99

DB2

January 2001

update

In this issue

3 24x7 DB2 applications – tips for good design

12 Cloning a DB2 subsystem using SnapShot

23 DB2 Version 6 stored procedures migration issues

28 Utility for generating recovery jobs using the REXX SQL interface – part 2

40 DB2 REXX Language Support

48 DB2 news

DB2 Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38342
From USA: 01144 1635 38342
E-mail: trevore@xephon.com

North American office

Xephon
PO Box 350100
Westminster, CO 80035-0100
USA
Telephone: 303 410 9344

Subscriptions and back-issues

A year's subscription to *DB2 Update*, comprising twelve monthly issues, costs £255.00 in the UK; $380.00 in the USA and Canada; £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 1997 issue, are available separately to subscribers for £22.50 ($33.50) each including postage.

DB2 Update on-line

Code from *DB2 Update* can be downloaded from our Web site at http://www.xephon.com/db2update.html; you will need the user-id shown on your address label.

Editor

Trevor Eddolls

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, and other contents of this journal before making any use of it.

Contributions

Articles published in *DB2 Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. To find out more about contributing an article, without any obligation, please contact us at any of the addresses above and we will send you a copy of our *Notes for Contributors*, or you can download a copy from www.xephon.com/contnote.html.

Printed in England.

24x7 DB2 applications – tips for good design

This article is my way of helping project teams in designing DB2 applications intended to work 24x7. Such applications are very important, and there are many ways to bring data to the end user, apart from traditional mainframe terminals. Some of the new technologies, like the Internet, just speed it up. As well as the Internet, there are also ATMs, answer machines, POSs, etc.

All of these erase the classical division between day and night jobs in the life of an application. With these tips and advice, I hope to help those teams to keep their attention on the key issues, which will ensure such applications can work 24x7.

The article is based on designing in a DB2 Version 5, CICS, batch, and PL/I environment.

HOW TO AVOID APPLICATIONS STOPPING BECAUSE THERE IS A FATAL SQL ERROR

When your application is meant to work 24x7, there is no excuse for any errors appearing – and there are many errors that can appear without your knowledge or you expecting them. Such errors are predicted by DB2 and placed at the end of the SQL error array. Any hardware errors that may occur are outside the scope of this article.

So what to do to avoid such errors, or, if you can't, what to do to ensure your application doesn't stop?

Although not the only solution, probably the best one and the easiest one for implementation is redundancy of your critical tables – creating tables that are reachable 24x7. It means we have to ensure that all our activities are carried out on both the original and duplicate tables. Of course, there are no differences between the original and the duplicate – they are, in fact, the first referenced and second referenced tables. See the code below for examples of a module for retrieving data from such tables.

Also, if you have frequent batch inserts of data during the on-line

period, it is a good idea to think about splitting that table in two – one only for changes from CICS and another for changes from the batch environment. Generally errors in batch tables do not occur very often because you can control them all the time. However, if an error does occur, you must deal with the recovery of that table and your application can continue. The data will be basically correct because you have control over the batch process during the recovery period. If you wish, you can always alert users about it.

As everyone can see, there is an overhead with tables, eg the space for them and of course the DBA's and programmer's efforts to ensure redundancy. As a result, when one of the fatal errors appears on a table, there is no harm done because our programs can 'identify' such a situation and behave as if nothing is happening.

Our end users don't know anything bad is going on and, most importantly, our data is correct and secure. But what about ourselves? When and in what way will we find out about an error? As shown below, all our modules must have some return code value for this situation, and, in the calling program, we must find a way to ALERT the DBA or other relevant authorities about it. Here are some possibilities for this:

- There are always some users who work for your company. For these users, we can show them the warning message which will be passed to the DBA and the others. The real end users, who are not employed by the company, do not need to know about anything.
- We can use the 'WRITE TO OPERATOR' command, but be careful about the frequency of warning message appearing on the console.
- We can write a separate transaction, which will be started on a specific terminal with the 'START TRAN' command.

THE PREREQUISITES FOR QUICK AND ACCURATE DATABASE RECOVERY

So, we have a duplicate table and all of our modules with a SQL DDL

statements are aware of it. What would happen when a fatal error appears on one of the tables? Of course you have to perform a recovery as quickly as possible. This means that you will restore data in the destroyed tables from the most recent copy, and use a DB2 utility to get as much data as possible from the DB2 log files. Every recovery process is a time-consuming operation.

So what should you do to be prepared for such an unpleasant situation? The answer is:

- Make copies of each database at least once a week, or more often if there are huge changes to tables over a single working day. In this way, you will cut down the time needed for recovery.
- Always take a control point with the QUIESCE utility to ensure recovery from an ABEND of a batch job. Some people use full copy, on tapes usually, but it is a time-consuming process for both taking and restoring data, while with the QUIESCE utility the DBA is working from his DB2 log, which is much more secure.
- Put every table in a separate tablespace because, if it is not, when doing recovery on one, the others are 'closed' – which is unacceptable for 24x7.

The scenario for the recovery process can be something like this (T*n* is a moment in time):

- T0 – everything is OK.
- T1 – a fatal error appears.
- T2 – we become aware of the fatal error.
- T3 – start of recovery process.
- T4 – end of recovery process.
- T5 – full copy of correct data to restored table.
- T6 – end of copy and application continues to work properly on both tables.

T3 has to be as quick as possible, so the way we ALERT the DBA is very important. The moment to start T5 has to be chosen carefully. It

does not have to be just after T4, but when such an operation will cost the least. If you do not have the time window necessary for T5, then warn the end users about a temporary break in service.

But what do you do when these suggestions cannot be implemented because there are fatal errors on both tables? or you can't recover from the last copy? or there is an error in the DB2 log so you have lost all changes since the last copy? My recommendation is to make your own log file for every change to the tables, using a VSAM ESDS file for example, where you can write all the changes to the tables. For recovery purposes you need one batch job with start and end time as parameters, which would supply all the changes to the tables, no matter whether the DB2 log file is OK, and a second batch job to implement these changes on the tables.

ENSURING FULL CONCURRENCY FOR PRODUCTION APPLICATIONS

Without full concurrency there is really no on-line application, not to mention a 24x7 application. Concurrency means that many different processes with fully secure data integrity can process the data, and every request will be correctly processed.

How do you ensure full concurrency? The most important aspect about concurrency is record-locking. Every time there is an INSERT, UPDATE, or DELETE SQL statement in process, your program has to demand locking on the chosen records so data integrity is ensured. Here are some tips for locking:

- Besides all programming there is one more thing that DB2 programs demand – binding, or, simply, the way to prepare SQL statements from a program to execute at run time. There are some parameters, which are strictly used for locking purposes. ISOLATION LEVEL (IS) is used to tell DB2 how to lock all programs that are bound to a particular plan. Values that reserve concurrency are READ STABILITY (RS) and CURSOR STABILITY (CS). Values for ACQUIRE and RELEASE parameters are those that decide *when* DB2 takes and releases resources needed for the execution of SQL statements. For programs that only read DB2 tables, avoid any locking.

- Because when a record is locked, no one can retrieve it, logically locking has to be for a short time. So, you have to avoid time-consuming SQL statements that:
  - use OR and LIKE operators (use IN and relational operators instead).
  - use ORDER BY statements, because sorting is involved, and we all know what that means.
- To speed up your SELECT statements, try using the OPTIMIZE parameter to force DB2 to use indexes as the best access path.
- If your application needs massive inserts in the tables, always use a batch program with INSERTs instead of the LOAD utility, because LOAD makes tables inaccessible for updating. With the Internet and offices all over the world, there is no day and night, so there is no free time during 'our' night.

HOW TO ENSURE QUICK RESPONSE TO NEW DEMANDS

Critical moments in the life of a 24x7 application are:

- Discovering the errors (fatal SQL codes) and recovering from them.
- Including new features in an application as the result of users' demands.

Both of these operations are time-consuming, so we have to find a way to eliminate any delay or to cut it down as much as we can.

It must be realized that the test phase is much more important in 24x7 application than any other. There are no excuses for SQL errors less than –900 in the real production environment. So test for as long as you can until you are sure that everything is OK. Transactions need to use the HANDLE ERROR CONDITION command for any other errors, which cannot be traced with standard handling.

Including new features in a program is a very sensitive issue because someone out there expects a quick response without any interruptions. There are three steps for putting a program into production:

- Compiling and linking (modules and called programs).
- Binding (plans and packages).
- Taking new versions (only in a CICS environment).

We have to find a way to, maybe, skip some of these steps.

Here are some tips:

- Because demands can come frequently and for random enhancements, it is a good idea to organize your programs as one main one with many called modules. As you will see below there are other reasons for this idea.
- Because there is a difference in using some PL/I features, it is good to make separate modules for batch and CICS environments, and when you do, you can:
  - Use LINK instead of the CALL command in CICS. In this way you do not need to compile and link all programs, but only take a new version of the module program. Also, trapping the errors is much easier in a separate program than in a linked module
  - Use FETCH and RELEASE commands in batch to make modules really external. Unfortunately, you cannot use these commands in a CICS environment.
- The most critical step is binding. To avoid –805 SQL errors, bind all modules as packages so you do not need to bind plans for any other program, which also bind that member. DB2 does it for you, and saves any additional time.

As with recovery, any new features have to be installed in some ‘dead time’, if there is any such time, and with an announcement to the end users in advance.

Below is an example of how to use redundant tables, and how to handle possible errors. CICS LINK program TLINK calculates account balances. Tables TERD01 and TERD02 are redundant and internal module SQLERR cares of the error level. Table TERD03 is a batch table.

```
TLINK: PROC(POINT) OPTIONS(MAIN);

%INCLUDE SQLCA;

/* COMMAREA FOR TRANSACTION TLINK                                    */
DCL POINT        PTR;
DCL 1 LINKAREA                    BASED(POINT),
      2 ACCOUNT  BIN FIXED(31),   /*  INPUT VARIABLE                 */
      2 SUMACC   DEC FIXED(15),   /* OUTPUT VARIABLE                 */
      2 CODE     BIN FIXED(15),   /* RETURN CODE                     */
                                  /*  Ø - OK                         */
                                  /*  1 - INFO ERROR ON TABLES 1 AND 3*/
                                  /* -1 -    SQL ERROR ABOVE -ØØØ    */
                                  /* -2 - SEVERE ERROR ON TABLE 1 OR 2*/
      2 TEXT     CHAR(2Ø);        /* TABLE NAME WITH SQLCODE         */

/* LOCAL VARIABLES IN PROGRAM                                        */
DCL SUMACC1      DEC FIXED(15);
DCL SUMACC2      DEC FIXED(15);
DCL IND          BIN FIXED(15);

/* LET SUM RECORDS FROM THE ONLINE TABLE                             */
EXEC SQL SELECT SUM( AMMOUNT )
         INTO :SUMACC1 :IND
         FROM TERDP.TBTERDØ1
         WHERE ACCOUNTNO = :ACCOUNT;
/* CHECK IF THE FIRST TABLE IS OK                                    */
CALL SQLERR(SUMACC1, 1, ERRCODE);

/* MAINTAIN RETURN CODE FROM PROGRAM                                 */
CODE=ERRCODE;
/* IF THERE IS AN ERROR BELOW -ØØØ TRY REDUNDANT TABLE               */
IF ERRCODE > Ø
THEN DO;
       IND=Ø;
       EXEC SQL SELECT SUM( AMMOUNT )
                INTO :SUMACC1 :IND
                FROM TERDP.TBTERDØ2
                WHERE ACCOUNTNO = :ACCOUNT;
       CALL SQLERR(SUMACC1, 2, ERRCODE);
       /* ALWAYS KEEP LAST ERROR                                    */
       IF CODE=Ø THEN CODE=ERRCODE;

       /* IF ANY ERROR OCCURS ON SECOND TABLE RETURN                */
       IF ERRCODE < Ø THEN GOTO ENDPROC;
     END;

/* IF THERE IS ANY OTHER ERROR RETURN                                */
IF ERRCODE < Ø THEN GOTO ENDPROC;
```

```
/* NOW SUM RECORDS FROM batch TABLE                                    */
IND=Ø;
EXEC SQL SELECT SUM( AMMOUNT )
         INTO :SUMACC2 :IND
         FROM TERDP.TBTERDØ3
         WHERE ACCOUNTNO = :ACCOUNT;
CALL SQLERR(SUMACC2, 3, ERRCODE);
IF CODE=Ø THEN CODE=ERRCODE;
IF ERRCODE < Ø THEN GOTO ENDPROC;
               ELSE SUMACC2=Ø;
/* CALCULATE REAL SUM ON THE ACCOUNT                                   */
SUMACC = SUMACC1 + SUMACC2;
ENDPROC:

EXEC CICS RETURN;

/* INTERNAL MODULE THAT CHECKS SQLCODE ON TABLES                       */
SQLERR: PROC( VAR, TABNO, ERRCODE );

DCL VAR     DEC FIXED(15);        /* FIELD FOR SUMMING                 */
DCL TABNO   DEC FIXED(1);         /* TABLE NUMBER                      */
DCL ERRCODE BIN FIXED(15);        /* RETURN CODE FROM MODULE           */
DCL PICCODE PIC'----9';           /* SQLCODE IN PIC FORMAT             */
DCL NAME(3) CHAR(6)               /* ARRAY OF TABLE NAMES              */
            INIT( 'TERDØ1', 'TERDØ2', TERDØ3');

ERRCODE=Ø;
SELECT (SQLCODE);
  WHEN( 1ØØ ) VAR = Ø;
  WHEN( Ø )   IF IND < Ø THEN VAR = Ø;
  OTHERWISE   DO;
                PICCODE = SQLCODE;
                /* IF THERE IS AN ERROR BELOW -9ØØ                    */
                /*    IF ON TABLE 1 OR 3 WE CAN CONTINUE WITH ALERT   */
                /*    IF ON TABLE 2 WE CAN NOT CONTINUE AT ALL        */
                /* ELSE THERE IS PROGRAM ERROR, MAYBE -8Ø5 OR -818    */
                IF SQLCODE < -9ØØ
                THEN DO;
                       IF TABNO = 1 | TABNO=3
                          THEN ERRCODE =  1;
                          ELSE ERRCODE = -2;
                     END;
                ELSE ERRCODE = -1;
                TEXT = NAME( TABNO ) ||', SQLCODE :' || PICCODE;
              END;
END;
END SQLERR;

END TLINK;
```

Below is a CICS transaction program, which calls TLINK. It shows how to ALERT the authorities about the error. A internal ALERT is raised whenever an error exists and an external one only when both tables are corrupted.

```
...

/* COMMAREA FOR TLINK PROGRAM                                          */
DCL 1 LINKAREA                    BASED(POINT),
      2 ACCOUNT  BIN FIXED(31),
      2 SUMACC   DEC FIXED(15),
      2 CODE     BIN FIXED(15),
      2 TEXT     CHAR(2Ø);
/* VARIABLES FOR DETERMINING WHAT TYPE OF USER IS CURRENT ONE          */
DCL TYPEUSER     DEC FIXED(1);
DCL EXTUSER      DEC FIXED(1)    INIT(1); /* OUTSIDE USER             */
DCL INTUSER      DEC FIXED(1)    INIT(2); /*  INSIDE USER             */

/* YOU CAN KNOW WHICH USERNAME IS FROM THE COMPANY AND WHICH IS NOT    */
IF some_condition
   THEN TYPEUSER=EXTUSER;
   ELSE TYPEUSER=INTUSER;
ACC = account_number;
SUM = Ø;
CODE = Ø;
TEXT= ' ';
EXEC CICS LINK PROGRAM('TLINK') COMMAREA(LINKAREA);

/* IF THERE IS FATAL ERROR ON FIRST OR batch TABLE                     */
/*    ALERT INTERNAL USERS AND CONTINUE                                */
IF CODE = 1 & TYPEUSER = INTUSER
   THEN MESSAGEO = 'CALL nnn , '|| TEXT;

/* IF BOTH TABLES ARE DAMAGED OR THERE IS PROGRAM ERROR                */
/*    ALERT ALL AND QUIT                                               */
IF CODE < Ø
THEN DO;
       IF TYPEUSER=INTUSER
          THEN MESSAGEO = 'CALL nnn, '|| TEXT;
          ELSE MESSAGEO = 'DATA IS TEMPORARILY INACCESSIBLE';
       EXEC CICS SEND MAP('MAP') MAPSET('MAPSET') DATAONLY FREEKB;
       EXEC CICS RETURN TRANSID('TRAN') COMMAREA(TRANAREA);
     END;
```

Predrag Jovanovic
Project Developer
Postal Savings Bank (Yugoslavia)

Cloning a DB2 subsystem using SnapShot

We developed this process to perform a DB2 subsystem copy in order to effect a system refresh of a SAP R/3 DB2 system, which contains thousands of tablespaces. We have a continuing need to refresh our SAP systems, so we wanted a quicker way to copy all the data from one DB2 subsystem to another (SAP is the only application running on the subsystem), because the traditional COPY/RECOVER method takes several days to complete and requires an extensive outage of the application.

THE SNAPSHOT FEATURE

This process uses the SnapShot feature of the IBM RAMAC Virtual Array Storage to clone a DB2 subsystem to an existing subsystem. The purpose of SnapShot is to make a rapid 'virtual copy' of DASD volumes or individual datasets. (For additional information on SnapShot, see SC26-7173 *IBM RAMAC SnapShot for MVS/ESA: Using Snapshot*.) With SnapShot, the entire process still takes us a matter of hours (12-15) to complete, but the downtime for the source DB2 is reduced to only 10 minutes.

(Note that it is supposedly possible under DB2 Version 6 to make a SnapShot copy without taking DB2 down, by using the '-SET LOG SUSPEND' command. This command was added to Version 6 by PTF UQ36695. The application will still be unavailable during the Snap, however.)

THE PEOPLE

In our shop, performing the process is a team effort involving the MVS systems programmer, the DB2 systems programmer, and the SAP DBAs. The MVS systems programmer makes dataset copies using the SnapShot feature, the DB2 systems programmer causes DB2 to recognize the system dataset copies as a viable DB2 subsystem, and the DBAs reset the high-level qualifier in all the stogroups so that DB2 will recognize the copied application data as viable.

THE PROCESS

The process as given here is for data sharing subsystems; that is, it is set up to copy a two-way data sharing subsystem to another two-way data sharing subsystem. Using a similar procedure, our team has also copied from data sharing to non-data sharing (although we end up with a data sharing target), and from one-way data sharing to two-way data sharing. (We merely re-run the original install jobs for the second member, after we finish copying the first member). If your DB2s are not data sharing, then you will be able to simplify the procedure somewhat; for instance, step 7 may be omitted.

Note that a similar process could be used to clone an existing DB2 to a completely new DB2. There would be some additional tasks involved as with any new DB2 subsystem – define a new ICF catalog, define new catalog aliases, define new SMF groups, define new SMF profiles with appropriate authority, create new PROCLIB members, make definitions in PARMLIB, and so forth.

THE SNAP

To facilitate the Snap, we have defined the DB2 datasets – and this includes the DB2 system datasets (BSDS, log datasets, catalog, directory) as well as the application data – to be managed by SMS and isolated as to SMS storage group (that is, there is a separate SMS storage group for each DB2 subsystem). In addition, we have defined a separate ICF catalog for each subsystem.

1 First, we stop the target DB2 subsystem and Snap (copy) its volumes, as a back-up measure in case we need to restore the target to its pre-cloned state.

2 The 'old' target datasets have to be deleted, and removed from the ICF catalog. We have a job set up to issue a TSO DELETE for each entry – this job takes about 2.5 hours' elapsed time. Alternatively we can get a list of aliases pointed to by the target catalog, connect the catalog from the master catalog, re-initialize all the target volumes, define a new ICF catalog, and redefine the aliases. This procedure takes about 30 minutes.

3 Stop the source DB2 subsystem MODE=QUIESCE, and do a SNAP of each volume with COPYVOLID(Y) to back up the source volumes. The copy volumes will go off-line. After the Snap of the source system volumes, data volumes, and ICF catalog, which takes about 10 minutes, we can bring the source DB2 subsystem back up.

4 Now we have to create new target volumes from the source back-ups; we do this from another OS/390 image because, on the image where we began working, there are outstanding ENQs on the source dataset names because the source DB2 is active. On the second image we Snap each source-copy volume by dataset, renaming each dataset to the new (ie the target's) high-level qualifier.

AFTER THE TARGET IS DOWN

5 Back up the following run-time target datasets, and then copy them from the source subsystem: SDSNEXIT, SDSNLOAD, CLIST, ISPMLIB, ISPPLIB, and ISPSLIB. This is a separate step because our run-time datasets do not follow the same naming convention as the rest of the DB2 datasets and hence do not participate in the Snap copy. Do this for all data sharing members. Also make a back-up copy of any subsystem-specific vendor load libraries, such as a BMC load library. You may also want to make a back-up copy of the target *.NEW.SDSNSAMP dataset since it will be overlaid later, in the SMP/E cloning.

6 Copy the target ZPARMS member DSNZxxxx from the 'old' target SDSNEXIT dataset (which is still unmodified) into the 'new' target run-time SDSNEXIT dataset. Do this for all data sharing members.

7 Clean up the SCA of the target system by issuing the following console command:

```
SETXCF FORCE,STRUCTURE,STRNAME=targetDCAT_SCA
```

(See GC28-1779 *OS/390 MVS Setting Up a Sysplex*.) This was not an obvious thing to do, but the first time(s) we performed the process, we received several messages DSNB232I

UNEXPECTED DATA SET LEVEL ID ENCOUNTERED. IBM Level 2 told us: “When you copy to this new subsystem member be sure to Force the SCA. The RBA for a dataset is found in the DBET of each member, the SYSLGRNX, and the group DBET. The other option is to use DSN1COPY to RESET the RBAs for each dataset.”

AFTER THE SNAP TO TARGET HAS COMPLETED

8 Rename the target BSDSs and active (LOGCOPYx) datasets to their correct names. This is necessary because these names have the data sharing member name embedded in them, but the Snap copy changes only the high-level qualifier, so we end up with dataset names of the form ‘targetDCAT.source.BSDSxx’ and ‘targetDCAT.source.LOGCOPYx.DS0y’. Repeat for other data sharing members.

9 Run a Print Log Map of the new target BSDS and examine the output to find the RBAs of the Active logs. Then run the DSNJU003 utility to specify NEWCAT, DELETE all of the old Active logs, and add (ie NEWLOG) the new Active logs. (There is an example job below.) Note that the delete step will get RC=4. The set-up for the utility job could be automated, but we have not done so, and it remains a somewhat tedious manual process to locate the required 12-digit RBAs in the Log Map printout and copy them to the DSNJU003 job input. Repeat for other data sharing members.

10 Run the DDF command in the DSNTLOG ‘Change Log Inventory’ step of job DSNTIJUZ. (Don’t bother to try to run the GROUP statement, because it won’t work; see APAR PN75791.) Repeat for other data sharing members.

11 Edit (using File-AID or some other VSAM editor) the new target BSDSs to change ‘source_ssid’ to ‘target_ssid’ in the Data Sharing records at the end of the BSDS. Take care not to change the archive log names located elsewhere in the BSDS. We originally planned to run just the Change Log Inventory GROUP statement at this point but we found that this cannot be done, as

indicated in step 10. Repeat for other data sharing members.

12 Start the target subsystem '-target START DB2'. Repeat for other data sharing members.

AFTER THE TARGET DB2 IS UP

13 Run the install job (DSNTIJTM) from your target SDSNSAMP, which deletes and redefines the WRK databases. (These have to be reallocated because the copied WRK files end up with strange names just like the BSDSs do.) This job should be modified to STOP/DROP database 'WRKsource' instead of 'WRKtarget'; also it should delete the targetDCAT.xxx.WRKsource.xxx datasets. Repeat for other data sharing members.

14 Rebind the plans for any third-party DB2 utilities you may have, as required.

15 Re-install any products that may have been 'lost'; that is, products which were installed on the target DB2, but were not installed on the source DB2, and which you wish to continue to use on the target DB2.

SET NEW HIGH-LEVEL QUALIFIER IN ALL TABLESPACES

Since we have only DB2-managed objects, we use the procedure given in the *DB2 Administration Guide* under the subhead *Changing DB2-managed objects to use the new qualifier*.

16 Stop all tablespaces '-STOP DATABASE(xxxx) SPACENAM(*)'. This job takes perhaps 1.5 hours to run, on account of the number of databases – almost 10,000 – and the fact that it is not possible to say '-STOP DATABASE(*)'.

17 Run jobs to set the new qualifier in all the tablespaces. These jobs do the following:

   i Convert to user-managed datasets with the USING VCAT clause of the ALTER TABLESPACE and ALTER INDEX statements. (Be sure that DSNDB04 is stopped.)

   ii Drop the storage groups 'DROP STOGROUP xxxxxxxx'.

iii Recreate the storage groups using the correct volumes and the new alias 'CREATE STOGROUP xxxxxxxx VOLUMES(*) VCAT zzzzzzzz'.

iv Convert the datasets back to DB2-managed by using the new storage groups 'ALTER TABLESPACE ddddd.ttttt USING STOGROUP xxxxxxxx PRIQTY ppppp SECQTY sssss' (there is an example job below).

It is not necessary to reset the qualifier for DSNDB01 or DSNDB06. Be sure that PTF UQ44954 is applied to DB2 V6 before attempting to run step 17(iv).

18 Start all tablespaces '-START DATABASE(*) SPACE(*)'. This is much simpler than step 16 because it *is* possible to start all of the databases with one command. The 'new' DB2 is now ready for meaningful work.

CLONING THE SMP/E ENVIRONMENT

Our SMP/E cloning process is independent of the SnapShot process, because of different naming conventions used for SPM/E datasets, etc. There is probably a defined SMP/E procedure for doing this, but we chose to use a dataset copy/edit technique similar to the one we are using with SnapShot. (Note: the only datasets that we copied in step 5 were DB2 run-time datasets. None of the DB2 install datasets have been copied yet, up to this point.)

Our SMP/E environment is such that each DB2 data sharing group has its own CSI dataset and its own set of libraries, although all DB2s are installed in the same global CSI. We use DFDSS to copy the source SMP/E datasets (SMPLTS, SMPSTS, etc) and the 'DLIB' and 'Target' DB2 datasets (source.V61.SDSN* and source.V61.ADSN*). There are about 30 of these. At the same time we make use of the Rename function of DFDSS to change the dataset prefix:

```
COPY DATASET (INCLUDE(SMPE.DB2.V61.source.**)) -
  RENAMEU ((SMPE.DB2.V61.source.**,SMPE.DB2.V61.target.**))
COPY DATASET (INCLUDE(source.V61.**)) -
  RENAMEU (target)
```

The source datasets should be recalled prior to the copy otherwise DFDSS just ignores them.

After the copy/rename job is run, we edit the target DLIB CSI and target CSI datasets to change the *zone names* and the dataset *prefixes* (such as *source*.V61.xxxxxxxx) to match the target DB2.

We use File-Aid Edit for this. (Note that all the zone names and most of the dataset names are in record keys, which are protected by File-Aid. It is necessary to enter U to unprotect before editing, and then P to protect.)

It is not advisable to postpone the SMP/E cloning just because it is not a requirement for bringing up and using the target DB2. On one occasion we delayed this step for a day or two, and inadvertently applied maintenance to the source DB2 in the meantime. Afterwards we could still copy the source SMP/E datasets, but they were no longer in sync with the target DB2's run-time datasets.

AND WE'RE DONE

There is additional application-specific DBA work to be done in the target R/3 system at this point before it can be turned over to the user community, but that is beyond the scope of this discussion.

Note that the target DB2, being a clone of the source DB2, will use the source DB2's archive log numbering from this time forward. So far we have not found this to be a problem.

RESOURCES

We had very little detailed documentation to guide us in developing this methodology, although there is a general discussion of *Cloning DB2 data using SnapShot* in the IBM redbook SG24-5333 *Using RVA and SnapShot for Business Intelligence Applications with OS/390 and DB2*; also there is a section on *Cloning DB2 Data* in the redbook SG24-2241 *Implementing Snapshot.* In addition, we found that after the data had been copied, we could/should follow the steps given in the *DB2 Administration Guide* under *Changing the High-Level Qualifier for DB2 Data Sets* starting on page 2-139 (V5) or page 180 (V6).

SAMPLE JOBS

```
//BSDSLOG   JOB  (11111,222),'BYARS',MSGCLASS=O
//* SSCOPY STEP 9 -- FIX ACTIVE LOG NAMES IN BSDS
//* COPY PRD TO QAS
//LISTBEF  EXEC PGM=DSNJU004  LIST BSDS BEFORE CHANGES
//STEPLIB   DD  DSN=SYS1.QAS1.DSNEXIT,DISP=SHR
//          DD  DSN=SYS1.QAS1.DSNLOAD,DISP=SHR
//SYSUT1    DD  DSN=QASDCAT.QAS1.BSDS01,DISP=SHR
//SYSPRINT  DD  SYSOUT=*
//*
//REPLLOG  EXEC PGM=DSNJU003
//STEPLIB   DD  DISP=SHR,DSN=SYS1.QAS1.DSNEXIT
//          DD  DISP=SHR,DSN=SYS1.QAS1.DSNLOAD
//SYSUT1    DD  DISP=OLD,DSN=QASDCAT.QAS1.BSDS01
//SYSUT2    DD  DISP=OLD,DSN=QASDCAT.QAS1.BSDS02
//SYSPRINT  DD  SYSOUT=*
//SYSUDUMP  DD  SYSOUT=*
//SYSIN     DD  *
NEWCAT         VSAMCAT=QASDCAT
*
DELETE          DSNAME=PRDDCAT.PRD1.LOGCOPY1.DS01
DELETE          DSNAME=PRDDCAT.PRD1.LOGCOPY1.DS02
DELETE          DSNAME=PRDDCAT.PRD1.LOGCOPY1.DS03
DELETE          DSNAME=PRDDCAT.PRD1.LOGCOPY1.DS04
DELETE          DSNAME=PRDDCAT.PRD1.LOGCOPY2.DS01
DELETE          DSNAME=PRDDCAT.PRD1.LOGCOPY2.DS02
DELETE          DSNAME=PRDDCAT.PRD1.LOGCOPY2.DS03
DELETE          DSNAME=PRDDCAT.PRD1.LOGCOPY2.DS04
*
*  REDEFINE COPY1 LOGS:
NEWLOG          DSNAME=QASDCAT.QAS1.LOGCOPY1.DS01,
                COPY1,
              STARTRBA=004063C53000,
                ENDRBA=00406AC61FFF
NEWLOG          DSNAME=QASDCAT.QAS1.LOGCOPY1.DS02,
                COPY1,
              STARTRBA=00406AC62000,
                ENDRBA=004080BF1FFF
NEWLOG          DSNAME=QASDCAT.QAS1.LOGCOPY1.DS03,
                COPY1,
              STARTRBA=004080BF2000,
                ENDRBA=004096B81FFF
NEWLOG          DSNAME=QASDCAT.QAS1.LOGCOPY1.DS04,
                COPY1,
              STARTRBA=004096B82000,
                ENDRBA=0040ACB11FFF
*  REDEFINE COPY2 LOGS:
NEWLOG          DSNAME=QASDCAT.QAS1.LOGCOPY2.DS01,
                COPY2,
```

```
              STARTRBA=004063C53000,
                ENDRBA=00406AC61FFF
NEWLOG          DSNAME=QASDCAT.QAS1.LOGCOPY2.DS02,
                COPY2,
              STARTRBA=00406AC62000,
                ENDRBA=004080BF1FFF
NEWLOG          DSNAME=QASDCAT.QAS1.LOGCOPY2.DS03,
                COPY2,
              STARTRBA=004080BF2000,
                ENDRBA=004096B81FFF
NEWLOG          DSNAME=QASDCAT.QAS1.LOGCOPY2.DS04,
                COPY2,
              STARTRBA=004096B82000,
                ENDRBA=0040ACB11FFF
//*   LIST BSDS AFTER CHANGES
//LISTAFT  EXEC PGM=DSNJU004
//STEPLIB   DD  DSN=SYS1.QAS1.DSNEXIT,DISP=SHR
//          DD  DSN=SYS1.QAS1.DSNLOAD,DISP=SHR
//SYSUT1    DD  DSN=QASDCAT.QAS1.BSDS01,DISP=SHR
//SYSPRINT  DD  SYSOUT=*
```

Note that, in practice, the following job stream consists of several jobs that are run separately.

```
//ALTERJ   JOB (11111,222),'DBAS',REGION=128M,MSGCLASS=O,CLASS=A
//*-------------------------------------------------------------------
//* SSCOPY STEP 17 - SET NEW QUALIFIER IN ALL TABLESPACES
//*-------------------------------------------------------------------
//*-------------------------------------------------------------------
//*A. ALTER TABLESPACES TO USER MANAGED
//*-------------------------------------------------------------------
//ALTER   EXEC DB2TMP,SYSTEM=QASD
//SYSTSIN  DD *
 DSN SYSTEM(QASD)
    RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA61) -
      LIB('QASD.V61.RUNLIB.LOAD')
//SYSIN    DD *
SET CURRENT SQLID= 'SAPR3';
 ALTER TABLESPACE A000#AAX.VIEA02 USING VCAT QASDDB              ;
 ALTER TABLESPACE A000#ABN.ZU008 USING VCAT QASDDB               ;
 ALTER TABLESPACE A000#AD6.BTXAUTH USING VCAT QASDDB             ;
      ....                          ....
 ALTER TABLESPACE U010#QZ8.ZZCONST$ USING VCAT QASDDB            ;
 ALTER TABLESPACE U010#WLC.Y#PERF# USING VCAT QASDDB             ;
 ALTER TABLESPACE U010#Z9D.ZZCONSA$ USING VCAT QASDDB            ;
 ALTER TABLESPACE BMCACP.BMCTPART USING VCAT QASDCAT             ;
 ALTER TABLESPACE BMCACP.BMCTSTAT USING VCAT QASDCAT             ;
 ALTER TABLESPACE BMCARM.BMCARMCR USING VCAT QASDCAT             ;
       ....                          ....
 ALTER TABLESPACE BMCUTIL.BMCXCOPY USING VCAT QASDCAT            ;
```

```
 ALTER TABLESPACE DB2.STATDBAS USING VCAT QASDCAT                ;
 ALTER TABLESPACE DSNDBØ4.DUMMY USING VCAT QASDCAT               ;
 ALTER TABLESPACE DSNDBØ4.MAPPING USING VCAT QASDCAT             ;
 ALTER TABLESPACE DSNDBØ4.MAPPING1 USING VCAT QASDCAT            ;
 ALTER TABLESPACE DSNDBØ4.PLANTABL USING VCAT QASDCAT            ;
 ALTER TABLESPACE DSNRGFDB.DSNRGFTS USING VCAT QASDCAT           ;
 ALTER TABLESPACE DSNRLST.DSNRLSØ1 USING VCAT QASDCAT            ;
//*-----------------------------------------------------------------
//*B. DROP STOGROUPS
//*-----------------------------------------------------------------
SET CURRENT SQLID = 'QASDDB';
  DROP STOGROUP GPPROEDT  ;
  DROP STOGROUP SAPBTD;
  DROP STOGROUP SAPU1I;
  DROP STOGROUP SAPU1D;
  ....       ....
  DROP STOGROUP SAPDOD;
  DROP STOGROUP SAPDII;
  DROP STOGROUP SYSDEFLT ;
//*-----------------------------------------------------------------
//*C. RECREATE STOGROUPS USING THE NEW CATALOG ALIAS
//*-----------------------------------------------------------------
SET CURRENT SQLID = 'SAPR3';
  CREATE STOGROUP SAPBTD  VOLUMES ('*')    VCAT QASDDB  ;   COMMIT;
  CREATE STOGROUP SAPU1I  VOLUMES ('*')    VCAT QASDDB  ;   COMMIT;
  CREATE STOGROUP SAPU1D  VOLUMES ('*')    VCAT QASDDB  ;   COMMIT;
            ....                                 ....
  CREATE STOGROUP SAPDOI  VOLUMES ('*')    VCAT QASDDB  ;   COMMIT;
  CREATE STOGROUP SAPDOD  VOLUMES ('*')    VCAT QASDDB  ;   COMMIT;
  CREATE STOGROUP SAPDII  VOLUMES ('*')    VCAT QASDDB  ;   COMMIT;
SET CURRENT SQLID = 'QASDDB';
  CREATE STOGROUP SYSDEFLT  VOLUMES ('*')    VCAT QASDCAT;  COMMIT;
  CREATE STOGROUP GPPROEDT  VOLUMES ('*')    VCAT QASDDB ;  COMMIT;
//*-----------------------------------------------------------------
//*D. ALTER TABLESPACES BACK TO DB2 MANAGED
//*-----------------------------------------------------------------
  ALTER TABLESPACE DSNRLST.DSNRLSØ1 USING STOGROUP SYSDEFLT      ;
  ALTER TABLESPACE DSNRGFDB.DSNRGFTS USING STOGROUP SYSDEFLT     ;
  ALTER TABLESPACE DB2.STATDBAS USING STOGROUP SYSDEFLT          ;
  ALTER TABLESPACE BMCUTIL.BMCUTIL   USING STOGROUP SYSDEFLT     ;
  ALTER TABLESPACE BMCUTIL.BMCLGRNX USING STOGROUP SYSDEFLT      ;
  ALTER TABLESPACE BMCACP.BMCTPART USING STOGROUP SYSDEFLT       ;
  ALTER TABLESPACE BMCACP.BMCTSTAT USING STOGROUP SYSDEFLT       ;
  ALTER TABLESPACE A14Ø#6KO.GLREFTX  USING STOGROUP SAPBTD       ;
  ALTER TABLESPACE A12Ø#MI5.RBVS     USING STOGROUP SAPBTD       ;
  ALTER TABLESPACE A14Ø#L6Z.S159X    USING STOGROUP SAPBTD       ;
          ....                              ....
  ALTER TABLESPACE AØØØXDQG.STGLTRA  USING STOGROUP SAPSTD       ;
  ALTER TABLESPACE A11ØXB15.DB2IXBA  USING STOGROUP SAPBTD       ;
```

```
  ALTER TABLESPACE A000XB15.DB2NORU  USING STOGROUP SAPSTD       ;
//*-------------------------------------------------------------------
//*ALTER INDEXES BACK TO DB2 MANAGED
//*-------------------------------------------------------------------
  ALTER INDEX SYSIBM."DSNARL01" USING STOGROUP SYSDEFLT          ;
  ALTER INDEX DSNRGCOL."DSN_REGISTER_APPLI" USING STOGROUP SYSDEFLT;
  ALTER INDEX DSNRGCOL."DSN_REGISTER_OBJTI" USING STOGROUP SYSDEFLT;
  ALTER INDEX SAPR3."SYSTBLSP~0" USING STOGROUP SYSDEFLT         ;
  ALTER INDEX SAPR3."SYSTABLE~0" USING STOGROUP SYSDEFLT         ;
  ALTER INDEX CANDLE."DSNATX99" USING STOGROUP SYSDEFLT          ;
  ALTER INDEX CDB."ALTCTRLX" USING STOGROUP SYSDEFLT             ;
  ALTER INDEX CDB."ALTENQUX" USING STOGROUP SYSDEFLT             ;
  ALTER INDEX CDB."ALTMESGX" USING STOGROUP SYSDEFLT             ;
  ALTER INDEX BMC."BMCDICTX" USING STOGROUP SYSDEFLT             ;
  ALTER INDEX BMC."DSNUCH01" USING STOGROUP SYSDEFLT             ;
  ALTER INDEX BMCACP."DSNDPX01" USING STOGROUP SYSDEFLT          ;
  ALTER INDEX SAPR3."RBVS~0" USING STOGROUP SAPBTI               ;
  ALTER INDEX SAPR3."FMIO~0" USING STOGROUP SAPBTI               ;
  ALTER INDEX SAPR3."MCDX~0" USING STOGROUP SAPBTI               ;
        ....                          ....
  ALTER INDEX SAPR3."STGLTRAN~0" USING STOGROUP SAPSTI           ;
  ALTER INDEX SAPR3."DB2IXBACK~0" USING STOGROUP SAPBTI          ;
  ALTER INDEX SAPR3."DB2NORUN~0" USING STOGROUP SAPSTI           ;
//*-------------------------------------------------------------------
//*RESET PRIQTY/SECQTY FOR ALL TABLESPACES
//*-------------------------------------------------------------------
 ALTER TABLESPACE BMCACP.BMCTPART PRIQTY 0000028 SECQTY 0000020  ;
 ALTER TABLESPACE BMCACP.BMCTSTAT PRIQTY 0000028 SECQTY 0000020  ;
             ....                                    ....
 ALTER TABLESPACE BMCUTIL.BMCUTIL PRIQTY 0000028 SECQTY 0000020  ;
 ALTER TABLESPACE BMCUTIL.BMCXCOPY PRIQTY 0000028 SECQTY 0015000 ;
 ALTER TABLESPACE DB2.STATDBAS PRIQTY 0011520 SECQTY 0001152     ;
 ALTER TABLESPACE DSNDB04.DUMMY PRIQTY 0000012 SECQTY 0000012    ;
 ALTER TABLESPACE DSNDB04.MAPPING PRIQTY 0072000 SECQTY 0007200  ;
 ALTER TABLESPACE DSNDB04.MAPPING1 PRIQTY 0072000 SECQTY 0007200 ;
 ALTER TABLESPACE DSNDB04.PLANTABL PRIQTY 0000012 SECQTY 0000012 ;
 ALTER TABLESPACE DSNRGFDB.DSNRGFTS PRIQTY 0000012 SECQTY 0000012;
 ALTER TABLESPACE DSNRLST.DSNRLS01 PRIQTY 0000012 SECQTY 0000012 ;
 ALTER TABLESPACE A000#AAX.VIEA02 PRIQTY 0000240 SECQTY 0000144  ;
 ALTER TABLESPACE A000#ABN.ZU008 PRIQTY 0000016 SECQTY 0000040   ;
 ALTER TABLESPACE A000#AD6.BTXAUTH PRIQTY 0000576 SECQTY 0000144 ;
             ....                                   ....
 ALTER TABLESPACE U010#QZ8.ZZCONST$ PRIQTY 0036000 SECQTY 0003600;
 ALTER TABLESPACE U010#WLC.Y#PERF# PRIQTY 0000016 SECQTY 0000160 ;
 ALTER TABLESPACE U010#Z9D.ZZCONSA$ PRIQTY 0000052 SECQTY 0000160;
```

H E Byars
Systems Associate
Eastman Chemical Company (USA)

DB2 Version 6 stored procedures migration issues

In recent months I had the opportunity to be the technical team leader on a project for upgrading DB2 OS/390 from Version 5 to Version 6. In preparing for this project, my team studied many of the DB2 V6 manuals, attended the DB2 Technical Conference in New Orleans, participated in the 'DB2 V6 Transition' course offered by IBM, consulted many colleagues, and 'surfed' the different DB2 mailing lists looking for information that would help us achieve a smooth migration.

Although all the above-mentioned information sources described the changes introduced with DB2 V6 stored procedures, the team did not find a single clear statement indicating the complexity and magnitude of the impact caused by these changes.

The purpose of this article is to present a list of issues related to the implementation of DB2 V6 stored procedures that we found at our installation. Hopefully, this list will help you at the time you migrate to DB2 Version 6.

DDL AND PROGRAMMING ISSUES

The transition from DB2 OS/390 Version 5 to Version 6 changes how DB2 stored procedures are defined. In DB2 Version 5, DB2 stored procedures were defined by inserting a row into a table named SYSIBM.SYSPROCEDURES. In DB2 Version 6, DB2 stored procedures are defined using Data Definition Language SQL statements.

DB2 V6 stored procedure names can be explicitly or implicitly qualified by a schema name. The concept of 'SCHEMA' represents a logical manner of grouping stored procedures, mostly for the purpose of simplifying their administration.

The DDL syntax to explicitly qualify a stored procedure is as follows:

```
CREATE  PROCEDURE Schema_name.Stored_procedure_name .. other parms
```

where schema_name is a short SQL identifier (eight characters maximum), and stored_procedure_name is a long SQL identifier (18 characters maximum).

The DDL syntax to implicitly qualify a stored procedure is as follows:

```
CREATE PROCEDURE stored_procedure_name … other parms
```

where the schema_name is derived as follows:

- If the DDL statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option. If QUALIFIER is not specified, then the name of the owner of the plan or package becomes the schema name.
- If the DDL is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register at the time the DDL is been executed.

Please refer to IBM's *DB2 for OS/390 – SQL Reference, Volume 2*, for additional information regarding the CREATE PROCEDURE DDL syntax.

CURRENT PATH is an additional concept that specifies an ordered list of schemas to be searched when resolving unqualified references to DB2 V6 stored procedures. A new special register supports this concept. This new special register can store up to 254 characters, so the number of schema names it can store varies depending on the length of the schema names themselves. For more information, refer to the SET CURRENT PATH DDL statement in IBM's *DB2 for OS/390 – SQL Reference, Volume 2*.

At the time of conversion from DB2 Version 5 to DB2 Version 6, all the existing definitions in SYSIBM.SYSPROCEDURES are migrated into DB2 Version 6 stored procedure definitions. These definitions are stored within two new system tables, SYSIBM.SYSROUTINES and SYSIBM.SYSPARMS. All of the migrated stored procedures are created under the 'SYSPROC' schema name, and the owner for all the migrated procedures is set to blanks (spaces).

DB2 V6 stored procedures DDL-related issues are listed as follows:

- If you deploy DB2 V6 on your test system before your production system (by the way, a highly recommended approach), you will now have an environment where your test stored procedures are defined in a different manner to your production stored procedures. If you have automated processes to migrate DB2 stored procedures into production, you will need to update these. If you manually migrate DB2 stored procedures into production, you will need to translate the V6 DDL (which can't execute in a DB2 V5 system) into an insert statement for the production V5 SYSIBM.SYSPROCEDURES table.

- Once you are under DB2 Version 6 across the board, what standards are you going to use for these new DB2 objects? For example, under DB2 V5, in order to execute a stored procedure all that was needed was EXECUTE authority on the DB2 packages. Under DB2 V6, an EXECUTE on the PROCEDURE is now required. (By the way, the manuals indicate that you need execute authority on the package, but we found that all it provided was a grant to execute on the stored procedure.) How are you going to change existing DB2 stored procedure definitions? Are you going to ALTER them or are you going to DROP and RE-CREATE them? If you drop and re-create them, you would need to save the permissions before the DROP, and then re-issue them. You need to take into consideration that if you need to change one of the stored procedure parameters, you will be forced to drop and recreate the procedure.

- Additional questions exist for the concepts of 'SCHEMA' and 'CURRENT PATH'. These two concepts introduce the need for programming standards. For example, which schemas are going to be used for the new procedures? SYSPROC or another schema more easily identified with the application groups. If you use schemas other than SYSPROC, you will need to change your application programs to use the 'SET CURRENT PATH' statement to point to the new schemas. At our installation, we migrated using only the SYSPROC schema, but we are planning to create separate schemas for different application development groups in order to provide a more granular control for administration of stored procedures.

- Ownership, administration, and implementation of DB2 stored procedures raises additional questions. Who is going to be the owner of the DB2 stored procedures (and the schemas) in the test and production environments? Who is going to be able to define new stored procedures – the DBAs, the application developers, or both groups? Are the DBAs going to be involved in the development and implementation of these programs executing as stored procedures, since now these are part of the database itself? These and many other questions need to be addressed prior to migration to DB2 V6.

MIGRATION ISSUES

IBM provides a mechanism to convert (migrate) the DB2 V5 stored procedures into DB2 V6 stored procedures. This mechanism is supported by the DB2 catalog maintenance utility, CATMAINT, which creates the new DB2 tablespaces and tables necessary to support the new DB2 objects (stored procedures, user defined functions, and triggers). This same utility, in the same step, populates the DB2 V6 stored procedure tables with the V5 information.

This mechanism has several shortcomings, listed as follows:

- During the migration of one of our subsystems, the CATMAINT utility abnormally ended with a message indicating that it was trying to create a DB2 table (for SYSOBJECTS) that already existed. The only way to correct this situation was to restart DB2 using the DB2 V5 load libraries, delete the V6 objects causing the error, and re-start the migration process. See IBM document II11442 on IBMLINK for additional information.
- No security is migrated for the stored procedures. You will have to manually generate and execute the SQL statements to grant access to the DB2 V6 stored procedures. The following SQL will generate the required grant statements:

```
SELECT 'GRANT EXECUTE ON PROCEDURE SYSPROC.' ||
       PROCEDURE || ' TO ' || GRANTEE || ' ;'
FROM
       SYSIBM.SYSPROCEDURES,
```

```
SYSIBM.SYSPACKAUTH
WHERE PROCEDURE = NAME
ORDER BY 1 FOR FETCH ONLY;
```

- No provision is made to grant access to the default schema, SYSPROC. You will need to determine who will have access to this schema. It is very likely that you may have to grant access to public while you develop your standards and procedures.
- If you have third-party tools to peruse DB2 catalog contents, make sure that these tools are Version 6-compliant – in other words, that these tools will be able to manage and display information related to the new DB2 V6 objects (namely DB2 stored procedures, DB2 user-defined functions, and DB2 V6 user-defined data types). If the tools are not Version 6 compliant, you may need to create a set of SQL queries to retrieve information from the DB2 catalog.

DB2 MAINTENANCE ISSUES

It is over a year since DB2 Version 6 has been generally available and IBM is still finding serious bugs in its code. During our testing phase, we found several bugs directly related to DB2 stored procedures. As of 11 September 2000, there are at least two serious outstanding problems related to stored procedures IBM is working to resolve.

If you must use DB2 V6 stored procedures after you migrate, my recommendations are that:

- You migrate to V6 using a very current DB2 maintenance level.
- All applications using stored procedures are thoroughly tested.

For more information about a good DB2 maintenance level to migrate to, please refer to IBM document II12343 on IBMLINK for a Recommended Maintenance Level (RML).

Some of the PTFs that had an impact on our implementation are as follows: PTF UQ44647 (APAR PQ38811), PTF UQ44509 (APAR PQ37582), and PTF UQ44155 (APAR PQ38457).

CONCLUSION

With DB2 Version 6, IBM delivers a powerful database management system, full of enhancements and new features. DB2 V6 stored procedures have significantly matured and have become an important component of mainframe and client/server applications. Other DB2 V6 enhancements, such as user functions and user data types, make DB2 V6 DBMS a superb candidate as central DBMS server of choice.

If your installation is currently using DB2 stored procedures on DB2 V5, it is very important that you pay close attention to the migration issues related to them. Use all possible sources of information available to learn about existing and emerging issues. Keep a close eye on the list of highly pervasive (HIPER) fixes published by IBM. More than ever, ensure that all of your DB2 stored procedures are exercised at your test installation before migrating DB2 V6 into your production sites.

Antonio L Salcedo
Lead DB2 System Programmer/DBA (USA)

Utility for generating recovery jobs using the REXX SQL interface – part 2

This month we conclude the utility that uses the REXX SQL interface to access the information from the catalog tables and to build the JCL.

```
Call PROCQRY
/*  if first fetch is successful, then process all rows       */
/*  otherwise retrieve next row from flat file for processing */
if ERRFLG = 0 then
   Call PROCRSLT
return
/*  Prepare Dynamic SQL using STMT and perform first fetch  */
/*  Call PROCRSLT if successful to process rows otherwise   */
/*  write error message and continue to next input row      */
/*  If severe error, then exit program                      */
PROCQRY:
ADDRESS DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
/* Say 'Declare SQLCODE = ' SQLCODE  */
```

```
if SQLCODE ¬= Ø then
do
   Call SQLCA
   exit
end
ADDRESS DSNREXX "EXECSQL PREPARE S1 FROM :STMT"
/*  Say 'Prepare SQLCODE = ' SQLCODE    */
if SQLCODE ¬= Ø then
do
   Call SQLCA
   exit
end
/* Say 'opening C1 '             */
ADDRESS DSNREXX "EXECSQL OPEN C1"
/*Say 'Open SQLCODE = ' SQLCODE  */
if SQLCODE ¬= Ø then
   Call SQLCA
ADDRESS DSNREXX "EXECSQL FETCH C1 INTO :A1, :A2, :A3, :A4, :A5,",
                ":A6, :A7, :A8 "
/*
IF  SQLCODE = Ø  Then  Do
       Line = ' '
       Line = Line||A1||A2||A3||A4||A5||A6||A7||A8
       say Line
end
*/
if SQLCODE ¬= Ø then
do
   Call ERRPT
   ERRFLG = 1
   return
end
/*********************************************************/
/******  DEBUG  TEXT - UNCOMMENT IF REQUIRED  ***********/
/*
IF  SQLCODE = Ø  Then  Do
       Line = ' '
       Line = Line||A1||A2||A3||A4||A5||A6||A7||A8
       say Line
end
*/
/*********************************************************/
return
/*  Process results of query and form output members  */
PROCRSLT:
memcnt=Ø; eof=Ø;row_cnt = Ø
prevts = ''
first = 1
k = Ø ; j = Ø ; l = Ø
I_dbname = strip(I_dbname)
I_tsname = strip(I_tsname)
```

```
If strip(I_tsname) = '' then
   Memname = I_dbname
else
   Memname = I_tsname
len= length(Memname);
say 'Processing subset 'Memname' ...'
Do until (SQLCODE ¬= Ø )
   db = strip(A1)       /* database name          */
   ts = strip(A2)       /* tablespace name        */
   prt = strip(A3)      /* partition num or DSNUM */
   fseq = strip(A4)     /* fileseq number         */
   dsn = strip(A5)      /* dataset name of imagecpy */
   rba = strip(A6)      /* HEX of START_RBA       */
   icd = strip(A7)      /* icdate of syscopy rec  */
   ict = strip(A8)      /* ictime of syscopy rec  */
   row_cnt = row_cnt + 1
   if  first = 1  then
   do
      prevts = ts
      Call  FILLJCL
      Call  BILDSTEP
      first = Ø
   end
   else /* first ¬= 1  */
   do
      if  prevts = ts  then
      do
         Call BILDSTEP
      end
      else /*  prevts ¬=  ts   */
      do
         if P_typ = 'Q' then
         do
            k=k+1
            out.k = prev_rba
         end
         tempts = ts
         ts = prevts
         Call BILDINDX
         ts = tempts
         Call BILDSTEP
         prevts = ts
      end
   end
ADDRESS DSNREXX "EXECSQL FETCH C1 INTO :A1, :A2, :A3, :A4, :A5,",
                ":A6, :A7, :A8"
/*********************************************************/
/***  DEBUGGING  TEXT   - UNCOMMENT IF REQUIRED  ********/
/*
   IF  SQLCODE = Ø  Then  Do
          Line = ' '
```

```
          Line = Line||A1||A2||A3||A4||A5||A6||A7||A8
          say Line
   end
*/
/*******************************************************/
end
if P_typ = 'Q' then
do
   k=k+1
   out.k = prev_rba
end
ts = prevts
Call BILDINDX
ADDRESS DSNREXX "EXECSQL CLOSE C1"
Call  WRITEMEM
return
CLEANUP:
ods_mem = ods_name||"(REPORT)"
address tso "alloc f(outdd) mod dsname('"ods_mem"')"
hdr.1 = 'ICDATE used ...'I_icdate
hdr.2 = 'ICTIMES between 'I_btime' and 'I_etime
hdr.3 = 'Input dataset 'I_lstdsn
hdr.4 = 'NO  DBNAME   TSNAME   MEMBER    STATUS'
hdr.5 = '-------------------------------------'
address tso "execio *  DISKW outdd (stem hdr.  "
address tso "execio *  DISKW outdd (stem rpt. FINIS "
address tso "FREE ddname(outdd)"
return
PERFCONN:
/* Set up STEPLIBs                                    */
/*                                                    */
ADDRESS TSO
lib.Ø = 2
/*     *** SITESPEC ***     */
lib.1 = "ABCD."||sid||".DSNLOAD"
lib.2 = "ABCD."||sid||".DSNEXIT"
L_Dsname = ""
  do i = 1 to lib.Ø
    L_Dsname = L_Dsname||"'"||∂ib.i||"' "
  end
/*  address tso "ALLOC F(STEPLIB) DA("L_Dsname") SHR REU "   */
  "STEPLIB DA("L_Dsname") SHR "
'SUBCOM  DSNREXX'            /* Is host command env avlbl ?  */
IF  RC then                  /*  If not, then add it         */
    S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
ADDRESS  DSNREXX             /* exec all further cmds in DSNREXX */
ADDRESS DSNREXX "CONNECT" sid
Say 'Connect RC ...' RC
return
/* Disconnect from DB2  */
PERFDISC:
```

```
ADDRESS  DSNREXX " DISCONNECT"
Say 'Disconnect RC ...' RC
if SQLCODE ¬= Ø then
do
   Call SQLCA
   exit
end
S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX')
return
/*  Report error in query processing  */
ERRPT:
   do while length(I_dbname) < 8
      I_dbname = ' '||I_dbname
   end
   do while length(I_tsname) < 8
      I_tsname = ' '||I_tsname
   end
   k = k+1
   sno=k
   do while length(sno) < 3
      sno = ' '||sno
   end
   rpt.k = sno||' '||I_dbname||' '||I_tsname||' '||Memname
   rpt.k = rpt.k||' - Query did not retrieve any rows'
return
/*  Useful in analyzing SQL codes    */
SQLCA:
TRACE O
SAY 'SQLCODE ='SQLCODE
SAY 'SQLERRMC ='SQLERRMC
SAY 'SQLERRP ='SQLERRP
SAY 'SQLERRD ='SQLERRD.1',',
|| SQLERRD.2',',
|| SQLERRD.3',',
|| SQLERRD.4',',
|| SQLERRD.5',',
|| SQLERRD.6
SAY 'SQLWARN ='SQLWARN.Ø',',
|| SQLWARN.1',',
|| SQLWARN.2',',
|| SQLWARN.3',',
|| SQLWARN.4',',
|| SQLWARN.5',',
|| SQLWARN.6',',
|| SQLWARN.7',',
|| SQLWARN.8',',
|| SQLWARN.9',',
|| SQLWARN.1Ø
SAY 'SQLSTATE='SQLSTATE
SAY 'SQLCODE ='SQLCODE
say  'SQLERRMC ='SQLERRMC';' ,
```

```
|| 'SQLERRP ='SQLERRP';' ,
|| 'SQLERRD ='SQLERRD.1',',
           || SQLERRD.2',',
           || SQLERRD.3',',
           || SQLERRD.4',',
           || SQLERRD.5',',
           || SQLERRD.6';' ,
|| 'SQLWARN ='SQLWARN.Ø',',
           || SQLWARN.1',',
           || SQLWARN.2',',
           || SQLWARN.3',',
           || SQLWARN.4',',
           || SQLWARN.5',',
           || SQLWARN.6',',
           || SQLWARN.7',',
           || SQLWARN.8',',
           || SQLWARN.9',',
           || SQLWARN.1Ø';' ,
           || 'SQLSTATE='SQLSTATE';'
return
/*  The JCL JOB card, EXEC card and work file definitions */
FILLJCL:
osn = osn+1
ldlst.q = cdsn
do while length(osn) < 4
   osn = 'Ø'||?sn
end
jcl.1 = "//"||substr(Memname,1,8)||" JOB ("||P_act||"),"
jcl.1 = jcl.1||"'"||Memname||" RECOVER',TIME=144Ø,"
/*    *** SITESPEC ***                              */
jcl.2 = "//           CLASS=A,MSGCLASS=T,REGION=2ØM,NOTIFY=&SYSUID, "
jcl.3 = "//           LINES=9999                                     "
jcl.4 = "//*                                                        "
/*    *** SITESPEC ***                               */
jcl.5 = "/*JOBPARM SYSAFF="||P_sysid
jcl.6 = "//* ---------------------------------------------    "
jcl.7 = "//*       RECOVER JOB                                "
jcl.8 = "//*                                                 "
jcl.9 = "//*       GENERATED USING RECVER on "||us_date||"        "
jcl.1Ø= "//* using "||I_lstdsn
jcl.11= "//* ---------------------------------------------    "
jcl.12= "//*                                                 "
jcl.13= "//RCVRSTEP   EXEC PGM=DSNUTILB,PARM='"||sid||","||Memname||"'"
jcl.14= "//*RCVRSTEP  EXEC                                   "
jcl.15= "//*
PGM=DSNUTILB,PARM='"||sid||","||Memname||",RESTART(PHASE)'"
/*    *** SITESPEC ***                                */
jcl.16= "//STEPLIB  DD DISP=SHR,DSN=ABCD."||sid||".DSNEXIT"
jcl.17= "//         DD DISP=SHR,DSN=ABCD."||sid||".DSNLOAD"
jcl.18= "//SYSPRINT DD SYSOUT=* "
jcl.19= "//SYSUDUMP DD SYSOUT=* "
```

```
jcl.20= "//UTPRINT  DD SYSOUT=* "
jcl.21= "//SYSOUT   DD SYSOUT=* "
jcl.22= "//SYSUT1   DD SPACE=(CYL,(500,500)),UNIT=(SYSDA,5)"
jcl.23= "//SORTWK01 DD SPACE=(CYL,(250,500)),UNIT=(SYSDA,5)"
jcl.24= "//SORTWK02 DD SPACE=(CYL,(250,500)),UNIT=(SYSDA,5)"
jcl.25= "//SORTWK03 DD SPACE=(CYL,(250,500)),UNIT=(SYSDA,5)"
jcl.26= "//SORTWK04 DD SPACE=(CYL,(250,500)),UNIT=(SYSDA,5)"
jcl.27= "//SORTWK05 DD SPACE=(CYL,(250,500)),UNIT=(SYSDA,5)"
jcl.28= "//SORTWK06 DD SPACE=(CYL,(250,500)),UNIT=(SYSDA,5)"
jcl.29= "//*  "
return
/**    end  of  FILLJCL     **/
/*  Build the COPY DD cards and SYSIN cards for RECOVERY */
BILDSTEP:
j = j+1
csno = strip(row_cnt)
do while length(csno) < 5
   csno = '0'||csno
end
cpy.j  = "//CPY"||csno||" DD DSN="||dsn||","
j = j+1
cpy.j  = "//         LABEL=("||fseq||",SL),DISP=(OLD,PASS)"
cpy.j = cpy.j||",VOL=(,RETAIN)"
k = k+1
if P_typ = 'Q' & prt > 1 then
   out.k = "          TABLESPACE "||db||"."||ts
else
   out.k = "  RECOVER TABLESPACE "||db||"."||ts
if prt > 0 then
   out.k = out.k||" DSNUM "||prt
if P_typ = 'F' then
do
   k=k+1
   out.k = "  TOCOPY "dsn
end
else
   prev_rba = "  TORBA (X'"rba"')"
return
/*  Build the SYSIN cards for rebuilding INDEXES        */
BILDINDX:
l = l+1
rbd.l = "     REBUILD INDEX (ALL) TABLESPACE "||db||"."||ts
return
/*  Allocate a member and write the JCL  */
WRITEMEM:
   ods_mem = ods_name||"("||Memname||")"
   address tso "alloc f(outdd) mod dsname('"ods_mem"')"
   address tso "execio *  DISKW outdd (stem jcl.  "
   address tso "execio *  DISKW outdd (stem cpy.  "
   address tso "execio *  DISKW outdd (stem sys.  "
   address tso "execio *  DISKW outdd (stem out.  "
```

```
   address tso "execio *  DISKW outdd (stem rbd. FINIS "
   drop jcl.
   drop cpy.
   drop out.
   drop rbd.
   xx = outtrap("zap.","*")
   address tso "FREE ddname(outdd)"
   xx = outtrap("OFF")
   s = s+1
   sno=s
   do while length(sno) < 3
      sno = ' '||sno
   end
   if I_tsname = '' then
      I_tsname = '        '
   do
      rpt.s = sno||' '||I_dbname||' '||I_tsname||' '||Memname
      rpt.s = rpt.s||'  Successfully written'
   end
return
```

RCVRERR

```
)ATTR
+ TYPE(TEXT) COLOR(WHITE)
)BODY WINDOW (6Ø,5)
+
+ &ERRMSG                    :+ &ERRCOL
+
+  Enter to continue ...
+
)INIT
)PROC
)END
```

RCVRPAN

```
)ATTR
+ TYPE(TEXT) COLOR(WHITE)
| TYPE(OUTPUT) INTENS(HIGH)  CAPS(OFF)  PAD(' ')
¬ TYPE(TEXT) COLOR(GREEN) INTENS(LOW) HILITE(REVERSE)
)BODY
+                                                    System: &MC  +
+           DB2 subsystem ID :_SSID+
+
+ Recover to QUIESCE POINT or FULL COPY?(Q/F) :_Z+
+
+ DBNAME/TSNAME List PDS :_DBLST                                    +
+ Edit this dataset (Y/N):_Z+
+
```

```
+ ICDATE value to recover to             :_ICDATE+(yymmdd)
+ ICTIME value AFTER  which to recover to :_ICT1  +(hhmmss)
+ ICTIME value BEFORE which to recover to :_ICT2  +(hhmmss)
+
+ Output Dataset    :_OUTDS                                        +
+
+
+
   |ERRMSG                    |ERRCOL
+
+
+
+
¬F3 - End +
+
)INIT
  .ZVARS= '(RECTYP,EDFLG)'
)PROC
   if (.PFKEY = 'PFØ3') &PF3 = EXIT
   VER(&SSID,NB,LIST,DBT3,DBP2,DBT1,DBT2,DBP5,DBP6,DBP1,DBP3)
   VER(&DBLST,NB)
   VER(&RECTYP,NB,LIST,F,Q)
   VER(&EDFLG,NB,LIST,Y,N)
   VER(&ICDATE,NB)
   VER(&ICT1,NB)
   VER(&ICT2,NB)
   VER(&OUTDS,NB)
)END
```

RECVER

```
/* rexx */
/*********************************************************************/
/*           Panel driven tool for recovery of databases             */
/*     This is the main REXX which invokes the panel. It collects    */
/*     necessary inputs and invokes RCVRCAL                          */
/*   Invocation: TSO RECVER                                          */
/*********************************************************************/
trace o
clear
address tso "ispexec  vget (zsysid)"
p_sysid=zsysid
if p_sysid = 'PROD' then
   MC = 'PROD'
else
   MC = 'TEST'
pref =strip(sysvar(syspref))
cd = date(U)
us_date = substr(cd,7,2)||substr(cd,1,2)||substr(cd,4,2)
ZWINTTL = 'UTILITY TO GENERATE RECOVERY JOBS'
DBLST = pref||'.'||USERID()||'.INLST'
```

```
OUTDS = pref||'.'|USERID()||'.RCVRPT.D'||us_date
cur_date = date(S)-1
cur_date = substr(cur_date,3)
ICDATE = cur_date
ICT1 = '190000'
ICT2 = '235900'
EDFLG= 'Y'
RECTYP = 'F'
BEGIN:
/*   *** SITESPEC ***                                  */
"ISPEXEC LIBDEF ISPPLIB DATASET ID('HLQ.USERID.PANELS')"
"ISPEXEC ADDPOP ROW(2) COLUMN(10)"
"ISPEXEC DISPLAY PANEL(RCVRPAN)"
"ISPEXEC REMPOP ALL "
clear
Upper DBLST
upper EDFLG
upper ICDATE
upper ICT1; upper ICT2; upper OUTDS
upper SSID
if PF3 = 'EXIT' then
  exit(0)
CHK_DSN:
DBLST = strip(DBLST,Leading,"'")
z = outtrap("zlst.","*")
x = SYSDSN("'"DBLST"'")
z = outtrap("OFF")
if x = 'OK' then
   nop
else
do
   ERRMSG = 'Dataset not found ...'
   ERRCOL = DBLST
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVRERR)"
   "ISPEXEC REMPOP ALL "
   signal BEGIN
end
CHK_ICDATE:
D_yy = substr(ICDATE,1,2)
D_mm = substr(ICDATE,3,2)
D_dd = substr(ICDATE,5,2)
if (D_mm > 13) | (D_dd > 31) then
do
   ERRMSG = 'Invalid date specified '
   ERRCOL = ICDATE
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVRERR)"
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
cur_date = date(S)
```

```
cur_date = substr(cur_date,3)
if ICDATE > cur_date then
do
   ERRMSG = 'Date must be < =  'cur_date
   ERRCOL = ICDATE
   "ISPEXEC ADDPOP ROW(4) COLUMN(12)"
   "ISPEXEC DISPLAY PANEL(RCVRERR)"
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
CHK_ICT1:
T_hh = substr(ICT1,1,2)
T_mm = substr(ICT1,3,2)
T_ss = substr(ICT1,5,2)
if (T_hh > 23) | (T_mm > 59) | (T_ss > 59) then
do
   ERRMSG = 'Invalid time specified '
   ERRCOL = ICT1
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVRERR)"
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
CHK_ICT2:
T_hh2 = substr(ICT2,1,2)
T_mm2 = substr(ICT2,3,2)
T_ss2 = substr(ICT2,5,2)
if ((T_hh2 > 23) | (T_mm2 > 59) | (T_ss2 > 59))  then
do
   ERRMSG = 'Invalid time specified '
   ERRCOL = ICT2
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVRERR)"
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
CHK_ICTS:
if (T_hh2 < T_hh) | (T_hh2 = T_hh & T_mm2 < T_mm) then
do
   ERRMSG = 'ICTIMES incorrect  '
   ERRCOL = ICT1||' < 'ICT2
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVRERR)"
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
/*   *** SITESPEC ***                                                */
CHKSSID:
sid = SSID
ssid_err = Ø
 if p_sysid  = 'PROD' then
    select
```

```
      when sid = 'PROD'
         then do
            nop
         end
      otherwise
         ssid_err =1
    end
 if p_sysid  = 'TEST' then
   select
      when sid = 'DBT1'
         then do
            nop
         end
      when sid = 'DBT2'
         then do
            nop
         end
      when sid = 'DBT3'
         then do
            nop
         end
      when sid = 'DBP1'
         then do
            nop
         end
      otherwise
         ssid_err =1
   end
if ssid_err = 1 then
do
   ERRMSG = 'Invalid SSID specified '
   ERRCOL = SSID
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVRERR)"
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
CHKEDT:
if EDFLG = 'Y' then
do
   ADDRESS ISPEXEC "EDIT  DATASET('"DBLST"')"
end
cmd = 'address tso "RCVRCAL" SSID DBLST ICDATE ICT1 ICT2 '
cmd = cmd||'OUTDS RECTYP'
interpret cmd
say 'Press Enter to quit...'; pull temp
exit(0)
```

Jaiwant K Jonathan
DB2 DBA
QSS Inc (USA)

DB2 REXX Language Support

INTRODUCTION

This article describes DB2 for OS/390 REXX Language Support, which is a new separately-orderable feature of DB2.

DB2 REXX Language Support provides the ability to write SQL application programs in the REXX programming language.

REXX Language support, which is a no-charge feature, is available by ordering the desired distribution media feature (DB2 610 = 5108 for a 3480 Cartridge).

This article will first explain programming techniques that are unique to coding SQL statements in a REXX procedure.

At the end of this article, you will find a complete DB2 REXX procedure sample.

CODING SQL STATEMENTS IN A REXX APPLICATION

SQL data areas

The SQL Communication Area (SQLCA)

When DB2 prepares a REXX procedure that contains SQL statements, DB2 automatically includes a SQL communication area (SQLCA) in the procedure.

The SQLCA is a structure used to provide an application program with information about the execution of its SQL statements.

The REXX SQLCA consists of a set of separate variables.

The folowing table lists the variables available in a REXX SQLCA:

- SQLCODE – the SQL return code.
- SQLERRMC – one or more tokens, separated by X'FF', that are substituted for variables in the descriptions of error conditions.

- SQLERRP – a product signature and, in the case of an error, diagnostic information such as the name of the module that detected the error. For DB2 for OS/390, the product signature is 'DSN'.
- SQLERRD.1 – an internal error code.
- SQLERRD.2 – an internal error code.
- SQLERRD.3 – the number of rows affected after INSERT, UPDATE, and DELETE (but not rows deleted as a result of CASCADE delete). Set to 0 if the SQL statement fails, indicating that all changes made in executing the statement were cancelled. Set to -1 for a mass delete from a table in a segmented table space. For SQLCODE -911 or -913, SQLERRD.3 can also contain the reason code for a timeout or deadlock.
- SQLERRD.4 – generally contains timerons, a short floating-point value that indicates a rough relative estimate of resources required. This value does not reflect an estimate of the time required to execute the SQL statement. After you prepare an SQL statement, you can use this field as an indicator of the relative cost of the prepared SQL statement. For a particular statement, this number can vary with changes to the statistics in the catalog. This value is subject to change between releases of DB2 for OS/390.
- SQLERRD.5 – the position or column of a syntax error for a PREPARE or EXECUTE IMMEDIATE statement.
- SQLERRD.6 – an internal error code.
- SQLWARN.0 – blank if all other indicators are blank; W if at least one other indicator also contains a W.
- SQLWARN.1 – W if the value of a string column was truncated when assigned to a host variable.
- SQLWARN.2 – W if null values were eliminated from the argument of a column function; not necessarily set to W for the MIN function because its results are not dependent on the elimination of null values.
- SQLWARN.3 – W if the number of result columns is larger than

the number of host variables. Z if the ASSOCIATE LOCATORS statement contains fewer locators than the stored procedure returned.

- SQLWARN.4 – W if a prepared UPDATE or DELETE statement does not include a WHERE clause.
- SQLWARN.5 – W if the SQL statement was not executed because it is not a valid SQL statement in DB2 for OS/390.
- SQLWARN.6 – W if the addition of a month or year duration to a DATE or TIMESTAMP value results in an invalid day (for example 31 June). Indicates that the value of the day was changed to the last day of the month to make the result valid.
- SQLWARN.7 – W if one or more non-zero digits were eliminated from the fractional part of a number that was used as the operand of a decimal multiply or divide operation.
- SQLWARN.8 – W if a character that could not be converted was replaced by a substitute character.
- SQLWARN.9 – W if arithmetic exceptions were ignored during COUNT DISTINCT processing. Z if the stored procedure returned multiple result sets.
- SQLWARN.10 – W if at least one character field of the SQLCA is invalid because of a character conversion error.
- SQLSTATE – a return code for the outcome of the most recent execution of an SQL statement.

DB2 sets the SQLCODE and SQLSTATE values after each SQL statement execution.

An application should check these variable values to determine whether the last SQL statement was successful.

The SQL Descriptor Area (SQLDA)

The SQLDA is a structure that describes input/output variables or the columns of a result table.

The following statements require a SQL descriptor area (SQLDA):

- CALL...USING DESCRIPTOR descriptor-name.
- DESCRIBE statement-name INTO descriptor-name.
- DESCRIBE CURSOR host-variable INTO descriptor-name.
- DESCRIBE INPUT statement-name INTO descriptor-name.
- DESCRIBE PROCEDURE host-variable INTO descriptor-name.
- DESCRIBE TABLE host-variable INTO descriptor-name.
- EXECUTE...USING DESCRIPTOR descriptor-name.
- FETCH...USING DESCRIPTOR descriptor-name.
- OPEN...USING DESCRIPTOR descriptor-name.
- PREPARE...INTO descriptor-name.

Unlike the SQLCA, a REXX procedure can contain more than one SQLDA.

Each SQLDA consists of a set of REXX variables with a common stem. The stem must be a REXX variable name that contains no periods (full stops) and is the same as the value of descriptor-name that you specify when you use the SQLDA in an SQL statement – see Figure 1.

Each SQLDA contains stem.SQLD with the variables shown in Figure 2.

Variable name	*Usage in DESCRIBE and PREPARE INTO*	*Usage in FETCH, OPEN, EXECUTE, and CALL*
stem.SQLD	The number of columns that are described in the SQLDA. Contains a 0 if the statement string is not a query. For DESCRIBE PROCEDURE, the number of result sets returned by the stored procedure. Contains a 0 if no result sets are returned.	The number of host variables that are used by the SQL statement.

Figure 1: SQLDA

Variable name	*Usage in DESCRIBE and PREPARE INTO*	*Usage in FETCH, OPEN, EXECUTE, and CALL*
stem.n.SQLTYPE	Indicates the data type of the column or parameter and whether it can contain null values.	Indicates the data type of the host variable and whether an indicator variable is provided. Host variables for datetime values must be character string variables. For FETCH, a datetime type code means a fixed-length character string.
stem.n.SQLLEN	For a column other than a DECIMAL or NUMERIC column, the length attribute of the column or parameter. For datetime data, the length of the string representation of the value.	For a host variable that does not have a decimal data type, the length attribute of the host variable.
stem.n.SQLPRECISION	For a DECIMAL or NUMERIC column, the precision of the column or parameter.	For a host variable with a decimal data type, the precision of the host variable.
stem.n.SQLSCALE	For a DECIMAL or NUMERIC column, the scale of the column or parameter.	For a host variable with a decimal data type, the scale of the host variable.
stem.n.SQLCCSID	For a string column or parameter, the CCSID of the column or parameter.	For a string host variable, the CCSID of the host variable.
stem.n.SQLLOCATOR	For DESCRIBE PROCEDURE, the value of a result set locator.	Not used.

Figure 2a: Variables in a REXX SQLDA

Therefore, 1<= n<= stem.SQLD. There is one occurrence of each variable for each column of the result table or host variable that is described by the SQLDA.

The DB2 REXX programming interface

DB2 REXX Language Support includes the following application programming interfaces:

Variable name	*Usage in DESCRIBE and PREPARE INTO*	*Usage in FETCH, OPEN, EXECUTE, and CALL*
stem.n.SQLDATA	Not used.	Before EXECUTE or OPEN, contains the value of an input host variable. The application must supply this value. After FETCH, contains the values of an output host variable.
stem.n.SQLIND	Not used.	Before EXECUTE or OPEN, contains a negative number to indicate that the input host variable in stem.n.SQLDATA is null. The application must supply this value. After FETCH, contains a negative number if the value of the output host variable in stem.n.SQLDATA is null.
stem.n.SQLNAME	The name of the nth column in the result table. For DESCRIBE PROCEDURE, contains the cursor name that is used by the stored procedure to return the result set. The values for SQLNAME appear in the order that the cursors were opened by the stored procedure.	Not used.

Figure 2: Variables in a REXX SQLDA

- CONNECT – connects the REXX procedure to a DB2 subsystem. You must execute CONNECT before you can execute SQL statements.
- EXECSQL – executes SQL statements in REXX procedures.
- DISCONNECT – disconnects the REXX procedure from a DB2 subsystem. You should execute DISCONNECT to release resources that are held by DB2.

CONNECT

The syntax of CONNECT is:

```
ADDRESS DSNREXX "CONNECT" subsystem-id
```

EXECSQL

The syntax of EXECSQL is:

```
ADDRESS DSNREXX "EXECSQL" sql-statement
```

DISCONNECT

The syntax of DISCONNECT is:

```
ADDRESS DSNREXX "DISCONNECT"
```

Using SQL statements in a REXX procedure

DB2 REXX Language Support allows all SQL statements that DB2 for OS/390 supports, except the following statements:

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

Each SQL statement in a REXX procedure must begin with EXECSQL, in upper, lower, or mixed-case.

One of the following items must follow EXECSQL:

- A SQL statement enclosed in single or double quotation marks.
- A REXX variable that contains a SQL statement. The REXX variable must not be preceded by a colon.

For example, you can use either of the following methods to execute the COMMIT statement in a REXX procedure:

```
EXECSQL "COMMIT"
rexxvar="COMMIT"
EXECSQL rexxvar
```

Variables names

You can use any valid REXX name that does not end with a period (full stop) as a host variable. However, host variable names should not begin with 'SQL', 'RDI', 'DSN', 'RXSQL', or 'QRW'. Variable names can be at the most 64 bytes long.

Cursors and statements names

In REXX SQL applications, you must use a predefined set of names for cursors or prepared statements. The following names are valid for cursors and prepared statements in REXX SQL applications:

- c1 to c100 – cursor names for DECLARE CURSOR, OPEN, CLOSE, and FETCH statements. Use c1 to c50 for cursors that are defined without the WITH HOLD option. Use c51 to c100 for cursors that are defined with the WITH HOLD option. All cursors are defined with the WITH RETURN option, so any cursor name can be used to return result sets from a REXX stored procedure.
- c101 to c200 – cursor names for ALLOCATE, DESCRIBE, FETCH, and CLOSE statements that are used to retrieve result sets in a program that calls a stored procedure.
- s1 to s100 – prepared statement names for DECLARE STATEMENT, PREPARE, DESCRIBE, and EXECUTE statements.

Use only the predefined names for cursors and statements. Do not use any of the predefined names for host variables.

REXX host variables

You do not declare host variables in REXX. When you need a new variable, you use it in a REXX command.

When you use a REXX variable as a host variable in a SQL statement, you must precede the variable with a colon.

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

Patrick Renard
CTRNE (France)

Omnis Software has announced Version 3.0 of its Omnis Studio 4GL rapid application development software. It has many changes to the Web server and Web client technologies to speed up Web-based business applications and it includes a range of developer productivity enhancements.

The core of the product has apparently been rewritten to take advantage of multi-threading when executing methods or directly accessing databases such as DB2, Oracle, and Sybase, which should improve the efficiency with which simultaneous users can access, use, and modify the same data.

Also, load-sharing extends this functionality across a number of servers, dynamically allocating the least-used server as each new client accesses the program.

The addition of client-side methods is said to eliminate the delays associated with browser-based server access, while the 18 new Web-specific components are said to create a richer, more responsive user interface.

A range of additional components include clocks, sliders, picture faders, and complex grids.

For further information contact:
Omnis Software, 981 Industrial Road, Building B, San Carlos, CA 94070, USA.
Tel: (650) 632 7100.
URL: http://www.omnis-software.com/v3/index.html.

* * *

Computer Associates is to resell SoftBase DB2 tools, including Database Rely, under the umbrella of its CA Back-up and Recovery Solution for DB2 for OS/390 products. Database Attach is designed to streamline the interaction of DB2 databases with mainframe processors, while Database Rely extends the restart and recovery capabilities of traditional mainframe DB2 databases.

For further information contact:
Computer Associates, 1 CA Plaza, Hauppauge, NY 11749, USA.
Tel: (516) 342 5224.
URL: http://www.cai.com/products/db2/factsht/dbrely_fs.htm.

* * *

Palm and IBM have announced plans to collaborate on mobile e-business software on a worldwide basis. As part of the agreement, IBM Global Services will create a consulting and systems integration competency group to focus on developing and deploying business applications for Palm OS devices, which include the IBM Workpad and related Palm-based data services, such as the Palm.Net wireless service.

The technology to deliver the new applications is based on IBM's just-released WebSphere Everyplace Suite, which allows mobile devices to connect to the Web. DB2 Everyplace, MQ Series products, VisualAge Micro Edition Java technology, and Tivoli Device Manager all support Palm OS.

For further information contact: your local IBM representative.
URL: http://www-3.ibm.com/pvc/.

* * *