Intro to DB2 UDB Programming Using REXX

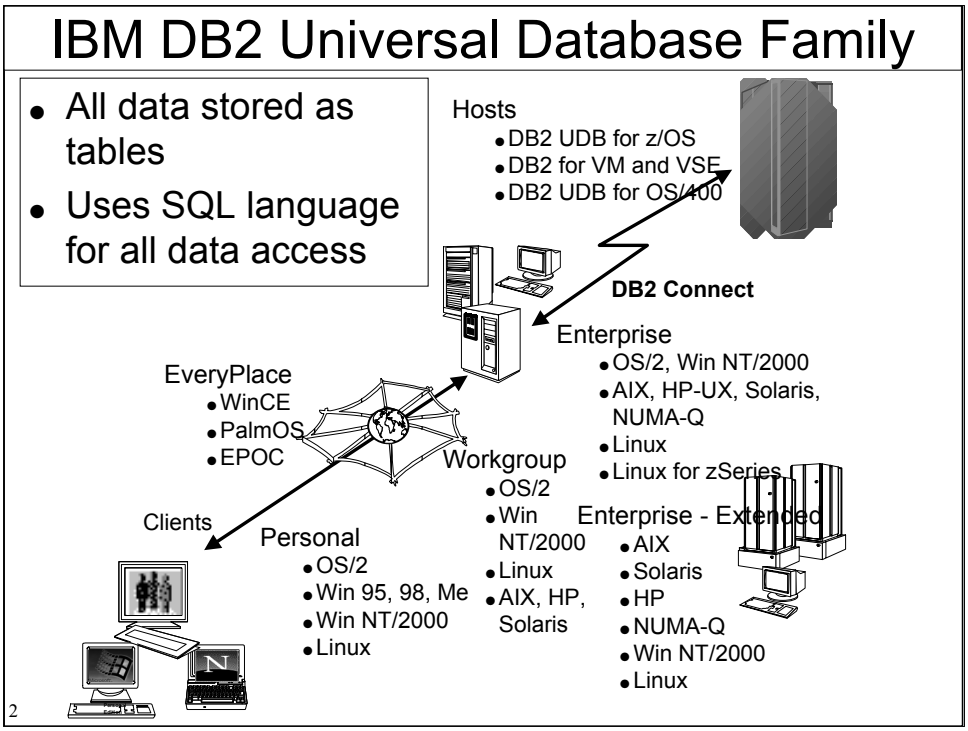
Abstract:

In this session, we overview the various DB2 REXX interfaces available to Win32 and UNIX/Linux platforms with Regina and Object REXX, including the DB2 UDB dynamic SQL interface and the REXX/SQL interface

Bob Stark bstark@protechtraining.com 412-445-8072

1

Notes:

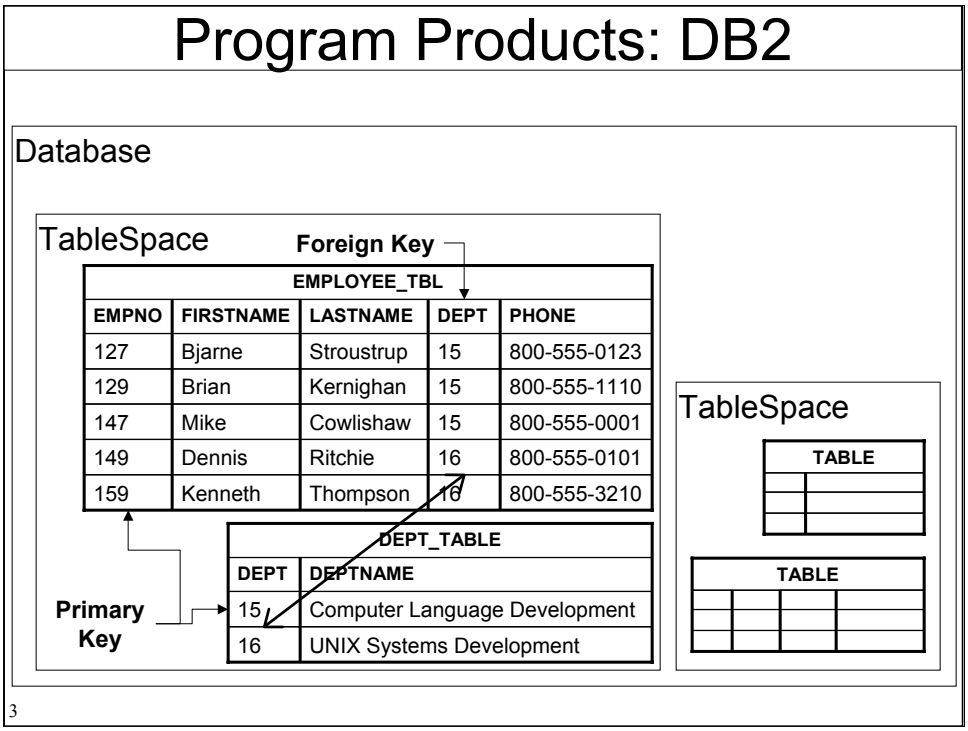


DB2 is a client-server database, so you can install a small DB2 client on your platform, which connects over the network (typically TCP/IP) to DB2 running on another host. The personal editions of DB2 allow you to have a DB2 server and client on the same host for development purposes.

DB2 is a multiple platform product family, with DB2 running on MVS as its premier platform, although the Windows and UNIX implementations are quite capable.

DB2 exploits the advanced features of MVS, such as Parallel Sysplex, data spaces, multiple address space architecture, and other key features to be able to access huge databases quickly, with full integrity and security.

Notes:



DB2 is IBM's relational database facility.

Data is retrieved and updated using the SQL language.

All DB2 data is stored in tables, with rows and columns.

All the data in a given column must be the same data type (text, integer, time, etc.).

Data in one table can be related or joined to data in another table, using the primary key data. Unlike other databases, the relationships are based on the actual data, not by physical pointers such as disk addresses.

No two rows in a table can have the same primary key; each must be unique.

A table space is an area in a database that contains one or more tables. The data is stored by MVS DB2 in VSAM linear datasets.

Notes:

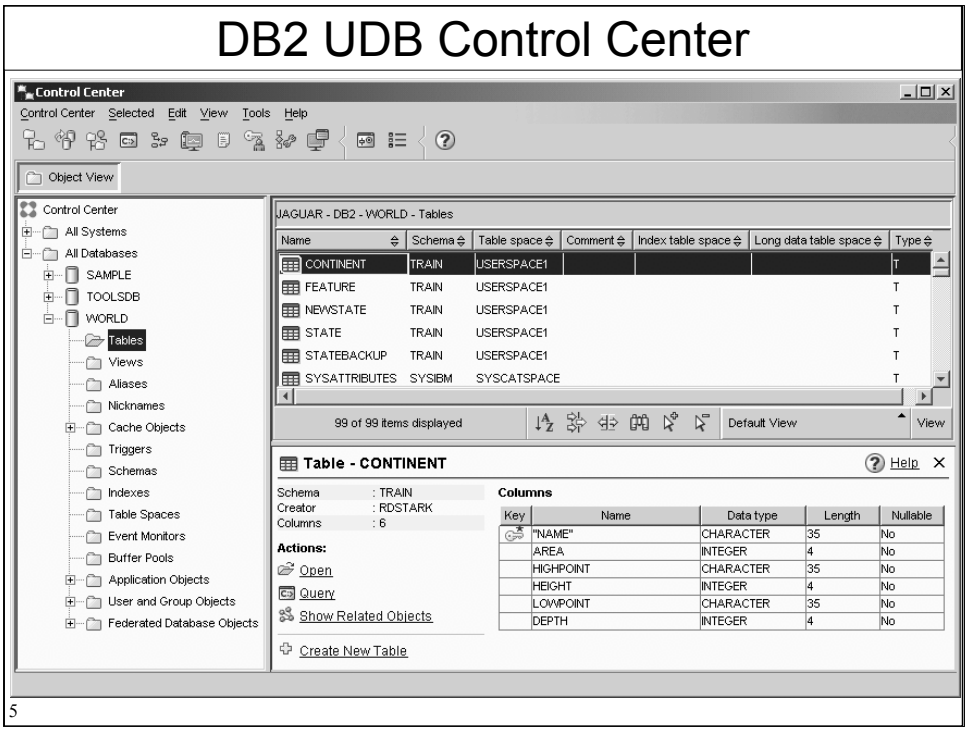
DB2 UDB Application Development Tools

- Control Center
  - Graphical interface to display databases, tables, etc.
  - Use to perform administrative tasks such as configuring the system, managing directories, backing up and restoring data, scheduling jobs
- Command Center
  - Enter DB2 commands and SQL statements in an interactive window, and see the results. You can scroll through the results and save the output to a file
- Script Center
  - Create scripts containing DB2 commands, SQL statements, and operating system commands, which can be stored for later execution
  - You can schedule scripts to run unattended, on a schedule, which is useful for backups
- Journal
  - View information about jobs that are pending execution, executing, or that have completed execution; the recovery history log; the alerts log; and the messages log

4

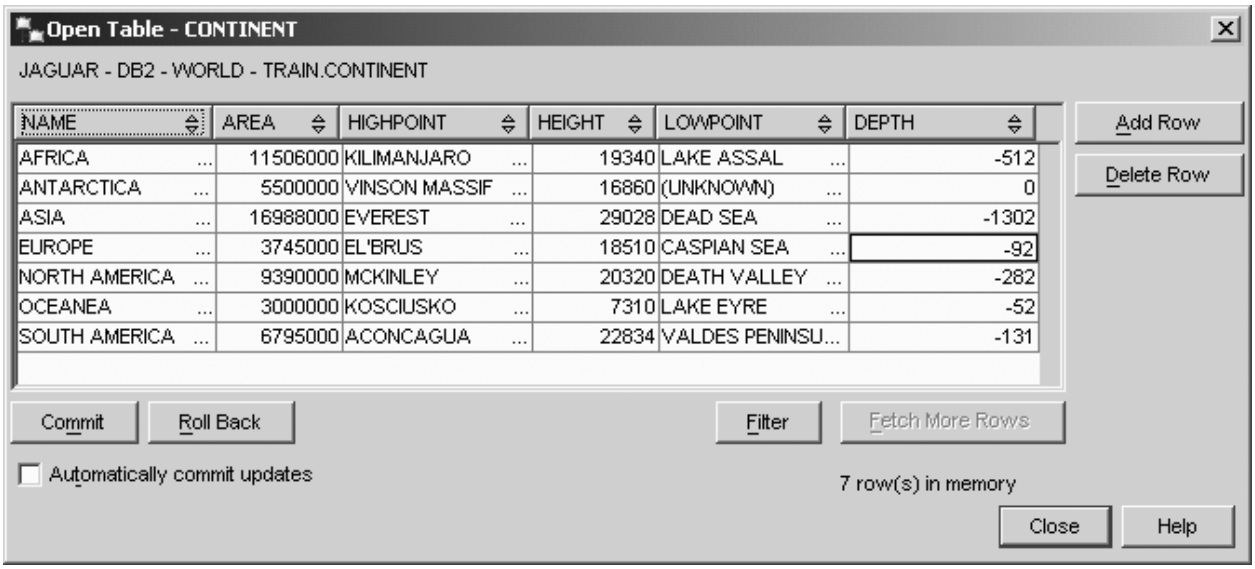
DB2 Universal Database supplies numerous application development tools to simplify writing and testing the SQL statements in your applications, and to help you monitor their performance. Note that not all tools are available on every platform.

Notes:

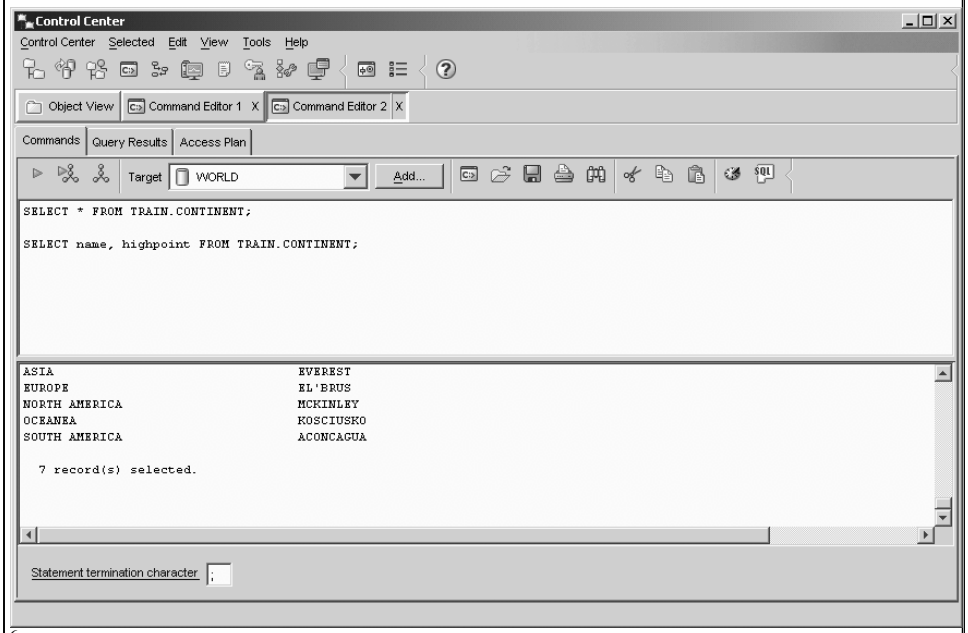


The DB2 Control center, shown above, is a Java Application (which, unfortunately, means that it requires a lot of memory and takes a long time to load). It allows you to navigate to various DB2 systems, display the databases defined to those systems, display the tables and objects in a given database, the characteristics of a given table, and even browse through and edit the data.

When you click "Open", you'll see the view of the selected table shown below.



DB2 UDB Command Center



6

The DB2 Command center, shown above, allows you to interactively enter DB2 commands and SQL statements in a window, and see the results. Just place your cursor on an SQL statement, and press Ctrl-Enter (or click the green arrow). You can scroll through the results and save the output to a file.

There is also a DB2 Command Line Processor and a DB2 Command Window (shown below), which is a custom CMD prompt window where you can run the DB2 command, as well as other commands. It is small and fast. I used it to issue the following command to create the "world" database; the file contains SQL create and insert statements.

```
C:\> db2 -tf World_Creates_And_Inserts_DB2_UDB.sql
```

SQL Programming APIs

- Embedded SQL - DB2 supports C/C++, COBOL, FORTRAN, Java™ (SQLj), and REXX
  - Static SQL - precompile, bind, then compile the language; more efficient; No REXX support
  - Dynamic SQL – SQL is built and executed at run time; more flexible; supports Perl, Java (JDBC), and REXX
    - IBM provides this REXX interface :
    - Non-MVS has not been enhanced since DB2 V5; does not support multi-threading
    - MVS interface first released in DB2 V6; different syntax
- DB2 Call Level Interface (CLI) – DB2 supports C/C++
  - Based on Microsoft® Open Database Connectivity (ODBC)
  - Mark Hessling's REXX/SQL is built on this DB2 API

7

The pre-compiler converts the embedded SQL statements in the source code into DB2 run-time API calls to the database manager. The pre-compiler also produces an access package in the database and, optionally, a bind file.

The access package contains access plans selected by the DB2 optimizer for the static SQL statements in your application. These access plans contain information required by the database manager to optimize the execution of the static SQL statements as determined by the optimizer.

For dynamic SQL statements, the optimizer creates access plans when you run your application.

The bind file contains the SQL statements and other data required to create an access package. You can use the bind file to re-bind your application later without having to pre-compile it first. The re-binding creates access plans that are optimized for current database conditions. You need to re-bind your application if it will access a different database from the one against which it was precompiled. You should re-bind your application if the database statistics have changed since the last binding.

You do not need to pre-compile or bind DB2 CLI applications; they use common access packages provided with DB2. You just compile and link your application.

The MVS DB2 Version 6 Refresh released in May 2000 introduced the MVS DB2 REXX Interface. It has been pretty heavily used by customers and IBM since then.

Notes:

Types of SQL Statements

DML – Data Manipulation Language	Modify data in tables
Select	Retrieve data from one or more tables
Insert	Put one/more new rows into table
Update	Modify the contents of a set of rows
Delete	Remove a set of rows from a table
DDL – Data Definition Language	Add tables, indexes, etc.
Create	Create a new database or database object
Alter	Modify characteristics of an existing database or object
Drop	Delete a database or database object
DCL – Data Control Language	Used to control security
Grant	Grant access to a given database object
Revoke	Remove access to a given object

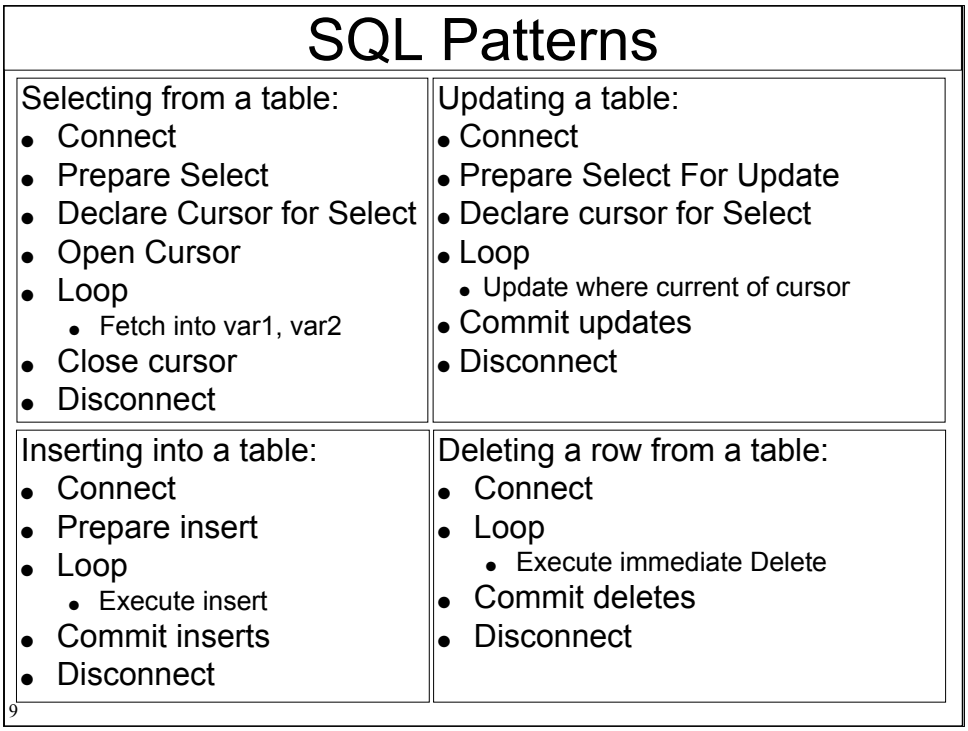
8

SQL statements conform to various levels of statements; the SQL 92 standard came out in 1996, while the SQL 99 standard came out around 2002.

The DML statements are pretty portable from database to database and platform to platform, although certain databases support different proprietary advanced features.

DDL statements are somewhat less portable, but still largely portable between platforms. DCL statements are often the least portable.

Notes:



The basic patterns shown above are generally used in the Create, Retrieve, Update, Delete (C.R.U.D.) operations.

Notes:

DB2 Definitions

- SQL communication area (SQLCA):
  - A structure that is used to provide an application program with information about the execution of its SQL statements
  - The SQLCA is created automatically by the first ADDRESS DSNREXX 'CONNECT' or CALL SQLDBS 'ATTACH TO'
  - The DSNREXX SQLCA is a set of variables (see next slide)
  - The CALL SQLDBS 'ATTACH TO' SQLCA is stem SQLCA.
- SQL descriptor area (SQLDA):
  - A structure that describes input variables, output variables, or the columns of a result table

10

In REXX, the SQLCA and SQLDA are just sets of stem variables.

Notes:

Selected SQLCA Fields

Variable	Value
SQLCODE	The SQL return code
SQLERRMC	Error condition description tokens
SQLERRP	Product signature ('DSN' for DB2 on OS/390) & error diagnostic information
SQLERRD.1	Rows in result table (when cursor at end)
SQLERRD.2	Same as SQLERRD.1 or an internal error code
SQLERRD.3	Number of rows affected by INSERT or UPDATE
SQLERRD.4	Rough estimate of resources required
SQLERRD.5	Position or column of a syntax error for PREPARE or EXECUTE IMMEDIATE statement
SQLERRD.6	Internal error code
SQLWARN.0-SQLWARN.10	SQL Warning messages (data conversion, truncation, data validation error, etc.)
SQLSTATE	Contains return code for most recent SQL statement.

11

This is documented in the SC26-9944 DB2 Universal Database for OS/390 and z/OS SQL Reference manual.

Error Checking using the SQLCA

```
/* REXX Pgm fragment showing DODB2 wrapper function & error checks */
rc = DODB2("EXECSQL EXECUTE S3 USING ",
           ":QUERYNO, :BIND_TIME, :APPLICATION_NAME, ",
           ":PROGNAME, :TB_CREATOR, :TB_NAME, :CARD, :PAGES")
IF rc < 0 THEN SIGNAL EXIT
/* .
   .
*/
EXIT: EXIT 0

/*:DODB2 function: Issue ADDRESS DSNREXX and check return codes */
DODB2:
IF ARG() < 1 THEN
  DO; SAY 'Missing statement to execute'; RETURN -1; END
ADDRESS DSNREXX arg(1)
IF WORDPOS(sqlcode,'0 100') THEN RETURN sqlcode
SAY 'Error, SQLCODE='sqlcode' in statement: 'ARG(1)
SAY 'SQLSTATE='sqlstate', SQLERRMC='sqlerrmc', 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', 'sqlwarn.7',',
              sqlwarn.8', 'sqlwarn.9', 'sqlwarn.10
RETURN sqlcode
```

12

The code fragment shown above does not show the connect or the prepare, just the execute statement. It is a DB2 for MVS example (due to the ADDRESS DSNREXX), and shows the use of wrapper function DODB2() to consolidate the return code checking into one place. In theory, the wrapper could be made portable across platforms as well.

Notes:

Understanding rc *vs.* sqlcode

- Test RC variable after each EXECSQL call
  - Note: This doesn't give any detail about the particular DB2 error

`rc = 0`	No SQL warnings or errors
`rc = +1`	SQL warning occurred
`rc = -1`	SQL error occurred

- Test SQLCODE after each SQL call
  - Note: This does not detect errors in the REXX interface to DB2

`SQLCODE = 0`	Execution was successful
`SQLCODE < 0`	Execution failed
`SQLCODE > 0`	Execution successful, with a warning
`SQLCODE 100`	Indicates no (more) data was found (EOF)

  - The meaning of SQLCODEs other than 0 and 100 varies with the particular product implementing SQL

13

Another variation between MVS and Windows/UNIX; On non-MVS platforms, we use SQLCA.SQLCODE to reference the SQLCODE variable.

Notes:

More DB2 Definitions

- Host Variable
  - A REXX variable that is referenced by the embedded SQL
  - Host variables must be in upper case and prefixed with : when referencing them from within SQL; for example:

```
Call SQLEXEC 'FETCH c1 INTO',
   ':NAME, :AREA, :HIGHPOINT, :HEIGHT, :LOWPOINT, :DEPTH'
```

- Cursor
  - Enables application to process sets of rows, one at a time
  - Each row is fetched into corresponding host variables
  - A program can have multiple active cursors
  - Cursors *must* be closed at program termination
- NULL
  - A special value that indicates the absence of information
  - SQL null is not the same as REXX `''`

14

Nulls are a key SQL concept that must be dealt with in your query, or in your code when you are processing the results of your query. It is much easier to deal with nulls when designing the SQL query.

Notes:

Using an SQL Cursor

The basic steps in using a cursor are:

1. Execute a DECLARE CURSOR statement to define the result table on which the cursor operates
2. Execute an OPEN CURSOR to make the cursor available to the application
3. Specify what the program does when all rows have been retrieved
4. Execute FETCH statements to get one row of data
5. Execute cursored SQL statements:
   UPDATE to update an existing row
   DELETE to delete matching row
6. Execute a CLOSE CURSOR statement to make the cursor unavailable to the application

15

Notes:

Pre-defined SQL Cursors

REXX SQL applications must use a predefined set of cursors:

c1 to c100

- Cursor names for DECLARE CURSOR, OPEN, CLOSE, and FETCH
- By default, c1 to c100 defined with the WITH RETURN clause; c51 to c100 defined with the WITH HOLD clause
- You can modify attributes: e.g. add attributes to make cursor scrollable

c101 to c200

- These are defined in MVS UDB only
- Cursors for ALLOCATE, DESCRIBE, FETCH, and CLOSE statements
- Used to retrieve result sets in programs that call stored procedures

s1 to s100

- Prepared statement names for DECLARE STATEMENT, PREPARE, DESCRIBE, and EXECUTE statements

Note: Do not use any of the predefined names as a host variable

16

Notes:

Example using cursor, SQLDA (1 of 2)

```
/* Below is the query we are going to execute */
sel = "SELECT * FROM TRAIN.CONTINENT ORDER BY NAME"

/* First, we must prepare the dynamic SQL statement */
Call SQLEXEC 'PREPARE s1 FROM :sel'
If sqlca.sqlcode <> 0 Then
  Do; Say 'Prepare failed:' sqlca.sqlcode sqlmsg; Signal Exit; End

/* Declare a cursor that will be used to loop through the results */
Call SQLEXEC 'DECLARE c1 CURSOR FOR s1'
If sqlca.sqlcode <> 0 Then
  Do; Say 'Declare failed:' sqlca.sqlcode sqlmsg; Signal Exit; End

Call SQLEXEC 'OPEN c1'      /* Now, open the cursor */
If sqlca.solcode = 0 Then
  Do; Say 'Open c1 failed:' sqlca.sqlcode sqlmsg; Signal Exit; End

Say Left('Name',15) Right('Area',9) Left('Highpoint',15),
    Right('Height',9) Left('Lowpoint',15) Right('Depth',9)
Say Copies('-',15) Copies('-',9) Copies('-',15),
    Copies('-',9) Copies('-',15) Copies('-',9)
```

17

You can see host variables at work here on the prepare statement.

Notes:

Example using cursor, SQLDA (2 of 2)

```
  Do i = 1 by 1 while (SQLCA.SQLCODE = 0)

  Call SQLEXEC 'FETCH c1 INTO',
   ':NAME, :AREA, :HIGHPOINT, :HEIGHT, :LOWPOINT, :DEPTH'

    Select
    When (SQLCA.SQLCODE = 0) Then
      Say Left(name,15) Right(area,9) Left(highpoint,15),
          Right(height,9) Left(lowpoint,15) Right(depth,9)
    When (SQLCA.SQLCODE = 100) Then Nop  /* Last record already
   found */
    Otherwise
      Say 'Fetch sqlcode:' sqlca.sqlcode sqlmsg
    End /* Select */
  End i

Call SQLEXEC 'CLOSE c1'
If sqlca.solcode = 0
Then Say 'Cursor c1 did not close' sqlca.sqlcode sqlmsg
```

18

```
W:\>rexx dselect1.rex
Name                Area Highpoint      Height Lowpoint          Depth
--------------- -------- ------------- ------- --------------- -------
AFRICA          11506000 KILIMANJARO     19340 LAKE ASSAL         -512
ANTARCTICA       5500000 VINSON MASSIF   16860 (UNKNOWN)             0
ASIA            16988000 EVEREST         29028 DEAD SEA          -1302
EUROPE           3745000 EL'BRUS         18510 CASPIAN SEA         -92
NORTH AMERICA    9390000 MCKINLEY        20320 DEATH VALLEY        -282
OCEANEA          3000000 KOSCIUSKO        7310 LAKE EYRE           -52
SOUTH AMERICA    6795000 ACONCAGUA       22834 VALDES PENINSUL     -131
```

Notes:

DB2 UDB Dynamic SQL Interface

- Advantages:
  - Installed with DB2 (built-in)
  - Supported, documented by IBM
- Disadvantages:
  - Not supported by any other DBMS; not portable
  - Not thread-safe
  - Functionally stabilized; no new features

19

This is the interface that is built into DB2 UDB.

Defining the Interface

```
rc = AddFuncs()   /* Add REXX function packages */
If rc <> 0 Then
  Do; Say rc; Signal Exit; End

/*-- snip, snip-------------------------- Many lines not shown */

Exit: Exit 0 /* Common exit point for errors */

AddFuncs: /* Add REXX function packages if not already defined */
Trace N
If RxFuncQuery('SysLoadFuncs') Then
  If rxfuncadd( 'SysLoadFuncs', 'RexxUtil', 'SysLoadFuncs' )
  Then Return "Unable to load RexxUtil functions"
  Else Call SysLoadFuncs

If RxFuncquery('SQLDBS') Then
  If RxFuncAdd('SQLDBS','DB2AR','SQLDBS')
  Then Return "Unable to load SQLDBS function"

If RxFuncquery('SQLDB2') Then
  If RxFuncAdd('SQLDB2','DB2AR','SQLDB2')
  Then Return "Unable to load SQLDB2 function"

If RxFuncquery('SQLEXEC') Then
  If RxFuncAdd('SQLEXEC','DB2AR','SQLEXEC')
  Then Return "Unable to load SQLEXEC function"

Return 0        /*End of REXX function registration.*/
```

20

Every DB2 UDB program needs a big blob of RxFuncAdd calls, as shown above. This is an internal function, you could also use an external function.

Notes:

Inserting Data Using ? Markers

```
sv = "INSERT INTO TRAIN.CONTINENT",
        "(NAME, AREA, HEIGHT, HIGHPOINT, DEPTH, LOWPOINT)",
        "VALUES (?, ?, ?, ?, ?, ?)"

/* First, we must prepare the dynamic SQL statement */
Call SQLEXEC 'PREPARE s2 FROM :sv'
If sqlca.sqlcode <> 0 Then
  Do; Say 'Prepare failed:' sqlca.sqlcode sqlmsg; Signal Exit; End

name = 'ATLANTIS'
area  = 1
height = 0
highpoint = 'SEA LEVEL'
depth = -20000
lowpoint = 'SEA FLOOR'

Call SQLEXEC 'EXECUTE s2',
  'USING :name, :area, :height, :highpoint, :depth, :lowpoint'
If sqlca.sqlcode <> 0 Then
  Do; Say 'Execute failed:' sqlca.sqlcode sqlmsg; Signal Exit; End

call SQLEXEC 'COMMIT'
If sqlca.sqlcode <> 0 Then
  Do; Say 'Commit failed:' sqlca.sqlcode sqlmsg; Signal Exit; End
```

21

You can also code the actual data in the values clause, in quotes. However, it gets quite tricky if the data itself has quotes.

The place markers don't scale up well; when you are inserting a row which contains 100 columns, the ? marker becomes quite messy!

Notes:

Deleting Data

```
su = "DELETE FROM TRAIN.CONTINENT",
       "WHERE NAME = 'ATLANTIS'"
Call SQLEXEC 'EXECUTE IMMEDIATE :su'
If sqlca.sqlcode <> 0 Then
  Do; Say 'Execute failed:' sqlca.sqlcode sqlmsg; Signal Exit;
  End

call SQLEXEC 'COMMIT'
If sqlca.sqlcode <> 0 Then
  Do; Say 'Commit failed:' sqlca.sqlcode sqlmsg; Signal Exit;
  End
```

22

You can also delete using an SQL cursor; which uses the "WHERE CURRENT OF C1 syntax, where C1 represents a cursor that we would open.

Notes:

Using the REXX/SQL Interface

- Advantages:
  - Supports multiple platforms and DBMS; portable
  - Laves development version is now thread-safe
  - Fairly mature; Open Source development continues
- Disadvantages:
  - Requires an additional end-user install
  - Limited Support (Mark does need to sleep!)

23

This is Mark Hessling's REXX Interface.

Installing Rexx/SQL on Windows

- Unpack the archive
- Copy all executables and DLLs to a new or existing directory already in the system PATH, or to a new directory
  - Remember to include any new directory in the PATH.

```
mkdir c:\programs\rexxsql
cd    c:\programs\rexxsql
pkzipc rexxsql.zip
```

- Go to Windows Control Panel, select Administrative tools; data sources (ODBC)
- Click Add, select the database type to add
- Once you have the Rexx/SQL executables and DLLs (or shared libraries) , installed, run the sample program; simple.cmd. simple.cmd takes 3
- optional parameters:
  - Username to connect to the database
  - password for username
  - database=dbName

24

This is pretty straightforward, but there is no setuo.exe.

Installing Rexx/SQL on Windows

- Next run simple.cmd directly (using regina as the example)

```
Unix>  regina simple.cmd user=user pass=pswd data=database
OS/2>  regina simple.cmd user=user pass=pswd data=database
Win32> regina simple.cmd user=user pass=pswd data=database
```

- If you have Object Rexx and Regina on the same machine, you need to tell Rexx/Trans (which is what allows Rexx/SQL to use multiple interpreters) to not use the default of Regina

```
set REXXTRANS_INTERPRETER=objectrexx
```

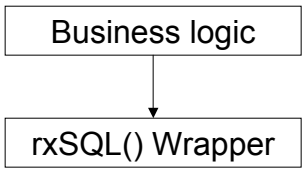
- To use multiple databases simultaneously, you can use environment-unique DLLS which bind to environment-specific REXX function names. For example:
- ODBCConnect is unique to ODBC, defined in rexxodbc.dll
- SQLConnect is generic, defined in rexxsql.dll

25

Notes:

REXXSQL Wrapper

- The RxSQL() wrapper handles data validation, errors, and thread management



26

rxSQL is an internal REXX function, so that variables can be passed into and returned by it.

```
/*RXCOPY RXSQL NODUP 100 LINES COPIED ON 10-27-04 AT 16:56************/
/*Start of RxSql-[rxCopy into EACH method w/ Database]--Version-01.05-*/
/*:RxSQL() function: Calls REXX/SQL and performs error checks         */
/* Example 1:                                                        */
/* rc = RxSQL('FETCH', 'S4')                                         */
/* Example 2, showing a data types and binds variable list:          */
/* rc = rxSQL('EXECUTE', 's12', 'CHAR', 'CB5-3', 'INTEGER', '73')    */
/*-------------------------------------------------------------------*/
RxSQL: Trace N

Parse Source . . _rxSqlpgm
_rxSqlValid = "CONNECT  DISCONNECT DEFAULT  COMMAND PREPARE  DISPOSE",
              "OPEN     CLOSE      FETCH    GETDATA EXECUTE  COMMIT ",
              "ROLLBACK DESCRIBE   VARIABLE GETINFO"
If WordPos(Translate(Arg(1)), _rxSqlValid) = 0 Then
  Return '-1 RxSQL Syntax error,' ARG(1) 'not one of' _rxSqlValid'.',
            'Called from line' sigl 'in' _rxSqlpgm
_rxSqlFunc = "SQL"Arg(1)

If 0 Then SAY 'Process ID:' SysQueryProcess('PID'),
              'Thread ID:' SysQueryProcess('TID') ARG(1)

/*-------------------------------------------------------------------*/
/* When RxSql is called in a multi-threaded environment, we must call */
/* SQLLoadFuncs() in each thread, so it gets the right instance data. */
/* We use a local environment object to track whether we have        */
/* initialized this thread.                                          */
/*-------------------------------------------------------------------*/
Signal Off NoString
Signal Off Any
If .nil = .local['rxsqltids'] Then .local['rxsqltids'] = ''
If WordPos(SysQueryProcess('TID'), .local['rxsqltids']) = 0 Then
```

DB2 Isolation Levels

The DB2 REXX interface provides four packages for accessing DB2, each with a different isolation level

- This controls the DB2 locks held for an application
- To change the isolation level for SQL statements:

```
"EXECSQL SET CURRENT  PACKAGESET='DSNREXCS'"
```

MVS Package	non-MVS Package	Isolation level
DSNREXRR	DB2ARXRR.BND	Repeatable read (RR)
DSNREXRS	DB2ARXRS.BND	Read stability (RS)
DSNREXCS	DB2ARXCS.BND	Cursor stability (CS)
DSNREXUR	DB2ARXUR.BND	Uncommitted read (UR)

27

Conceptual background: Concurrency must be controlled to prevent lost updates and undesirable effects as unrepeatable reads and access to uncommitted data.

Lost updates. Without concurrency control, two processes, A and B, might both read the same row from the database, and both calculate new values for one of its columns, based on what they read. If A updates the row with its new value, and then B updates the same row, A's update is lost.

Access to uncommitted data. Also without concurrency control, process A might update a value in the database, and process B might read that value before it was committed. Then, if A's value is not later committed, but backed out, B's calculations are based on uncommitted (and presumably incorrect) data.

Unrepeatable reads. Some processes require the following sequence of events:
A reads a row from the database and then goes on to process other SQL requests. Later, A reads the first row again and must find the same values it read the first time. Without control, process B could have changed the row between the two read operations.

DB2ARXNC.BND Supports the "no commit" isolation level. This isolation level is used when working with AS/400 (iSeries) database servers. On other databases, it behaves like the uncommitted read isolation level.

When you use the SQLEXEC routine, the cursor stability package is used by default. If you require one of the other isolation levels, you can change isolation levels with
SQLDBS CHANGE SQL ISOLATION LEVEL
before connecting to the database. This will cause subsequent calls to the SQLEXEC routine to be associated with the specified isolation level.

Notes:

MVS DB2 Dynamic SQL Interface

28

Example using cursor, SQLDA

```
/* REXX fragment to retrieve DB2 table using an SQLDA */
sqlstmt= ,
'SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,' ,
' WORKDEPT, PHONENO, HIREDATE, JOB,' ,
' EDLEVEL, SEX, BIRTHDATE, SALARY,' ,
' BONUS, COMM' ,
' FROM EMP'
ADDRESS DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
ADDRESS DSNREXX "EXECSQL PREPARE S1 INTO :OUTSQLDA FROM
   :SQLSTMT"
ADDRESS DSNREXX "EXECSQL OPEN C1"
  DO UNTIL sqlcode <> 0
  ADDRESS DSNREXX "EXECSQL FETCH C1 USING DESCRIPTOR :OUTSQLDA"
  IF sqlcode = 0 THEN
    DO
    line = ''
      DO i = 1 To outsqlda.sqld
      line = line outsqlda.i.sqldata
      End i
    SAY line
    END
  END
```

29

Notes:

Embedding SQL into MVS REXX

- Use the following to embed SQL into your program:
  `ADDRESS DSNREXX "EXECSQL` *`sql statement`*`"`
- The REXX interface supports all DB2 SQL, except:
  - BEGIN DECLARE SECTION
  - DECLARE STATEMENT
  - END DECLARE SECTION
  - INCLUDE
  - SELECT INTO
  - WHENEVER
- You cannot include REXX comments (/* ... */) or SQL comments (--) within SQL statements
- Use the normal REXX rules to continue long SQL statements

30

Notes:

DB2 REXX: ADDRESS DSNREXX

- MVS DB2 provides ADDRESS DSNREXX environment
  - Note: There are also function interfaces:
    - CALL SQLDBS
    - CALL SQLEXEC
- Must define via the TSO/E RXSUBCOM() function
  - RXSUBCOM() adds or deletes DSNREXX to the REXX host command environment table
- Syntax:

```
RXSUBCOM(    'ADD'          , 'DSNREXX' ,    'DSNREXX' )
             'DELETE'
```

31

Notes:

DB2 REXX: Defining DSNREXX

```
/* REXX exec to define the ADDRESS DSNREXX environment      */
ssid = 'DSNT'         /* Desired DB2 Subsystem  id          */
'SUBCOM DSNREXX'      /* Is DSNREXX environment available?  */
IF rc = 1             /* IF NOT, MAKE IT AVAILABLE */
THEN rc = RXSUBCOM('ADD','DSNREXX','DSNREXX')
ADDRESS DSNREXX  /* Change default environment to DSNREXX  */

ADDRESS DSNREXX 'CONNECT' ssid   /* Connect to DB2         */

/* Call EXECSQL, and DISCONNECT interfaces here...         */

/* REMOVE DSNREXX WHEN DONE                                */
rc = RXSUBCOM('DELETE','DSNREXX','DSNREXX')
```

32

Notes:

DB2 REXX: ADDRESS DSNREXX

ADDRESS DSNREXX "CONNECT *subsystem*"

- Connects the REXX procedure to a DB2 subsystem. You must execute CONNECT before you can execute SQL statements
- Note: CALL SQLDBS 'ATTACH TO' *ssid* is equivalent to ADDRESS DSNREXX 'CONNECT' ssid.

ADDRESS DSNREXX "EXECSQL *SQL-statement*"

- Executes SQL statements in REXX procedures
- Note: CALL SQLEXEC is equivalent to ADDRESS DSNREXX EXECSQL

ADDRESS DSNREXX 'DISCONNECT'

- Disconnects the REXX procedure from a DB2 subsystem
- You should execute DISCONNECT when finished to release DB2 resources (memory, threads, locks, etc.)
- Note: CALL SQLDBS 'DETACH' is equivalent to ADDRESS DSNREXX DISCONNECT

33

Notes:

Writing a REXX stored procedure

- REXX stored procedures follow the same rules as other languages
- A stored procedure:
  - Receives input parameters
  - Executes REXX commands and SQL statements
  - Returns at most one output parameter
- Differences between REXX stored procedures and regular execs:
  - Connect statements are not used (DB2 establishes the connection for you)
  - On MVS, REXX stored procedures run in a WLM-established stored procedures address space

34

Notes:

Additional DSNREXX Information

DB2 Universal Database for OS/390 and z/OS Bookshelf

- SC26-9944 - SQL Reference
- SC26-9933 - Application Programming and SQL Guide

Redbooks:

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

35

Notes:

Third-Party MVS DB2 REXX Interfaces

36

Third-party MVS REXX-DB2 Interfaces

- MAX/REXX
  - REXX interface to VSAM, SAM, PDS and DB2 data. Uses SQL syntax to access DB2 data

    MAX Software LLC Littleton,Colorado, USA 80123
    303-933-5827 http://www.max2000.com
- CA-ProBuild
  - Embed SQL directly in REXX exec. Produces ISPF table containing results
- RLX/SQL
  - REXX Interface to DB2 (Dynamic and Static SQL)

    Relational Language Architects
    www.relarc.com
- REXXTOOLS/MVS
  - REXX Interface to VSAM, DB2 (Dynamic and Static SQL), QSAM, BPAM, IDCAMS. WEB and TCP/IP support

    Open Software Technologies, Inc. Longwood, Florida 32779 USA
    407-788-7173 http://www.open-softech.com

37

Notes:

MAX/REXX Example Page 1 of 2

```
/* REXX ***************************************************/
/* DOC: Lists tables defined to a specific DB2 subsystem */
/* ************************************************** */
                                    /* Connect to DB2 */
IF RXSQL("CONNECT DB2P")<>0 THEN CALL CLEANUP
/* ************************************************** */
/* List name, dbname, tsname, from sysibm.systables      */
/* ************************************************** */
RC=RXSQL("DECLARE C1 CURSOR FOR",
  "SELECT NAME, DBNAME, TSNAME FROM SYSIBM.SYSTABLES",
  "ORDER BY DBNAME, TSNAME, NAME")
  IF RC<>0 THEN CALL CLEANUP              /*             */
IF RXSQL("OPEN C1")<>0 THEN CALL CLEANUP

  DO WHILE RXSQL("FETCH C1")=0      /* Fetch rows  */
  SAY 'DBNAME='SUBSTR(DBNAME,1,12), /* List columns*/
     'TSNAME='SUBSTR(TSNAME,1,12),  /*             */
     'NAME='NAME                    /*             */
  END                               /*             */
IF SQLCODE<>100 THEN CALL CLEANUP
```

38

Notes:

MAX/REXX Example Page 2 of 2

```
RC=RQSQL("CLOSE C1")            /* Close CURSOR          */
                                /*                       */
CALL CLEANUP                    /* Clean up & exit       */

/* ************************************************** */
/* Clean up and exit                                  */
/* ************************************************** */
CLEANUP:
  IF SQLCODE<>0 THEN DO                  /*             */
     SAY 'RXSQL failed RC='SQLCODE       /*             */
     SAY SQLMSG                          /*             */
  END                                    /*             */
  CALL "RXSQL" "DISCONNECT"       /* Disconnect from DB2 */
  EXIT 0                                 /*             */
RETURN
```

39

Notes:

This page intentionally left blank