109

DB2 update

November 2001

In this issue

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 ($33.75) 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.

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

When Xephon is given copyright, 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 download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

Printed in England.

COPYTOCOPY – a new DB2 utility for Version 7

DB2 Version 7 introduced several new utilities and utility functions, one of which is the utility called COPYTOCOPY. The purpose of COPYTOCOPY is to make additional image copies of currently existing image copy datasets.

The primary benefit of COPYTOCOPY is a reduction in the amount of time required to run the COPY utility. Remember that COPY A formatting routine for DSNTIAUL unload filescan be used to take up to four image copies with a single execution of the utility. But with COPYTOCOPY available, instead of using COPY to make four image copy back-ups, the DBA can use it to make a single image copy, and then run COPYTOCOPY to make additional image copies. The COPY utility will take less time to create a single image copy back-up than it will to take multiple image copy back-ups. And the combination of COPY plus COPYTOCOPY can be used to increase availability.

When COPYTOCOPY is running, the database object (table space or index space) being copied is placed in Utility Restricted Read/Write state (URRW). This prevents other users from dropping the table space or index while COPYTOCOPY is running.

Individual data and index partitions are treated as distinct target objects by the COPYTOCOPY utility. Any other utilities operating on different partitions of the same table space or index space can be run concurrently with COPYTOCOPY.

The following utilities cannot be run concurrently on the same database object as the COPYTOCOPY utility:

- COPY
- LOAD
- MERGECOPY
- MODIFY
- RECOVER

- REORG INDEX
- REORG TABLESPACE.

Furthermore, COPYTOCOPY is flexible enough to run against any DB2 image copy dataset. This includes inline copies made during the execution of the REORG and LOAD utilities. COPYTOCOPY must start with a primary image copy back-up – either the local primary or recovery site primary copy. From that image copy, the COPYTOCOPY utility can make up to three copies of one or more of the following types:

- Local primary
- Local back-up
- Recovery site primary
- Recovery site back-up.

Copies created by COPYTOCOPY can be used by the RECOVER utility just like regular image copies created using the COPY utility. Both table space and index space copies can be made using the COPYTOCOPY utility. Any DB2 utility process that uses image copy datasets can use the image copy datasets created by COPYTOCOPY. This includes MERGECOPY, UNLOAD, and subsequent runs of COPYTOCOPY. However, bear in mind that image copies created with the CONCURRENT option of the COPY utility are not supported by the COPYTOCOPY utility.

Just like COPY, COPYTOCOPY records information about the image copies that it creates in the SYSIBM.SYSCOPY system catalog table. The COPYTOCOPY utility will insert the values in the DSNAME, GROUP_MEMBER, JOBNAME, AUTHID, DSVOLSER, and DEVTYPE columns as appropriate, depending on the copies that are being created.

You cannot run COPYTOCOPY to create additional image copies for certain DB2 catalog (SYSCOPY in DSNDB06) and DB2 directory (DSNDB01 and SYSUTILX both in DSNDB01) objects.

The COPYTOCOPY utility operates in these distinct phases:

- UTILINIT – initialization and set-up.
- CPY2CPY – copying the image copy.
- UTILTERM – clean-up.

REQUIRED AUTHORIZATIONS

To execute COPYTOCOPY, the process or user running the utility must have been granted one of the following privileges:

- SYSADM.
- SYSCTRL.
- DBADM, DBCTRL, or DBMAINT for the database in which the index or table space resides.
- IMAGCOPY for the database in which the index or table space resides.

The only exception to the above is for the DB2 directory (DSNDB01) and the DB2 catalog (DSNDB06). Any process or user having INSTALL SYSOPR authority can run COPYTOCOPY for table spaces in the directory or catalog.

TERM AND RESTART ISSUES

The use of the TERM command to terminate a COPYTOCOPY step that has abended is not recommended. A current restart should be done instead to allow COPYTOCOPY to pick up where it left off. Terminating COPYTOCOPY in such a situation might cause inconsistencies between the ICF catalog and DB2 catalog when GDGs are used.

You cannot use RESTART(PHASE) for a COPYTOCOPY job. It is fine to use RESTART(CURRENT) as long as you avoid using the TERM UTILITY command to terminate a COPYTOCOPY step. When you use RESTART(CURRENT), COPYTOCOPY will restart from the last commit point with the same image copy dataset, so be sure to code a dataset disposition of DISP=(MOD,CATLG,CATLG) on your JCL DD statements.

INLINE COPY EXCEPTION

When using COPYTOCOPY to copy an inline image copy that was made by the REORG utility with the part range option, you will need to specify individual DSNUMs for the partitions to be copied. The COPYTOCOPY utility does not support part range. COPYTOCOPY will copy only the specified partition data from the input inline image copy dataset into the output image copy dataset.

COPYTOCOPY EXECUTION

To run the COPYTOCOPY utility, it is not necessary to provide the explicit dataset name of the image copy being copied. Instead, the input to the COPYTOCOPY utility is the name of the table space, index space, or index for which the original copy was made, and an indication of which image copy in the catalog should be copied. To specify this information COPYTOCOPY provides three options:

- FROMLASTCOPY – indicates that the most recent image copy taken for the table space or index space is to be used as input to the COPYTOCOPY utility. The input could be either a full image copy or incremental copy. The utility will retrieve the information from the SYSIBM.SYSCOPY system catalog table.
- FROMLASTFULLCOPY – indicates that the most recent full image copy taken for the object is to be used as the input to the COPYTOCOPY job. Once again, this information is obtained by querying the DB2 Catalog.
- FROMLASTINCRCOPY – indicates that the most recent incremental image copy taken for the object is to be used as the input to the COPYTOCOPY job. FROMLASTINCRCOPY is not valid for index spaces or indexes. If FROMLASTINCRCOPY is specified for an index space or index, COPYTOCOPY will use the last full image copy that was taken for the index, if one is available. And once again, this information is obtained by querying the DB2 catalog.

Of course, you may choose instead to specify the dataset name for the image copy that is to be copied by the COPYTOCOPY utility. This can be accomplished by using the FROMCOPY clause. But bear in mind that when you are using COPYTOCOPY with a list of objects defined using the LISTDEF statement, the FROMCOPY clause is not valid.

If the FROMCOPY keyword is not used, the COPYTOCOPY utility must determine which specific image copy is to be copied. Before COPYTOCOPY can execute it may have to choose between the local site primary copy, local site back-up copy, recovery site primary copy, and recovery site back-up copy datasets. COPYTOCOPY will search image copies in the following order to determine the input dataset to be used:

- If you are running COPYTOCOPY at your local site, the search order will be local site primary copy, local site back-up copy, recovery site primary copy, recovery site back-up copy.
- If you are running the utility at your recovery site, the search order will be recovery site primary copy, recovery site back-up copy, local site primary copy, then finally local site back-up copy.

If the input dataset cannot be allocated or opened, the COPYTOCOPY utility will try to use the next image copy data with the same START_RBA value in SYSIBM.SYSCOPY column, in the search order as indicated previously. When the FROMCOPY keyword is used, though, only the explicitly specified dataset can be used as the input to COPYTOCOPY.

AN EXAMPLE OF THE COPYTOCOPY UTILITY

Let’s take a quick look at a sample JCL job step to run the COPYTOCOPY utility. The following code can be run to make a back-up local image copy of the table space DSN8S71E in the sample DB2 database DSN8D71A. This will be either a full or incremental image copy, whichever was last run for this object:

```
//STEP1     EXEC DSNUPROC,UID='DBAPCSM.CPY2CPYT',
//          UTPROC='',
//          SYSTEM='V71A',DB2LEV=DB2A
//SYSIN     DD *
//COPY2     DD DSN=COPYØØ2F.IFDYØ1,UNIT=SYSDA,VOL=SER=CPYØ2I,
//          SPACE=(CYL,(15,1)),DISP=(NEW,CATLG,CATLG)
//SYSIN     DD *
```

```
   COPYTOCOPY TABLESPACE DSN8D71A.DSN8S71E COPYDDN(,COPY2)
/*
```

BOTTOM LINE

The COPYTOCOPY utility provides a useful new feature for the toolkit of DB2 DBAs. Using COPYTOCOPY to create additional image copies from existing image copies can enhance availability and assist DBAs in creating an optimal back-up and recovery plan for their DB2 applications and databases.

Craig S Mullins
BMC (USA)

Improve DSN1COPY performances

INTRODUCTION

The DB2 utility DSN1COPY can be used to copy DB2 VSAM datasets to sequential datasets.

In order to improve DSN1COPY execution, you should code an AMP parameter in the SYSUT1 DD statement. The AMP parameter is used to specify information in an Access method Control Block (ACB) for a VSAM dataset.

In order to get better I/O performance, you should code an optimized BUFND parameter. This parameter specifies the number of I/O buffers that VSAM is to use for data records.

By default, the BUFND value is too low – it is 2!

A DB2 VSAM dataset uses 12 CI/track of a 3390 DASD. So, in order to match a cylinder, you should code BUFND=180 (12 * 15 = 180).

BENCHMARK

To test the effect of this optimization, we will use DSN1COPY to ‘download’ the SYSPKAGE tablespace (5115 tracks).

Test with BUFND = 2

First, we are not going to code the AMP parameter, so we will use the default BUFND value of 2. The JCL for this is shown below:

```
//STEPØ1   EXEC PGM=DSN1COPY
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DISP=SHR,DSN=DB2P.DSNDBC.DSNDBØ6.SYSPKAGE.IØØØ1.AØØ1
//SYSUT2   DD DUMMY
```

The DSN1COPY results are shown below:

```
1     JES2  JOB  LOG  –  S Y S T E M  F M V S  –  NODE  JES2MVØØ
Ø
 15.Ø9.26 JOBØ4661 —— THURSDAY,  16 AUG 2ØØ1 ——
 15.Ø9.26 JOBØ4661  TSS7ØØØI SXSDØØ1 Last-Used 16 Aug Ø1 15:Ø6
System=MVØP Facility=VNM
 15.Ø9.26 JOBØ4661  $HASP373 SXSDØØ1A STARTED - WLM INIT  - SRVCLASS
JOB_6Ø   - SYS MVØP
 15.Ø9.26 JOBØ4661  TSS7ØØ1I Count=248Ø3 Mode=Fail Locktime=None
Name=MORTIMER MIKE
 15.Ø9.26 JOBØ4661  IEF4Ø3I SXSDØØ1A - STARTED - TIME=15.Ø9.26
 15.14.22 JOBØ4661  -                                       –TIMINGS
(MINS.)–
 15.14.22 JOBØ4661  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU
SRB  ELAPS   SERV
 15.14.22 JOBØ4661  -SXSDØØ1A STEPØ1              Ø4  58354    .Ø4
.Ø2    4.9   665K
 15.14.22 JOBØ4661  IEF4Ø4I SXSDØØ1A - ENDED - TIME=15.14.22
 15.14.22 JOBØ4661  -SXSDØØ1A ENDED.  NAME-SYSTEM TEAM          TOTAL
CPU TIME=   .Ø4  TOTAL
 15.14.22 JOBØ4661  $HASP395 SXSDØØ1A ENDED
Ø——— JES2 JOB STATISTICS ———
-  16 AUG 2ØØ1 JOB EXECUTION DATE
-          15 CARDS READ
-          63 SYSOUT PRINT RECORDS
-           Ø SYSOUT PUNCH RECORDS
-           4 SYSOUT SPOOL KBYTES
-        4.93 MINUTES EXECUTION TIME
```

This first execution of DSN1COPY used 58354 EXCP, 2.78 CPU seconds, and its elapsed time was 4.93 minutes.

Test with BUFND = 180

The JCL with BUFND = 180 is shown below:

```
//STEP01   EXEC PGM=DSN1COPY
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DISP=SHR,DSN=DB2P.DSNDBC.DSNDB06.SYSPKAGE.I0001.A001,
//            AMP=('BUFND=180')
//SYSUT2   DD DUMMY
```

The DSN1COPY results are shown below:

```
1        JES2  JOB  LOG  --  S Y S T E M  P R O D  --  NODE  JES2MV00
0
 15.11.04 JOB04662 -- THURSDAY,  16 AUG 2001 --
 15.11.04 JOB04662  TSS7000I SXSD001 Last-Used 16 Aug 01 15:10
System=MV0P Facility=BATCH
 15.11.04 JOB04662  $HASP373 SXSD001B STARTED - WLM INIT  - SRVCLASS
JOB_10   - SYS MV0P
 15.11.04 JOB04662  TSS7001I Count=24805 Mode=Fail Locktime=None
Name=MORTIMER MIKE
 15.11.04 JOB04662  IEF403I SXSD001B - STARTED - TIME=15.11.04
 15.11.37 JOB04662  -                                       --TIMINGS
(MINS.)--
 15.11.37 JOB04662  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU
SRB  ELAPS   SERV
 15.11.37 JOB04662  -SXSD001B STEP01              04    681    .02
.00     .5   173K
 15.11.37 JOB04662  IEF404I SXSD001B - ENDED - TIME=15.11.37
 15.11.37 JOB04662  -SXSD001B ENDED.  NAME-SYSTEM TEAM          TOTAL
CPU TIME=   .02  TOTAL
 15.11.37 JOB04662  $HASP395 SXSD001B ENDED
0------ JES2 JOB STATISTICS ------
-  16 AUG 2001 JOB EXECUTION DATE
-          16 CARDS READ
-          64 SYSOUT PRINT RECORDS
-           0 SYSOUT PUNCH RECORDS
-           4 SYSOUT SPOOL KBYTES
-        0.54 MINUTES EXECUTION TIME
```

This second execution of DSN1COPY used 681 EXCP, 1.62 CPU seconds, and its elapsed time was 0.54 minutes.

MEMORY CONSTRAINTS

If the region of your job is too small to allocate 180 data buffers, your job will fail (RC=8) with the following messages:

```
IEC161I Ø2Ø-Ø54,I99Ø557B,STEPØ1,SYSUT1,,,
IEC161I DB2P.DSNDBC.DSNDBØ6.SYSPKAGE.IØØØ1.AØØ1,
IEC161I DB2P.DSNDBD.DSNDBØ6.SYSPKAGE.IØØØ1.AØØ1,CATALOG.DB2P
….
DSN1999I START OF DSN1COPY FOR JOB I99Ø557B STEPØ1
DSN1996I VSAM OPEN ERROR, ACBERRFLG = 136.  INPUT
DSN1993I DSN1COPY TERMINATED,        Ø PAGES PROCESSED
```

CONCLUSION

The results of these two executions of DSN1COPY are summarized in Figure 1.

BUFND	*EXCP*	*CPU*	*Elapsed*
2	58354	2.78	4.93
180	681	1.62	0.54
	-98,83%	-41.73%	-89.05%

Figure 1: Result summary

You should notice that you can save 89.05% execution time using an optimized BUFND parameter!

(France)

LASTDATE and LASTDAY functions

I have prepared two more user-defined functions (see *DB2 Update* issues 103 and 104, May and June 2001, *Sample user-defined functions*). The first function is called LASTDATE and the second function is LASTDAY. I wrote all these functions in PL/I.

LASTDATE

The schema is SYSADM.

The LASTDATE function returns the last date of the month in date format.

The expression must be a date, for example:

```
SELECT LASTDATE(CURRENT DATE)       AS LDATE1,
       LASTDATE(DATE('1987-03-17')) AS LDATE2
FROM SYSIBM.SYSDUMMY1
```

This gives the following result:

```
LDATE1      LDATE2
----------  ----------
2001-05-31  1987-03-31
```

The current date for this example was ‘2001-05-28’.

LASTDAY(expression)

The schema is SYSADM.

The LASTDAY function returns the last day of the month in smallint (small integer) format.

The expression must be a date, for example:

```
SELECT LASTDAY(CURRENT DATE)       AS LDAY1,
       LASTDAY(DATE('1987-03-17')) AS LDAY2
FROM SYSIBM.SYSDUMMY1
```

This gives the following result:

```
 LDAY1   LDAY2
```

```
------  ------
31
```

The current date for this example was ‘2000-11-27’.

THE UDF CREATE DEFINITIONS

LASTDATE

```
CREATE FUNCTION SYSADM.LASTDATE
       ( DATE )
RETURNS  SMALLINT
SPECIFIC LASTDATE
EXTERNAL NAME 'LASTDATE'      -- MVS load module
LANGUAGE PLI
PARAMETER STYLE DB2SQL
DETERMINISTIC
READS SQL DATA
DBINFO
FENCED
COLLID LASTDATE
WLM ENVIRONMENT DSNNWLM1    -- WLM application
STAY RESIDENT YES
PROGRAM TYPE MAIN
NO EXTERNAL ACTION
RETURNS NULL ON NULL INPUT
NO SCRATCHPAD
NO FINAL CALL
DISALLOW PARALLEL
ASUTIME NO LIMIT
SECURITY DB2;
```

LASTDAY

```
CREATE FUNCTION SYSADM.LASTDAY
       ( DATE )
RETURNS  SMALLINT
SPECIFIC LASTDAY
EXTERNAL NAME 'LASTDAY'      -- MVS load module
LANGUAGE PLI
PARAMETER STYLE DB2SQL
DETERMINISTIC
READS SQL DATA
DBINFO
FENCED
COLLID LASTDAY
WLM ENVIRONMENT DSNNWLM1    -- WLM application
STAY RESIDENT YES
```

```
PROGRAM TYPE MAIN
NO EXTERNAL ACTION
RETURNS NULL ON NULL INPUT
NO SCRATCHPAD
NO FINAL CALL
DISALLOW PARALLEL
ASUTIME NO LIMIT
SECURITY DB2;
```

LOAD MODULES

LASTDATE – PL/I source code

```
* PROCESS SYSTEM(MVS);
 LASTDAT: PROC(UDF_PARM1, UDF_RESULT,
               UDF_IND1,  UDF_INDR,
               UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
               UDF_DIAG_MSG, UDF_SCRATCHPAD,
               UDF_CALL_TYPE, UDF_DBINFO)
          OPTIONS(MAIN NOEXECOPS REENTRANT);
 /********************************************************************/
 /*    UDF   : LASTDATE                                              */
 /*    INPUT : UDF_PARM1    CHAR(1Ø)                                 */
 /*    OUTPUT: UDF_RESULT   DATE                                     */
 /********************************************************************/
  DCL UDF_PARM1      CHAR(1Ø);          /* INPUT PARAMETER           */
  DCL UDF_RESULT     CHAR(1Ø);          /* RESULT PARAMETER          */
  DCL UDF_IND1       BIN FIXED(15);     /* INDICATOR FOR INPUT PARM  */
  DCL UDF_INDR       BIN FIXED(15);     /* INDICATOR FOR RESULT      */
  DCL 1 UDF_SCRATCHPAD,                 /* SCRATCHPAD                */
        3 UDF_SPAD_LEN    BIN FIXED(31),
        3 UDF_SPAD_TEXT   CHAR(1ØØ);

  DCL 1 UDF_DBINFO,                        /* DBINFO                */
      3 UDF_DBINFO_LLEN BIN FIXED(15),     /* LOCATION LENGTH       */
      3 UDF_DBINFO_LOC  CHAR(128),         /* LOCATION NAME         */
      3 UDF_DBINFO_ALEN BIN FIXED(15),     /* AUTH ID LENGTH        */
      3 UDF_DBINFO_AUTH CHAR(128),         /* AUTHORIZATION ID      */
      3 UDF_DBINFO_CCSID CHAR(48),         /* CCSIDS FOR DB2 OS/39Ø */
       5 UDF_DBINFO_ESBCS BIN FIXED(31),   /* EBCDIC SBCS CCSID     */
       5 UDF_DBINFO_EMIXED BIN FIXED(31),  /* EBCDIC MIXED CCSID    */
       5 UDF_DBINFO_EDBCS BIN FIXED(31),   /* EBCDIC DBCS CCSID     */
       5 UDF_DBINFO_ASBCS BIN FIXED(31),   /* ASCII SBCS CCSID      */
       5 UDF_DBINFO_AMIXED BIN FIXED(31),  /* ASCII MIXED CCSID     */
       5 UDF_DBINFO_ADBCS BIN FIXED(31),   /* ASCII DBCS CCSID      */
       5 UDF_DBINFO_RESERV1 CHAR(2Ø),      /* RESERVED              */
      3 UDF_DBINFO_QLEN BIN FIXED(15),     /* QUALIFIER LENGTH      */
      3 UDF_DBINFO_QUALIF CHAR(128),       /* QUALIFIER NAME        */
      3 UDF_DBINFO_TLEN BIN FIXED(15),     /* TABLE LENGTH          */
      3 UDF_DBINFO_TABLE  CHAR(128),       /* TABLE NAME            */
      3 UDF_DBINFO_CLEN BIN FIXED(15),     /* COLUMN LENGTH         */
      3 UDF_DBINFO_COLUMN CHAR(128),       /* COLUMN NAME           */
      3 UDF_DBINFO_RELVER CHAR(8),         /* DB2 RELEASE LEVEL     */
      3 UDF_DBINFO_PLATFORM BIN FIXED(31), /* DATABASE PLATFORM     */
      3 UDF_DBINFO_NUMTFCOL BIN FIXED(15), /* # OF TF COLS USED     */
      3 UDF_DBINFO_RESERV1 CHAR(24),       /* RESERVED              */
      3 UDF_DBINFO_TFCOLUMN PTR,           /* -> TABLE FUN COL LIST */
      3 UDF_DBINFO_RESERV2 CHAR(24);       /* RESERVED              */
  DCL (ADDR,LENGTH,SUBSTR,NULL)   BUILTIN;
  EXEC SQL INCLUDE SQLCA;

  /* ********************************************************** */
  /* RETURNS THE LAST DATE OF THE MONTH IN FORMAT 'YYYY-MM-DD'  */
  /* LASTDATE(DATE('2ØØØ-Ø8-18'))   ->   '2ØØØ-Ø8-31'           */
  /* ********************************************************** */

  EXEC SQL
    SELECT STRIP(CHAR(YEAR(:UDF_PARM1)))||'-'||
           SUBSTR(DIGITS(MONTH(:UDF_PARM1)), 9, 2)||'-'||
           CHAR(DAY(DATE(STRIP(CHAR(YEAR(:UDF_PARM1)))||'-'||
           SUBSTR(DIGITS(MONTH(:UDF_PARM1)), 9, 2)||'-Ø1')
           + 1 MONTH - 1 DAY))
    INTO :UDF_RESULT
  FROM SYSIBM.SYSDUMMY1 WITH UR;
 END LASTDAT;
```

LASTDAY – PL/I source code

```
* PROCESS SYSTEM(MVS);
 LASTDAY: PROC(UDF_PARM1, UDF_RESULT,
               UDF_IND1,  UDF_INDR,
               UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
               UDF_DIAG_MSG, UDF_SCRATCHPAD,
               UDF_CALL_TYPE, UDF_DBINFO)
          OPTIONS(MAIN NOEXECOPS REENTRANT);
 /********************************************************************/
 /*    UDF   : LASTDAY                                               */
 /*    INPUT : UDF_PARM1    CHAR(1Ø)                                 */
 /*    OUTPUT: UDF_RESULT   SMALLINT                                 */
 /********************************************************************/
  DCL UDF_PARM1      CHAR(1Ø);          /* INPUT PARAMETER           */
  DCL UDF_RESULT     BIN FIXED(15);     /* RESULT PARAMETER          */
  DCL UDF_IND1       BIN FIXED(15);     /* INDICATOR FOR INPUT PARM  */
  DCL UDF_INDR       BIN FIXED(15);     /* INDICATOR FOR RESULT      */
  DCL 1 UDF_SCRATCHPAD,                 /* SCRATCHPAD                */
       3 UDF_SPAD_LEN    BIN FIXED(31),
       3 UDF_SPAD_TEXT   CHAR(1ØØ);
 DCL 1 UDF_DBINFO,                         /* DBINFO                */
     3 UDF_DBINFO_LLEN BIN FIXED(15),     /* LOCATION LENGTH       */
     3 UDF_DBINFO_LOC  CHAR(128),         /* LOCATION NAME         */
     3 UDF_DBINFO_ALEN BIN FIXED(15),     /* AUTH ID LENGTH        */
     3 UDF_DBINFO_AUTH CHAR(128),         /* AUTHORIZATION ID      */
     3 UDF_DBINFO_CCSID CHAR(48),         /* CCSIDS FOR DB2 OS/390 */
      5 UDF_DBINFO_ESBCS BIN FIXED(31),   /* EBCDIC SBCS CCSID     */
      5 UDF_DBINFO_EMIXED BIN FIXED(31),  /* EBCDIC MIXED CCSID    */
      5 UDF_DBINFO_EDBCS BIN FIXED(31),   /* EBCDIC DBCS CCSID     */
      5 UDF_DBINFO_ASBCS BIN FIXED(31),   /* ASCII SBCS CCSID      */
      5 UDF_DBINFO_AMIXED BIN FIXED(31),  /* ASCII MIXED CCSID     */
      5 UDF_DBINFO_ADBCS BIN FIXED(31),   /* ASCII DBCS CCSID      */
      5 UDF_DBINFO_RESERV1 CHAR(20),      /* RESERVED              */
     3 UDF_DBINFO_QLEN BIN FIXED(15),     /* QUALIFIER LENGTH      */
     3 UDF_DBINFO_QUALIF CHAR(128),       /* QUALIFIER NAME        */
     3 UDF_DBINFO_TLEN BIN FIXED(15),     /* TABLE LENGTH          */
     3 UDF_DBINFO_TABLE  CHAR(128),       /* TABLE NAME            */
     3 UDF_DBINFO_CLEN BIN FIXED(15),     /* COLUMN LENGTH         */
     3 UDF_DBINFO_COLUMN CHAR(128),       /* COLUMN NAME           */
     3 UDF_DBINFO_RELVER CHAR(8),         /* DB2 RELEASE LEVEL     */
     3 UDF_DBINFO_PLATFORM BIN FIXED(31), /* DATABASE PLATFORM     */
     3 UDF_DBINFO_NUMTFCOL BIN FIXED(15), /* # OF TF COLS USED     */
     3 UDF_DBINFO_RESERV1 CHAR(24),       /* RESERVED              */
     3 UDF_DBINFO_TFCOLUMN PTR,           /* -> TABLE FUN COL LIST */
     3 UDF_DBINFO_RESERV2 CHAR(24);       /* RESERVED              */
 DCL (ADDR,LENGTH,SUBSTR,NULL)   BUILTIN;
 EXEC SQL INCLUDE SQLCA;

 /* ********************************************************** */
 /* RETURNS THE LAST DAY OF THE MONTH IN FORMAT 'DD'           */
 /* LASTDAY(DATE('2000-08-18'))   ->   31                      */
 /* ********************************************************** */

 EXEC SQL SELECT DAY(DATE(STRIP(CHAR(YEAR(:UDF_PARM1)))||'-'||
        SUBSTR(DIGITS(MONTH(:UDF_PARM1)),9,2)||'-01')
        + 1 MONTH - 1 DAY)
      INTO :UDF_RESULT
 FROM SYSIBM.SYSDUMMY1 WITH UR;

END LASTDAY;
```

Bernard Zver
DBA (Slovenia)

Generating recovery control cards

This REXX EXEC helps DBAs in building recovery control cards when the recovery to be performed is of the type TOCOPY. Based on a selection criterion given by the DBA, this EXEC reads the SYSIBM.SYSCOPY catalog table, selects image copy records based on the selection criterion, and displays these entries to the DBA in another panel. A DBA can browse these entries in the panel and select the entries required for recovery. Once these entries are selected, the EXEC generates SYSIN control cards (RECOVER TABLESPACE …. TOCOPY …..) to be used by the DB2 RECOVER utility. These control statements are generated in the ascending file sequence number order. This reduces tapes mount and dismount processes when multiple image copy datasets exist on the same tape. This EXEC also generates control cards for the recovery of the corresponding indexes (RECOVER INDEX …).

In the first step, it displays a panel, PRCVR, where the user can provide the values for:

1 Subsystem ID – this is the DB2 subsystem name from where SYSCOPY entries are to be retrieved. In cases of data sharing, this is the group name.

2 Database name – this is the database containing tablespaces requiring recovery.

3 SYSCOPY rows choice – this has three options:

   – Retrieve all image copy rows (ICTYPE='F') for a given tablespace name.

   – Retrieve all image copy rows (ICTYPE='F') for a given date and time range.

   – Retrieve all image copy rows (ICTYPE = 'F') for a combination of a given tablespace name and given date and time range.

4 Tablespace name – for choices 1 and 3.

5 Date range – for choice 2 and 3.

6 Time range – for choice 2 and 3.

7 Dataset name – name of dataset for generating recovery control cards. Allocate this dataset first.

By default, the display allows you to select all the entries for the current date.

In the second step (once required entries are given in the PRCVR panel), it reads the DB2 catalog table SYSIBM.SYSCOPY rows for *ICTYPE = 'F'* and qualifying SYSCOPY row's choice (given in the panel), orders them, and displays these entries in a second panel, PRCVR2. The second panel allows the DBA to browse through all of these entries and select the required ones (depending on the date and time the image copy was taken for a tablespace, and how the DBA wishes to recover a tablespace from that image copy dataset) by typing 'S' to select that entry or by making it blank (to de-select an entry that was previously selected). The DBA can also use F7/F8 keys to browse through multiple screens. Once he selects all his entries, he can press F3 to return to the PRCVR panel. In the first panel, he can again make selections for another set of tablespaces. Once you have selected all the required entries in the first panel, press F3 and the EXEC generates the required recovery control cards for the dataset specified in the first panel.

In order to configure this REXX EXEC for your installation, make the following changes in this REXX program:

- For variable DYNM_SQL_PGM_LOADLIB, provide the name of the load library containing the Dynamic SQL program load module.
- For variable DYNM_SQL_PGM, provide the name of the Dynamic SQL program.
- For variable FILEO, provide the name of a dataset where you want to build your recovery control cards.
- ROWS_LIMIT is a variable which allows you to control the total number of rows selected by your query.
- In cases where you are selecting a large number of rows, you have

to change the sizes of the work datasets used inside the REXX EXEC appropriately.

```
/* REXX */
  DYNM_SQL_PGM_LOADLIB = 'DYNSQL.RUNLIB.LOAD'
                         /*  LOAD LIBRARY FOR DYNAMIC SQL PROGRAM */
  FILEO = 'SYSIN.OUTPUT'
                         /*  DATASET FOR GENERATING CONTROL CARD */
  DYNM_SQL_PGM = 'DYNSQL'
                         /*  DYNAMIC SQL PROGRAM */
  SSID = '    '          /*  DB2 SUBSYSTEM NAME    */
  DBNAME = '      '      /*  DATABASE NAME    */
  PTSNAME = '      '
  ROWS_LIMIT = 6ØØ       /*  MAX LIMIT ON SYSCOPY RECORDS */
  W = ' '
  ROW_COUNT = Ø
  COUNT_IND ='Y'
  CUR_DATE = SUBSTR(DATE('S',,),3)
  STDATE = CUR_DATE
  ENDDATE = CUR_DATE
  STTIME = 'ØØØØØ1'
  ENDTIME = '235959'
  MSG = ''
  EOF = 'NO'
  W_VOLSER = ''
  W_VOLIND = Ø
  ADDRESS "ISPEXEC"
  "LIBDEF ISPPLIB DATASET ID ( 'EXT.EXT1SKP.TEMP.LIBP' )"
                         /*  ISPF PANEL LIBRARY */
  "LIBDEF ISPTLIB DATASET ID ( 'EXT.EXT1SKP.ISPTLIB' )"
                         /*  ISPF TABLE LIBRARY */
  DO WHILE EOF = 'NO'
    ADDRESS "ISPEXEC"
    "DISPLAY PANEL(PRCVR)"
     MSG = ''
    IF  SSID = ' '  THEN
        DO
           EOF = 'YES'
           LEAVE
        END
    IF DBNAME = ' '          THEN
       DO
           MSG = 'DATABASE NAME INVALID'
           ITERATE
       END
    IF W = '1' | W = '2' | W = '3'   THEN
       NOP
    ELSE
       DO
           MSG = 'CHOICE INVALID'
       ITERATE
   END
IF W = '1' | W = '3' THEN
   IF PTSNAME = ' '  THEN
      DO
          MSG = 'ENTER TABLESPACE NAME, FOR CHOICE 1 OR 3'
          ITERATE
      END
IF W = '2' | W = '3' THEN
   IF STDATE < 'ØØØ1Ø1'  THEN
      DO
          MSG = 'COPY DATE IS TOO OLD'
          ITERATE
      END
IF W = '2' | W = '3' THEN
   IF STDATE > CUR_DATE  THEN
      DO
          MSG = 'START DATE IS FUTURE DATE'
          ITERATE
      END
IF W = '2' | W = '3' THEN
   IF ENDDATE < 'ØØØ1Ø1'  THEN
      DO
          MSG = 'COPY DATE IS TOO OLD'
          ITERATE
      END
IF W = '2' | W = '3' THEN
   IF ENDDATE > CUR_DATE  THEN
      DO
          MSG = 'END DATE IS FUTURE DATE'
          ITERATE
      END
IF W = '2' | W = '3' THEN
   IF ENDDATE < STDATE  THEN
      DO
          MSG = 'DATE RANGE IS INVALID'
          ITERATE
      END
IF W = '2' | W = '3' THEN
   IF STTIME < 'ØØØØØ1'  THEN
      DO
          MSG = 'START TIME IS INVALID'
          ITERATE
      END
IF W = '2' | W = '3' THEN
   IF STTIME > '235959'  THEN
      DO
          MSG = 'START TIME IS INVALID'
          ITERATE
      END
```

```
    IF W = '2' | W = '3' THEN
       IF ENDTIME < 'ØØØØØ1'  THEN
          DO
              MSG = 'END TIME IS INVALID'
              ITERATE
          END
    IF W = '2' | W = '3' THEN
       IF ENDTIME > '235959'  THEN
          DO
              MSG = 'END TIME IS INVALID'
              ITERATE
          END
    IF SYSDSN("'"FILEO"'") <> "OK" THEN
      DO
        MSG = 'CONTROL CARD DATASET DOES NOT EXIST'
        ITERATE
      END
    CALL PROCESS_LOOP
  END /* DO WHILE */
  ADDRESS TSO
  "EXECIO Ø DISKW DATAOUT (FINIS"
  "FREE F(DATAOUT)"
  ADDRESS "ISPEXEC"
  "LIBDEF ISPPLIB "
  EXIT

/****************************************************************/
/* PROCESS FOR TABLESPACE / DATE-TIME SELECTED ON PANEL         */
/****************************************************************/
  PROCESS_LOOP:
  ADDRESS TSO
  "ALLOC DA('"||FILEO||"')    F(DATAOUT) SHR"
  ADDRESS TSO
  "EXECIO Ø DISKW DATAOUT (OPEN"
  LOOP_EOF = 'NO'
  TOT_ROWS = Ø
  SQLID = ''
  J_IND = Ø
  I = Ø
  J = Ø
  SQLOK='YES'
  ADDRESS "ISPEXEC" "TBCREATE TSCOPY"||,
      " KEYS (TSNAME DSM ICDATE ICTIME LVL)"||,
      " NAMES (VOLSER SRL DSNAME A)"||,
      " NOWRITE REPLACE"
  CALL PØØØ1_BLD_TSCOPY
  IF ROW_COUNT = Ø THEN
     DO
         MSG = 'NO ROWS SELECTED '
         RETURN
     END
  IF ROW_COUNT > ROWS_LIMIT THEN
     DO
         MSG = 'NUMBER OF ROWS LIMIT EXCEEDED '
         RETURN
     END
  DO WHILE LOOP_EOF = 'NO'
     ADDRESS "ISPEXEC" "TBDISPL TSCOPY PANEL (PRCVR2)"
     IF  RC = 8  THEN
         LOOP_EOF = 'YES'
     IF ZTDSELS > Ø THEN
        DO
          CALL PROCESS_DTL
        END
     SELCOUNT = ZTDSELS
     DO WHILE ZTDSELS  > 1
        ADDRESS "ISPEXEC" "TBDISPL TSCOPY "
        IF ZTDSELS > Ø THEN
           DO
              CALL PROCESS_DTL
           END
     END
  END
  CALL PROCESS_SEL
  ADDRESS "ISPEXEC"
  "LIBDEF ISPTLIB "
  RETURN
/****************************************************************/
/* CREATE RECORDS IN ISPF TABLE                                 */
/****************************************************************/
PØØØ1_BLD_TSCOPY:
    CALL PROCESS_SQL
    IF ROW_COUNT = Ø  THEN
       RETURN
    IF ROW_COUNT > ROWS_LIMIT THEN
       RETURN
    DO I = 1 TO TOT_ROWS
      A = ''
      TSNAME = TTSNAME.I
      DSM    = TDSM.I
      ICDATE = TICDATE.I
      ICTIME = TICTIME.I
      LVL    = TLVL.I
      VOLSER = TVOLSER.I
      SRL    = TSRL.I
      DSNAME = TDSNAME.I
      ADDRESS "ISPEXEC" "TBADD TSCOPY"
      IF RC > Ø  THEN
        DO
          SAY 'TBADD ERROR '||RC
          ADDRESS "ISPEXEC" "TBEND TSCOPY"
          EXIT
        END
    END
  ADDRESS "ISPEXEC" "TBSORT TSCOPY "||,
                    "FIELDS(TSNAME,C,A,DSM,C,A,ICDATE,C,A,ICTIME,C,A)"
  ADDRESS "ISPEXEC" "TBTOP TSCOPY"
  IF RC =4 THEN
    DO
      SAY 'TBTOP ERROR '||RC
      ADDRESS "ISPEXEC" "TBEND TSCOPY"
      EXIT
    END
RETURN;
/****************************************************************/
/* PROCESS RECORDS IN ISPF TABLE                                */
/****************************************************************/
PROCESS_DTL:
    PARSE UPPER VAR A ACT
    A = ACT
    ADDRESS "ISPEXEC" "TBMOD TSCOPY"
RETURN;
/****************************************************************/
/* CREATE SYSIN AND RUN DYNAMIC SQL                             */
/****************************************************************/
PROCESS_SQL:
  COUNT_IND = 'Y'
  ADDRESS TSO
  "DELSTACK"
  "ALLOC DD(SYSPRINT) NEW DELETE UNIT(VIO)"
  IF RC <> Ø THEN DO
    SAY 'ERROR IN SYSPRINT ALLOC *' || RC
    EXIT
  END
  "ALLOC DD(SYSIN) NEW DELETE UNIT(VIO) RECFM (F B) LRECL (8Ø)"
  IF RC <> Ø THEN DO
     SAY 'ERROR IN SYSIN ALLOC *' || RC
     RETURN
  END
  CALL BUILD_SYSIN
  ADDRESS TSO
  "DELSTACK"
  CALL P_RUNPGM
  ADDRESS TSO
  "FREE  FI(SYSIN)"
  "FREE  FI(SYSPRINT)"
  IF ROW_COUNT = Ø THEN
     RETURN
  IF ROW_COUNT >  ROWS_LIMIT THEN
     RETURN
  COUNT_IND = 'N'
  ADDRESS TSO
  "DELSTACK"
  "ALLOC DD(SYSPRINT) NEW DELETE UNIT(VIO)"
  IF RC <> Ø THEN DO
    SAY 'ERROR IN SYSPRINT ALLOC *' || RC
    EXIT
  END
  "ALLOC DD(SYSIN) NEW DELETE UNIT(VIO) RECFM (F B) LRECL (8Ø)"
  IF RC <> Ø THEN DO
     SAY 'ERROR IN SYSIN ALLOC *' || RC
     RETURN
  END
  CALL BUILD_SYSIN
  ADDRESS TSO
  "DELSTACK"
  CALL P_RUNPGM
  ADDRESS TSO
  "FREE  FI(SYSIN)"
  "FREE  FI(SYSPRINT)"

RETURN

/****************************************************************/
/* BUILD SYSIN CONTROL CARD                                     */
/****************************************************************/
BUILD_SYSIN:

  IF COUNT_IND = 'N'  THEN
     DO
        RECORD = "SELECT TSNAME||'  '||"
        QUEUE RECORD
        RECORD = "       SUBSTR(DIGITS(DSNUM),8,3)||'  '||"
        QUEUE RECORD
        RECORD = "       ICDATE||'  '||"
        QUEUE RECORD
        RECORD = "       ICTIME||'  '||"
        QUEUE RECORD
        RECORD = "       SHRLEVEL||'  '||"
        QUEUE RECORD
        RECORD = "       SUBSTR(DSVOLSER,1,6)||'  '||"
        QUEUE RECORD
        RECORD = "       SUBSTR(DIGITS(FILESEQNO),8,3)||'  '||"
        QUEUE RECORD
        RECORD = "       STRIP(DSNAME) "
        QUEUE RECORD
     END
  ELSE
     DO
```

```
        RECORD = "SELECT COUNT(*)      "
        QUEUE RECORD
     END
  RECORD = "FROM   SYSIBM.SYSCOPY "
  QUEUE RECORD
  RECORD = "WHERE  DBNAME = "||"'"||DBNAME||"'"||'  '
  QUEUE RECORD
  RECORD = "AND    ICTYPE = 'F' "
  QUEUE RECORD
  IF W = '1' | W = '3'  THEN
     DO
        RECORD = "AND    TSNAME = "||"'"||PTSNAME||"'"||'  '
        QUEUE RECORD
     END
  IF W = '2' | W = '3'  THEN
     DO
        RECORD = "AND    ICDATE >= "||"'"||STDATE||"'"||'  '
        QUEUE RECORD
        RECORD = "AND    ICDATE <= "||"'"||ENDDATE||"'"||'  '
        QUEUE RECORD
        RECORD = "AND    ICTIME >= "||"'"||STTIME||"'"||'  '
        QUEUE RECORD
        RECORD = "AND    ICTIME <= "||"'"||ENDTIME||"'"||'  '
        QUEUE RECORD
     END
  RECORD = "       ORDER BY 1 "
  QUEUE RECORD

   QUEUE ''
   "EXECIO * DISKW SYSIN (FINIS"
   IF RC <> Ø THEN DO
      SAY     '* ERROR WRITING SYSIN FILE *RC* ' || RC
      EXIT
   END

RETURN

/*****************************************************************/
/* RUN DYNAMIC SQL PROGRAM                                       */
/*****************************************************************/
P_RUNPGM:
    DUMMY = OUTTRAP("OUTPUT_LINE.","*")
    Q1 = " RUN PROGRAM ("||DYNM_SQL_PGM||") PLAN("||DYNM_SQL_PGM||") "
    Q1 = Q1||"  LIB ('"||DYNM_SQL_PGM_LOADLIB||"')"
    QUEUE Q1
    QUEUE "END"
    ADDRESS TSO
    "DSN SYSTEM("||SSID||")"
    IF RC > 4 THEN
      DO
```

```
        SQLOK='NO'
        SAY 'DSN COMMAND  FAILED,  RERTURN CODE = '||RC
      END
    "EXECIO * DISKR SYSPRINT (FINIS"
    IF RC <> Ø THEN DO
       SAY     '* ERROR READING INPUT FILE   *RC* ' || RC
       EXIT
    END
    NREC = QUEUED()
    DO J= 1 TO NREC
       PULL RECORD
       IF SQLOK = 'YES' THEN
          DO
             REC1 = RECORD
             IF POS('_|',REC1) = Ø    THEN
                ITERATE
             ELSE
                DO
                   IF COUNT_IND = 'N'  THEN
                      CALL PROCESS_REC1
                   ELSE
                      DO
                         NPOS =  POS('_|',REC1)
                         REC2 = SUBSTR(REC1,(NPOS+2))
                         REC2 = TRANSLATE(REC2,' ','|')
                         ROW_COUNT = WORD(REC2,1)
                      END
                END
          END
       ELSE
          DO
             SAY RECORD
          END
     END
     ADDRESS TSO
     "DELSTACK"
    /*CALL BUILT_REPORT */
    /* CALL BROWSE_REPORT */
RETURN

/****************************************************************/
/* EXTRACT DETAILS FROM DYNAMIC SQL RUN                        */
/****************************************************************/
PROCESS_REC1:
   IF TOT_ROWS >= ROWS_LIMIT THEN
      RETURN
   NPOS =  POS('_|',REC1)
   REC2 = SUBSTR(REC1,(NPOS+2))
   REC2 = TRANSLATE(REC2,' ','|')
   TOT_ROWS         = TOT_ROWS + 1
   NWORDS           = WORDS(REC2)
   TTSNAME.TOT_ROWS = WORD(REC2,1)
   TDSM.TOT_ROWS    = WORD(REC2,2)
   TICDATE.TOT_ROWS = WORD(REC2,3)
   TICTIME.TOT_ROWS = WORD(REC2,4)
   TLVL.TOT_ROWS    = WORD(REC2,5)
   IF NWORDS = 8 THEN
      DO
         TVOLSER.TOT_ROWS = WORD(REC2,6)
         TSRL.TOT_ROWS = WORD(REC2,7)
         TDSNAME.TOT_ROWS = WORD(REC2,8)
      END
   ELSE
      DO
         TVOLSER.TOT_ROWS = ' '
         TSRL.TOT_ROWS    = WORD(REC2,6)
         TDSNAME.TOT_ROWS = WORD(REC2,7)
      END
RETURN

/****************************************************************/
/* PROCESS RECORDS SELECTED BY USER                             */
/****************************************************************/
PROCESS_SEL:
ADDRESS "ISPEXEC" "TBSORT TSCOPY "||,
                  "FIELDS(A,C,D,SRL,C,A,VOLSER,C,A)"
ADDRESS "ISPEXEC" "TBTOP TSCOPY"
TB_EOF='NO'
DO WHILE TB_EOF='NO'
    ADDRESS "ISPEXEC" "TBSKIP TSCOPY"
    IF RC=8 THEN
       TB_EOF='YES'
    ELSE
       DO
          IF A = 'S' THEN
             CALL PROCESS_GEN_TS
       END
END
DO I = 1 TO 5
   REC3=' '
   CALL PROCESS_WRITE
END
ADDRESS "ISPEXEC" "TBTOP TSCOPY"
TB_EOF='NO'
DO WHILE TB_EOF='NO'
    ADDRESS "ISPEXEC" "TBSKIP TSCOPY"
    IF RC=8 THEN
       TB_EOF='YES'
    ELSE
       DO
```

```
          IF A = 'S' THEN
             CALL PROCESS_GEN_IX
       END
END

RETURN

/*****************************************************************/
/* GENERATE CONTROL CARDS FOR TABLESPACE RECOVERY               */
/*****************************************************************/
PROCESS_GEN_TS:
IF W_VOLIND = Ø THEN
   DO
      W_VOLIND = 1
      W_VOLSER = VOLSER
   END
IF VOLSER <> W_VOLSER THEN
   DO
      REC3=' '
      CALL PROCESS_WRITE
   END
REC3=' RECOVER TABLESPACE  '
REC3=REC3||SUBSTR(DBNAME||'.'||TSNAME,1,2Ø)
REC3=REC3||'  DSNUM '
IF DSM = 'ØØØ' THEN
   REC3=REC3||' ALL '
ELSE
   REC3=REC3||DSM
CALL PROCESS_WRITE
REC3='         TOCOPY  '
REC3=REC3||SUBSTR(DSNAME,1,5Ø)
CALL PROCESS_WRITE
IF VOLSER = ' ' THEN
   DO
      REC3='                 TOVOLUME  CATALOG '
   END
ELSE
   DO
      REC3='                 TOVOLUME  '
      REC3=REC3||VOLSER
      IF SRL <> 'ØØØ' THEN
         REC3=REC3||'  TOSEQNO  '||SRL
   END
CALL PROCESS_WRITE

RETURN

/*****************************************************************/
/* GENERATE CONTROL CARDS FOR INDEX RECOVERY                    */
/*****************************************************************/
```

```
PROCESS_GEN_IX:
REC3=' RECOVER INDEX  (ALL) TABLESPACE  '
REC3=REC3||SUBSTR(DBNAME||'.'||TSNAME,1,2Ø)
IF DSM = 'ØØØ' THEN
   NOP
ELSE
   REC3=REC3||' PART '||DSM
CALL PROCESS_WRITE

RETURN

/****************************************************************/
/* WRITE IN OUTPUT DATASET                                     */
/****************************************************************/
PROCESS_WRITE:
OUTLIST.1 = REC3
ADDRESS TSO
"EXECIO 1 DISKW DATAOUT (STEM OUTLIST."
RETURN
```

PANEL PRCVR

```
)ATTR
/**********************************************************************/
/*        PANEL NAME - PRCVR                                          */
/*        FIRST PANEL FOR TABLESPACE / DATE & TIME OPTIONS            */
/*                                                                    */
/**********************************************************************/
   + TYPE(TEXT)   INTENS(LOW)   COLOR(BLUE)         SKIP(ON)
   % TYPE(TEXT)   INTENS(HIGH)  COLOR(WHITE)        SKIP(ON)
   $ TYPE(TEXT)   INTENS(HIGH)  COLOR(RED)          SKIP(ON)
   * TYPE(TEXT)   INTENS(HIGH)  COLOR(YELLOW)       SKIP(ON)
   # TYPE(OUTPUT) INTENS(HIGH)  COLOR(TURQUOISE)    CAPS(ON)
   ~ TYPE(TEXT)   INTENS(HIGH)  COLOR(RED)          CAPS(ON)
   ! TYPE(INPUT)  INTENS(HIGH)  COLOR(GREEN)   CAPS(ON) HILITE(USCORE)
)BODY CMD(C)
%——————————+ RECOVERY  SPECIFICATIONS %—————————
%OPTION ===_C                                             %SCR-_AMT +
+                                                                     +
+  *To end dialog, Enter blanks in Subsystem Id+                      +
+                                                                     +
+                                                                     +
+        %Subsystem Id        : !z   +                                +
+        %Database Name       : !z       +                            +
+        %SYSCOPY rows Choice : !z+                                   +
+                                    %1 - For a specific Tablespace    +
+                                    %2 - For a Date & Time Range      +
+                                    %3 - For a Combination of Both    +
+ ~————————————————————————————————— +
+                                                                     +
+        %Choice 1 / 3 -  Tablespace Name : !z       +                +
+                                                                     +
+        %Choice 2 / 3 -  Date Range      : !z     +%- !z     +       +
+                        %Time Range      : !z     +%- !z     +       +
+                                                                     +
+   %Data set Name: !z                                              + +
+   %Control card                                                     +
+                                                                     +
+                                                                     +
+        #msg                                                         +
)INIT
 .ZVARS = '(ssid,dbname,w,ptsname,stdate,enddate,sttime,endtime,fileo)'
)END
```

PANEL PRCVR2

```
)ATTR
/**********************************************************************/
/*                 PANEL NAME - PRCVR2                                */
/*                 PANEL FOR SYSCOPY RECORDS                          */
/*                                                                    */
/**********************************************************************/
   + TYPE(TEXT)   INTENS(LOW)   COLOR(BLUE)         SKIP(ON)
   % TYPE(TEXT)   INTENS(HIGH)  COLOR(WHITE)        SKIP(ON)
   $ TYPE(TEXT)   INTENS(HIGH)  COLOR(RED)          SKIP(ON)
   # TYPE(OUTPUT) INTENS(HIGH)  COLOR(TURQUOISE)    CAPS(ON)
   @ TYPE(OUTPUT) INTENS(HIGH)  COLOR(YELLOW)       CAPS(ON)
   ! TYPE(INPUT)  INTENS(HIGH)  COLOR(GREEN) CAPS(ON) HILITE(USCORE)

)BODY CMD(C)
%——————————+  LIST OF SYSCOPY RECORDS  %——————————
%OPTION ===_C                                             %SCR-_AMT +
+                                                                     +
+         Subsystem: @SSID +    Database: @DBNAME                     +
+                                                                     +
$A  Table   DS  ICDate  ICTime  SHR Volser  Seq Dataset Name  +
$   Space   Num                 LVL         No                +
%
%———————————————————————————————————
)MODEL
!A+#TSNAME #DSM#ICDATE+#ICTIME+#LVL#VOLSER+#SRL#DSNAME
)INIT
)PROC
  VER (&C LIST,END,' ',CAN,CANCEL)
)END
```

Sharad K Pande
Senior DBA
PriceWaterhouseCooper (USA)

Unique and primary key constraints in DB2 for OS/390 Version 7

It was another one of those Sunday night DB2 application implementations – complete with coffee, donuts, chocolate, dumps, and my dog, Skip, helping himself to everything he could reach. My partner, Charles, was braving the cold to fetch some pizza and soft drinks for us because it was already beginning to look like a longer evening than we expected. Doing nothing more than waiting for a long-running job to finish, it was becoming harder and harder to keep my eyes open. I was bolted alert by a blast of the north wind. Charles burst through the door with the promised pizza, soft drinks, and a lot on his mind. …

“Prepare yourself, Wilburt, for a singular situation we haven’t seen the likes of in a new version of DB2 since farther back than my fading memory can recall. The DDL you are planning to run at 11pm tonight may not execute successfully in DB2 Version 7 without modifications. In fact, a negative SQLCODE may be about to display on your screen just when you think you are ready to logoff. Be assured, though, that a simple change to the DDL will quickly get you on your way to a full set of zero return codes and a full night’s sleep.”

“Absolutely true, Charles, I’m already informed of this difference! The specifics are that SQLCODE –538 (FOREIGN KEY relname DOES NOT CONFORM TO THE DESCRIPTION OF A PARENT KEY OF TABLE owner.name) is issued in Version 7 when you try to define a foreign key that refers to a unique non-primary index on a parent table that does not have a unique constraint.

“I have also analysed the impact and found that foreign keys referring to non-primary unique indexes are a rather infrequent occurrence for us. Only three such foreign keys are affected in the thousands of lines of DDL I’m running tonight, and I spent no more than a few minutes making adjustments. However, the changes to unique constraints in Version 7 involve more than just this.”

HIGHLIGHTS

"Yes indeed, Wilburt, a good bit more. Changes in Version 7 to unique and primary key constraints also include the following:

- A user-specified name may be optionally assigned to constraints, similar to the way that names are assigned to foreign keys.
- A constraint may be added to an existing table with the ALTER TABLE statement, after the enforcing unique index has been created.
- A constraint must be dropped before its enforcing index can be dropped.
- Information about constraints that are created in Version 7 is now stored in two new DB2 catalog tables – SYSIBM.SYSTABCONST and SYSIBM.SYSKEYCOLUSE.
- When a DB2 subsystem is migrated from Version 5 or 6 to Version 7, existing constraints remain untouched by the CATMAINT process. In other words, after the CATMAINT process, existing constraint definitions are still stored in the same places. Immediately following a CATMAINT upgrade, the two new catalog tables are empty, because the CATMAINT process does not convert existing information into the new tables."

NEW RESTRICTIONS

"All of this deserves a thorough examination, Charles, but let's first understand how to avoid an SQLCODE –538. Before Version 7, the only prerequisite for creating a foreign key referencing a non-primary parent key was a unique index on the parent table. In Version 7, we still must have a unique index, but in addition a unique constraint must also exist on the parent. Let's look at some DDL that creates parent and child tables, along with a couple of foreign keys and the required indexes.

"The following code shows a unique constraint (in italics) defined in the CREATE TABLE statement. You'll recall that the UNIQUE parameter existed prior to Version 7, but it was not a prerequisite for creating a foreign key. However, omitting it from this sample DDL in

Version 7 results in an SQLCODE –538."

```
CREATE TABLE  V7OWNR.PARENT_TB
        (COL_PKEY        SMALLINT NOT NULL
        ,COL_UNIQ1       INTEGER  NOT NULL
        ,COL_STUFF       CHAR(5)
   ,PRIMARY KEY
        (COL_PKEY)
   ,UNIQUE
        (COL_UNIQ1)  )

CREATE  UNIQUE
       INDEX  V7OWNR.INDEX_PK
          ON V7OWNR.PARENT_TB
               (COL_PKEY)

CREATE  UNIQUE
       INDEX  V7OWNR.INDEX_UC1
          ON V7OWNR.PARENT_TB
               (COL_UNIQ1)

CREATE TABLE  V7OWNR.CHILD_TB1
         (COL_CHLD1_A            SMALLINT NOT NULL
         ,COL_MORE_STUFF     CHAR(1Ø) NOT NULL)

CREATE TABLE  V7OWNR.CHILD_TB2
         (COL_CHLD2_B            INTEGER  NOT NULL
         ,COL_GOOD_STUFF     CHAR(15) NOT NULL)

ALTER  TABLE  V7OWNR.CHILD_TB1
       FOREIGN KEY  CHLDFK1
            (COL_CHLD1_A)
       REFERENCES V7OWNR.PARENT_TB
            (COL_PKEY)

ALTER  TABLE  V7OWNR.CHILD_TB2
       FOREIGN KEY  CHLDFK2
            (COL_CHLD2_B)
       REFERENCES  V7OWNR.PARENT_TB
            (COL_UNIQ1)
```

NEW CREATE TABLE PARAMETER CONSTRAINT

"Good job getting straight to the point of the minimum DDL changes that are required to get around SQLCODE -538, Wilburt. I will build on your example by demonstrating that Version 7 now allows you to optionally specify a name for unique and primary key constraints. This is accomplished with the new parameter CONSTRAINT, shown in italics below. If the CONSTRAINT parameter is omitted, as it is in the example above, the constraint name defaults to the name of the first column in the key.

“Notice in the following code that the CONSTRAINT keyword is coded once for each constraint, if the table has multiple constraints. Two unique constraints are named *MYUNIQ1* and *MYUNIQ2*. Tables don’t have to have primary keys, even if they have unique constraints. However, another way to define this table would have been with a primary key and one unique constraint, just like in the first example.”

```
CREATE TABLE  V7OWNR.PARENT_TB
      (COL_UNIQ1     INTEGER  NOT NULL
      ,COL_UNIQ2     DATE     NOT NULL
      ,COL_STUFF     CHAR(5)
,CONSTRAINT  MYUNIQ1
     UNIQUE
        (COL_UNIQ1)
 ,CONSTRAINT  MYUNIQ2
     UNIQUE
        (COL_UNIQ2)  )
```

ADDING CONSTRAINTS WITH ALTER TABLE STATEMENT

“Even better, Charles, is that the same parameter has been added to the ALTER TABLE statement – so it can also be optionally used to define the named primary key and unique constraints on an existing table. Just as in the CREATE TABLE statement, the name defaults to the name of the first column of the key if the CONSTRAINT keyword is omitted.

“Sample DDL that adds a primary key constraint named *MYPKEY* is shown below. Remember in prior DB2 versions how we could add a primary key with an ALTER TABLE statement? Now with the new CONSTRAINT parameter in Version 7, we can name the primary key as well. Observe that the statements must be executed in the order shown: the unique index must be created before the constraint.

```
CREATE  UNIQUE
       INDEX  V7OWNR.INDEX_PK
          ON V7OWNR.PARENT_TB
               (COL_PKEY)

ALTER  TABLE  V7OWNR.PARENT_TB
       ADD  CONSTRAINT  MYPKEY
```

```
             PRIMARY KEY  (COL_PKEY)
```

“A unique constraint named *MYUNIQ3* is added to a table in the code below. Prior to Version 7, we could not use an ALTER TABLE statement to add unique constraints. So naturally, this DDL will prove itself priceless when the occasion arises to add a foreign key that references a parent without a unique constraint. In that instance, we can alter the constraint into the table instead of having to drop and re-create the table. Notice again that the unique index must be created before the unique constraint can be added.”

```
CREATE  UNIQUE
       INDEX V7OWNR.INDEX_UC3
          ON V7OWNR.PARENT_TB
               (COL_UNIQ3)

ALTER  TABLE V7OWNR.PARENT_TB
       ADD  CONSTRAINT  MYUNIQ3
             UNIQUE  (COL_UNIQ3)
```

DROPPING CONSTRAINTS WITH ALTER TABLE STATEMENT

“DDL we’ve used in the past to drop primary keys is still valid. To refresh our memories, Wilburt, the code below shows a sample:

```
ALTER TABLE  V7OWNR.PARENT_TB
      DROP PRIMARY KEY
```

“To drop a unique constraint, we must use the ALTER TABLE statement with the CONSTRAINT parameter. The new syntax shown below works for primary key constraints, too. Note that the constraint name must be specified, even though a default name may have been used when it was created.”

```
ALTER TABLE  V7OWNR.PARENT_TB
      DROP CONSTRAINT  MYPKEY
```

CHANGING EXISTING CONSTRAINTS

“You can’t exactly ‘change’ constraints, Charles. The ALTER TABLE statement can only add and drop constraints. Changes to existing constraints such as adding, deleting, renaming, or reordering columns, or renaming the constraint, require the constraint to be dropped and recreated. Dropping a primary or unique constraint makes dependent foreign keys disappear too, so be prepared to recreate them as well."

DROPPING INDEXES THAT ENFORCE CONSTRAINTS

"Speaking of dropping constraints, Wilburt, we have one more important new rule to tuck away in our memories. When we drop an index (but not the table) that enforces a primary key or unique constraint, we must first drop the constraint. If the constraint exists when the DROP INDEX statement is executed, DB2 issues SQLCODE -669, ERROR: THE OBJECT CANNOT BE EXPLICITLY DROPPED. Before Version 7, if we dropped such an index, DB2 issued SQLCODE 625, WARNING: THE DEFINITION OF TABLE owner.name HAS BEEN CHANGED TO INCOMPLETE, and the index was dropped.

"The exception to this is that if the objects were created before Version 7, a unique constraint won't exist, just a unique index and a foreign key. In this case, Version 7 now requires us to drop the foreign key before we drop the index, otherwise DB2 reminds us of this fact with SQLCODE –669.

"Just remember Wilburt, 'Create the index first, then the constraint. When you want to drop the index, you must first drop either the constraint (if one exists), or the foreign key (if no constraint exists)'. Of course, if we drop the table, then the constraints, indexes, and foreign keys all go down with the table, as always."

NEW DB2 CATALOG TABLES

"Your earlier point about specifying the constraint name reminds me to bring to your attention, Charles, my discoveries from querying the new DB2 catalog tables, where everything about constraints is stored. SYSIBM.SYSTABCONST contains one row for each constraint. SYSIBM.SYSKEYCOLUSE contains one row for each column in each constraint. By joining these two tables on columns CONSTNAME, TBCREATOR, and TBNAME, we can obtain a comprehensive report of both our primary and unique constraints. An SQL statement that selects my favourite columns from these two tables is displayed below:

```
SELECT
      A.CONSTNAME
     ,A.TBCREATOR
     ,A.TBNAME
     ,A.IXOWNER
     ,A.IXNAME
     ,A.TYPE
     ,B.COLNAME
     ,B.COLSEQ
     ,B.COLNO
FROM  SYSIBM.SYSTABCONST   A
            ,SYSIBM.SYSKEYCOLUSE B
WHERE  A.CONSTNAME  =  B.CONSTNAME
      AND A.TBCREATOR    =  B.TBCREATOR
      AND A.TBNAME           =  B.TBNAME
ORDER BY
      A.TBCREATOR
     ,A.TBNAME
     ,A.TYPE
     ,A.CONSTNAME
```

“Charles, let’s look at the report produced by this query. It shows the constraints we created in the second and third examples, and also some constraints created by the people in Department 123.”

```
TBCREATOR TBNAME CONSTNAME TYPE COLNAME COLSEQ COLNO IXOWNER  IXNAME
DEPT123  EMPL_HIST SURROGATE P IDENT_COL   1    1    DEPT123   EMPIX2
DEPT123  EMPL_HIST EMPLHIST  U EMP_NMBR    1    5    DEPT123   EMPIX1
DEPT123  EMPL_HIST EMPLHIST  U EMP_START_DT2    4    DEPT123   EMPIX1
V7OWNR   PARENT_TB MYPKEY    P COL_PKEY    1    1    V7OWNR    INDEX_PK
V7OWNR   PARENT_TB MYUNIQ1   U COL_UNIQ1   1    2    V7OWNR    INDEX_UC1
V7OWNR   PARENT_TB MYUNIQ2   U COL_UNIQ2   1    3    V7OWNR    INDEX_UC2
```

The columns are described as follows:

- TBCREATOR – owner of the parent table.
- TBNAME – name of the parent table.
- CONSTNAME – name of the constraint.
- TYPE – type of constraint (P = Primary, U = Unique).
- COLNAME – name of the column.
- COLSEQ – numeric position of the column in the key.
- COLNO – numeric position of the column in the table.
- IXOWNER – owner of the index enforcing the constraint.
- IXNAME – name of the index enforcing the constraint.

NAMING CONVENTIONS FOR CONSTRAINTS

“Viewing the information in your report, Wilburt, I know enough to reconstruct the constraint parameters in DEPT123’s CREATE TABLE DDL, shown above. The names they chose for their constraints and column names give us good clues about the reason for two unique keys on the same table. Referring back to the report, it appears that they are using an identity column for a surrogate key as the primary key. Looking at the columns in unique constraint EMPLHIST, I would say this table is a history of the dates of service of all the employees who have worked in the department. With EMP_STRT_DT as part of the key, it allows for employees who leave the department and return at a later date.

```
CREATE TABLE  DEPT123.EMPL_HIST
      ( <column definitions>
,CONSTRAINT  SURROGATE
     PRIMARY
        (IDENT_COL)
 ,CONSTRAINT  EMPLHIST
     UNIQUE
        (EMP_NMBR
        ,EMP_STRT_DT)  )
```

“Instituting good naming conventions for constraints, which helps describe their meaning and purpose, is a good idea, as it is in general for DB2 objects. It gives us a new opportunity to self-document and convey information about the design of our structures for developers and others who rely on the DB2 catalog to answer their questions.”

SOME THINGS NEVER CHANGE

“I did a little digging into the DB2 catalog myself, Wilburt, and found that the KEYSEQ column in SYSIBM.SYSCOLUMNS is still being maintained in Version 7. Therefore, all of our existing primary key reports are still reliable. They just won’t show the constraint name unless we update them to reference the new catalog tables. Of course, since we couldn’t tell from the catalog before Version 7 the exact state of unique constraints, we didn’t write queries about them.”

Cathy Lappe
BMC Software (USA)

DB2 stored procedures and dynamic cursors

With a more complete implementation of stored procedures in DB2 Versions 5 and 6, many new distributed access opportunities have been given to the DB2 developer. Many shops, ours included, severely limited or completely denied Dynamic SQL access to operational Online Transaction processing (OLTP) databases. The fear of reducing CICS response time by even tenths of a second was enough to keep Dynamic SQL out of all mission-critical databases.

At the same time, a great deal of pressure was mounting to give greater access to mission-critical data via the Web. Statically bound stored procedures were viewed as the optimal solution for this problem. Utilizing stored procedures with a combination of Active Server Pages (ASP) and ActiveX Data Objects (ADO) produced guaranteed access paths, with known performance results.

The newly-allowed DB2 access from the Web platform resulted in stored procedures becoming an integral member in the tool set used to produce distributed applications. In a very short time, hundreds of stored procedures were written, providing much-needed DB2 access and extending the OLTP database information to the Web.

However, as with any solution, stored procedure use was pushed until a definite weakness was exposed. Statically bound SQL has the advantage of pre-defined access paths, but it lacks the ability to change in a reporting type of environment. Many of the Web applications being developed needed the ability to perform sorts and look-ups based on a varying number of predicates.

Using Dynamic SQL, you would create a new query based on the user's input and re-issue the SQL statement to DB2. In the stored procedure environment, the same task was being performed by inserting additional SQL into the COBOL programs. What started out as one cursor turned into multiple cursors with different predicates and sort orders.

Below is just one example of the problem that faced the stored procedure programmer.

Initial query:

```
SELECT Fst_NM, Lst_NM, Addr_1_TXT, Addr_2_TXT, City_NM, State_CD, Zip_CD
WHERE Last_NM LIKE WS_LST_NM
AND   Martial_CD = 'M'
AND   Age_NUM > 65
ORDER BY Lst_NM, State_CD
```

Some sort and predicate variations:

```
WHERE Last_NM LIKE WS_LST_NM
AND   Martial_CD = 'M'
AND   Age_NUM < 65
ORDER BY Lst_NM, State_CD

WHERE Last_NM LIKE WS_LST_NM
AND   Martial_CD = 'M'
AND   Age_NUM > 65
ORDER BY Lst_NM, Zip_CD

WHERE Last_NM LIKE WS_LST_NM
ORDER BY Lst_NM, Age_NUM

WHERE Last_NM LIKE WS_LST_NM
AND   Income_AMT > 20000
ORDER BY Income_AMT
```

In each case, the access path selected utilized an index on Lst_NM; the only difference was sort order and/or the number of predicates. As one can see, the number of cursors needed to support the user request is equal to the number of possible sort orders multiplied by the number of predicate combinations. A simple Dynamic query coded as Static SQL in a stored procedure can result in an unmanageable number of cursors. One example we encountered had two predicate variations, with nine possible sort orders resulting in 18 different cursors. Clearly this was not acceptable.

We determined that it was necessary to combine the use of stored procedures with Dynamic SQL. Using Dynamic SQL within statically bound modules is nothing new. However, using it with the new cursor capabilities within stored procedures is.

Stored procedures use the RESULT-SET-LOCATOR in order to address where the answer set from a cursor is returned to. We combined this capability with a dynamically-prepared cursor. Our only requirement was that the dynamically-prepared cursor has a known access path. This meant that one or more predicates that resulted in a known access path had to be coded in the COBOL program. Other predicates and sort orders could be added or changed, but those predicates that were pre-determined to provide efficient access had to be hard-coded.

Below is one way this can be accomplished using the COBOL STRING function.

Defined in Working Storage:

```
Ø1  EMPLOYEE-LOC-1                  USAGE SQL TYPE IS
                                    RESULT-SET-LOCATOR VARYING.

Ø1  WS-DYNAMIC-STATEMENT.
    49  WS-DYNAMIC-STATEMENT-LEN   PIC S9(Ø4)  COMP VALUE +999.
    49  WS-DYNAMIC-STATEMENT-TXT   PIC  X(999).


EXEC SQL DECLARE EMPLOYEE-CUR-1 CURSOR
     WITH RETURN FOR DYNAM1
     END-EXEC.
```

Coded in the Procedure Division:

```
STRING "SELECT"
       SQL-SELECT-VAR-TEXT(1:SQL-SELECT-VAR-LENGTH)
       " FROM ACME_EMPLOYEE "
       " WHERE"
       " LST_NM LIKE 'S%'"
       " AND "
       SQL-PREDICATE_VAR_TEXT(1:SQL-PREDICATE-VAR-LENGTH)
       SQL-ORDERBY-VAR-TEXT(1:SQL-ORDERBY-VAR-LENGTH)
     DELIMITED BY SIZE
     INTO WS-DYNAMIC-STATEMENT-TXT.

EXEC SQL
     PREPARE DYNAM1 FROM :WS-DYNAMIC-STATEMENT
     END-EXEC.

EXEC SQL
     OPEN EMPLOYEE-CUR-1
     END-EXEC.
```

This example demonstrates the extreme amount of flexibility that this solution can provide. Here, the developer can choose to supply either no additional predicates or as many as they like. The same can be said of the sort order. This example will work if no sort order is provided, or if many columns are included in the order by clause. However, in some instances, the column(s) specified in the order by statement need to be referenced by integer position, instead of column name (for example: ORDER BY 2,1).

In this example, the columns selected are being determined by the developer. This means one cursor could satisfy a *SELECT count(*)* or return a list of all employees with a last name beginning with *S%*. In any case, since LST_NM is included as a hard-coded predicate, the access path should use the index associated with the LST_NM column.

By utilizing Dynamic cursors in a controlled manner, the distributed developer can be given as much control as the DBA deems necessary and safe. If the developer codes an incorrect column name, a SQL error will occur in the PREPARE statement. This error situation can be eliminated by forcing the stored procedure caller to specify parameters that indicate which columns they want returned, instead of the column name. This method would also control which columns are accessible via the stored procedure, thus adding an additional level of security.

Cursors could also be asked to return a list of columns separated by commas (CSV file). The developer may want to feed an answer set directly into MS Excel. He could request that the answer set return *column1* || *','* || *column2* || *','* etc. Again, one cursor could be used for an almost limitless combination of answer sets as long as the caller can handle them.

An additional benefit of using Dynamic cursors in a stored procedure environment is that you can use Dynamic SQL without having to grant direct access to the underlying objects. A developer's best security work in a distributed application can be useless if the user is capable of bypassing the application. Most distributed applications using Dynamic SQL require that all users have access to the actual DB2 objects. With this access, users can use MS Access or some other ODBC tool instead of the application. In the stored procedure environment, access to only the stored procedure is given to the caller. No authorization is needed to any object referenced within the stored procedure.

We have found including Dynamic cursors within DB2 stored procedures to be an extremely valuable tool. We now have the flexibility of Dynamic SQL, in a very controlled environment. Depending on the application, and the DB2 knowledge that specific developers have, you can give a varying amount of control to what the Dynamic SQL can perform.

Below is a complete working example of a DYNAMIC cursor with somewhat more limited flexibility:

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.    SPDYNACR.
*****************************************************************
*  ——THIS PROGRAM IS A COBOL II/DB2 STORED PROCEDURE ———-
*  PROGRAM:  SPDYNACR - TAKES INPUT TO PROCESS A RECEIVED
*                        OR SENT LIST.
*  PURPOSE:
*    BASED ON THE PARMS PASSED EITHER A RECEIVED OR SENT LIST
*    CURSOR IS OPENED FOR THE INTERNET/INTRANET VB DLL.
*    DYNAMIC SQL IS USED BECAUSE A SORT COLUMN NAME AND SORT
*    TYPE (ASCENDING OR DESCENDING) ARE PASSED ALLOWING FOR
*    9 DIFFERENT SORT ORDERS FOR EACH LIST TYPE (RECEIVED OR
*                                                SENT).
*  PROCESS LOGIC:
*      READ THE PARM PASSED AND EITHER PROCESS A RECEIVED OR
*      SENT LIST.
*  CALLED MODULES/COPYBOKS:
*      DB2CPER1  SQLCA     TABC1Ø2   TABC1Ø8   DB2CPER1
*      DB2ERRØ1 - DB2 ERROR HANDLER
 ENVIRONMENT DIVISION.
 DATA DIVISION.
*****************************************************************
*      W O R K I N G     S T O R A G E      S E C T I O N
*****************************************************************
 WORKING-STORAGE SECTION.
*
 Ø1  WS-WORKING-STORAGE-LITERAL          PIC X(36)   VALUE
     'SPDYNACR WORKING STORAGE BEGINS HERE'.
*
 Ø1  WS-SQL.
     Ø5  WS-SQLCODE                      PIC —99  VALUE ZERO.
     Ø5  WS-SQLCA-DESC                   PIC X(7Ø)  VALUE SPACES.
*
 Ø1  WS-DYNAMIC-SQL.
```

```
     49  WS-DYNAMIC-SQL-LEN         PIC S9(Ø4)  COMP VALUE +6ØØ.
     49  WS-DYNAMIC-SQL-TXT         PIC  X(6ØØ).
*** RECEIVED CURSOR SQL TEXT
 Ø1  WS-DYNAMIC-SQL-RECEIVED.
     Ø5  FILLER                          PIC X(Ø7) VALUE 'SELECT'.
     Ø5  FILLER                          PIC X(19)
         VALUE ' REFRL_ID'.
     Ø5  FILLER                          PIC X(19)
         VALUE ',CLNT_ID'.
     Ø5  FILLER                          PIC X(19)
         VALUE ',CLNT_NM'.
     Ø5  FILLER                          PIC X(19)
         VALUE ',CLNT_LAST_NM'.
     Ø5  FILLER                          PIC X(19)
         VALUE ',ORGN_DT'.
     Ø5  FILLER                          PIC X(21)
         VALUE ',A.STAT_CD'.
     Ø5  FILLER                          PIC X(19)
         VALUE ',STAT_CD_DESC'.
     Ø5  FILLER                          PIC X(21)
         VALUE ',A.DTL_STAT_CD'.
     Ø5  FILLER                          PIC X(19)
         VALUE ',DTL_STAT_CD_DESC'.
     Ø5  FILLER                          PIC X(21)
         VALUE ',A.LAST_UPDT_TS'.
     Ø5  FILLER                          PIC X(19)
         VALUE ',EXPN_DT'.
     Ø5  FILLER                          PIC X(19)
         VALUE ',FOLLOW_UP_DT'.
     Ø5  FILLER                          PIC X(19)
         VALUE ',REFRR_NM'.
     Ø5  FILLER                          PIC X(19)
         VALUE ',REFRR_LAST_NM'.
     Ø5  FILLER                          PIC X(23)
         VALUE 'FROM TABC1Ø2_REFRL   A '.
     Ø5  FILLER                          PIC X(23)
         VALUE '    ,TABC1Ø8_STAT_CD B '.
     Ø5  FILLER                          PIC X(33)
         VALUE 'WHERE A.STAT_CD       = B.STAT_CD'.
     Ø5  FILLER                          PIC X(38)
         VALUE '  AND A.DTL_STAT_CD   = B.DTL_STAT_CD '.
     Ø5  FILLER                          PIC X(24)
         VALUE '  AND   RECIP_CLNT_ID = '.
     Ø5  WS-RECIP-CLNT-ID-R              PIC 9(Ø9).
     Ø5  FILLER                          PIC X(25)
         VALUE "  AND   ORGN_DT BETWEEN '".
     Ø5  WS-BEGIN-ORGN-DT-R              PIC X(1Ø).
     Ø5  FILLER                          PIC X(Ø7)
         VALUE "' AND '".
     Ø5  WS-END-ORGN-DT-R                PIC X(1Ø).
     Ø5  FILLER                          PIC X(25)
```

```
         VALUE "' AND A.STAT_CD BETWEEN '".
     Ø5  WS-BEGIN-STAT-CD-R              PIC X(Ø2).
     Ø5  FILLER                          PIC X(Ø7)
         VALUE "' AND '".
     Ø5  WS-END-STAT-CD-R                PIC X(Ø2).
     Ø5  FILLER                          PIC X(Ø1)
         VALUE "'".
     Ø5  WS-ORDER-BY-SECTION-R.
         1Ø  FILLER                      PIC X(1Ø)
             VALUE ' ORDER BY '.
         1Ø  WS-ORDER-BY-COLUMN-R        PIC X(18).
         1Ø  FILLER                      PIC X(Ø1)
             VALUE ' '.
         1Ø  WS-ORDER-ASC-DESC-R-1       PIC X(Ø4).
         1Ø  FILLER                      PIC X(18)
             VALUE ' ,DTL_STAT_CD_DESC'.
         1Ø  FILLER                      PIC X(Ø1)
             VALUE ' '.
         1Ø  WS-ORDER-ASC-DESC-R-2       PIC X(Ø4).
*** SENT CURSOR SQL TEXT
 Ø1  WS-DYNAMIC-SQL-SENT.
     Ø5  FILLER                          PIC X(Ø7) VALUE 'SELECT'.
     Ø5  FILLER                          PIC X(18)
         VALUE ' REFRL_ID'.
     Ø5  FILLER                          PIC X(18)
         VALUE ',CLNT_ID'.
     Ø5  FILLER                          PIC X(18)
         VALUE ',CLNT_NM'.
     Ø5  FILLER                          PIC X(18)
         VALUE ',CLNT_LAST_NM'.
     Ø5  FILLER                          PIC X(18)
         VALUE ',ORGN_DT'.
     Ø5  FILLER                          PIC X(18)
         VALUE ',A.STAT_CD'.
     Ø5  FILLER                          PIC X(19)
         VALUE ',STAT_CD_DESC'.
     Ø5  FILLER                          PIC X(18)
         VALUE ',A.DTL_STAT_CD'.
     Ø5  FILLER                          PIC X(19)
         VALUE ',DTL_STAT_CD_DESC'.
     Ø5  FILLER                          PIC X(18)
         VALUE ',A.LAST_UPDT_TS'.
     Ø5  FILLER                          PIC X(18)
         VALUE ',EXPN_DT'.
     Ø5  FILLER                          PIC X(18)
         VALUE ',FOLLOW_UP_DT'.
     Ø5  FILLER                          PIC X(18)
         VALUE ',RECIP_NM'.
     Ø5  FILLER                          PIC X(18)
         VALUE ',RECIP_LAST_NM'.
     Ø5  FILLER                          PIC X(23)
```

```
           VALUE 'FROM TABC1Ø2_REFRL   A '.
       Ø5  FILLER                          PIC X(23)
           VALUE '    ,TABC1Ø8_STAT_CD B '.
       Ø5  FILLER                          PIC X(33)
           VALUE 'WHERE A.STAT_CD      = B.STAT_CD'.
       Ø5  FILLER                          PIC X(38)
           VALUE '  AND A.DTL_STAT_CD   = B.DTL_STAT_CD '.
       Ø5  FILLER                          PIC X(24)
           VALUE '  AND (REFRR_CLNT_ID  = '.
       Ø5  WS-REFRR-CLNT-ID-S              PIC 9(Ø9).
       Ø5  FILLER                          PIC X(24)
           VALUE ' OR  ORGNTR_CLNT_ID   = '.
```

Editor's note: this article will be concluded in the next issue.

Tim Albrecht
DB2 Database Administrator (USA)

November 1998 – November 2001 index

Items below are references to articles that have appeared in *DB2 Update* since November 1998. References show the issue number followed by the page number(s). Back-issues of *DB2 Update* are available back to Issue 73 (November 1998). See page 2 for details.

DB2 news

IBM has announced QMF High Performance Option (HPO) as a direct feature of DB2 for OS/390 V6 and DB2 for z/OS and OS/390 V7, and of QMF, QMF for Windows, and QMF HPO for z/OS and OS/390 V7.

QMF has new capabilities for the workstation environment and enhancements for the mainframe that help access and present data better. V7 works with data from DB2 for OS/390, DB2 for VSE and VM, DB2 for AS/400 to workstation servers running OS/2, Windows NT, AIX, and other Unix operating systems and massively parallel processors.

When coupled with DB2 DataJoiner, it allows access to non-relational and other data sources.

QMF V7 enhancements include improved command processing and defaults, new DB2 data types for large objects, VSE DRDA RUOW Application Requestor and DB2 for AS/400 support, Java-based query from a Web browser, aggregation, grouping, and formatting in the query results, point-and-click interface to QMF Form creation, and a personal data portal for the desktop, which can be used for launching centrally-shared queries and reports, sending results to desktop tools and browsers, and sharing already-formatted spreadsheets.

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

* * *

Savvion is to integrate DB2 UDB with the Savvion BusinessManager, promising the means to automate and manage varied business processes, ranging from enterprise-wide procurement through asset management to human resource systems.

BusinessManager helps automate operations both internally and across enterprises. Integrated management systems, including balanced scorecards, allow managers to continuously monitor, measure, and improve operational efficiencies.

Integrating DB2 as the database, says the firm, helps simplify database integration while optimizing system operation.

For further information contact:
Savvion, 5000 Old Ironsides Drive, Santa Clara, CA 95054, USA.
Tel: (40) 330 3400.
URL: http://www.savvion.com.

* * *

IBM has announced new tools, functions, and tool integration for its DB2 UDB and IMS databases. These include the DB2 Administration Tool for z/OS, Version 3.1, DB2 Automation Tool for z/OS, Version 1.2, DB2 High Performance Unload for z/OS, Version 1.2, DB2 Log Analysis Tool for z/OS, Version 1.2, DB2 Object Restore for z/OS, Version 1.2, DB2 Table Editor for iSeries, Version 4.2, and DB2 Table Editor for z/OS, Version 4.2.

Other tools include IBM DataPropagator for z/OS, Version 3.1, DB2 Buffer Pool Analyzer for z/OS, DB2 SQL Performance Analyzer for z/OS, Version 1.2, DB2 Data Export Facility for z/OS, DB2 Web Query Tool for iSeries, Version 1.2, and DB2 Web Query Tool for z/OS, Version 1.2.

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