136

DB2

February 2004

update

In this issue

DB2 Update

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690
Fax: 214-341-7081

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Nicole Thomas
E-mail: nicole@xephon.com

Subscriptions and back-issues

A year's subscription to *DB2 Update*, comprising twelve monthly issues, costs $380.00 in the USA and Canada; £255.00 in the UK; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 2000 issue, are available separately to subscribers for $33.75 (£22.50) each including postage.

DB2 Update on-line

Code from *DB2 Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at http://www.xephon.com/db2; you will need to supply a word from the printed issue.

Contributions

When Xephon is given copyright, articles published in *DB2 Update* are paid for at the rate of $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

Printed in England.

Homemade replication

Core banking applications run on OS/390 DB2 V6.1 in our environment. When a new branch application project development is started, it will use DB2/UDB V7.2 for its local data. But some core banking data is also needed for branch applications, for example personnel account information for local authentication.

We had no replication tool in our shop, so we created our real-time replication tool on base tables. After the initial synchronization, we captured changes (insert/update/delete) on the original table with the help of triggers. Triggers call WLM-managed SQL stored procedures, which connect to the remote location and execute the same DML. Connection is made via DRDA and TCP/IP to the remote location.

Table definition:

```
   CREATE TABLE TDB2.DB2_TTRIG
  (SICIL                INTEGER
       NOT NULL WITH DEFAULT,
   UNVAN                CHAR    (8)
       NOT NULL WITH DEFAULT,
   FLAG                 CHAR    (1)
       NOT NULL WITH DEFAULT,
   ZAMAN                TIMESTAMP
       NOT NULL WITH DEFAULT) ;
```

Tables created on both sides are identical.

Remote location definitions look like this:

```
SYSIBM.LOCATIONS :

     LOCATION  LINKNAME  PORT

     TESTDB    TESTDB    60000
```

TESTDB is a DB2/UDB database name containing table TDB2.DB2_TTRIG.

Port number of TESTDB is 60000.

```
SYSIBM.IPNAMES :
```

```
    LINKNAME  SECURITY_OUT  USERNAMES  IPADDR

    TESTDB    P             O          172.16.4.150
```

The IP number of the server containing the TESTDB database is 172.16.4.150.

```
SYSIBM.USERNAMES :

    Y    AUTHID    LINKNAME  PASSWORD

    O    SDBA1     TESTDB    xxxxx
    O    STCUSR    TESTDB    yyyyy
```

SDBA1 is the BIND owner.

STCUSR is the stored procedure address space started task user.

The users mentioned above are defined on DB2/UDB with passwords *xxxxx* and *yyyyy*.

TRIGGER DEFINITION

```
//STEP1    EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DB0T)
 RUN PROGRAM(DSNTEP2) PLAN(DSNTEP61) LIB('DSN610.RUNLIB.LOAD') -
  PARMS('SQLTERM($)')
//SYSIN    DD *
CREATE TRIGGER TTDB202 AFTER INSERT ON TDB2.DB2_TTRIG REFERENCING NEW N
 FOR EACH ROW MODE DB2SQL
 BEGIN ATOMIC
  CALL DB2L002(N.SICIL,N.UNVAN,N.FLAG,N.ZAMAN) ;
 END
$
--
CREATE TRIGGER TTDB203 AFTER UPDATE ON TDB2.DB2_TTRIG REFERENCING NEW N
 OLD O FOR EACH ROW MODE DB2SQL
 BEGIN ATOMIC
  CALL DB2L003(N.SICIL,N.UNVAN,N.FLAG,N.ZAMAN,O.SICIL) ;
 END
$
--
```

```
CREATE TRIGGER TTDB2Ø4 AFTER DELETE ON TDB2.DB2_TTRIG REFERENCING OLD O
 FOR EACH ROW MODE DB2SQL
 BEGIN ATOMIC
  CALL DB2LØØ4(O.SICIL);
 END
$
```

TTDB202 is for capturing inserts. *TTDB203* is for capturing updates. *TTDB204* is for capturing deletes.

STORED PROCEDURE DEFINITION

DB2L002 (for insert)

```
CREATE PROCEDURE SYSPROC.DB2LØØ2
   (IN H_SICIL INTEGER,
    IN H_UNVAN CHAR(8),
    IN H_FLAG  CHAR(1),
    IN H_ZAMAN TIMESTAMP)
    LANGUAGE SQL
    MODIFIES SQL DATA
    COLLID CDDB299
    EXTERNAL NAME 'DB2LØØ2'
    WLM ENVIRONMENT DB3TWLM2
    ASUTIME NO LIMIT
    STAY RESIDENT YES
    RESULT SETS 2
-----------------------------------------------------------------
-- SQL STORED PROCEDURE
-----------------------------------------------------------------
BEGIN
CONNECT TO TESTDB;
INSERT INTO TDB2.DB2_TTRIG (SICIL,UNVAN,FLAG,ZAMAN)
       VALUES(H_SICIL,H_UNVAN,H_FLAG,H_ZAMAN);
END
```

DB2L003 (for update)

```
CREATE PROCEDURE SYSPROC.DB2LØØ3
   (IN N_SICIL INTEGER,
    IN N_UNVAN CHAR(8),
    IN N_FLAG  CHAR(1),
    IN N_ZAMAN TIMESTAMP,
    IN O_SICIL INTEGER)
    LANGUAGE SQL
    MODIFIES SQL DATA
    COLLID CDDB299
```

```
    EXTERNAL NAME 'DB2LØØ3'
    WLM ENVIRONMENT DB3TWLM2
    ASUTIME NO LIMIT
    STAY RESIDENT YES
    RESULT SETS 2
-----------------------------------------------------------------
-- SQL STORED PROCEDURE
-----------------------------------------------------------------
BEGIN
CONNECT TO TESTDB;
UPDATE TDB2.DB2_TTRIG SET (SICIL,UNVAN,FLAG,ZAMAN) =
                          (N_SICIL,N_UNVAN,N_FLAG,N_ZAMAN)
 WHERE SICIL = O_SICIL;     END
```

DB2L003 (for delete)

```
CREATE PROCEDURE SYSPROC.DB2LØØ4
    (IN O_SICIL INTEGER)
    LANGUAGE SQL
    MODIFIES SQL DATA
    COLLID CDDB299
    EXTERNAL NAME 'DB2LØØ4'
    WLM ENVIRONMENT DB3TWLM2
    ASUTIME NO LIMIT
    STAY RESIDENT YES
    RESULT SETS 2
-----------------------------------------------------------------
-- SQL STORED PROCEDURE
-----------------------------------------------------------------
BEGIN
CONNECT TO TESTDB;
DELETE FROM TDB2.DB2_TTRIG WHERE SICIL = O_SICIL;
END
```

DB2L002 is for applying inserts. *DB2L003* is for applying updates. *DB2L004* is for applying deletes.

These stored procedures are WLM managed and their application environment is DB3TWLM2.

These stored procedures must bind with location information:

```
BIND PACKAGE(TESTDB.CDDB299) MEMBER(DB2LØØn) OWNER(SDBA1)
```

Serdar Sabri Özkubulay
DB2 Systems Programmer
Akbank (Turkey)

Sequence objects and identity columns

When designing DB2 databases, a frequently-asked request is for a column that contains sequentially-generated numbers. For example, each row has a counter associated with it. When a new row is inserted, the counter should be incremented by 1 for the new row. This way, each new DB2 row has a unique 'row number' associated with it. Until recently, such a design was difficult to deliver.

Without sequence objects or identity columns, an application program can implement similar functionality, but usually not in a manner that performs adequately as database usage scales. A common technique is to maintain a one-row table that contains the sequence number. Each transaction locks that table, increments the number, and then commits the change to unlock the table. In this scenario only one transaction at a time can increment the sequence number. A variation uses something like this:

```
SELECT MAX()+ 1
FROM   ONEROW_TABLE
WITH RR;
```

The result is the next highest number to be used. This value is used by the application and ONEROW_TABLE must be updated with the incremented value. Performance bottlenecks will occur with this method when a lot of concurrent usage is required.

But now DB2 offers two methods of automatically generating sequential numbers for a column:

- Identity columns
- SEQUENCE objects.

IDENTITY COLUMNS

Identity columns were formally added to DB2 as of Version 7, but were actually available as of the DB2 Version 6 refresh. The

identity property is applied to a DB2 column using the IDENTITY parameter. A column defined in this way will cause DB2 to automatically generate a sequential value for that column when a row is added to the table. For example, identity columns might be used to generate primary key values or a value that somewhat mimics Oracle's row number capability. Using identity columns helps to avoid some of the concurrency and performance problems that can occur when application programs are used to populate sequential values for a 'counter' column.

When inserting data into a table that uses an identity column, the program or user will not provide a value for the identity column. Instead, DB2 automatically generates the appropriate value to be inserted.

Only one identity column can be defined per DB2 table. Additionally, the data type of the column must be SMALLINT, INTEGER, or DECIMAL with a zero scale, that is DECIMAL(*n*,0). The data type also can be a user-defined DISTINCT type based on one of these numeric data types. The designer has control over the starting point for the generated sequential values, and the number by which the count is incremented.

An example creating a table with an identity column follows:

```
CREATE TABLE EXAMPLE
   (ID_COL INTEGER NOT NULL
           GENERATED ALWAYS AS IDENTITY
           START WITH 100
           INCREMENT BY 10
    ...);
```

In this example, the identity column is named ID_COL. The first value stored in the column will be 100 and subsequent INSERTs will add 10 to the last value. So the identity column values generated will be 100, 110, 120, 130, and so on.

Note, too, that each identity column has a property associated with it assigned using the GENERATED parameter. This parameter indicates how DB2 generates values for the column. You must specify GENERATED if the column is to be considered an identity column or the data type of the column is a ROWID.

This means that DB2 must be permitted to generate values for all identity columns. There are two options for the GENERATED parameter – ALWAYS and BY DEFAULT:

- GENERATED ALWAYS indicates that DB2 will always generate a value for the column when a row is inserted into the table. You will usually specify ALWAYS for your identity columns unless you are using data propagation.
- GENERATED BY DEFAULT indicates that DB2 will generate a value for the column when a row is inserted into the table unless a value is specified. So, if you want to be able to insert an explicit value into an identity column, you must specify GENERATED BY DEFAULT.

Additionally, you can specify what to do when the maximum value is hit. Specifying the CYCLE keyword will cause DB2 to begin generating values from the minimum value all over again. Of course, this can cause duplicate values to be generated and should be used only when uniqueness is not a requirement.

Actually, the only way to ensure uniqueness of your identity columns is to create a unique index on the column. The IDENTITY property alone will not guarantee uniqueness.

Sometimes it is necessary to retrieve the value of an identity column immediately after it is inserted. For example, if you are using identity columns for primary key generation you may need to retrieve the value to provide the foreign key of a child table row that is to be inserted after the primary key is generated. DB2 provides the IDENTITY_VAL_LOCAL() function, which can be used to retrieve the value of an identity column after insertion. For example, you can run the following statement immediately after the INSERT statement that sets the identity value:

```
VALUES IDENTITY_VAL_LOCAL() INTO :IVAR;
```

The host variable IVAR will contain the value of the identity column.

Problems with identity columns

Identity columns can be useful, depending on your specific

needs, but the problems that accompany identity column are numerous. Some of these problems include:

- Handling the loading of data into a table with an identity column defined as GENERATED BY DEFAULT. The next identity value stored by DB2 to be assigned may not be the correct value that should be generated. This can be especially troublesome in a testing environment.
- LOAD INTO PART *x* is not allowed if an identity column is part of the partitioning index.
- What about environments that require regular loading and reloading (REPLACE) for testing? The identity column will not necessarily hold the same values for the same rows from test to test.
- Prior to V8, it was not possible to change the GENERATED parameter (such as from GENERATED BY DEFAULT to GENERATED ALWAYS).
- The IDENTITY_VAL_LOCAL() function returns the value used for the last insert to the identity column. But it works only after a single INSERT. This means you cannot use INSERT INTO SELECT FROM or LOAD, if you need to rely on this function.
- When the maximum value is reached for the identity column, DB2 will cycle back to the beginning to begin reassigning values – which might not be the desired approach.

If you can live with these caveats, then identity columns might be useful to your applications. However, in general, these 'problems' make identity columns a very niche solution. IBM has intentions to rectify some of these problems over time in forthcoming versions of DB2.

SEQUENCE OBJECTS

Remember, DB2 has two methods of automatically generating sequential numbers. The first method is to define an identity

column for the table, the second is to create a SEQUENCE object. A SEQUENCE object is a separate structure that generates sequential numbers.

New to DB2 V8, a SEQUENCE is a database object specifically created to generate sequential values. So, using a SEQUENCE object requires the creation of a database object; using an identity column does not.

A SEQUENCE object is created using the CREATE SEQUENCE statement.

When the SEQUENCE object is created it can be used by applications to 'grab' the next sequential value for use in a table. SEQUENCE objects are ideal for generating sequential, unique, numeric key values. A sequence can be accessed and incremented by many applications concurrently without the hot spots and performance degradation associated with other methods of generating sequential values.

Sequences are designed for efficiency and to be used by many users at the same time without causing performance problems. Multiple users can concurrently and efficiently access SEQUENCE objects because DB2 does not wait for a transaction to COMMIT before allowing the sequence to be incremented again by another transaction.

An example creating a SEQUENCE object follows:

```
CREATE SEQUENCE ACTNO_SEQ
   AS SMALLINT
   START WITH 1
   INCREMENT BY 1
   NOMAXVALUE
   NOCYCLE
   CACHE 10;
```

This creates the SEQUENCE object named ACTNO_SEQ. Now it can be used to generate a new sequential value, for example:

```
INSERT INTO DSN8810.ACT
     (ACTNO, ACTKWD, ACTDESC)
   VALUES
     (NEXT VALUE FOR ACTNO_SEQ, 'TEST', 'Test activity');
```

The NEXT VALUE FOR clause is known as a sequence expression. Coding the sequence expression causes DB2 to use the named SEQUENCE object to automatically generate the next value. You can use a sequence expression to request the previous value that was generated. For example:

```
SELECT PREVIOUS VALUE FOR ACTNO_SEQ
INTO   :IVAR
FROM   DSN8810.ACT;
```

As you can see, sequence expressions are not limited to INSERT statements, but can be used in UPDATE and SELECT statements too.

Caution: if you specify the NEXT VALUE FOR clause more than once in the same SQL statement DB2 will return the same value for each NEXT VALUE FOR specification.

SEQUENCE object parameters

Similar to identity columns, a SEQUENCE object has parameters to control the starting point for the generated sequential values, and the number by which the count is incremented. You can also specify the data type to be generated (the default is INTEGER). You can also specify a minimum value (MINVALUE) and a maximum value (MAXVALUE) if you wish to have further control over the values than is provided by the data type chosen.

Again, as with identity columns, you can specify how the SEQUENCE should handle running out of values when the maximum value is hit. Specifying the CYCLE keyword will cause the SEQUENCE object to wrap around and begin generating values from the minimum value all over again.

A final consideration for SEQUENCE objects is cacheing. Sequence values can be cached in memory to facilitate better performance. The size of the cache specifies the number of sequence values that DB2 will pre-allocate in memory. In the previous example CACHE 10 indicates that ten sequence values will be generated and stored in memory for subsequent use. Of course, you can turn off cacheing by specifying NO CACHE.

With cacheing turned off, each new request for a sequence number will cause I/O to the DB2 catalog (SYSIBM.SYSSEQUENCES) to generate the next sequential value.

SEQUENCE object guidelines

DB2 does not wait for an application that has incremented a sequence to commit before allowing the sequence to be incremented again by another application. Applications can use one sequence for many tables, or create multiple sequences to be used by each table requiring generated key values. In either case, the applications control the relationship between the sequences and the tables.

The name of the SEQUENCE object indicates that we are going to use it to generate activity numbers (ACTNO), but its usage is not limited to that. Of course, failure to control the use of a SEQUENCE object can result in gaps in the sequential values. For example, if we use the ACTNO_SEQ object to generate a number for a different column, the next time we use it for ACTNO there will be a gap where we generated that number.

Other scenarios can cause gaps in a SEQUENCE, too. For example, issuing a ROLLBACK after acquiring a sequence number will not roll back the value of the sequence generator – so that value is lost. A DB2 failure can also cause gaps because cached sequence values will be lost.

Please note, too, that when sequences were introduced in non-mainframe DB2, syntax was supported that did not conform to the SQL standard. This non-standard syntax is supported on the mainframe as well:

- NEXTVAL can be used in place of NEXT VALUE.
- PREVVAL can be used in place of PREVIOUS VALUE.

CHOOSING BETWEEN IDENTITY AND SEQUENCE

Although both identity columns and SEQUENCE objects are

useful for generating incremental numeric values, you will be confronted with situations where you will have to choose between the two. Consider the following criteria for when to use one instead of the other. Identity columns are useful when:

- Only one column in a table requires automatically-generated values.
- Each row requires a separate value.
- An automatic generator is desired for a primary key of a table.
- The LOAD utility is not used to load data into the table.
- The process of generating a new value is tied closely to inserting into a table, regardless of how the insert happens.

SEQUENCE objects are useful when:

- Values generated from one sequence are to be stored in

Identity columns	*SEQUENCE objects*
Internal objects generated and maintained by DB2	Stand-alone database objects created by a DBA
Associated with a single table	Not associated with a specific table; usable across tables
Use IDENTITY_VAL_LOCAL() to get last value assigned	Use PREVIOUS VALUE FOR *seq-expr* to get last value assigned
N/A	Use NEXT VALUE FOR *seq-expr* to get next value to be assigned
Add/change using ALTER TABLE ...ALTER COLUMN (*DB2 V8 only*) ...ALTER COLUMN (*DB2 V8 only*)	Administer using ALTER SEQUENCE, DROP, COMMENT, GRANT, and REVOKE
Version 6 refresh; Version 7	Version 8

Figure 1: Identity columns and SEQUENCE objects

more than one table.

- More than one column per table requires automatically-generated values (multiple values may be generated for each row using the same sequence or more than one sequence).
- The process of generating a new value is independent of any reference to a table.

Unlike SEQUENCE objects, which are more flexible, identity columns must adhere to several rigid requirements. For example, an IDENTITY column is always defined on a single table and each table can have, at most, one IDENTITY column. Furthermore, when you create an IDENTITY column, the data type for that column must be numeric – not so for sequences. If you used a SEQUENCE object to generate a value you could put what's generated into a CHAR column, for example. Finally, when defining an IDENTITY column you cannot specify the DEFAULT clause, and the column is implicitly defined as NOT NULL. Remember, DB2 automatically generates the IDENTITY column's value, so default values and nulls are not useful concepts.

Figure 1 shows a summary comparison of SEQUENCE objects and identity column characteristics.

SUMMARY

Both identity columns and SEQUENCE objects can be used to automatically generate sequential values for DB2 columns. Prior to Version 8, identity columns are your only option. However, after you move to V8, SEQUENCE objects will provide more flexibility and be easier to use than the identity column option.

Craig S Mullins
Director, Technology Planning
BMC Software (USA)

Capturing dynamic SQL on DB2 for z/OS and OS/390 distributed processing

Nowadays, most applications use distributed processing in some form or other to access data from the mainframe host that is running DB2 for OS/390 or z/OS. The access mechanisms could be either ODBC or JDBC. There are many products for database connectivity, such as DB2 Connect, HIS-2000 server, Shadow Direct, etc. All these products use DB2's internal mechanisms for accessing the data and it is invariably DRDA (Distributed Relational Database Architecture).

With the Web-enablement of most applications, dynamic SQL has come to be used more. The issue with dynamic SQL is that it cannot be kept in a DBRM like the static SQL. Since developers tend to churn out dynamic SQL without much thought for performance, problems surface when the programs start running in production. Distributed processing threads are executed in DB2 under a plan called DISTSERV, and it is common to have several threads running under this at one time. Depending on which type of product is being used for the host connection, and a variety of other parameters, it may be difficult or impossible to identify which thread is executing which query. In a development environment, it is difficult to debug and also tune such queries. Hence the need arises for capturing dynamic SQL for analysis, performance tuning, etc. A few methods will be discussed to achieve this.

It is assumed that the reader is familiar with using traces. For details about traces and IFCIDs, please refer to the *DB2 Universal Database for OS/390 and z/OS Administration Guide, Version 7*.

USING DB2 TRACES IN THE SPECIAL UTILITY JOB DSN1SDMP

This is a crude way of capturing SQL, but it has been useful in several instances as a quick way of getting trace data for solving special problems. The utility is known as IFC Selective Dump

and is used to produce a dump. The *DB2 UDB for OS/390 and z/OS, Utility Guide and Reference, Version 7* lists details about this job and its parameters.

Caution: the manual suggests that this must be used under the directive of IBM Support Center.

In the sample JCL provided below, the dump control cards are given in the SDMPIN DD card:

```
START TRACE(MONITOR) DEST(OPX)
AUTHID(*)
IFCID(63)
PLAN(DISTSERV)
FOR(5000)
```

This states that we start a monitor trace for IFCID(63) using OPX (the next available output buffer) as the destination and for PLAN name DISTSERV for any AUTHID. It also says that we want it to terminate after dumping 5,000 records. The plan name DISTSERV is the one used for distributed processing.

The output or the dump dataset is defined on the SDMPTRAC DD card.

The SYSTSIN DD card specifies the subsystem on which to execute the command and also the program and plan.

Sample JCL for DSN1SDMP:

```
//JOBCARD your job card
//IFCSD EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//STEPLIB  DD DISP=SHR,DSN=SYS1.DB2T.DSNLOAD
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SDMPPRNT DD SYSOUT=*
//*-------------------------------------------------
//*    SDMPTRAC DD USED ONLY FOR DEST(OPX)
//*-------------------------------------------------
//SDMPTRAC DD DISP=(NEW,CATLG,CATLG),
//            DSN=your output dataset
//            UNIT=HSM,SPACE=(CYL,(50,100),RLSE),
//            DCB=(DSORG=PS,LRECL=8188,RECFM=VB,BLKSIZE=8192)
//*-------------------------------------------------
//*    DO NOT USE SUBSYSTEM IDENTIFIER CHARACTER ON
//*      TRACE STATEMENT
//*-------------------------------------------------
```

```
//SDMPIN DD *
 START TRACE(MONITOR) DEST(OPX)
 AUTHID(*)
IFCID(63)
 PLAN(DISTSERV)
 FOR(5000)
/*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
 DSN SYSTEM(DB2T)
 RUN PROGRAM(DSN1SDMP) PLAN(DSNEDCL)
 END
/*
//
```

Important: if you need to stop or cancel the program before it has collected the dump information, you need to do it only with the **–STOP TRACE** command. Cancelling the job in SDSF will not work. Take care to correctly identify the trace and fully qualify the trace using trace number, trace type, destination, etc, before you issue the **–STOP TRACE** command. To identify the trace number issued by this job, look in the job log under SDSF option display active (DA). The **–STOP TRACE** command could be like this:

```
    -STOP TRACE(MONITOR) DEST(OPx) TNO(n)
```

Note that the *OPx* destination and *TNO(n)* will have to be determined from the **–DIS TRACE(*)** command or from the SDSF active job log. The **STOP TRACE** and/or the **DISPLAY TRACE** commands may be issued from the DB2I (DB2 Interactive) panel under the DB2 commands option, which is Option 7.

Once the job has terminated, we can browse the output dataset and identify the SQL. A partial sample output dataset is shown below with the hexadecimal display:

```
     ----+----1----+----2----+----3----+----4----+----5----+----6----+--
--7----+----8
------------------------------------------------------------------------
--------------------------------------------------
. . . . . . . . . . . . . . . . . . . . SELECT * FROM EMPLY.< . . . . .
e .'. . DBT2¾P.An93c...a...(....DBT2
000200000000101000001ECDCCE454CDDD4ECCEC00401030051733CCEFBD269FF800040000400000
000C1E0100040701400052535330C0696405312500C160F215DE8423297F45933000600D00004232
 ------------------------------------------------------------------------
--------------------------------------------------
```

```
. . .O. . . . . . . . . . N. . . . . LSELECT TBCREATOR as TABLE_OWNER,
TBNAME as TABLE_NAME, NAME
000E000000010D00000DECDCCE4ECCDCCEDD48A4ECCDC6DEDCD64ECDCDC48A4ECCDC6DCDC64DCD
000C1E01000405014003253533032395136901203123 5D66559B032514501203123 5D5145B051450
 ------------------------------------------------------------------------
------------------------------------------------------------------------
...o.........I.....ESELECT SYSIBM.SYSINDEXES.TBCREATOR as
TABLE_QUALIFIER, SYSIB
0008000000010700000 7ECDCCE4EEECCD4EEECDCCECE4ECCDCCEDD48A4ECCDC6DECDCCCCD64EEEC
002C1E01000425014023253533028 2924B2829545752B32395136901203123 5D841396959B028292
 ------------------------------------------------------------------------
------------------------------------------------------------------------
```

In the output dataset, the first 5 lines pertain to header information. The following lines contain the SQL statements. The SQL statements begin in column 21. The length of the SQL statement is stored in binary form in column 19 as shown in hex format above. Note that this length value takes column 19 as the starting position. For example, for the SQL shown in the example above:

```
SELECT * FROM EMPLY
```

the length is hex(0015), which is 21 in decimal. Counting from Column 19, we find that it is the length of the SQL statement in this case. With this information, we can write a REXX routine to extract only the SQL and dump it to another file. The maximum length for the SQL statement is 5,000 characters.

I used IFCID 63 for an application that uses the OLEDB technique to access data from DB2 on the host. This essentially helped us identify the sequence of events that were happening on the DB2 server after the OLEDB call was issued. Using monitors like DB2-PM or TMON for DB2, the thread did not stay long enough for us to view the SQL.

Please note host variables or symbolic variables will be represented by '?' in these statements. We need to use IFCID 247 to capture host variable data. It is possible to capture both IFCID 63 and 247 records in the same DSN1SDMP dump. However, identifying them requires more knowledge of traces and header records. Also, the host variables for decimal data are stored in a special format and it needs to be debugged differently. In the output dataset, columns 3 and 4 contain the offset at which

the relevant IFCID header information is. Going back to our example, the value is hex(002C), which is decimal 44. At column 44, the IFCID 63 record details are stored. This can be verified by the IFCID value in columns 45 and 46, namely hex(003F).

More details about IFCID 63 and other IFCID record layouts may be found in the IBM-supplied member DSNWMSGS in the DSNSAMP dataset. Look specifically under RMID 22 or performance trace records to identify which IFCIDs you will have to activate. For JDBC applications that use prepares, you may want to use IFCID 64, 65, etc.

Normally, IFCID 58 signifies the end of IFCID 63 and must also be captured. However, depending on your requirements, this may or may not be included.

USING CA UNICENTER'S DETECTOR PRODUCT

The Detector feature of CA (Computer Associates) Unicenter offers a simple way to capture all dynamic SQL. The advantage of this is we have all the information readily available, including accurate timestamp data. At our site, we have a mechanism whereby the collected data is archived by the hour and recycled every 72 hours.

The following section explains how to set up an exception profile for capturing dynamic SQL using Detector.

From the Detector main menu, choose Option 4, *View/modify profile*. From the next panel choose Option 1, *View Collection Profiles*. This will navigate to the Detector Collection Profiles Display panel. Enter 'Y' against the *Create Profile* field near the top right of this screen. In the following panel, specify valid values for *Profile SSID*, *Profile ID* (say TESTPROF), and *Profile desc*. Specify the high-level qualifier(s) and other parameters like volume name, allocation units, and primary and secondary spaces for allocating the exception dataset. The message 'Detector profile successfully initialized' will be displayed if all input is correct.

Press the *Exit* key to return to the previous screen. The newly-

created profile will be displayed here. Type **S** (for view/modify profile) against the newly created profile. This will navigate to the *Detector View/Modify application groups* panel. Type **S** against *Set Global Defs* in the second line to go to the *Global Defaults* panel. This is where we set the exception thresholds for dynamic and static SQL. Specify the exception limit for CPU time under dynamic SQL to a very low value, say 00:00:00.001. Specify the exception limit for *Getpage Reqs* and *Rows Returned* to 1. Specify **S** against the field *Update Global Defs* at the top to go through with the update. The updated screen is displayed when you press *Enter*. Press the *Exit* key three times to come to the *Detector Collection Profiles Display* panel. You are now ready to load your profile. Enter **L** (load) against the new profile that you created. If all necessary authorization is available then this becomes the active profile.

Now all dynamic SQL that exceeds the threshold specified will be captured in the Detector data store. Since we have set the thresholds so low, we will virtually trap every dynamic SQL that is executed. To view the dynamic SQL that was captured, utilize the exception tracking feature of Detector. Detector lists a bunch of useful accounting statistics like getpages, I/O wait times, synchronous reads, etc, for the dynamic SQL. One can also run Explain on the SQL utilizing the Explain product of Unicenter for DB2. It is also possible to load the SQL into DB2 tables utilizing the features of Index Expert, also from CA.

DB2 PM (PERFORMANCE MONITOR) FOR THE WORKSTATION

A detailed discussion of the features of this product can be found in the IBM Red Book, *Squeezing the most out of Dynamic SQL with DB2 for z/OS and OS/390*.

Essentially, this runs on the workstation and interfaces with DB2 on the host and collects and reports various performance data, such as accounting statistics, SQL activity, EDM pool, etc. However, it utilizes a graphical user interface to present its data. It also has the ability to start and stop traces and issue any authorized start command except **START DB2**.

A notable feature of this product is the ability to capture the SQL in the dynamic cache and also to run an explain on any of it. IFCID 316 needs to be activated to analyse the statement cache. You will also need IFCID 317 for viewing the entire SQL statement because 316 logs only the first 60 bytes of the statement text. Note that 316 must be captured in order to get 317 as well. They too may be linked using the statement identifier that is generated by DB2. Since the trace information for these two IFCIDs cannot be externalized to SMF or GTF like other conventional traces, using an online monitor like DB2 PM or an IFI program is the only way to view this information. Another trace that may be required is IFCID 318, which acts like a trigger to fill in the values for IFCID 316. DB2 PM for the workstation provides easy controls in the interface to activate and view this data in a real-time fashion. Additionally, DB2 PM for the workstation can download the collected trace data from the host. We can also set filters to show only a subset of the data. Most of these features are incorporated as point and click options or buttons, making it easy to use.

CONCLUSION

Three mechanisms for capturing Dynamic SQL were introduced and discussed. There are also other DB2 support products from other vendors that have features similar to Detector in CA Unicenter or DB2 PM for Workstation to capture and analyse SQL. These mechanisms should complement each other and help the DBA to deal in a competent manner with dynamic SQL in a distributed environment.

Jaiwant K Jonathan
DB2 DBA
QSS Inc (USA)

Refreshing test and development environments with the most current production data

There is always a requirement at some time to refresh various test and development environments with the most current production data – particularly for fixing defects reported in the production environment. Appropriate production data is very important to help developers reduce the time taken to fix a problem or deliver a solution. Moving a database from one system to another can be a complex job and requires a DB2 DBA. DBAs are often too busy to look at development and test environments prior to seeing to production issues. This article intends to help system engineers who understand the workings of DB2 to automate the movement of a recoverable production database to a test machine in order to perform tests on the most current production data. An important thing to remember here is to work with the DBA responsible for the production environment and take a recoverable production database and perform an online back-up using a script via DB2's Script Center GUI tool. A script can then be executed on the Test machine to perform a redirected restore followed by a rollforward to apply the log files. The article also explains how incremental delta back-ups can be taken in the test environments. Incremental back-ups provide an option to go back to the state of the test data at a particular date/ time in the past. This is particularly useful as an initial refresh of test from production, depending on the size of the database, might take quite a long time.

ADVANTAGES AND DISADVANTAGES OF VARIOUS UTILITIES

There are different ways to move a database from one system to another. The db2move utility allows movement of tables via the export and load/import APIs. This is a great method if you need to move a database across heterogeneous platforms. However, it does not move other database objects such as triggers, sequences, tablespaces, bufferpools, and indexes.

These objects would have to be re-created in a separate operation with the aid of the db2look utility. And for LOB tables, there is a limitation of 26,000 rows per table.

Performing a split mirror is another way to move a database. Although this is also a great way to clone a database, it is more complicated and requires a storage vendor's facilities to access the split mirror. In addition, you must have the exact directory paths on the development machine for the database directory, the tablespace containers, and log files as they would appear on the production system. Often, the necessary drive letters are not even available on development machines to reproduce the pathing required. Also, the DB2 server's instance must be the same name.

DB2's back-up and restore utility moves all objects in the database, and you can specify alternative paths to where your tablespace containers will reside. This allows for maximum flexibility when moving your database.

Note: some objects, such as User Defined Functions (UDFs) and stored procedures that are stored externally, will have to be moved separately since they are not included in the back-up image.

ENVIRONMENT

The example provided later uses two Windows machines that are mapped to each other via a local network (it is required that both machines have the same OS platform). In our example, each machine has DB2 Version 7.2.3, Enterprise Edition, with fixpak 6 applied. The Control Center is included by default during installation of DB2 and is required for running the Script Center.

If you have a Unix environment, the same rules apply. Both machines must have the same OS platform (an exception is a restore between SunOS and HP) and must have file systems mounted via the network during the back-up/restore procedure. Additionally, the Control Center component is not installed on Unix by default. So it will be necessary to install it before using

the scheduler (an alternative would be to execute the scripts using a cron job).

INITIAL STEPS

Some initial directories should be created before executing the scripts.

On the production machine create *\scripts* (to hold the script that will perform the online database back-up).

Note: this should already exist, because the DBA must be taking back-ups of production databases.

On the test machine create *\backups* (to hold the backed-up image of the production database), *\scripts* (to hold the script to restore the database), and *\tablespaces* (to hold the containers for the tablespaces).

Create the back-up script on the production machine. The DBA should have the *\backups* directory dedicated for this database only and back up.

```
db2cmd "db2 backup db proddb online to G:\backups"
```

This command executes the **db2** command window session, runs the database back-up command, and saves the image on the test machine. Since it is assumed that the production database must be running 24 hours a day, seven days a week, an online back-up is necessary. In the instance above, the G: drive letter is pointing to the C: drive on the test machine. It is required to have at least one full off-line database back-up of the production database before you can execute an on-line back-up.

Create the restore scripts on the development machine. There are two scripts for the restore. The first script will call a DB2 command window session and execute the second script, which, in turn, will execute multiple DB2 and OS commands.

Script 1:

```
\scripts\restore_testdb.cmd ...executes the following...
db2cmd restore_testdb_2.cmd
```

Script 2:

\scripts\restore_testdb_2.cmd ...executes the following...

1 **db2 "force application all"**

Before dropping the old database on the test machine, it is necessary to force any applications off the DB2 instance where the database resides. Currently, there is no automated way of forcing all applications off a single database. If there are any other services running that are dependent on the DB2 instance, it may be necessary to stop those services first before stopping the DB2 instance.

2 **db2 "restore db proddb from C:\backups into testdb redirect without prompting"**

The restore will now create the new database testdb and indicate a redirect to allow containers to be specified for the tablespaces. Do not be worried when receiving the message SQL1277N. This is only a warning that containers can be defined for the tablespaces (see Step 3).

The above REDIRECT option allows us to specify alternative tablespace container paths from the production system. If REDIRECT is not specified, you must create the same database *<drive letter>\<instance name>\NODE0000* as is shown on the production system. And if the production tablespace containers are located in a different path, they too must have paths set up for them on the development machine.

3 **db2 "set tablespace containers for 0 using (path "C:\tablespaces\tbspc0")"**

db2 "set tablespace containers for 1 using (path"C:\tablespaces\tbspc1")"

db2 "set tablespace containers for 2 using (path "C:\tablespaces\tbspc2")"

In order to automate the restore process, we will redirect all tablespaces associated with the production database by setting new container paths on the development machine. To determine what tablespaces reside in the database, you must first connect to the production database and execute the following command at the DB2 command window.

4 **db2 "list tablespaces show detail"**

The output will look something like this:

```
Tablespace ID                        = 0
Name                                 = SYSCATSPACE
Type                                 = System managed space
Contents                             = Any data
State                                = 0x0000
Detailed explanation:
Normal

Tablespace ID                        = 1
Name                                 = TEMPSPACE1
Type                                 = System managed space
Contents                             = System Temporary data
State                                = 0x0000
Detailed explanation:
Normal

Tablespace ID                        = 2
Name                                 = USERSPACE1
Type                                 = System managed space
Contents                             = Any data
State                                = 0x0000
Detailed explanation:
Normal
```

5 In the SET commands in Step 3, the tablespace IDs 0, 1, and 2 are used to specify the tablespace that will be assigned the new container path. All three tablespaces are System Managed Spaces (SMS) with the containers pointing to *C:\tablespaces* directory. Note, you need to create only the *C:\tablespaces* directory. Executing the SET commands will create the directories tbspc1, tbspc2, and tbspc3. If you have Database Managed Spaces (DMS), the syntax is slightly different. Let's say there's a fourth tablespace that is a DMS tablespace. We would perform something like the following.

6 **db2 "set tablespace containers for 0 using (path"C:\tablespaces\tbspc0")"**

7 **db2 "set tablespace containers for 1 using (path "C:\tablespaces\tbspc1")"**

8 **db2 "set tablespace containers for 2 using (path "C:\tablespaces\tbspc2")"**

9 **db2 "set tablespace containers for 3 using (file "C:\tablespaces\tbspc3" 50000)"**

10 Note: the path is changed to **file** and **50000** is the number of pages you are allocating for the container. Make sure the number of pages you assign is at least the same number of pages as the production database. You can also consolidate multiple containers into one during a redirected restore if your development machine uses fewer disks than the production server.

11 **db2 "restore database proddb continue"**

12 This is the final process of the redirected database restore.

13 `copy /y E:\DB2\NODE0000\SQL00010\SQLOGDIR\*.* C:\DB2\NODE0000\SQL00002\SQLOGDIR\*.*`

14 Above, we copy the log files from the production machine to the development machine. Continuing with the assumption that your production database uses the on-line back-up method, it is a requirement that a recovered database rolls forward all log files to ensure database consistency. Because of this, we will need to copy all log files from the production machine to the development machine and place them in the directory path where the database manager can find them. The path to the log files can be determined by performing the following command:

15 **db2 "get database configuration for <database name>"**

16 Look for ‘path to log files’ or ‘changed path to log files’. You should execute the above command on both the production and development servers to determine the log path to specify.

17 **db2 "rollforward database devdb to end of logs and stop"**

18 Finally, we can perform a rollforward operation after all log files are copied to the D:\DB2INST\NODE0000\SQL00002\SQLOGDIR on the development machine. Rolling forward the log files is necessary since the database back-up is performed on-line. Remember that the log path will be based on the path indicated in the database configuration file.

IMPORTANT RESTORE CONSIDERATIONS

If you plan to move a database from the production machine to a development machine and it does not have the default code page 1252 (for Windows), you will need to create your database on the development machine with the correct code page before you perform the restore. If you do not do this and your production code page is different, such as 1208, the restore utility will assume the default code page of 1252 and try to restore the code page 1208 database into a code page 1252 database. This will result in an SQL2548N error.

SCHEDULING THE JOB

Now that we have detailed the steps for writing the scripts, a scheduled job can be created to run the back-up and restore operation at a specific time of the day, week, or month. Our strategy is to run the back-up script on the production machine before running the restore script on the development machine. Depending on the size of your production database, it may be prudent to schedule your back-up several hours before you run the restore operation. This will ensure the restore does not start before the back-up has completed.

Using the development machine as our example, let us go through the steps of creating a schedule using the Script Center GUI tool.

1 Open the Control Center. Left mouse click on *Tools* from the menu bar and select *Script Center*.

2 Script Center opens. Left mouse click on *Script* from the menu bar and select *Import...*.

3 Select your *scripts* directory where the scripts are located and select restore_devdb.cmd:

   – In the *Instance list* box, select the DB2 instance where the database was created.

   – Type a new script name for the script you selected. This

is just a copy of the script you selected in Step 3 and will be used as the executing script.

- Provide a description of the script in the *Script Description* text box.
- The *Working Directory* text box gives you the option of specifying where you would like the output from your scripts (ie error and warning messages).
- Make sure you select the *OS command* radio button since we are using a command file.

4 Now that the script is prepared, we can schedule it to run at a predefined date. Right mouse click on the script to be run and select *Schedule...*.

After you run the scripts, it is very important to know whether they executed successfully. When executing scripts from the Script Center, it is a general practice to use the Journal to determine whether the script is successful. But since we are running OS scripts that execute other OS scripts, the results of the output will not be displayed in the Journal. To alleviate this problem, you can send the DB2 messages to output within your script. For instance, in the back-up script above, we back up the database in the following manner:

```
db2cmd "db2 backup db proddb online to G:\backups"
```

We could include the following to allow the results of the command to be sent to backup_results.msg:

```
db2cmd " "db2 backup db proddb online to g:\backups" >
backup_resultslog.msg"
```

INCREMENTAL DELTA BACK-UPS

Incremental back-up on the test database can help in troubleshooting problems in applications specific to the state of data from a certain date/time. We can choose different storage media for saving a back-up image. The most often used solutions are local or remote disk file system or TSM (Tivoli Storage

Manager). The TSM solution is widely used for large databases. Below I will explain the TSM solution in detail and give an example of hard disk usage. Before we start, we will check what needs to be installed and configured on the test server:

- Tivoli Storage Manager Client API.
- C Compiler for compiling user exit program db2uext2.c.
- TSM management classes for full back-up, delta back-up, and DB2 logs.
- Disk space on a separate file system in case we back-up on a hard disk.

We need to configure the user exit program db2uext.c to ensure that archived log files are correctly handled and saved on TSM. Usually you need to change only the log destination before compiling it.

Edit file ~/c/db2uext2.c and create directory structure (*/logs/**):

```
#define ARCHIVE_PATH “/logs/archive”
#define RETRIEVE_PATH “/logs/retrieve”
#define AUDIT_ERROR_PATH “/logs/log”
```

Compile the source file. Take this warning into account:

```
IY09505:INCORRECT
COMPILE INSTRUCTIONS IN DB2UEXT2.CADSM FOR ADSM 3.1.6 OR HIGHER.
```

In the db2uext2.cadsm skip the documentation that tells you to use “cc -o db2uext2 db2uext2.c libApiDs.a” and use the following:

```
"cc_r -o db2uext2 db2uext2.c libApiDs.a"
```

This will use the re-entrant (threadsafe) compiler.

Copy the final compiled version to destination ~/sqllib/adm/db2uext2. Now we are ready to go to the database.

Our test database, AMSTEST, is running in no-logging mode.

```
db2 => get db cfg for AMSTEST

 Database Configuration for Database AMSTEST

 Track modified pages (TRACKMOD) = OFF
```

```
 Log retain for recovery enabled (LOGRETAIN) = OFF
 User exit for logging enabled (USEREXIT) = OFF
We are going to change the configuration for Archive Logging and check
the back-up pending indicator.
db2 =>update db cfg for AMSTEST using USEREXIT ON

DB20000I The UPDATE DATABASE CONFIGURATION command completed
successfully.

db2 =>update db cfg for AMSTEST using LOGRETAIN ON

DB20000I The UPDATE DATABASE CONFIGURATION command completed
successfully.

$db2 "get db cfg for AMSTEST" | grep -i "BACKUP PENDING"

Log retain for recovery enabled (LOGRETAIN) = BACKUP PENDING
```

The back-up pending indicator (LOGRETAIN) now has the value 'BACKUP PENDING', which is the new recovery point for the database. DB2 requires an off-line back-up to establish this new recovery point and get the database out of the BACKUP PENDING state. Before making an off-line back-up we have to close all connections and restart the database.

```
$ db2 force application all

DB20000I The FORCE APPLICATION command completed successfully.
DB21024I This command is asynchronous and may not be effective
immediately.

$db2 connect reset
DB20000I The SQL command completed successfully.

$db2stop
SQL1064N DB2STOP processing was successful

$db2start
SQL1063N DB2START processing was successful.

TSM:
$db2 backup db AMSTEST to tsm
Backup successful. The timestamp for this backup image is :
20021311141448001

FILE SYSTEM

$db2 backup db AMSTEST to /backup_fs/amstestdb/
Backup successful.
```

```
The timestamp for this backup image is :20021311141448001
```

If the back-up ends successfully, then the updated history file will reset the back-up pending flag from BACKUP PENDING to RECOVERY.

We've just produced a database image, which will be a starting point for the recovery process if we need to rebuild the database to a consistent state. The image is:

```
 Database configuration release level = 0x0900
 Database release level = 0x0900
 Log retain for recovery enabled (LOGRETAIN) = RECOVERY
 User exit for logging enabled (USEREXIT) = ON
```

Finally we have everything prepared for an online back-up.

```
TSM:
db2 => backup database AMSTEST online use tsm
Backup successful. The timestamp for this backup image is:
20021311141448001
FILE SYSTEM

$db2 backup db AMSTEST online to /backup_fs/amstest/

Backup successful. The timestamp for this backup image is :
20021311141448001
```

If we were to try to run the delta back-up now, the database log file would display the message: 'Incremental backup not permitted for tablespace 0 (SYSCATSPACE). TRACKMOD not enabled'.

We need to update the TRACKMOD parameter:

```
db2 => update db cfg for AMSTEST using TRACKMOD ON
DB20000I The UPDATE DATABASE CONFIGURATION command completed
successfully.
```

Restart the database after that change is made and make the off-line back-up.

```
$ db2 force application all
DB20000I The FORCE APPLICATION command completed successfully.
DB21024I This command is asynchronous and may not be effective
immediately.

$db2 connect reset
DB20000I The SQL command completed successfully.
```

```
$db2stop
SQL1064N DB2STOP processing was successful

$db2start
SQL1063N DB2START processing was successful.

TSM:
$db2 backup db AMSTEST to tsm
Backup successful. The timestamp for this backup image is :
20021311161448001
$db2 backup db AMSTEST online incremental delta use tsm
Backup successful. The timestamp for this backup image is :
20021311161448001

FILE SYSTEM :

$db2 backup db AMSTEST online incremental delta to /backup_fs/amstest
Backup successful. The timestamp for this backup image is :
20021311161448001
```

BACK-UP CONTROL

All important information is stored in one file, called the history file (db2rhist.asc). DB2 handles duplicated versions of the same file (db2rhist.bak) for recovery reasons.

For example, the history file contains information of all the back-ups for database AMSTEST:

```
$ db2 "list history backup all for AMSTEST"

Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log  Backup ID
 -- -- ------------------ ---- -- ------------ ------------ ------------
  B  D  20021111162015003      E    A  S0002207.LOG S0002207.LOG
 ------------------------------------------------------------------------
  Contains 3 tablespace(s):

  00001 SYSCATSPACE
  00002 OLTP_1A
  00003 OLTB_1B
--------------------------------------------------------------------------
  Comment: DB2 BACKUP AMSTEST ONLINE
 Start Time: 20021311141448001
  End Time: 20021311141848001
 ------------------------------------------------------------------------
```

The suggested restore order of images using timestamp 20021311141448001 for database AMSTEST is:

```
=====================================================
 restore db amstest incremental taken at 20021206010134
 restore db amstest incremental taken at 20021204010129
 restore db amstest incremental taken at 20021204172723
 restore db amstest incremental taken at 20021205010133
 restore db amstest incremental taken at 20021206010134
```

It is recommended that every DBA checks and compares the back-up size on TSM or file system for delta and full back-ups.

Compare on TSM:

```
 >> dsmc query backup "/AMSTEST/DELTA.*.*"

 Size Backup Date Mgmt Class A/I File
 ---- ---------- ---------- -- ----
API 7.944.346 K 02.11.2002 01:02:47 MC3650 A /AMSTEST/NODE0000/
DELTA.20021111010247.1


>> dsmc query backup “/AMSTEST/FULL.*.*”

 Size Backup Date Mgmt Class A/I File
 ---- ---------- ---------- -- ----
API 7.944.363 K 11.11.2002 01:05:52 MC3650 A /AMSTEST/NODE0000/
FULL.20021111010552.1
```

This information is critical in making a final decision. In our case, delta back-up is almost as large as a full back-up and will not be the right solution for us.

CONCLUSION

Moving a production database to a test machine can be a complicated and frustrating process. This article attempts to alleviate the potential pitfalls by providing a step-by-step guide for creating your own back-up/restore scripts and automating them to run without any user intervention. As with any major movement of data, it is highly recommended to practise the back-up and restore scripts using a test database for both the production and test machine before implementing a large-scale back-up and restore.

Vikas Baruah
Senior Technical Specialist
American Management Systems (USA)

A program to fix tablespaces/indexes with RESTRICTed access

DB2 objects with RESTRICTed access are tablespaces/indexes that became inaccessible through (for example) the cancellation of a job (LOAD for example), or because a utility executing on a resource will not permit other processes to access the same resource.

When a DB2 object is RESTRICTed in our production environment, we need to quickly identify which utility is executing and solve the problem because it negatively affects the availability of the system. We decided to provide the operators with a tool to assist them in this situation.

In the development and test environments, a RESTRICTed DB2 object causes delay in the work of the programmers. It was desirable to have a tool for them to resolve the problem themselves.

I developed the program DB2RES to automatically fix RESTRICTed DB2 objects, which can be executed by the person who detects the problem (operators or programmers).

This program may be executed by people with little knowledge of DB2 because they don't need to know DB2 commands or DB2 utilities to use it. The program is actually executed by operators (in the production environment) and programmers (in development and test environments).

We now quickly and easily resolve incidents caused by DB2 objects becoming RESTRICTed (and inaccessible), cutting down the number of calls to and interventions by the DB2 support staff.

As a consequence, we have improved the quality of the service and the availability of the system.

DB2RES (REXX/ISPF) shows in the main panel all the DB2 objects in RESTRICTed mode, the various statuses possible (RO/STOP/CHKP/COPY/RECP/... etc), the name of the object,

the database, the partition, and the type of object (tablespace/ index).

The program is initiated by means of a unique option R (fix or resolve), and in some cases (RECP for example) with suboptions.

The program analyses the status of the DB2 object selected and builds a combination of commands and/or utility jobs to fix the problem and make it accessible (RW). The DB2 commands are executed automatically. In the case of the job(s) these are displayed on an ISPF window (not editable), and they are submitted for execution by pressing *Enter*. After running the job(s), the user may again check the DB2 objects by pressing *Enter* in the main panel. The object should now be accessible (ie it should not appear on the panel).

DB2RES fixes DB2 objects in any combination of the following RESTRICTed statuses: RO, STOP, STOPP, UT, UTRO, UTRW, GRECP, LPL, COPY, CHKP, PSRBD, RBDP, RBDP*, RECP. It may be extended to resolve other RESTRICTed statuses too.

The only condition is that it first fixes the tablespace and then its index(es) in the case where both the tablespace and its index(es) are RESTRICTed.

The program may accept many options together on the main panel (it uses the ISPF variable ZTDSELS) and processes them one-by-one until all the commands and/or utilities are completed.

This version of the DB2RES program uses BMC utilities, but it could be changed to generate IBM utilities or any third-party's software.

The program recognizes/checks when a DB2 object is in use by a utility (and as a consequence of this could be RESTRICTed – a load, for example IBM's Load or BMC's Load Plus – and refuses to start, not building any commands and/or utilities.

If the utility is not active (STOPPED for example), the program terminates the utility and then fixes the DB2 object.

The program supplies help panels (F1 key). In addition, it has

explanatory and error messages in two versions – short and long (accessible by pressing the F1 key when it shows a short message).

It uses a library of messages (ISPMLIB).

It maintains a log in which to write all the users' actions. It stores information about the option chosen (**D** to display the main panel and **R** to fix/resolve), the user who executes DB2RES, DB2 subsystem, status (restrict) of the DB2 object, name of the DB2 object, database, tablespace or index, and date and time of the DB2RES execution.

At the start of its execution the program searches the DB2 subsystems defined on OS/390 (reads the DB2 vectors from storage), which are shown in an initial panel – where the user can choose the DB2 subsystem.

DB2RES is a REXX/ISPF program that uses tables and windows ISPF. At present the program is executed with OS/390 V2.9, DB2 V6.1.

It uses BMC commands BMCDSN V2.R3.00 to drive the BMC utilities. The BMC utilities used are BMC Copy Plus V6.2 and BMC Recover Plus V3.4. It can be any version of the BMC utilities or you can change the program to use IBM utilities.

I installed the program DB2RES as an option on the DB2 panel DSNEPRI (beside SPUFI, QMF).

```
/* REXX    carlos-osorio@excite.com */
TRACE OFF
NUMERIC   DIGITS 12;
CVT     = C2X(STORAGE(10,4))                /* ADDR DE CVT          */
CVTJESCT= D2X((X2D(CVT))+296)               /* POINTER A CVTJESCT   */
JESCT   = C2X(STORAGE(CVTJESCT,4))          /* ADDR DE JESCT        */
JESSSCT = D2X((X2D(JESCT))+24)              /* POINTER A JESSSCT    */
SSCVT   = C2X(STORAGE(JESSSCT,4))           /* ADDR DE SSCVT        */
J = 0
DO I = 1  WHILE (SSCVT <> 00000000)
          SSCTSNAM=D2X((X2D(SSCVT))+8)      /* POINTER A SSCTSNAM   */
          ERLY  = D2X((X2D(SSCVT))+20)      /* POINTER A ERLY       */
          ERLYAD= C2X(STORAGE(ERLY,4))      /* ADDR DE ERLY         */
          ERLYSCOM= D2X((X2D(ERLYAD))+56)   /* POINTER A ERLYSCOM   */
          IF SUBSTR(STORAGE(ERLYSCOM,64),29,8) = 'DSN3EPX ' THEN
```

```
            DO
            J = J + 1
            SSIDDB2.J = STORAGE(SSCTSNAM,4) /* ADDR DE SSCTSNAM     */
            END
          SSCTSCTA = D2X((X2D(SSCVT))+4)     /* POINTER AL SGTE SSCVT*/
          SSCVT    = C2X(STORAGE(SSCTSCTA,4))
END
"ISPEXEC LIBDEF ISPPLIB DATASET ID('YOUR.PANELI')"
"ISPEXEC LIBDEF ISPMLIB DATASET ID('YOUR.MSGLBI')"
"ISPEXEC TBCREATE TSSIDDB2",
"NAMES(O SDB2)",
"NOWRITE REPLACE"
O= '';
DO J = 1 TO J
SDB2 = SSIDDB2.J
"ISPEXEC TBADD   TSSIDDB2"
END
"ISPEXEC TBTOP   TSSIDDB2"
"ISPEXEC TBDISPL TSSIDDB2 PANEL(DB2RESSP)"
IF RC = 8 THEN EXIT;
"ISPEXEC LIBDEF ISPPLIB"
"ISPEXEC LIBDEF ISPMLIB"
"ISPEXEC TBEND  TSSIDDB2"
SSID = SDB2
SELECT
  WHEN SSID = 'DB2P' THEN
       DO
      "ISPEXEC LIBDEF ISPPLIB DATASET ID('YOUR.PROD.PANELLIB.USER')"
      "ISPEXEC LIBDEF ISPMLIB DATASET ID('YOUR.PROD.MSGLIB')"
       LOGPREFIX = 'PROD'
       END
  WHEN SSID = 'DB2D' THEN
       DO
      "ISPEXEC LIBDEF ISPPLIB DATASET ID('YOUR.DESA.PANELLIB.USER')"
      "ISPEXEC LIBDEF ISPMLIB DATASET ID('YOUR.DESA.MSGLIB')"
       LOGPREFIX = 'DESA'
       END
  WHEN SSID = 'DB2T' THEN
       DO
      "ISPEXEC LIBDEF ISPPLIB DATASET ID('YOUR.TEST.PANELLIB.USER')"
      "ISPEXEC LIBDEF ISPMLIB DATASET ID('YOUR.TEST.MSGLIB')"
       LOGPREFIX = 'MBVD'
       END
  OTHERWISE
       DO
      "ISPEXEC LIBDEF ISPPLIB DATASET ID('YOUR.PROD.PANELLIB.USER')"
      "ISPEXEC LIBDEF ISPMLIB DATASET ID('YOUR.PROD.MSGLIB')"
       LOGPREFIX = 'PROD'
       END
END
```

```
"ISPEXEC TBCREATE TRES",
"NAMES(O STATUS NAME DATABASE PART TYPE)",
"NOWRITE REPLACE"
RESCOPY_EXIT = 'N';
RESCHKP_EXIT = 'N';
RESRBDP_EXIT = 'N';
RESREC1_EXIT = 'N';
RESREC2_EXIT = 'N';
RESREC3_EXIT = 'N';
RESREC4_EXIT = 'N';
DO FOREVER
O = ''
IF  RESCOPY_EXIT = 'N' & ,
    RESCHKP_EXIT = 'N' & ,
    RESRBDP_EXIT = 'N' & ,
    RESREC1_EXIT = 'N' & ,
    RESREC2_EXIT = 'N' & ,
    RESREC3_EXIT = 'N' & ,
    RESREC4_EXIT = 'N'    THEN CALL DISRES;
"ISPEXEC TBDISPL TRES PANEL(DB2RESP)";
IF   RC = 8 THEN
     DO
    "ISPEXEC LIBDEF ISPPLIB"
    "ISPEXEC LIBDEF ISPMLIB"
    "ISPEXEC TBEND  TRES"
     EXIT Ø;
     END
ELSE DO
     TDSELS = ZTDSELS
     IF          TDSELS =  Ø  THEN
        DO
        RESCOPY_EXIT = 'N';
        RESCHKP_EXIT = 'N';
        RESRBDP_EXIT = 'N';
        RESREC1_EXIT = 'N';
        RESREC2_EXIT = 'N';
        RESREC3_EXIT = 'N';
        RESREC4_EXIT = 'N';
        END
     IF         PFKEYIN = 'S' THEN "ISPEXEC DISPLAY PANEL(DB2RESH)"
     IF          TDSELS >  Ø  THEN  CALL TRATA_RESTRICT
        DO WHILE TDSELS >  1
          "ISPEXEC TBDISPL TRES"
                 TDSELS =  ZTDSELS
           IF    TDSELS >  Ø  THEN  CALL TRATA_RESTRICT
        END
     IF RESCOPY_EXIT = 'N' & ,
        RESCHKP_EXIT = 'N' & ,
        RESRBDP_EXIT = 'N' & ,
        RESREC1_EXIT = 'N' & ,
```

```
        RESREC2_EXIT = 'N' & ,
        RESREC3_EXIT = 'N' & ,
        RESREC4_EXIT = 'N'    THEN CALL TBDELROWS
     END
END
/*-------------------------------------------------------------------*/
/*-------------------- R U T I N A S - carlos-osorio@excite.com -*/
/*-------------------------------------------------------------------*/
DISRES:
W = OUTTRAP('RES.')
    QUEUE "-DIS DB(*) SPACE(*) RES LIMIT(*)"
    QUEUE "END"
   "DSN SYSTEM("SSID")"
W = OUTTRAP('OFF')
IF RC >= 12 THEN
   DO
   "DELSTACK"
   ADDRESS ISPEXEC "SETMSG MSG(DBC3ØØ)"                /* DB2 ABAJO */
   RETURN
   END
RESMAX = RES.Ø
IF WORD(RES.RESMAX,1) = 'DSN9Ø22I' THEN        /* NORMAL     DIS RES */
   DO
   IF SUBSTR(RES.1,1,8) = 'DSNT365I' THEN      /* NO HAY OBJETOS RES */
      DO
      ADDRESS ISPEXEC "SETMSG MSG(DBC3Ø1)"
      DISRES_ZERO = 1
      CALL GRABALOG
      RETURN
      END
   END
ELSE
   IF WORD(RES.RESMAX,1) = 'DSN9Ø23I' THEN     /* ABNORMAL   DIS RES */
      DO
      ADDRESS ISPEXEC "SETMSG MSG(DBC3Ø2)"
      RETURN
      END
I = 1;
DO I = I WHILE I < RES.Ø
   IF (SUBSTR(RES.I,1,8) = 'DSNT362I') THEN             /* DATABASE */
      DO
      DATABASE = WORD(RES.I,5)
        DO K = 1 UNTIL (SUBSTR(RES.I,1,8) = 'DSNT397I' |,
                        SUBSTR(RES.I,1,8) = '******* ') /*FIN DISPLAY*/
        I = I + 1;
        END
        IF (SUBSTR(RES.I,1,8) = '******* ') THEN
           DO
           ADDRESS ISPEXEC "SETMSG MSG(DBC3Ø1)"
           DISRES_ZERO = 1
```

```
            CALL GRABALOG
            RETURN
            END
          DO K = 1 UNTIL (SUBSTR(RES.I,1,8) = '--------' |,   /* TS/IX */
                          SUBSTR(RES.I,1,8) = '******* ') /*FIN DISPLAY*/
          I = I + 1;
          END
          IF (SUBSTR(RES.I,1,8) = '******* ') THEN
             DO
             ADDRESS ISPEXEC "SETMSG MSG(DBC301)"
             DISRES_ZERO = 1
             RETURN
             END
        I = I + 1;
          DO K = 1 UNTIL (SUBSTR(RES.I,1,8) = '******* ')
          NAME   = WORD(RES.I,1)
          TYPE   = WORD(RES.I,2)
          SELECT
            WHEN TYPE = 'TS' THEN TYPE = 'TABLESPACE'
            WHEN TYPE = 'IX' THEN TYPE = 'INDEX     '
            OTHERWISE             TYPE = '          '
          END
          PART   = SUBSTR(RES.I,15,4)
          STATUS = SUBSTR(RES.I,20,18)
          IF SUBSTR(NAME,1,1) <> '-' THEN
             DO
            "ISPEXEC TBADD TRES"
             DISRES_ADD = 1
             CALL GRABALOG
             END
          I = I + 1;
          END
        END
END
"ISPEXEC TBSORT TRES FIELDS(DATABASE,C,A,NAME,C,A,TYPE,C,D,PART,N,A)"
"ISPEXEC TBTOP TRES"
RETURN
/*-------------------------------- carlos-osorio@excite.com --*/
TRATA_RESTRICT:
IF O = 'R' THEN
   DO
     SELECT
       WHEN SUBSTR(DATABASE,1,4) = 'DSND' THEN
            DO
            ADDRESS ISPEXEC "SETMSG MSG(DBC400)"
            RETURN
            END
       WHEN SUBSTR(DATABASE,1,3) = 'DSQ'  THEN
            DO
            ADDRESS ISPEXEC "SETMSG MSG(DBC401)"
```

```
            RETURN
            END
       WHEN SUBSTR(DATABASE,1,3) = 'BMC'  THEN
            DO
            ADDRESS ISPEXEC "SETMSG MSG(DBC4Ø2)"
            RETURN
            END
       OTHERWISE NOP
     END
     CALL STATUS_PARSE
     DO R = 1 TO NRES
        SELECT
              WHEN STATUSRES.R = 'RW'    THEN NOP
              WHEN STATUSRES.R = 'RO'    THEN CALL RESSTA
              WHEN STATUSRES.R = 'STOP'  THEN CALL RESSTA
              WHEN STATUSRES.R = 'STOPP' THEN CALL RESSTA
              WHEN STATUSRES.R = 'UT'    THEN CALL RESSTA
              WHEN STATUSRES.R = 'UTRO'  THEN CALL RESSTAFORCE
              WHEN STATUSRES.R = 'UTRW'  THEN CALL RESSTAFORCE
              WHEN STATUSRES.R = 'GRECP' THEN CALL RESSTAFORCE
              WHEN STATUSRES.R = 'LPL'   THEN CALL RESSTAFORCE
              WHEN STATUSRES.R = 'COPY'  THEN CALL RESCOPY
              WHEN STATUSRES.R = 'CHKP'  THEN CALL RESCHKP
              WHEN STATUSRES.R = 'PSRBD' THEN CALL RESRBDP
              WHEN STATUSRES.R = 'RBDP'  THEN CALL RESRBDP
              WHEN STATUSRES.R = 'RBDP*' THEN CALL RESRBDP
              WHEN STATUSRES.R = 'RECP'  THEN CALL RESRECP
              OTHERWISE DO
                        ADDRESS ISPEXEC "SETMSG MSG(DBC4Ø3)"
                        RETURN
                        END
        END
     END
   END
IF O = 'D' THEN NOP            /* PARA FUTURA AMPLIACION DB2RES  */
RETURN
/*-----------------------------------------------------------------*/
STATUS_PARSE:
STATUST = TRANSLATE(STRIP(STATUS),' ',',')
NRES    = WORDS(STATUST)
DO I = 1 TO NRES
   STATUSRES.I = WORD(STATUST,I)
END
RETURN
/*-----------------------------------------------------------------*/
RESSTA:
CALL DISUTIBMC;
IF STATUSRBMC = 'X' THEN RETURN;
CALL TERUTIIBMSTOP;
W = OUTTRAP('STA.')
```

```
QUEUE "-START DATABASE("DATABASE") SPACE("NAME") ACCESS(RW)"
QUEUE "END"
"DSN SYSTEM("SSID")"
W = OUTTRAP('OFF')
IF RC = Ø THEN
   DO
   ADDRESS ISPEXEC "SETMSG MSG(DBC3Ø4)"       /* START SATISFACTORIO */
   CALL GRABALOG
   END
ELSE
   ADDRESS ISPEXEC "SETMSG MSG(DBC3Ø3)"       /* START CON PROBLEMAS */
RETURN
/*-------------------------------------------------------------*/
RESSTAFORCE:
CALL DISUTIBMC;
IF STATUSRBMC = 'X' THEN RETURN;
CALL TERUTIIBMSTOP;
W = OUTTRAP('STA.')
QUEUE "-START DATABASE("DATABASE") SPACE("NAME") ACCESS(FORCE)"
QUEUE "END"
"DSN SYSTEM("SSID")"
W = OUTTRAP('OFF')
IF RC = Ø
   DO
   ADDRESS ISPEXEC "SETMSG MSG(DBC3Ø4)"       /* START SATISFACTORIO */
   CALL GRABALOG
   END
ELSE
   ADDRESS ISPEXEC "SETMSG MSG(DBC3Ø3)"       /* START CON PROBLEMAS */
RETURN
/*-------------------------------------------------------------*/
RESCOPY:
CALL DISUTIBMC;
IF STATUSRBMC = 'X' THEN RETURN;
CALL TERUTIIBMSTOP;
"NEWSTACK"
CALL LIBDEFWINDJ;
DSNJOB   = USERID() || '.RES' || '.JOB'  || TIME('S')
ADDRESS TSO "ALLOC FILE(JOB) DATASET('"DSNJOB"') " ,
            "NEW CAT REUSE UNIT(SYSDA)" ,
            "LRECL(8Ø) BLKSIZE(2792Ø) RECFM(F B) SPACE(1,1) CYL"
             IF RC <> Ø THEN
                DO
                ADDRESS ISPEXEC "SETMSG MSG(DBCØ21)"
                RETURN
                END
JOBNAME = SUBSTR((USERID() || 'IC'),1,8)
QUEUE "//"JOBNAME" JOB (DB2),'OSORIO',MSGCLASS=X,"
QUEUE "// CLASS=S,MSGLEVEL=(1,1),NOTIFY=&SYSUID"
QUEUE "//STEP1 EXEC PGM=ACPMAIN,PARM='"SSID",,NEW,MSGLEVEL(1)',"
```

```
QUEUE "// REGION=ØM"
QUEUE "//SYSPRINT DD SYSOUT=*"
QUEUE "//SYSIN DD *"
SELECT
  WHEN SSID = 'DB2P' THEN
     DO
     QUEUE "  OUTPUT LOCALP UNIT SYSDA"
     QUEUE "         DSNAME PROD.P.&DB.&TS.D&DATE.H&TIME"
     QUEUE "  OUTPUT RECOVP UNIT SYSDA"
     QUEUE "         DSNAME BRS.P.&DB.&TS.D&DATE.H&TIME"
     END
  WHEN SSID = 'DB2D' THEN
     DO
     QUEUE "  OUTPUT LOCALP UNIT SYSDA"
     QUEUE "         DSNAME DESA.A.&DB.&TS.D&DATE.H&TIME"
     END
  WHEN SSID = 'DB2T' THEN
     DO
     QUEUE "  OUTPUT LOCALP UNIT SYSDA"
     QUEUE "         DSNAME TEST.A.&DB.&TS.D&DATE.H&TIME"
     END
  OTHERWISE
     DO
     QUEUE "  OUTPUT LOCALP UNIT SYSDA"
     QUEUE "         DSNAME PROD.P.&DB.&TS.D&DATE.H&TIME"
     QUEUE "  OUTPUT RECOVP UNIT SYSDA"
     QUEUE "         DSNAME BRS.P.&DB.&TS.D&DATE.H&TIME"
     END
END
IF PART = '' THEN
QUEUE "  COPY   TABLESPACE "DATABASE"."NAME
ELSE
QUEUE "  COPY   TABLESPACE "DATABASE"."NAME" DSNUM "PART;
SELECT
  WHEN SSID = 'DB2P' THEN QUEUE "  COPYDDN(LOCALP) RECOVERYDDN(RECOVP)"
  WHEN SSID = 'DB2D' THEN QUEUE "  COPYDDN(LOCALP)"
  WHEN SSID = 'DB2T' THEN QUEUE "  COPYDDN(LOCALP)"
  OTHERWISE               QUEUE "  COPYDDN(LOCALP) RECOVERYDDN(RECOVP)"
END
QUEUE "         SHRLEVEL CHANGE"
QUEUE "         FULL YES"
QUEUE "         RESETMOD NO"
QUEUE "//"
QUEUE ""
ADDRESS TSO "EXECIO * DISKW JOB (FINIS"
ADDRESS TSO "EXECIO * DISKR JOB (STEM JOBCOPQ. FINIS"
"ISPEXEC TBCREATE TJOBCOP",
"NAMES(JOBCOP)",
"NOWRITE REPLACE"
DO I = 1 TO JOBCOPQ.Ø
```

```
   JOBCOP = JOBCOPQ.I
  "ISPEXEC TBADD  TJOBCOP"
END
RESCOPY_EXIT = 'N'
ADDRESS TSO "FREE DDNAME(JOB)"
ADDRESS ISPEXEC 'TBTOP   TJOBCOP';
ADDRESS ISPEXEC 'ADDPOP  ROW(3) COLUMN(16)';
ADDRESS ISPEXEC 'TBDISPL TJOBCOP PANEL(DB2TCOPJ)';
IF RC = 8 THEN
   DO
   RESCOPY_EXIT = 'S'
   END
ELSE
   DO
   Y = OUTTRAP(DELSUB.)
  "SUBMIT '"DSNJOB"'"
   Y = OUTTRAP('OFF')
       IF RC <> Ø THEN
          DO
          ADDRESS ISPEXEC "SETMSG MSG(DBCØ22)"
          RETURN
          END
   CALL GRABALOG
   END
ADDRESS ISPEXEC 'REMPOP';
"DELSTACK"
"ISPEXEC TBEND    TJOBCOP"
CALL LIBDEFPANEL;
CALL DELJOB;
RESCOPY_EXIT = 'S'
RETURN
/*-----------------------------------------------------------------*/
RESCHKP:
CALL DISUTIBMC;
IF STATUSRBMC = 'X' THEN RETURN;
CALL TERUTIIBMSTOP;
IF TYPE = 'INDEX' THEN CALL TRAEIXCREATOR;
"NEWSTACK"
"ISPEXEC ADDPOP ROW(3) COLUMN(16)"
CALL LIBDEFWINDJ;
DSNJOB   = USERID() || '.RES' || '.JOB'  || TIME('S')
ADDRESS TSO "ALLOC FILE(JOB) DATASET('"DSNJOB"') " ,
            "NEW CAT REUSE UNIT(SYSDA)" ,
            "LRECL(8Ø) BLKSIZE(2792Ø) RECFM(F B) SPACE(1,1) CYL"
             IF RC <> Ø THEN
                DO
                ADDRESS ISPEXEC "SETMSG MSG(DBCØ21)"
                RETURN
                END
JOBNAME = SUBSTR((USERID() || 'CK'),1,8)
```

```
QUEUE "//"JOBNAME" JOB (DB2),'OSORIO',MSGCLASS=X,"
QUEUE "// CLASS=S,MSGLEVEL=(1,1),NOTIFY=&SYSUID"
QUEUE "//STEP1 EXEC PGM=DSNUTILB,PARM='"SSID",,',REGION=ØM"
QUEUE "//SYSPRINT DD SYSOUT=*"
QUEUE "//UTPRINT  DD SYSOUT=*"
QUEUE "//SYSUDUMP DD SYSOUT=*"
QUEUE "//SORTOUT  DD DSN=&&SORTOUT,"
QUEUE "//         DISP=(NEW,DELETE,CATLG),UNIT=SYSDA,"
QUEUE "//         SPACE=(TRK,(1ØØ,1ØØ),,,ROUND)"
QUEUE "//SYSUT1   DD DSN=&&SYSUT1,"
QUEUE "//         DISP=(NEW,DELETE,CATLG),UNIT=SYSDA,"
QUEUE "//         SPACE=(TRK,(1ØØ,1ØØ),,,ROUND)"
QUEUE "//SYSERR   DD DSN=&&SYSERR,"
QUEUE "//         DISP=(NEW,DELETE,CATLG),UNIT=SYSDA,"
QUEUE "//         SPACE=(TRK,(1ØØ,1ØØ),,,ROUND)"
QUEUE "//SYSIN DD *"
IF TYPE = 'TABLESPACE' & PART  = '' THEN
QUEUE " CHECK DATA TABLESPACE "DATABASE"."NAME;
IF TYPE = 'TABLESPACE' & PART <> '' THEN
QUEUE " CHECK DATA TABLESPACE "DATABASE"."NAME" PART "PART;
IF TYPE = 'INDEX'      & PART  = '' THEN
QUEUE " CHECK INDEX ("IXCREATOR"."NAME")";
IF TYPE = 'INDEX'      & PART <> '' THEN
QUEUE " CHECK INDEX ("IXCREATOR"."NAME") PART "PART;
QUEUE "//"
QUEUE ""
ADDRESS TSO "EXECIO * DISKW JOB (FINIS"
ADDRESS TSO "EXECIO * DISKR JOB (STEM JOBCHKQ. FINIS"
"ISPEXEC TBCREATE TJOBCHK",
"NAMES(JOBCHK)",
"NOWRITE REPLACE"
DO I = 1 TO JOBCHKQ.Ø
   JOBCHK = JOBCHKQ.I
  "ISPEXEC TBADD  TJOBCHK"
END
RESCHKP_EXIT = 'N'
ADDRESS TSO "FREE DDNAME(JOB)"
"ISPEXEC TBTOP    TJOBCHK"
"ISPEXEC TBDISPL  TJOBCHK PANEL(DB2TCHKJ)"
IF RC = 8 THEN
   DO
   RESCHKP_EXIT = 'S'
   END
ELSE
   DO
   Y = OUTTRAP(DELSUB.)
  "SUBMIT '"DSNJOB"'"
   Y = OUTTRAP('OFF')
       IF RC <> Ø THEN
          DO
```

```
          ADDRESS ISPEXEC "SETMSG MSG(DBCØ22)"
          RETURN
          END
   CALL GRABALOG
   END
"DELSTACK"
"ISPEXEC REMPOP"
"ISPEXEC TBEND    TJOBCHK"
CALL LIBDEFPANEL;
CALL DELJOB;
RESCHKP_EXIT = 'S'
RETURN
/*-----------------------------------------------------------------*/
RESRBDP:
CALL DISUTIBMC;
IF STATUSRBMC = 'X' THEN RETURN;
CALL TERUTIIBMSTOP;
CALL TRAEIXCREATOR;
"NEWSTACK"
"ISPEXEC ADDPOP ROW(3) COLUMN(16)"
CALL LIBDEFWINDJ;
DSNJOB   = USERID() || '.RES' || '.JOB'  || TIME('S')
ADDRESS TSO "ALLOC FILE(JOB) DATASET('"DSNJOB"') " ,
            "NEW CAT REUSE UNIT(SYSDA)" ,
            "LRECL(8Ø) BLKSIZE(2792Ø) RECFM(F B) SPACE(1,1) CYL"
             IF RC <> Ø THEN
                DO
                ADDRESS ISPEXEC "SETMSG MSG(DBCØ21)"
                RETURN
                END
JOBNAME = SUBSTR((USERID() || 'RB'),1,8)
QUEUE "//"JOBNAME" JOB (DB2),'OSORIO',MSGCLASS=X,"
QUEUE "// CLASS=S,MSGLEVEL=(1,1),NOTIFY=&SYSUID"
QUEUE "//STEP1 EXEC PGM=AFRMAIN,REGION=ØM,"
QUEUE "// PARM='"SSID",,NEW,MSGLEVEL(1),,RDB2STAT(RW)'"
QUEUE "//SYSPRINT DD SYSOUT=*"
QUEUE "//UTPRINT  DD SYSOUT=*"
QUEUE "//SYSIN DD *"
IF PART  = '' THEN
QUEUE " RECOVER INDEX ("IXCREATOR"."NAME")";
IF PART <> '' THEN
QUEUE " RECOVER INDEX ("IXCREATOR"."NAME") DSNUM "PART;
QUEUE "         SORTDEVT SYSDA"
QUEUE "         SORTNUM  12"
QUEUE "         NOWORKDDN"
QUEUE "         ANALYZE  YES"
QUEUE "         REDEFINE YES"
QUEUE "//"
QUEUE ""
ADDRESS TSO "EXECIO * DISKW JOB (FINIS"
```

```
ADDRESS TSO "EXECIO * DISKR JOB (STEM JOBRBDQ. FINIS"
"ISPEXEC TBCREATE TJOBRBD",
"NAMES(JOBRBD)",
"NOWRITE REPLACE"
DO I = 1 TO JOBRBDQ.Ø
   JOBRBD = JOBRBDQ.I
  "ISPEXEC TBADD  TJOBRBD"
END
RESRBDP_EXIT = 'N'
ADDRESS TSO "FREE DDNAME(JOB)"
"ISPEXEC TBTOP    TJOBRBD"
"ISPEXEC TBDISPL  TJOBRBD PANEL(DB2TRBDJ)"
IF RC = 8 THEN
   DO
   RESRBDP_EXIT = 'S'
   END
ELSE
   DO
   Y = OUTTRAP(DELSUB.)
  "SUBMIT '"DSNJOB"'"
   Y = OUTTRAP('OFF')
       IF RC <> Ø THEN
          DO
          ADDRESS ISPEXEC "SETMSG MSG(DBCØ22)"
          RETURN
          END
   CALL GRABALOG
   END
"DELSTACK"
"ISPEXEC REMPOP"
"ISPEXEC TBEND    TJOBRBD"
CALL LIBDEFPANEL;
CALL DELJOB;
RESRBDP_EXIT = 'S'
RETURN
/*-------------------------------------------------------------*/
RESRECP:
IF TYPE = 'INDEX' THEN
   DO
   CALL RESRBDP
   RETURN
   END;
CALL DISUTIBMC;
IF STATUSRBMC = 'X' THEN RETURN;
CALL TERUTIIBMSTOP;
CALL LIBDEFWINDJ;
"NEWSTACK"
E = ''
RESREC1_EXIT = 'N'
"ISPEXEC ADDPOP ROW(2) COLUMN(2)"
```

```
"ISPEXEC DISPLAY PANEL(DB2TRECO)"
IF   RC = 8 THEN
     DO
     RESREC1_EXIT = 'S'
     R = NRES + 1
     END
ELSE DO
     IF E = '1' THEN
        DO
        DO I = 1 TO 9
           SYSIN.I = ''
        END
        CALL ALLOCSYS;
```

Editor's note: this article will be concluded next month.

Carlos German Osorio Montoya
Database Administrator
BBVA Banco Continental (Peru)

Safe Software has announced that it has added support for DB2 Universal Database Spatial Extender to its complete product line, including its Feature Manipulation Engine (FME), FME Objects, SpatialDirect, and FME SDP Server. The result is a direct connection between DB2 Universal Database and over 100 FME-supported GIS (Geographic Information System), CAD and database formats.

DB2 UDB with the DB2 Spatial Extender enables businesses to store, manage, and analyse spatial data (information about geographic features) with traditional business data. Customers can generate, analyse, and exploit spatial information about geographic features, such as the locations of office buildings or the size of flood zones, and integrate that information with any business data to add another element of business intelligence to the enterprise.

For further information contact:
Safe Software, Suite 2017, 7445 132nd Street, Surrey, BC, Canada, V3W 1J8.
Tel: (604) 501 9985.
URL: http://www.safe.com/products/fme/index.htm.

* * *

IBM has announced Version 8 (technically, Version 8.1.4) of its DB2 Everyplace middleware to enable the flow of information to handheld computing devices, and has also introduced an SMB-focused mid-market version of the product, DB2 Everyplace Express.

The new release features changes to simplify both applications development and administration. There's now support for Microsoft's .Net Framework and .Net Compact framework, and there are significant improvements to the Java side of the product. New support bundles IBM's J9 Java Virtual Machine for improved connectivity and performance to Java databases to help developers build mobile applications faster.

For further information contact your local IBM representative.
URL: http://www-306.ibm.com/software/data/db2/everyplace.

* * *

ACCPAC International (a CA subsidiary) has announced Version 5.6 of ACCPAC CRM, its customer relationship management application.

The new version includes enhancements to improve integration capabilities with other business management applications, including DB2, Lotus Notes, and Microsoft Outlook.

For further information contact:
ACCPAC, 6700 Koll Center Parkway, Third Floor, Pleasanton, CA 94566, USA.
Tel: (925) 461 2625.
URL: http://www.accpac.com/products/crmsfa/.

* * *