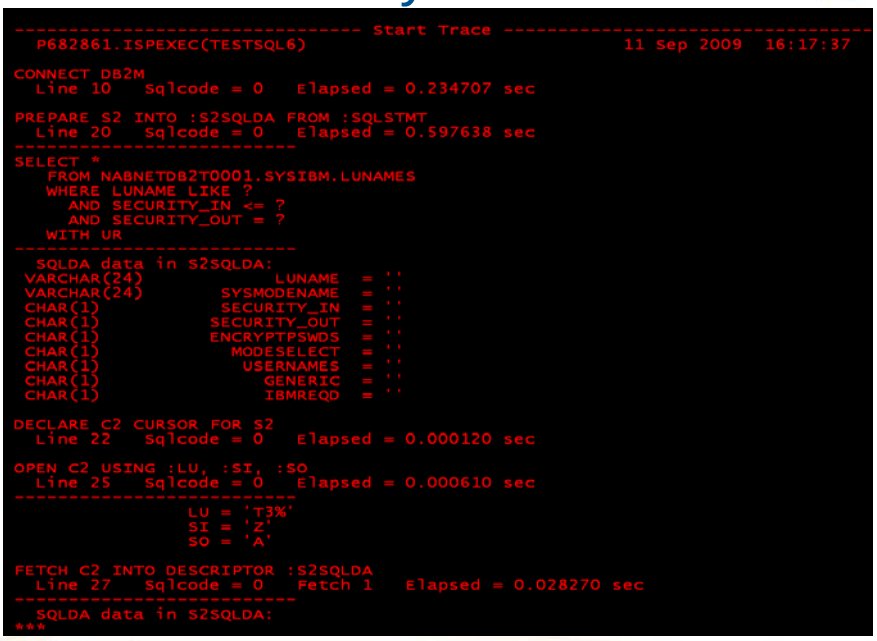
d) The text of the SQL statement text for cursor C2 is shown (after any Rexx variable substitutions).

e) The names and values of the input host variables are shown

The program ran successfully until line 30, where it abended. It is clear that the FETCH is missing a colon before “VAR5”.

Therefore, we will add the colon and try it again. We will also set **runsql_trace = ‘YES’** at the start of the program to display each SQL statement being processed.

DB2 and the Joy of Rexx

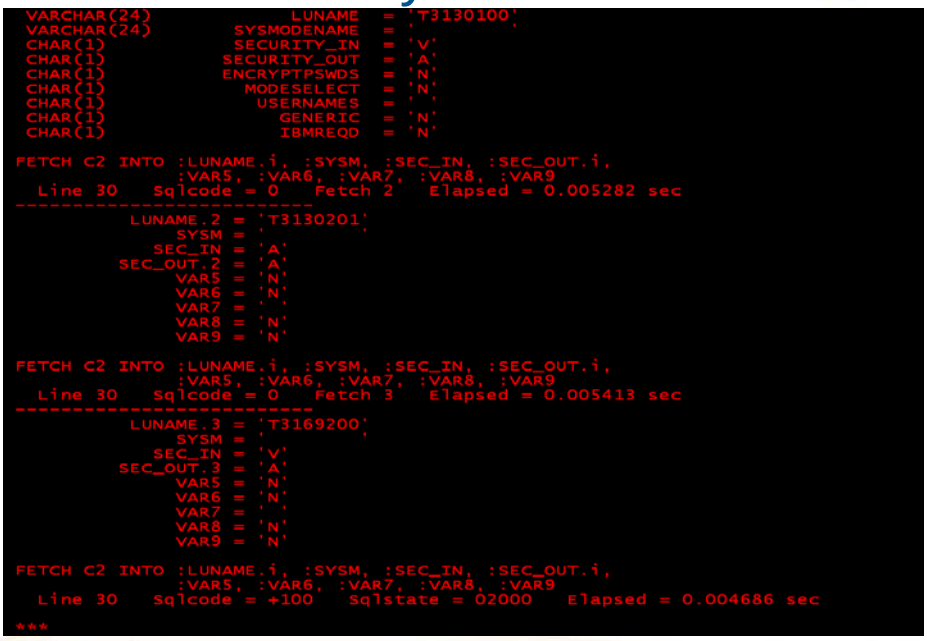


Here is the corrected program's RUNSQL trace output.

runsql_trace = 'YES' was set at the beginning of the program, and there was no subsequent **runsql_trace = 'NO'** to turn the tracing off again, hence the trace output shows all SQL statements.

The text of each SQL statement is shown, plus it's location in the program, the SQLCODE and the elapsed time it took. If there are related Rexx host variables they are shown directly underneath (fully interpreted if it is an SQLDA), and a blank line follows.

DB2 and the Joy of Rexx



Note that the first fetch was into an SQLDA compound variable structure (an associative array). Each column of the returned row is written as follows:

stem.SQLD

stem.n.SQLTYPE

stem.n.SQLLEN

stem.n.SQLLEN.SQLPRECISSION

stem.n.SQLLEN.SQLSCALE

stem.n.SQLCCSID

stem.n.SQLLOCATOR

stem.n.SQLDATA

stem.n.SQLIND

stem.n.SQLNAME

Reference: DB2 version 8.1 for z/OS - SQL Reference, Appendix E SQL Communication Area (SQLCA), The REXX SQLDA

DB2 version 9.1 for z/OS - SQL Reference, Appendix F SQL Communication Area (SQLCA), The REXX SQLDA

After all the trace output, there are finally 3 lines of output showing what LUNAME values were fetched. The first fetch was into an SQLDA structure, and the subsequent fetches were into compound variable with stem: "LUNAME." , as indicated in the way they are displayed.

RUNSQL is particularly useful for running

-Rexx Stored Procedures because it provides diagnostic error output in dynamically allocated DDNAMES matching the program names (to help separate and identify them)

-Programs with complex SQL, statements with variable substitution, place-holders or just extensive use of host variables in SQL

DB2 and the Joy of Rexx

- Invocation on LUW
  - **AIX: Initialise environments before connecting**
    - Rc = SysAddFuncPkg(‘db2rexx')
    - If rc <> 0 Then Return "Unable to initialise DB2REXX"
  - **Windows: Initialise environments before connecting**
    - If RxFuncquery('SQLDBS') Then
      If RxFuncAdd('SQLDBS','DB2AR','SQLDBS')
      Then Return "Unable to register SQLDBS"
    - If RxFuncquery('SQLDB2') Then /* CLP */
      If RxFuncAdd('SQLDB2','DB2AR','SQLDB2')
      Then Return "Unable to register SQLDB2"
    - If RxFuncquery('SQLEXEC') Then /* SQL */
      If RxFuncAdd('SQLEXEC','DB2AR','SQLEXEC')
      Then Return "Unable to register SQLEXEC"

Reference: IBM DB2 9.7 for Linux, Unix and Windows, **Developing Embedded SQL Applications,** SC27-2445-00

2 books were published about Object Rexx, but they are now both out of print:

Object REXX for Windows 95/NT, by Ulrich Wahl, Prentice Hall (March 1997)

Object-Oriented Programming with REXX, by Tom Ender, John Wiley & Sons (January 1997)

DB2 and the Joy of Rexx

- SQL invocation on LUW
  - Connect to DB2
    - `Call SQLDBS "ATTACH TO" db2name`
    - `Call SQLEXEC "CONNECT" db2name`
  - SQLEXEC must be used
    - `Call SQLEXEC sql_statement`
  - SQLCA is written to stem variables called SQLCA.xxxxx (where 'xxxxx' are the same names as the variables created by DSNREXX)
    - `Say 'SQLCODE =' sqlca.sqlcode`
    - `Say 'SQLSTATE =' sqlca.sqlstate`
  - Connect to DB2
    - `Call SQLDBS "DETACH"`
    - `Call SQLEXEC "CONNECT RESET"`

If a userid/password is required - the connection to DB2 can be done:

Call SQLEXEC "CONNECT TO" db2name "USER" userid "USING" mypassword

To disconnect it is also valid to use

Call SQLEXEC "DISCONNECT

The **SQLDA** is identical to the structure used on z/OS.

Statement names must be only **S1 – S100** and cursor names must be only **C1 – C100** as for z/OS ;(cursor names C101 –C200 are not allowed).

Reference: IBM DB2 9.7 for Linux, Unix and Windows, **Developing Embedded SQL Applications,** SC27-2445-00

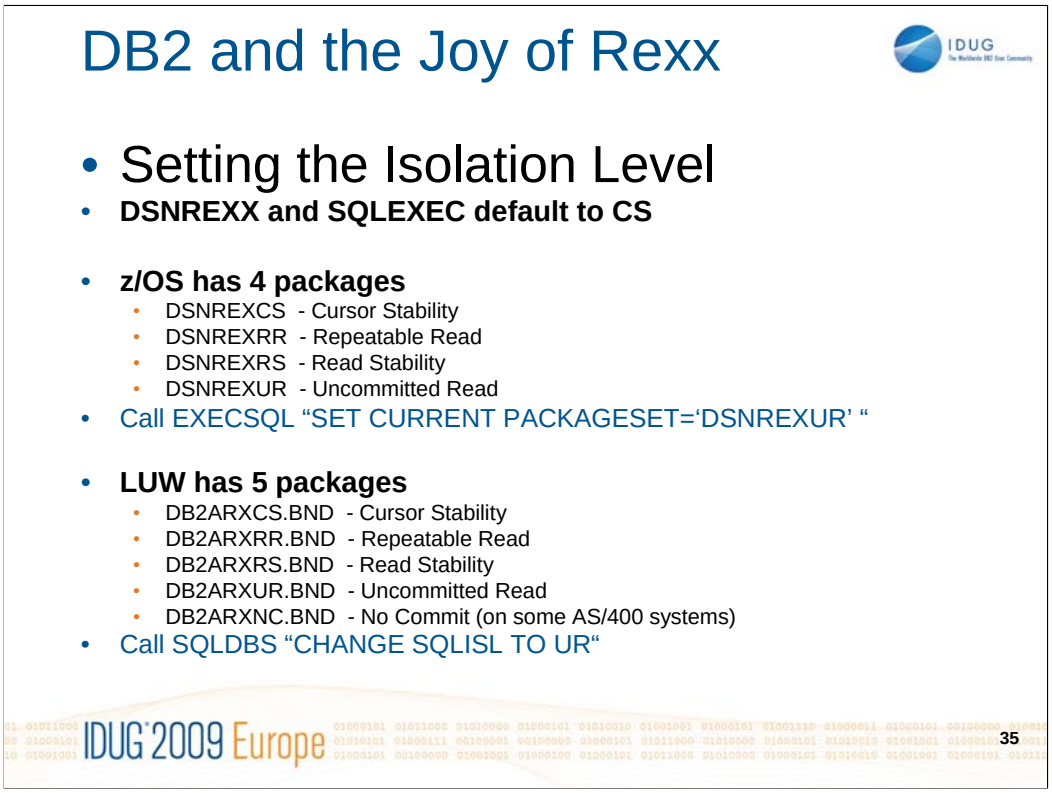
DB2 and the Joy of Rexx

- Debugging on LUW
- Return Code, SQL Code & SQL State are the same as on z/OS, and SQLCA is updated after every SQL statement or API call.
- Error messages can be produced by either of the following APIs, which use the contents of the SQLCA structure to obtain information.
- The REXX API syntax for Get Error Message is:
  `Call SQLDBS "GET MESSAGE INTO :msg [LINEWIDTH width]"`

  The REXX API syntax for Get the SQLSTATE Message is:
  `Call SQLDBS`
- `"GET MESSAGE FOR SQLSTATE sqlstate INTO :msg . . . "`

Note that SQL Code 0 and +100 are the same everywhere but the meaning of other codes can vary, whereas SQL State is more consistent.

Reference: IBM DB2 9.7 for Linux, Unix and Windows, **Administrative API Reference** SC27-2435-00



Default isolation is Cursor Stability, which can be overridden in an individual SQL statement, for example:

```
SELECT col1, col2
       FROM table
       WHERE predicate
       WITH UR
```

Otherwise, change the Isolation as shown on the slide.

When creating a Rexx Stored Procedure on z/OS: in the **CREATE PROCEDURE** statement specify the package name as **DSNREXX** to get the default, or else specify one of the 4 package names listed on the slide to get a particular Isolation Level.

Reference: IBM DB2 9.7 for Linux, Unix and Windows, **Administrative API Reference** SC27-2435-00

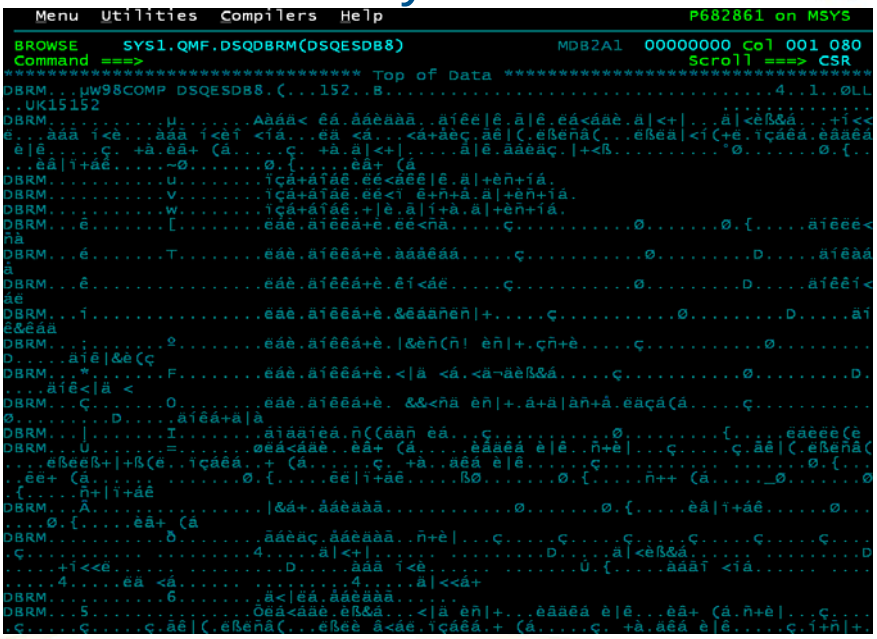
DB2 and the Joy of Rexx

- Unicode Considerations
  - This is usually only an issue on DB2 for z/OS, starting with version 8.1
  - SELECT from DB2 Catalog (or other tables with CCSID UNICODE)
  - Interpret DBRMLIB members


```
-The Catalog tables in DB2 version 8.1 (and later) are in UNICODE, but if you run an SQL
SELECT using DSNREXX on z/OS you will get the results in EBCDIC. There are some exceptional
columns like SYSIBM_SYSPACKSTMT which is VARCHAR(..) FOR BIT DATA.

-Other TSO and batch utilities like SPUFI and DSNTEP2 behave the same - showing EBCDIC for
almost all columns.

-The members of DBRMLIB libraries are entirely in EBCDIC until DB2 version 8.1, when they
contain a mixture of EBCDIC and UNICODE.  Therefore, the SQL STATEMENT text is no longer easy
to read if you BROWSE the member.  See IDUG Code Place for exec DBRMMAP which interprets DBRM
members.
```

References:

DB2 and the Joy of Rexx



This is an example of what you see when your BROWSE a member of a DBRM library.

Prior to DB2 version 8.1, the contents were all in EDCDIC, but since then the SQL statements and variable names are in UNICODE (UTF-8).

It is difficult to interpret this!

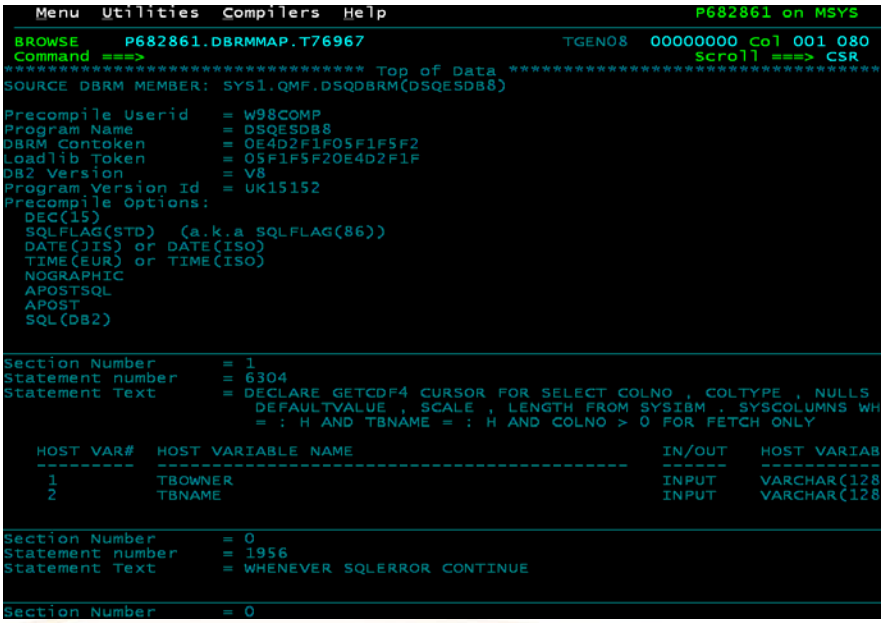
DB2 and the Joy of Rexx

- DBRMMAP

```
 Menu  Functions  Confirm  Utilities  Help                    P682861 on MSYS
VIEW              SYS1.QMF.DSQDBRM                 MDB2A1  Row 00030 of 00046
Command ===>                                                Scroll ===> CSR
          Name      Prompt       Size   Created          Changed          ID
dbrmmap__ DSQESDB8 *Browsed
_________ DSQESDGN
_________ DSQESDTA
_________ DSQESDT7
_________ DSQESDT8
_________ DSQESQ44
_________ DSQESQ45
_________ DSQESSDS
_________ DSQESSQL
_________ DSQESSQ2
_________ DSQESSQ5
_________ DSQESUSR
_________ DSQESV
_________ DSQESV2
_________ DSQETMGT
_________ DSQETSQL
_________ DSQEUPRF
          **End**
```


-Now DBRMMAP is invoked to find out more about the contents of that member. It will reformat the header information and show the SQL statements and variable names in EDCDIC

DB2 and the Joy of Rexx



-Here is the first screen of the output

DB2 and the Joy of Rexx

- Unicode Considerations

```
tablei = XRANGE('00'x,'FF'x)
tableo =  /* conversion table for Unicode UTF-8 to EBCDIC */,
'00'x'01'x'02'x'03'x'37'x'2D'x'2E'x'2F'x'16'x'05'x'25'x'0B'x ||,
'0C'x'0D'x'0E'x'0F'x'10'x'11'x'12'x'13'x'3C'x'3D'x'32'x'26'x ||,
'18'x'19'x'3F'x'27'x'1C'x'1D'x'1E'x'1F'x'40'x'4F'x'7F'x'7B'x ||,
'5B'x'6C'x'50'x'7D'x'4D'x'5D'x'5C'x'4E'x'6B'x'60'x'4B'x'61'x ||,
'F0'x'F1'x'F2'x'F3'x'F4'x'F5'x'F6'x'F7'x'F8'x'F9'x'7A'x'5E'x ||,
'4C'x'7E'x'6E'x'6F'x'7C'x'C1'x'C2'x'C3'x'C4'x'C5'x'C6'x'C7'x ||,
'C8'x'C9'x'D1'x'D2'x'D3'x'D4'x'D5'x'D6'x'D7'x'D8'x'D9'x'E2'x ||,
'E3'x'E4'x'E5'x'E6'x'E7'x'E8'x'E9'x'4A'x'E0'x'5A'x'5F'x'6D'x ||,
'79'x'81'x'82'x'83'x'84'x'85'x'86'x'87'x'88'x'89'x'91'x'92'x ||,
'93'x'94'x'95'x'96'x'97'x'98'x'99'x'A2'x'A3'x'A4'x'A5'x'A6'x ||,
'A7'x'A8'x'A9'x'C0'x'BB'x'D0'x'A1'x'07'x'20'x'21'x'22'x'23'x ||,
'24'x'15'x'06'x'17'x'28'x'29'x'2A'x'2B'x'2C'x'09'x'0A'x'1B'x ||,
'30'x'31'x'1A'x'33'x'34'x'35'x'36'x'08'x'38'x'39'x'3A'x'3B'x ||,
'04'x'14'x'3E'x'FF'x'41'x'AA'x'B0'x'B1'x'9F'x'B2'x'6A'x'B5'x ||,
'BD'x'B4'x'9A'x'8A'x'BA'x'CA'x'AF'x'BC'x'90'x'8F'x'EA'x'FA'x ||,
'BE'x'A0'x'B6'x'B3'x'9D'x'DA'x'9B'x'8B'x'B7'x'B8'x'B9'x'AB'x ||,
'64'x'65'x'62'x'66'x'63'x'67'x'9E'x'68'x'74'x'71'x'72'x'73'x ||,
'78'x'75'x'76'x'77'x'AC'x'69'x'ED'x'EE'x'EB'x'EF'x'EC'x'BF'x ||,
'80'x'FD'x'FE'x'FB'x'FC'x'AD'x'AE'x'59'x'44'x'45'x'42'x'46'x ||,
'43'x'47'x'9C'x'48'x'54'x'51'x'52'x'53'x'58'x'55'x'56'x'57'x ||,
'8C'x'49'x'CD'x'CE'x'CB'x'CF'x'CC'x'E1'x'70'x'DD'x'DE'x'DB'x ||,
'DC'x'8D'x'8E'x'DF'x
text = TRANSLATE(text,tableo,tablei) /* Unicode -> EBCDIC */
```

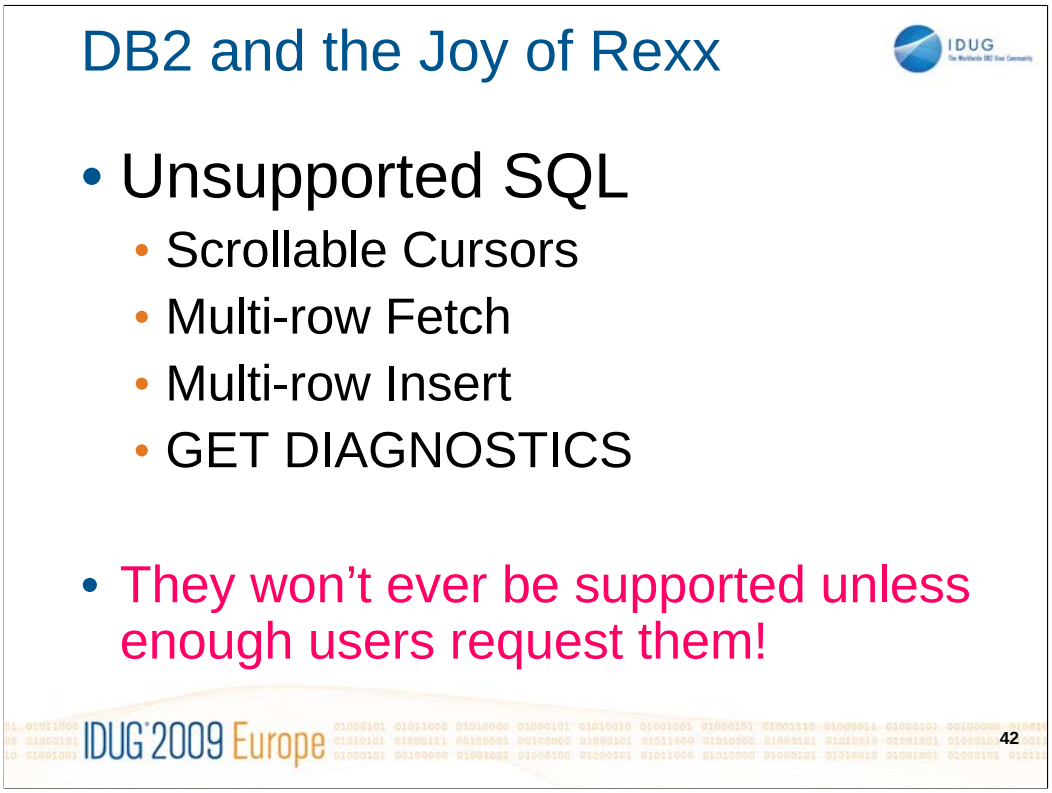

This is an example of converting text from Unicode to EBCDIC (similar to what is used in DBRMMAP).

DB2 and the Joy of Rexx

- **SPUFI**
  - SPUFI is designed to run under the DSN program, but only online in TSO/ISPF
  - Rexx can use its ISPEXEC interface to seed some ISPF variables, then invoke SPUFI with a modified SPUFI panel to run SQL immediately

```
Address TSO "NEWSTACK"              /* start a new (empty) stack */
Parse Value '; 2500 4092 4096 VB SYSDA',
            '33 256 NAMES C SECOND TIME',
   With DSNESV2B DSNESV2D DSNESV2C DSNESV21 DSNESV22 DSNESV2E,
        DSNESV24 DSNESV25 DSNESV26 DSNESV3Z DSNESV1W
"VPUT (DSNESV2B DSNESV2D DSNESV2C DSNESV21 DSNESV22 DSNESV2E",
      "DSNESV24 DSNESV25 DSNESV26 DSNESV3Z DSNESV1W) PROFILE"
Push ''
Push 'END'
Push 'SPUFI'
"SELECT CMD(DSN SYSTEM("dsneov01") TEST(0) RETRY(0))"
Address TSO "DELSTACK"               /* finished with that stack */
```


IDUG Code Place has an example of code to run SPUFI in batch using a similar approach, except that they call the standard IBM CLIST (called DSNESC01) to invoke SPUFI.



Scrollable cursors have been possible on DB2 for z/OS for a long time, but unfortunately not for DSNREXX or SQLEXEC.

Similarly, DB2 for z/OS version 8.1 and 9 provide multi-row FETCH or multi-row INSERT for almost all supported languages, but not for Rexx, and there is no plan to offer support in the next version.

Writing your own multi-row fetch for Rexx

An external program must receive parameters (including the SQL statement), then create the required Rexx stem output variables and invoke the multi-row fetch to populate them before returning to the main Rexx exec. It would also need to run **GET DIAGNOSTICS** to provide some information which is not in the SQLCA.

Multi-row with Rexx (on z/OS):

DSNTEP4 is a sample dynamic SQL program supplied with DB2. It is basically the same as DSNTEP2 with the added ability to do multi-row processing. It reads an input file of SQL statements and produces an output file with the results. A Rexx exec could invoke it. An exec could also read the output file to extract the results, but that is not very efficient!

DSNTIAUL defaults to multi-row fetch of 100 rows, which can be changed by PARM('SQL,####') or PARM('####'). For SELECTING a result set into a file it would better than DSNTEP4.

DB2 and the Joy of Rexx

- Disadvantages of Rexx
  - performance
  - no static SQL
  - some SQL not supported
  - possible “write-only code”
  - not widely used on LUW

-Interpreted Rexx is generally slower than compiled programs. Compiled Rexx programs are not as fast as most other compiled languages, partly because any calls to host environments are still interpreted (on z/OS they are not linked in).

-Dynamic SQL requires extra work by the optimizer before execution, reducing performance too.

-Scrollable cursors, multi-row SQL not supported

-The flexibility and power of Rexx make it possible to create programs which are difficult to understand.

-Not many people have experience using OO-Rexx on LUW (compared with other languages).

DB2 and the Joy of Rexx

- Why should you use Rexx?
  - Quick development
  - Can use many types of interfaces
  - Runs on many different platforms
  - Widely used by DBAs and DB2 Systems Programmers on z/OS

Session 17829

DB2 and the Joy of Rexx

Ron Brown

Ronek Consulting

Ronek.Consulting@gmail.com