

# Using REXX to Build Your Own DB2 Tools

**David Simpson**  
*Themis Training*

*Download samples and slides at:*

[www.themisinc.com/rexx](http://www.themisinc.com/rexx)

## Agenda

- Who (what) is REXX?
- Language elements
- Using REXX and DB2
- Error Handling
- Sample applications

# What is REXX?

## REXX Language

- Scripting Language supplied with z/OS
- Can be run using TSO or batch
- Can be used to create custom scripts
- DB2 interface since DB2 Version 5

# Rexx

From Wikipedia, the free encyclopedia



This article **needs additional citations for verification**. Please help [improve this article](#) by [adding citations to reliable sources](#). Unsourced material may be challenged and removed. *(December 2012)* [\(Learn how and when to remove this template message\)](#)

**Rexx (Restructured Extended Executor)** is an [interpreted programming language](#) developed at IBM by [Mike Cowlshaw](#). It is a structured, high-level [programming language](#) designed for ease of learning and reading. Proprietary and [open source REXX interpreters](#) exist for a wide range of computing platforms; [compilers](#) exist for IBM [mainframe computers](#).<sup>[2]</sup>

Rexx is used as a [scripting](#) and [macro](#) language, and is often used for processing data and text and generating reports; these similarities with [Perl](#) mean that Rexx works well in [Common Gateway Interface](#) (CGI) programming and it is indeed used for this purpose. Rexx is the primary scripting language in some operating systems, e.g. [OS/2](#), [MVS](#), [VM](#), [AmigaOS](#), and is also used as an internal macro language in some other software, such as [KEDIT](#), [THE](#) and the [ZOC](#) terminal emulator. Additionally, the Rexx language can be used for scripting and macros in any program that uses Windows Scripting Host ActiveX scripting engines languages (e.g. [VBScript](#) and [JScript](#)) if one of the Rexx engines is installed.

Rexx is supplied with VM/SP on up, TSO/E Version 2 on up, OS/2 (1.3 and later, where it is officially named *Procedures Language/2*), AmigaOS Version 2 on up, PC DOS (7.0 or 2000), and Windows NT 4.0 (Resource Kit: Regina). REXX scripts for OS/2 share the filename extension `.cmd` with other scripting languages, and the first line of the script specifies the interpreter to be used. REXX macros for REXX-aware applications use extensions determined by the application. In the late 1980s Rexx became the common scripting language for [IBM Systems Application Architecture](#), where it was renamed "SAA Procedure Language REXX."

A Rexx script or command is sometimes referred to as an *EXEC* in a nod to Rexx's role as a replacement for the older [EXEC](#) command language on [CP/CMS](#) and VM/370 and [EXEC 2](#) command language on VM/SP.

## Rexx



<b>Paradigm</b>	multiparadigm: procedural, structured
<b>Designed by</b>	<a href="#">Mike Cowlshaw</a>
<b>Developer</b>	Mike Cowlshaw, IBM
<b>First appeared</b>	1979; 38 years ago
<b>Stable release</b>	<a href="#">ANSI X3.274 / 1996</a> ; 21 years ago
<b>Typing discipline</b>	Dynamic
<b>Filename extensions</b>	<code>.cmd</code> <code>.exec</code> <code>.rexx</code> <code>.rex</code>

### Major implementations

VM/SP, TSO/E V2, SAAREXX, ARexx, BREXX, KEXX, Regina<sup>[1]</sup>

### Dialects

NetRexx, Object REXX, now ooREXX

### Influenced by

[PL/I](#), [ALGOL](#), [EXEC](#), [EXEC 2](#)

## DB2 REXX Extension

- What's Bad about it?
  - Can't do a “singleton” SELECT
  - It's interpretive: Slower than Cobol
  
- What's Good about it?
  - It's FREE
  - Can define up to 200 Cursors
  - Don't pre-define host Variables!!!! (except SPs)
  - Supports Almost all DB2 for z/OS SQL Syntax
  - It's interpretive: quickly modifiable, easy to rollout

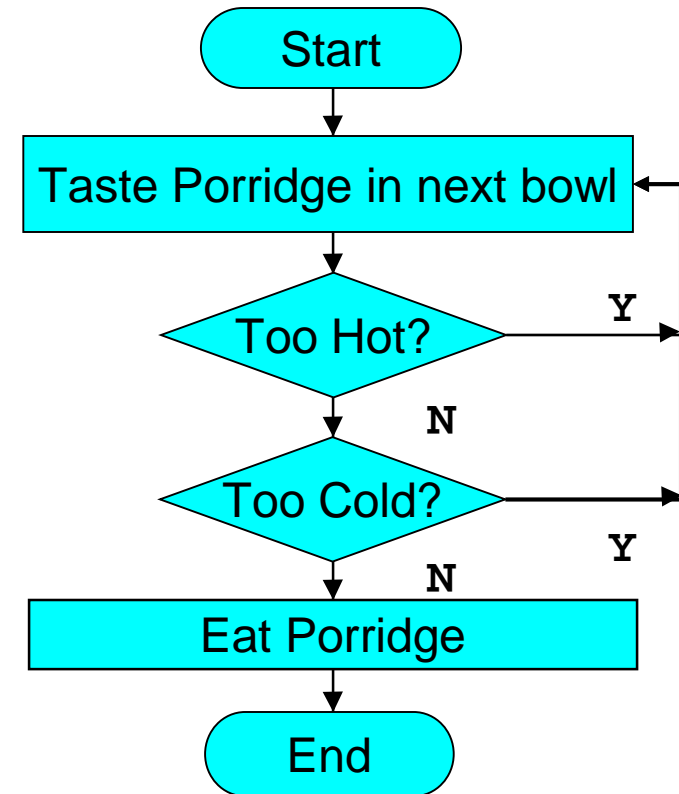


**No Bind  
Required!**

# REXX Coding Basics

## REXX Basics - Coding

```
/* REXX - Goldilocks -----*/  
Call SELECT_PORRIDGE_RTN  
Exit(0)  
  
SELECT_PORRIDGE_RTN:  
  
EAT_PORRIDGE_FLG = "N"  
  
Do Until EAT_PORRIDGE_FLG = "Y"  
  
  Call TASTE_RTN  
  If TASTE_ID > 5 Then Iterate  
  If TASTE_ID < 5 Then Iterate  
  If TASTE_ID = 5 Then EAT_PORRIDGE_FLG = "Y"  
  
End  
Call GOBBLE_PORRIDGE_ALL_UP_RTN  
Return;
```

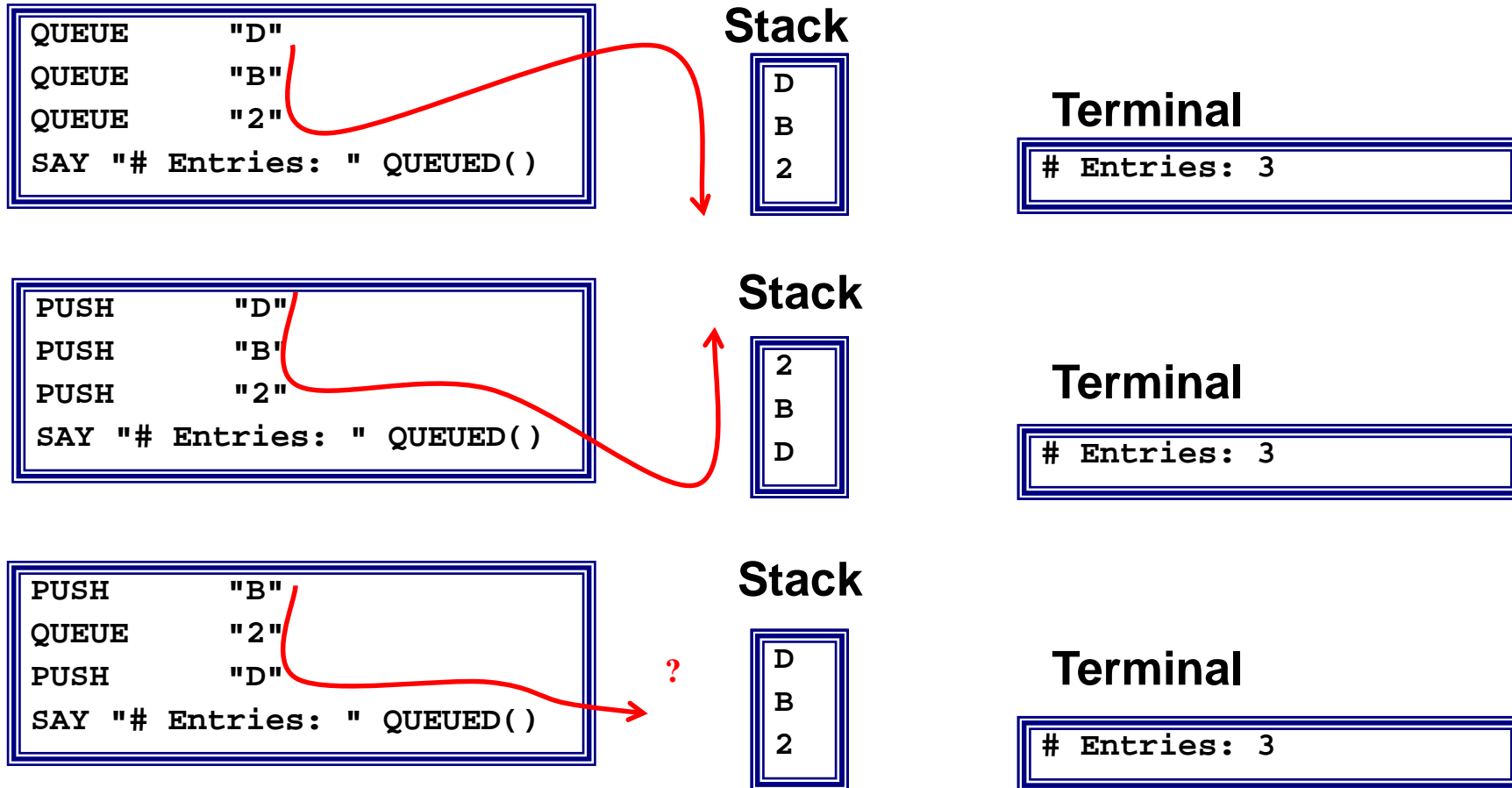




## REXX Basics - Comments

```
/* ----- Select and eat Porridge ----- */
SELECT_PORRIDGE_RTN:
EAT_PORRIDGE_FLG = "N"          /* Set flag designating that
                                Goldilocks has not found
                                acceptable porridge          */
Do Until EAT_PORRIDGE_FLG = "Y" /* Keep testing porridge          */
    Call TASTE_RTN              /* Set TASTE_ID to value range 1-9.
                                A Value of "1" is very COLD,
                                A Value of "9" is very HOT    */
/* Too Hot          */ If TASTE_ID > 5 Then Iterate /* Check next Bowl */
/* Too Cold         */ If TASTE_ID < 5 Then Iterate /* Check next Bowl */
/* Just Right       */ If TASTE_ID = 5 Then,        /* Bowl Selected   */
    EAT_PORRIDGE_FLG = "Y"
End                          /* End of Porridge test loop, check first bowl */
Call GOBBLE_PORRIDGE_ALL_UP_RTN /* Consume Porridge */
Return; /* Porridge has been selected and gobbled - back to main */
```

## REXX Basics – The Stack



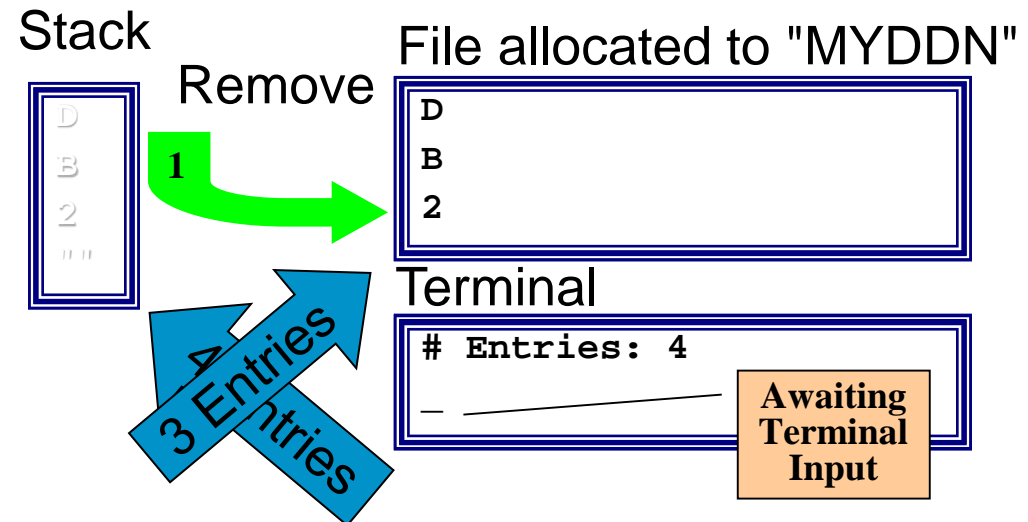
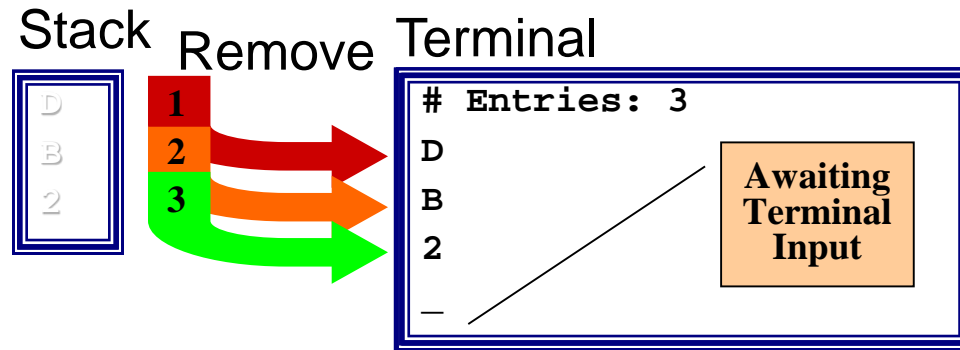
## REXX Basics – More Stack

```

QUEUE "D"
QUEUE "B"
QUEUE "2"
SAY "# Entries: " QUEUED()
DO QUEUED()
  PULL stackdata
  SAY stackdata
END
PULL termdata
  
```

```

QUEUE "D"
QUEUE "B"
QUEUE "2"
QUEUE ""
SAY "# Entries: " QUEUED()
EXECIO * DISKW MYDDN (FINIS
PULL termdata
  
```

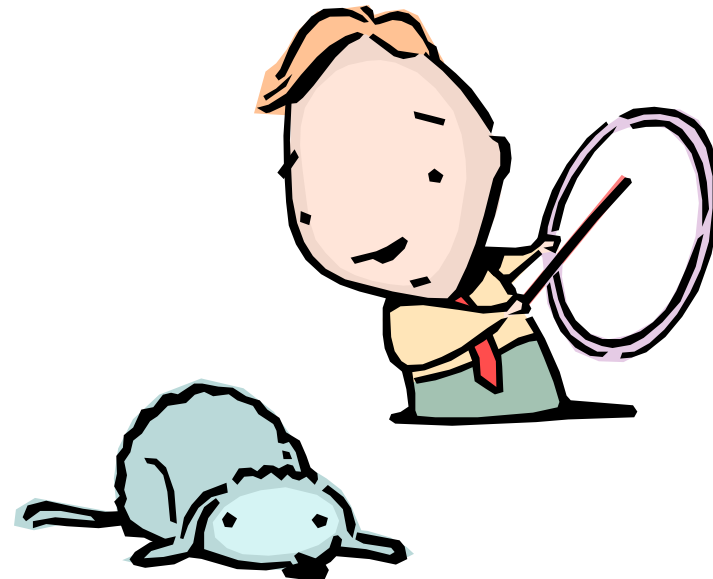


# REXX and DB2

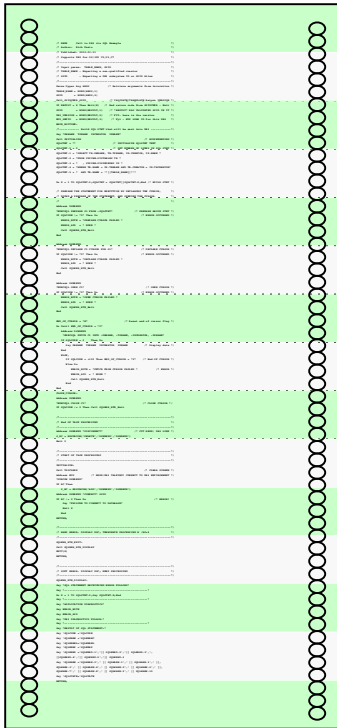
## REXX & DB2

- DB2 doesn't have a native "Scripting language" for use on ISPF
- Third party tools fill the gap
- Homegrown
  - Custom Code: Cobol, ASM,...
  - DSNTEP2 / DSNTEP4
  - DSNTIAUL

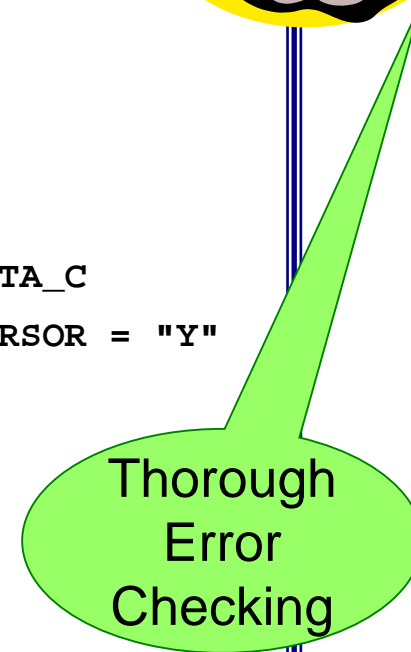
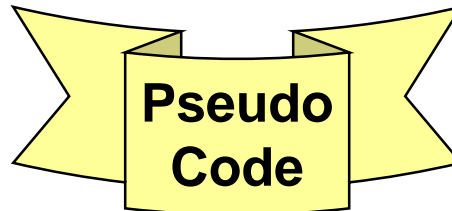
**FETCH** me some data!



# REXX SQL Interface



```
/* Build SQLSTMT - Not shown */  
DECLARE CURSOR  
PREPARE STMT  
OPEN CURSOR  
END_OF_CURSOR = "N"  
DO UNTIL END_OF_CURSOR = "Y"  
    FETCH INTO :DATA_A, :DATA_B, :DATA_C  
    If SQLCODE = +100 Then END_OF_CURSOR = "Y"  
    Else Call SQLERR_RTN  
End  
Exit;  
SQLERR_RTN:  
xxx  
Return;
```



## Parlez-vous DB2?



Before I can "talk" to DB2, I need to understand the language!

Add "DSNREXX" to my Command Environment.

Create the REXX/DB2 command interface

```
ADDRESS MVS "SUBCOM DSNREXX"  
IF RC THEN S_RC = RXSUBCOM('ADD', 'DSNREXX', 'DSNREXX')
```

Remove the REXX/DB2 command interface

```
S_RC = RXSUBCOM('DELETE', 'DSNREXX', 'DSNREXX')
```

## Connect to DB2

Establish thread between  
REXX and DB2



The following statements do the same thing.

```
ADDRESS DSNREXX "CONNECT DB2A"  
IF SQLCODE /= 0 THEN CALL error_routine  
  
-----  
  
DB2_SSID = "DB2A"  
ADDRESS DSNREXX "CONNECT" DB2_SSID  
IF SQLCODE /= 0 THEN CALL error_routine  
  
-----  
  
DB2_SSID = "DB2A"  
ADDRESS DSNREXX "CONNECT" DB2_SSID  
IF RC = -3 THEN . . . /* Missed SUBCOM setup */
```



## Run SQL

### Example of Execute Immediate

```
ADDRESS DSNREXX
"EXECSQL UPDATE TEST.MYTABLE SET INQ_IND = 'Y'"
IF SQLCODE \= "0" THEN,
DO
  ERROR_NOTE = "UPDATE FAILED "
  ERROR_AID = " NONE "
  CALL SQLERR_RTN_EXIT
END
```

Test your  
SQLCODE!

More on this  
stuff later



## Building an SQL Statement

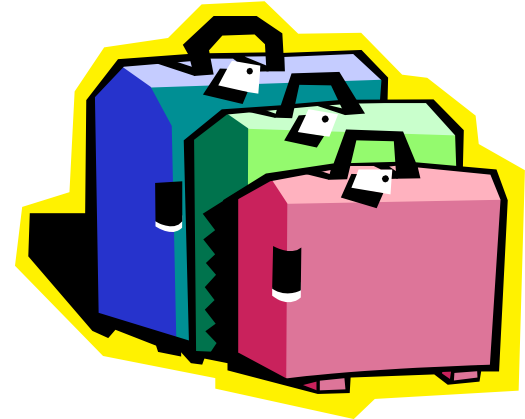
```
SQLSTMT = "" /* INITIALIZE SQLSTMT TEXT */
SQLSTMT.0 = 3 /* SET NUMBER OF LINES IN SQL STMT */
SQLSTMT.1 = "SELECT TB.DBNAME, TB.TSNAME, IX.CREATOR,
SQLSTMT.2 = " FROM " || SOURCE_TABLE
SQLSTMT.3 = "WHERE NAME = \" || HOST_VARIABLE | | \"' "

/* BUILD THE SQLSTMT FOR EXECUTION; EACH LINE APPENDED TO THE PRIOR */
```

Glue literal into the statement

```
DO X = 1 TO SQLSTMT.0; SQLSTMT = SQLSTMT || SQLSTMT.X | | " " ;END
```

```
ADDRESS DSNREXX "EXECSQL " SQLSTMT
IF SQLCODE \= "0" THEN,
DO
  ERROR_NOTE = "QUERY FAILED "
  ERROR_AID = " NONE "
  CALL SQLERR_RTN_EXIT
END
```



## Building an SQL Statement

```
SQLSTMT = ""  
SQLSTMT.0 = 3  
SQLSTMT.1 = "SELECT TB.DBNAME, TB.TSNAME, IX.CREATOR, IX.NAME"  
SQLSTMT.2 = " FROM " || SOURCE_TABLE  
SQLSTMT.3 = "WHERE NAME = \" || HOST_VARIABLE || "\"
```

```
DO X = 1 TO SQLSTMT.0; SQLSTMT = SQLSTMT || SQLSTMT.X || " " ;END
```

### Memory area for SQLSTMT

```
SELECT TB.DBNAME, TB.TSNAME, IX.CREATOR, IX.NAME FROM SYSIBM.SYSTA  
BLES WHERE NAME = 'fred'
```

```
DO X = 1 TO SQLSTMT.0; SQLSTMT = SQLSTMT || SQLSTMT.3 || " " ;END
```

## Cursors

```
000063 SQLSTMT = ""
000064 SQLSTMT.0 = 5
000065 SQLSTMT.1 = "SELECT NAME, COLTYPE, LENGTH, SCALE, NULLS      "
000066 SQLSTMT.2 = "  FROM SYSIBM.SYSCOLUMNS                        "
000067 SQLSTMT.3 = "  WHERE TBNAME = ? "
000068 SQLSTMT.4 = "  AND   TBCREATOR = ? "
000069 SQLSTMT.5 = "  ORDER BY COLNO "
000070 DO X = 1 TO SQLSTMT.0;SQLSTMT = SQLSTMT||SQLSTMT.X||" ";END
000071 Address DSNREXX
000072 "EXECSQL PREPARE S1 FROM :SQLSTMT" /* PREPARE ABOVE STMT */
000073 If SQLCODE \= "0" Then Do /* ERROR OCCURRED */
000074   ERROR_NOTE = "PREPARE CURSOR FAILED "
000075   ERROR_AID = " NONE "
000076   Call SQLERR_RTN_Exit
000077 End
```

Use parameter marker in place of literal

## Cursors

```
000079 Address DSNREXX
000080 "EXECSQL DECLARE C1 CURSOR FOR S1" /* DECLARE CURSOR */
000081 If SQLCODE \= "0" Then Do /* ERROR OCCURRED */
000082     ERROR_NOTE = "DECLARE CURSOR FAILED "
000083     ERROR_AID = " NONE "
000084     Call SQLERR_RTN_Exit
000085 End
000086
000087 Address DSNREXX
000088 "EXECSQL OPEN C1 USING :TBSN, :TBCR" /* OPEN CURSOR */
000089 If SQLCODE \= "0" Then Do /* ERROR OCCURRED */
000090     ERROR_NOTE = "OPEN CURSOR FAILED "
000091     ERROR_AID = " NONE "
000092     Call SQLERR_RTN_Exit
000093 End
```

Fill in values for  
parameter markers

## Cursors

```
000094 EOF_SW = "N"
000095 SAY "COLUMNS: "
000096 SAY "-----"
000097 DO WHILE EOF_SW = "N"
000098     Address DSNREXX
000099     "EXECSQL FETCH C1 INTO :CN, :TY, :LN, :SC, :NUL"
000100     IF SQLCODE = "100" THEN EOF_SW = "Y"
000101     ELSE
000102         DO
000103             If SQLCODE \= "0" Then Do
000104                 ERROR_NOTE = "FETCH JOBS FAILED "
000105                 ERROR_AID = " NONE "
000106                 Call SQLERR_RTN_Exit
000107             End
000108             IF TY = "DATE" | TY = "TIME" | tu="SMALLINT" | TY="INTEGER" | ,
```

## Cursors

- Cursor names are pre-defined
  - C1 thru C100 defined WITH RETURN
  - C51 thru C100 also have WITH HOLD
- Statement names are also pre-defined
  - S1 thru S100

That's NOT like COBOL!



## Display Table Definition

```
Table: THEMIS81.PROJ  
Database: DTHM81  
Tablespace: TS00PROJ  
Tablespace Type: Classic Segmented
```

### COLUMNS:

```
-----  
PROJNO      CHAR      (6)  NOT NULL  
PROJNAME    VARCHAR   (24) NOT NULL  
DEPTNO      CHAR      (3)  NOT NULL  
EMPNO       CHAR      (6)  NOT NULL  
PRSTAFF     DECIMAL   (5,2)  
PRSTDATE    DATE  
PRENDATE    DATE  
MAJPROJ     CHAR      (6)
```

\*\*\*

```
ISPF Primary Option Menu  
Option ==> TSO TB DB1A THEMIS81.PROJ  
Settings Terminal and user parameters  
Display source data as listings
```



## Calling a DB2 Command from REXX

Command goes on the stack with blank line to complete

```
/*----- STUFF DB2 COMMAND STRING INTO THE STACK ----- */
"NEWSTACK"                /* Create input stack for TSO CMD processor */
X      = OUTTRAP(CMD_OUTPUT.)          /* Capture to STEM Variable */
QUEUE "-DISPLAY UTILITY("UTIL_ID")" /* "UTIL_ID" is a REXX Variable */
QUEUE "END"
QUEUE ""
/*      Execute the contents of the command stack using DSN processor */
ADDRESS TSO "DSN SYSTEM ("SSID")"
X      = OUTTRAP(OFF)                /* Turn off capture */
"DELSTACK"                /* Clear the previously created stack */
```

# Error Handling

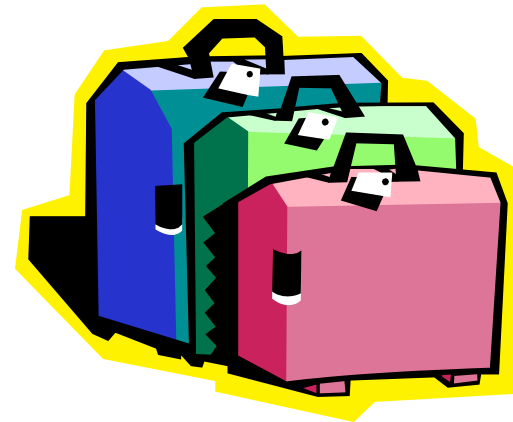
## Building an SQL Statement

```
SQLSTMT = "" /* INITIALIZE SQLSTMT TEXT */
SQLSTMT.0 = 3 /* SET NUMBER OF LINES IN SQL STMT */
SQLSTMT.1 = "SELECT TB.DBNAME, TB.TSNAME, IX.CREATOR, IX.NAME"
SQLSTMT.2 = " FROM " || SOURCE_TABLE
SQLSTMT.3 = "WHERE NAME = \" | |HOST_VARIABLE| |\"'"

/* BUILD THE SQLSTMT FOR EXECUTION; EACH LINE APPENDED TO THE PRIOR */

DO X = 1 TO SQLSTMT.0; SQLSTMT = SQLSTMT || SQLSTMT.X || " " ;END

ADDRESS DSNREXX "EXECSQL " SQLSTMT
IF SQLCODE \= "0" THEN,
DO
  ERROR_NOTE = "QUERY FAILED "
  ERROR_AID = " NONE "
  CALL SQLERR_RTN_EXIT
END
```



## Error Handling



SQL STATEMENT RECEIVING ERROR FOLLOWS

```
SELECT TB.DBNAME, TB.TSNAME, IX.CREATOR, IX.NAME  
FROM SYSIBM.SYSTABLES  
WHERE NAME = 'fred'
```

```
APPLICATION DIAGNOSTICS  
QUERY FAILED  
NONE
```

Shouldn't this be an  
SQLCODE -206?!

DB2 DIAGNOSTICS FOLLOW:

RESULT OF SQL STATEMENT:

SQLCODE = -104

SQLERRM = SQLERRM = SQLSTMT: FETCH, INSERT, UPDATE, DELETE, OPEN, CLOSE, SET, CALL

SQLERRML = SQLERRML

SQLERRP = DSN

SQLERRD = 0, 0, 0, 0, 0, 0

SQLWARN = , , , , , , , , , ,

SQLSTATE = 42601

## Error Handling

```
SQLERR RTN_DISPLAY:
Say "SQL STATEMENT RECEIVING ERROR FOLLOWS"
Say "-----"
Do X = 1 TO SQLSTMT.0;Say SQLSTMT.X;END
Say "-----"
Say "APPLICATION DIAGNOSTICS"
Say ERROR_NOTE
Say ERROR_AID
Say "DB2 DIAGNOSTICS FOLLOW:"
Say "-----"
Say "RESULT OF SQL STATEMENT:"
Say 'SQLCODE = 'SQLCODE
Say 'SQLERRM = 'SQLERRMC
Say 'SQLERRML = 'SQLERRML
Say 'SQLERRP = 'SQLERRP
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', '
SQLTEXT
```

Display the offending  
Statement

Supplied by the caller

Dump the SQLCA

## DSNTIAR?

```
Say "-----"
/* Package SQLCA for DSNTIAR usage */
NUMERIC DIGITS 10 /* Allow for big numbers in SQLCA */
SQL_ERRD = "";Do I = 1 To 6;SQL_ERRD = SQL_ERRD || D2C(SQLERRD.I,4);End
SQL_WARN = "";Do I = 0 To 10;SQL_WARN = SQL_WARN || LEFT(SQLWARN.I,1);End
SQLCA = 'SQLCA ' || D2C(136,4) || D2C(SQLCODE,4) || D2C(70,2),
|| LEFT(SQLERRMC,70) || 'DSN ' || SQL_ERRD || SQL_WARN || LEFT(SQLSTATE,5)
/* If the length is beyond DSNTIAR possible values (72-240), reset */
If MSG_LEN < 72 | MSG_LEN > 240 Then MSG_LEN = 120 /* Outside scope */
If MSG_LEN = "MSG_LEN" Then MSG_LEN = 120 /* Default msg length 120 */
DB2_ERR_MSG = D2C(MSG_LEN * 12,2) || COPIES(' ',MSG_LEN * 12)
DB2_ERR_LEN = D2C(MSG_LEN,4)
Address /* Execute DSNTIAR program with SQLCA/Parm data */
Address LINKPGM "DSNTIAR SQLCA DB2_ERR_MSG DB2_ERR_LEN"
If RC < 5 Then,
Do
If RC = 4 Then Say "DSNTIAR RC=4 Message Area Truncated"
```

## User Messages



```
RFMS0015 - rexxname Failure to connect to database. RC: return code  
SSID: ssid
```

**Explanation:**

Attempt to connect to DB2 Subsystem failed. Return code and DB2 SSID displayed.

**Severity:**

Severe

**Application Action:**

Connection to DB2 could not occur. Message is displayed; processing halts.

**User Response:**

Check to see if the subsystem name is spelled correctly.

Check to see if the DB2 subsystem is active

Check to see if the REXX is executing on the proper LPAR where the DB2 is supposed to be running.



# Examples...



## Running REXX in Batch

```
//STEP1 EXEC PGM=IKJEFT1B,REGION=3M,  
// PARM='CLEANUP DBTHMA1'  
//SYSEXEC DD DISP=SHR,DSN=THEMIS.BATCH.REXX  
//SYSTSPRT DD SYSOUT=*  
//SYSPRINT DD SYSOUT=*  
//SYSUDUMP DD SYSOUT=*  
//SYSTSIN DD DUMMY  
//*
```

First word of the PARM is the REXX being run.  
Next come any arguments expected by the script.  
SYSEXEC is the location of the script.

## Running REXX in Batch

```
//STEP1 EXEC PGM=IKJEFT1B,REGION=3M,  
// PARM='CLEANUP DBTHMA1'  
//SYSEXEC DD DISP=SHR,DSN=THEMIS.BATCH.REXX  
//SYSTSPRT DD SYSOUT=*  
//SYSPRINT DD SYSOUT=*  
//SYSUDUMP DD SYSOUT=*  
//SYSTSIN DD DUMMY  
//*
```

This particular script does the following.

- Deletes any datasets owned by DBTHMA1
- Queries the DB2 catalog and drops any objects owned by DBTHMA1
  - Tables
  - Views
  - Stored Procedures

## -DISPLAY UTILITY(\*)

```
DSNU100I  -DB2A DSNUGDIS - USERID = ROGUE
          MEMBER =
          UTILID = ROGUE.ROUGE1
          PROCESSING UTILITY STATEMENT 1
          UTILITY = LOAD
          PHASE = RELOAD    COUNT = 61197984
          NUMBER OF OBJECTS IN LIST = 1
          LAST OBJECT STARTED = 1
          STATUS = STOPPED

DSNU100I  -DB2A DSNUGDIS - USERID = JOEPGMR
          MEMBER =
          UTILID = JOEPGMR.JOEPGMR2
          PROCESSING UTILITY STATEMENT 1
          UTILITY = LOAD
          PHASE = RELOAD    COUNT = 0
          NUMBER OF OBJECTS IN LIST = 1
          LAST OBJECT STARTED = 1
          STATUS = STOPPED

DSNU105I  -DB2A DSNUGDIS - USERID = COOLDBA
          MEMBER =
          UTILID = COOLDBA.REORG1
          PROCESSING UTILITY STATEMENT 1
          UTILITY = REORG
```



```
PHASE = BUILD    COUNT = 3260046018
NUMBER OF OBJECTS IN LIST = 1
LAST OBJECT STARTED = 1
DSNU111I -DB2A DSNUGDIS - SUBPHASE = COPY COUNT = 9044775
DSNU111I -DB2A DSNUGDIS - SUBPHASE = SORTOUT COUNT = 2056453640
DSNU111I -DB2A DSNUGDIS - SUBPHASE = RUNSTATS COUNT = 1720416
DSNU105I -DB2A DSNUGDIS - USERID = RFAZIO
MEMBER =
UTILID = REORG001
PROCESSING UTILITY STATEMENT 1
UTILITY = REORG
PHASE = LOG    COUNT = 0
NUMBER OF OBJECTS IN LIST = 1
LAST OBJECT STARTED = 1
STATUS = ACTIVE
DSNU347I -DB2A DSNUGDIS -
DEADLINE = NONE
DSNU384I -DB2A DSNUGDIS -
MAXRO = DEFER
LONGLOG = CONTINUE
DELAY = 1200 SECONDS
DSNU383I -DB2A DSNUGDIS - CURRENT ITERATION NUMBER = 207
WRITE ACCESS ALLOWED IN THIS ITERATION = YES
```

```
ITERATION BEFORE PREVIOUS ITERATION:  
  ELAPSED TIME = 01:41:12  
  NUMBER OF LOG RECORDS PROCESSED = 3746459374  
PREVIOUS ITERATION:  
  ELAPSED TIME = 00:23:10  
  NUMBER OF LOG RECORDS PROCESSED = 84763764  
CURRENT ITERATION:  
  ESTIMATED ELAPSED TIME = 04:10:11  
  ACTUAL ELAPSED TIME SO FAR = 03:01:03  
  ACTUAL NUMBER OF LOG RECORDS BEING PROCESSED = 5857634451  
CURRENT ESTIMATE FOR NEXT ITERATION:  
  ELAPSED TIME = 01:34:11  
  NUMBER OF LOG RECORDS TO BE PROCESSED = 36465284634  
DSNU111I  -DB2A DSNUGDIS - SUBPHASE = COPY COUNT = 5613883  
DSN9022I  -DB2A DSNUGCCC '-DISPLAY UTILITY' NORMAL COMPLETION
```



DB2 Utility Report for DB2 Subsystem ID DBT4 Display Mode: TERSE MOD5  
UTILID Mask(\*)

REF	USERID	UTILID	TYPE	PHASE	COUNT	STM--LIST			STATUS	MEMBER
NBR						NBR	CUR	TOT		
1	ROGUE	ROGUE.ROUGE1	LOAD	RELOAD	61,197,984	1	1	1	STOPPED	
2	JOEPGMR	JOEPGMR.JOEPGMR2	LOAD	RELOAD		0	1	1	STOPPED	
3*	COOLDBA	COOLDBA.REORG1	REORG	BUILD	3,260,046,018	1	1	1	ACTIVE	
4*	RFAZIO	REORG001	REORG	LOG		0	1	1	ACTIVE	

\* - Additional utility info available for display  
<Enter> to loop, <Quit> to exit, <Help> for more options

Commas?!?!

```

DB2 Utility Report for DB2 Subsystem ID DBT4 Display Mode: VERBOSE MOD5
                                UTILID Mask(*)

REF                                STM--LIST
NBR  USERID  ----UTILID-----  ---TYPE  ---PHASE  ----COUNT-----  NBR CUR TOT  STATUS  MEMBER
-----
1    ROGUE    ROGUE.ROUGE1      LOAD   RELOAD    61,197,984      1  1  1  STOPPED
-----
2    JOEPGMR  JOEPGMR.JOEPGMR2      LOAD   RELOAD           0  1  1  1  STOPPED
-----
3    COOLDBA  COOLDBA.REORG1      REORG   BUILD    3,260,046,018   1  1  1  ACTIVE
3    COOLDBA  COOLDBA.REORG1  SUBPHASE   COPY      9,044,775
3    COOLDBA  COOLDBA.REORG1  SUBPHASE  SORTOUT   2,056,453,640
3    COOLDBA  COOLDBA.REORG1  SUBPHASE  RUNSTATS  1,720,416
-----
4    RFAZIO    REORG001      REORG   LOG           0  1  1  1  ACTIVE
4    RFAZIO    REORG001  SUBPHASE   COPY    5,613,883
-----
----ITERATION----  Time      LOG RECORDS
205 ACTUAL    01:41:12  3,746,459,374
206 ACTUAL    00:23:10   84,763,764
207 ACTUAL    03:01:03  5,857,634,451 WRITE ACCESS ALLOWED
207 ESTIMATE  04:10:11           N/A
208 FORECAST  01:34:11  36,465,284,634

REORG DEADLINE = NONE
MAXRO = DEFER, LONGLOG = CONTINUE, DELAY = 1200 SECONDS
-----
<Enter> to loop, <Quit> to exit, <Help> for more options

```

## A REXX SQL Processor

```
//REXX      EXEC PGM=IRXJCL,PARM='SQLPROC DB2A'  
//SYSEXEC  DD DISP=SHR,DSN=MY.REXX.EXEC ←Your REXX source lib.  
//SYSPRINT DD SYSOUT=*  
//SQLSTMT  DD *  
-- Drop my.index  
-- If it's gone already...great; keep going  
DROP INDEX MY.INDEX;  
--sqltest(-204,Dropped already...ok)  
INSERT INTO MY.TABLE VALUES(...);  
--sqltest(-803,Dup is ok)  
DELETE FROM MY.TABLE WHERE X='Y';  
UPDATE MY.TABLE  
  SET X='Y'  
  WHERE X='N';
```



Note:  
This is an  
input script.

DSNTEP2 says  
+100 on update  
is GOOD...is it?

The REXX code  
is not shown!

Bad Conditions:	+100 Update	+100 Delete
Good Conditions:	-803 Insert	-204 Drop



## The “Enforcer” - APPLCHKR



### Define Rules

- Check the objects against the rules regularly
- Report warnings
- Correct errors
- Simplify your daily routine

For every table in a given database...

Verify the CREATOR is 'x'

Ensure each table has the 5 required aliases: A, B, C, D, E

Some tables may have optional alias of Q or P

All tables need select granted to authid U1, update to U2

All views on these tables need different security profile

## Dropping a tablespace

Drop a tablespace with 254 parts  
 Lock DB2 Directory  
 Lock DB2 Catalog  
 Lock DataBase Descriptor (DBD)

Can take 10 min each tablespace.

REXX Name: TSDROP

Control card Input: DBNAME.TSNAME Masking allowed

JCL Output:

```

DB2 CMD -STOP DATABASE( ) SPACE(Tablespace)
DB2 CMD -STOP DATABASE( ) SPACE(Indexspace)
IDCAMS DELETE Tablespace Part x
IDCAMS DELETE Indexspace Part x
SQL DROP TABLESPACE
```



Directory



DB2  
Catalog



DBD

**LOCKDOWN**  
1 Second



## Reference

### DB2

????-???? DB2 for OS/390 REXX Language Support *Version 5 (1999)\**

**\* SQLDA IS WRONG!!!**

SC27-8859 DB2 12 for z/OS SQL Reference

SC27-8845 DB2 12 for z/OS Application Programming and SQL Guide

### Rexx

SA-22-7790 z/OS TSO/E REXX Reference

### Redbook

SG24-6465 DB2 UDB for z/OS Version 8 Performance Topics

SG24-6108 DB2 UDB Server for OS/390 Version 6 Technical Update

SG24-6418 DB2 for z/OS and OS/390 : Squeezing the Most Out of Dynamic SQL

### IDUG Presentation

REXX and DB2 – The Stuff NOT in the Manuals – IDUG North America 2005



**RTFM!**


## David Simpson

Themis Training

[dsimpson@themisinc.com](mailto:dsimpson@themisinc.com)

E16

Using REXX to Build Your Own DB2 Tools



*Please fill out your session  
evaluation before leaving!*