143

DB2 update

September 2004

In this issue

3 Perl script to force users off a DB2 UDB LUW database
8 SQL scalar functions – part 2
12 A strategy for image copying large partitioned tablespaces using Real-Time Statistics
23 Modify column attributes
40 DB2 UDB LUW – High Performance Unload
46 DB2 news

DB2 Update

Published by
Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

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

Editor
Trevor Eddolls
E-mail: trevore@xephon.com

Publisher
Nicole Thomas
E-mail: nicole@xephon.com

Subscriptions and back-issues

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

DB2 Update on-line

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

Disclaimer

Contributions

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

Perl script to force users off a DB2 UDB LUW database

This article describes a Perl script that can be used to force off all users of a specified DB2 UDB LUW database.

Let's look at what you would have to do if you didn't use the script. I ran all the SQL in this article on a Windows XP machine running ActiveState ActivePerl 5.8 and DB2 UDB 8.1 FP2 using the db2admin userid.

Say we have a database called HMDB and we want to force off all users from it. The first thing we have to do is list out all the applications that are connected to the database. We do this by using the **list applications** command:

```
>db2 list applications

Auth Id  Application    Appl.    Application Id             DB       # of
         Name           Handle                              Name     Agents
-------- -------------- -------- -------------------------- -------- -----
DB2ADMIN db2bp.exe      12       *LOCAL.DB2.0065C2152237    HMDB     1
DB2ADMIN db2bp.exe      11       *LOCAL.DB2.008F42152212    SAMPLE   1
DB2ADMIN db2dasstm.exe  10       *LOCAL.DB2.00B282151956    TOOLSDB  1
DB2ADMIN db2dasstm.exe  9        *LOCAL.DB2.00B282151955    TOOLSDB  1
DB2ADMIN db2dasstm.exe  8        *LOCAL.DB2.00B282151950    TOOLSDB  1
```

You can see that we have three 'connected to' databases on our system – SAMPLE, HMDB, and TOOLSDB. To force off the user on the HMDB database we would use the **force application** command and supply the application handle (appl handle) as a parameter, which in the above example is 12, as shown below:

```
>db2 force application(12)
```

This is fine if we have only one user and our list of connections is small (as above), but what if we have many databases and connections? You could use the above method and try to write down or cut and paste all the application handle numbers, but a simpler approach would be to use the Perl script shown below.

This script can be run in two modes – reporting mode and action mode. As the names suggests, when run in reporting mode, the script just reports on the number of connections for each database. In action mode you could force off all users for a particular database. The script is called fap01.pl. There is a short help available with the script, which can be accessed by typing:

```
>perl fapØ1.pl -help
```

Let's first run it in reporting mode:

```
>perl fapØ1.pl

*****************************************************************************************
No database parameter passed.  Will therefore only list out the nos of
connections for each db.
*****************************************************************************************
*****************************************************************************************
>>> Number of databases is 3
Database     HMDB has this many connections    1
Database   SAMPLE has this many connections    1
Database  TOOLSDB has this many connections    3
*****************************************************************************************
>>> These are the application numbers for each database:
The list for HMDB is db2 force application (12)
The list for SAMPLE is db2 force application (11)
The list for TOOLSDB is db2 force application (1Ø,9,8)
*****************************************************************************************
```

We can see there are connections to three databases – the number of connections to each database is given, together with the application handle numbers for each connection.

Now let's run it in action mode and force all the applications for the HMDB database. You simply pass the database name as a parameter to the script, as shown below:

```
>perl fapØ1.pl hmdb

*****************************************************************************************
Will force all applications for database: HMDB
*****************************************************************************************
*****************************************************************************************
>>> Number of databases is 3
Database     HMDB has this many connections    1
Database   SAMPLE has this many connections    1
Database  TOOLSDB has this many connections    3
>>> Will now execute:
db2 force application (12)
DB20000I  The FORCE APPLICATION command completed successfully.
DB21024I  This command is asynchronous and may not be effective
immediately.
```

And let's check that the application has indeed been forced:

```
>db2 list applications

Auth Id  Application    Appl.      Application Id             DB    # of
         Name           Handle                                Name  Agents
-------- -------------- ------- --------------------------- ------ -----
DB2ADMIN db2bp.exe      11      *LOCAL.DB2.008F42152212      SAMPLE   1
DB2ADMIN db2dasstm.exe  10      *LOCAL.DB2.00B282151956      TOOLSDB  1
DB2ADMIN db2dasstm.exe  9       *LOCAL.DB2.00B282151955      TOOLSDB  1
DB2ADMIN db2dasstm.exe  8       *LOCAL.DB2.00B282151950      TOOLSDB  1
```

We can see that the database HMDB no longer has any connections to it.

The source for the script is shown below. I have tried to keep it fairly simple, so that you can modify it to suit your site's requirements. I just build a single string containing the force application command and all the application handles – if you have hundreds of connections, you may need to change this (I tried it with 101 connections, and it seemed to work fine).

I hope you find the script useful.

```
#!/usr/bin/perl -w
#
#####################################################################################$count
= @ARGV;
$dbp = uc $ARGV[0];
if ($dbp eq "") {
  $dbp = "XXXX";
}
$help = '-HELP';
if (index($help,$dbp,0) >= 0) {
#  system 'cls';
  print (('='x101),"\n");
  print (("="x45)," Help page ",("="x45),"\n");
  print (('='x101),"\n");
  print "This script can be run in 2 modes: reporting mode and action
mode.\n";
  print " \n";
  print "In reporting mode just enter the command without any parameters
```

```
>$0 \n";
  print "What you get back is a list of all databases that have
connections and the number of connections \n";
  print " \n";
  print "In action mode just enter >$0 <database-alias>\n";
  print "This will force off all applications connected to the
<database-alias> database\n";
  print (('='x101),"\n");
  exit 99;
}
print (('*'x101),"\n");
if ($count == 0) {
  print "No database parameter passed.  Will therefore only list out the
nos of connections for each db.\n";
  $path = 0;
}
else {
  print "Will force all applications for database: $dbp\n";
  $path = 1;
}
print (('*'x101),"\n");
#
open(FH1,"db2 list applications|");
#
$nos = 0;
$ndb = 0;
#
while(<FH1>) {
  $output=$_;
  chomp $output;
  $nos ++;
  $lin[$nos] = $output;
}
close FH1;
#
$lmax = $nos - 1;
for ($sc=5; $lmax >= $sc ; $sc +=1){
# print "lin $sc is $lin[$sc]\n";
  $slin = $lin[$sc];
  $x = 0;
  foreach $char (split ' ',$slin) {
    $x = $x + 1;
    if ($x == 3) {$appl = $char;}
    if ($x == 5) {$db   = $char;}
  }
#  print " $appl and $db\n";
#
# We have now got the application number and the database name.
#
# Add up the number of connections for each database.
```

```
#
  $ex = Ø;
  $jk = 1;
  while ($ndb >= $jk) {
    if ($dbnam[$jk] eq $db) {
      $dbnos[$jk] = $dbnos[$jk] + 1;
      $ex = 1;
      $na = $na + 1;
      $tota[$jk][$na] = $appl;
      $tota[$jk][2] = $tota[$jk][2] + 1;
    }
    $jk = $jk + 1;
  }
  if ($ex == Ø) {
    $ndb = $ndb + 1;
    $dbnam[$ndb] = $db;
    $dbnos[$ndb] = 1;
    $tota[$ndb][1] = $db;
    $tota[$ndb][2] = 1;
    $tota[$ndb][3] = $appl;
    $na = 3;
  }
}
#
# Now print out the results.
#
print (('*'x1Ø1),"\n");
print ">>> Number of databases is $ndb\n";
foreach $jk (1..$ndb) {
  printf ("Database %8s has this many connections
%4s\n",$dbnam[$jk],$dbnos[$jk]);
}
if ($path ==Ø) {
  print (('*'x1Ø1),"\n");
  print ">>> These are the application numbers for each database:\n";
}
#
$ihf = Ø;
foreach $xa (1..$ndb){
  $str = "db2 force application (";
  foreach $xb (3..(2+$tota[$xa][2])){
    if ($xb == (2+$tota[$xa][2])) {
      $str = $str . $tota[$xa][$xb] . ")";
    }
    else {
      $str = $str . $tota[$xa][$xb] . ",";
    }
  }
  if ($path == Ø) {
    print "The list for $tota[$xa][1] is $str\n";
  }
  else {
    if ($tota[$xa][1] eq $dbp ) {
      print ">>> Will now execute:\n";
      print "$str\n";
      system"$str";
      $ihf = 1;
    }
  }
}
if (($ihf == 0) & ($dbp ne "XXXX")) {
  print ">>> The database alias entered: $dbp does not have any
connections\n";
}
if ($path ==0) {
  print (('*'x101),"\n");
}
```

C Leonard
Freelance Consultant (UK)

SQL scalar functions – part 2

This month we conclude the code for an SQL scalar function.

WORD

```
* PROCESS SYSTEM(MVS);
 WORD: PROC(UDF_PARM1, UDF_PARM2, UDF_RESULT,
               UDF_IND1,  UDF_INDR,
               UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
               UDF_DIAG_MSG, UDF_SCRATCHPAD,
               UDF_CALL_TYPE, UDF_DBINFO)
          OPTIONS(FETCHABLE NOEXECOPS REENTRANT);
  /******************************************************************/
 /*    UDF    : WORD                                                */
 /*    INPUT : UDF_PARM1    CHAR     INPUT STRING                   */
 /*    INPUT : UDF_PARM2    INTEGER  NTH WORD                       */
 /*    OUTPUT: UDF_RESULT   CHAR     NTH BLANK-DELIMITED WORD       */
  /******************************************************************/
  DCL UDF_PARM1      CHAR(2000) VAR;    /* INPUT PARAMETER          */
  DCL UDF_PARM2      BIN FIXED(31);     /* INPUT PARAMETER          */
  DCL UDF_RESULT     CHAR(2000) VAR;    /* RESULT PARAMETER         */
  DCL UDF_IND1       BIN FIXED(15);     /* INDICATOR FOR INPUT PARM  */
  DCL UDF_IND2       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);
  %INCLUDE UDFINFO;                     /* DBINFO                    */
  DCL (LENGTH,SUBSTR)     BUILTIN;
  DCL (I,IS,J,J1,W,ST,EN) BIN FIXED(31);
  UDF_RESULT='';
  ST,EN=Ø;
  W=Ø; IS=1;
  DO I=IS TO LENGTH(UDF_PARM1) WHILE(W < UDF_PARM2);
     IF SUBSTR(UDF_PARM1,I,1)¬=' '
     THEN DO;
       W=W+1;
       IF UDF_PARM2=W THEN ST=I;
       J1=I;
       DO J=J1 TO LENGTH(UDF_PARM1) WHILE(SUBSTR(UDF_PARM1,J,1)¬=' ');
       END;
       IS,I=J;
     END;
  END;
  EN=J;
  IF W < UDF_PARM2
  THEN UDF_RESULT=' ';
  ELSE UDF_RESULT=SUBSTR(UDF_PARM1,ST,EN-ST);
  IF UDF_PARM1=' ' THEN UDF_RESULT=' ';
 END WORD;
```

WORDINDX

```
* PROCESS SYSTEM(MVS);
 WORDIN: PROC(UDF_PARM1, UDF_PARM2, UDF_RESULT,
              UDF_IND1,  UDF_INDR,
              UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
              UDF_DIAG_MSG, UDF_SCRATCHPAD,
              UDF_CALL_TYPE, UDF_DBINFO)
         OPTIONS(FETCHABLE NOEXECOPS REENTRANT);
 /*****************************************************************/
 /*    UDF    : WORDINDEX                                          */
 /*    INPUT : UDF_PARM1    CHAR     INPUT STRING                  */
 /*    INPUT : UDF_PARM2    INTEGER  NTH WORD                      */
 /*    OUTPUT: UDF_RESULT   INTEGER  POS BLANK-DELIMITED WORD      */
 /*****************************************************************/
 DCL UDF_PARM1      CHAR(2ØØØ) VAR;    /* INPUT PARAMETER           */
 DCL UDF_PARM2      BIN FIXED(31);     /* INPUT PARAMETER           */
 DCL UDF_RESULT     BIN FIXED(31);     /* RESULT PARAMETER          */
 DCL SSTR           CHAR(2ØØØ) VAR;    /* SEARCH STRING             */
 DCL NWORDS          BIN FIXED(31);     /* NO. OF WORDS                 */
 DCL UDF_IND1        BIN FIXED(15);     /* INDICATOR FOR INPUT PARM     */
 DCL UDF_IND2        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(100);
 EXEC SQL INCLUDE SQLCA;
 %INCLUDE UDFINFO;                      /* DBINFO                       */
 DCL (LENGTH,SUBSTR,ADDR,NULL)     BUILTIN;
 EXEC SQL SET :NWORDS=SYSADM.WORDS(:UDF_PARM1);
 IF UDF_PARM2 > NWORDS | UDF_PARM2 < 1
 THEN UDF_RESULT=0;
 ELSE DO;
    EXEC SQL SET :SSTR = SYSADM.WORD(:UDF_PARM1,:UDF_PARM2);
    EXEC SQL SET :UDF_RESULT = POSSTR(:UDF_PARM1,:SSTR);
    IF SQLCODE¬=0 THEN UDF_RESULT=0;
 END;
END WORDIN;
```

WORDS

```
* PROCESS SYSTEM(MVS);
 WORDS: 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(FETCHABLE NOEXECOPS REENTRANT);
 /******************************************************************/
 /*    UDF   : WORDS                                               */
 /*    INPUT : UDF_PARM1    CHAR     INPUT STRING                  */
 /*    OUTPUT: UDF_RESULT   INTEGER  NUMBER OF BLANK-DELIMITED WORDS */
 /******************************************************************/
 DCL UDF_PARM1       CHAR(2000) VAR;    /* INPUT PARAMETER              */
 DCL UDF_RESULT      BIN FIXED(31);     /* 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(100);
 %INCLUDE UDFINFO;                      /* DBINFO                       */
 DCL (LENGTH,SUBSTR)   BUILTIN;
 DCL (I,D)             BIN FIXED(31);
 UDF_RESULT=1;
 DO I=1 TO LENGTH(UDF_PARM1);
    D=0;
    IF SUBSTR(UDF_PARM1,I,1)=' '
    THEN DO;
```

```
      D=1;
      IF I=1 THEN UDF_RESULT=1;
      ELSE DO;
        IF SUBSTR(UDF_PARM1,I-1,1)¬=' '
        THEN UDF_RESULT=UDF_RESULT+1;
        D=1;
      END;
    END;
  END;
  IF D=1 THEN UDF_RESULT=UDF_RESULT-1;
  IF UDF_PARM1=' ' THEN UDF_RESULT=Ø;
 END WORDS;
```

UDBINFO – include udbinfo declaration from SYSLIB

```
  DCL UDF_SQLSTATE CHAR(5);              /* SQLSTATE RETURNED TO DB2 */
  DCL UDF_NAME CHAR(137) VARYING;        /* QUALIFIED FUNCTION NAME */
  DCL UDF_SPEC_NAME CHAR(128) VARYING;   /* SPECIFIC FUNCTION NAME */
  DCL UDF_DIAG_MSG CHAR(7Ø) VARYING;     /* DIAGNOSTIC STRING */
  DCL UDF_CALL_TYPE BIN FIXED(31);       /* CALL TYPE */
  DCL DBINFO PTR;
  /* CONSTANTS FOR DB2_ENCODING_SCHEME */
  DCL SQLUDF_ASCII BIN FIXED(15) INIT(1);
  DCL SQLUDF_EBCDIC BIN FIXED(15) INIT(2);
  DCL SQLUDF_MIXED BIN FIXED(15) INIT(3);
  DCL Ø1 UDF_DBINFO BASED(DBINFO),      /* DBINFO */
      Ø3 UDF_DBINFO_LLEN BIN FIXED(15), /* LOCATION LENGTH */
      Ø3 UDF_DBINFO_LOC CHAR(128),      /* LOCATION NAME */
      Ø3 UDF_DBINFO_ALEN BIN FIXED(15), /* AUTH ID LENGTH */
      Ø3 UDF_DBINFO_AUTH CHAR(128),     /* AUTHORIZATION ID */
      Ø3 UDF_DBINFO_CDPG,               /* CCSIDS FOR DB2 FOR OS/39Ø*/
        Ø5 DB2_CCSIDS(3),
         Ø7 R1 BIN FIXED(15),          /* RESERVED */
         Ø7 DB2_SBCS BIN FIXED(15),    /* SBCS CCSID */
         Ø7 R2 BIN FIXED(15),          /* RESERVED */
         Ø7 DB2_DBCS BIN FIXED(15),    /* DBCS CCSID */
         Ø7 R3 BIN FIXED(15),          /* RESERVED */
         Ø7 DB2_MIXED BIN FIXED(15),   /* MIXED CCSID */
       Ø5 DB2_ENCODING_SCHEME BIN FIXED(31),
       Ø5 DB2_CCSID_RESERVED CHAR(8),
     Ø3 UDF_DBINFO_SLEN BIN FIXED(15),     /* SCHEMA LENGTH */
     Ø3 UDF_DBINFO_SCHEMA CHAR(128),       /* SCHEMA NAME */
     Ø3 UDF_DBINFO_TLEN BIN FIXED(15),     /* TABLE LENGTH */
     Ø3 UDF_DBINFO_TABLE CHAR(128),        /* TABLE NAME */
     Ø3 UDF_DBINFO_CLEN BIN FIXED(15),     /* COLUMN LENGTH */
     Ø3 UDF_DBINFO_COLUMN CHAR(128),       /* COLUMN NAME */
     Ø3 UDF_DBINFO_RELVER CHAR(8),         /* DB2 RELEASE LEVEL */
     Ø3 UDF_DBINFO_PLATFORM BIN FIXED(31), /* DATABASE PLATFORM*/
     Ø3 UDF_DBINFO_NUMTFCOL BIN FIXED(15), /* # OF TF COLS USED*/
```

```
    Ø3 UDF_DBINFO_RESERV1 CHAR(24),        /* RESERVED */
    Ø3 UDF_DBINFO_TFCOLUMN PTR,            /* -> TABLE FUN COL LIST*/
    Ø3 UDF_DBINFO_APPLID PTR,              /* -> APPLICATION ID */
    Ø3 UDF_DBINFO_RESERV2 CHAR(2Ø);        /* RESERVED */
```

Bernard Zver (bernard.zver@informatika.si)
DBA
Informatica (Slovenia)

A strategy for image copying large partitioned tablespaces using Real-Time Statistics

This article looks at using the information from the Real-Time Statistics tables to determine the frequency of image copies for large partitioned tablespaces, although the approach works equally well for any tablespace. No REXX is required, just SQL.

THE REQUIREMENTS

Most strategies for image-copying tablespaces are period-based – copy everything every day, perhaps, or do full image copies once a week and incremental copies every day. This is an approach that works well until constraints such as batch window time and sheer amount of data backed up come into play. Then, a more tailored approach is required, backing up frequently-updated objects more than the others. This approach requires either a large amount of administration or an intelligent automated process, which this article attempts to provide.

LARGE TABLESPACES

The time and space constraints start to bite as the DB2 objects get bigger and more numerous.

We have a large number of 32KB tablespaces, which have been partitioned into 250 parts – with the partitions varying

Partition	*Inserts/updates*
97	4
98	58
99	10,022
100	0
101	43
102	0

Figure 1: The profile of a few partitions

from several thousand tracks to several gigabytes, and growing all the time. Rebalancing takes place periodically. In this particular application, one partition in each tablespace on a particular day is active, taking all inserts and most updates. Other partitions may have some updates, but these will be fairly low. The profile of a few partitions on a given day might look like Figure 1.

Looking at these partitions, we definitely want to do a full image copy of part 99. We may want to copy parts 97, 98, and 101. If parts 100 and 102 were copied yesterday, copying them again is a waste of time, because the copies will be exactly the same as yesterday's.

PREVIOUS FACILITIES: CHANGELIMIT

Version 5 of DB2 made some attempt at addressing the problem. The CHANGELIMIT keyword of the COPY utility decides whether to take a full copy, an incremental copy, or no copy at all, depending on its input parameters and the percentage of pages in the tablespace that have changed.

For example, the following statement:

```
    COPY TABLESPACE dbname.tsname CHANGELIMIT(15,35)
```

will cause a full image copy to be taken if 35% or more of the pages in the tablespace have changed, an incremental copy if 15% or more have changed, and no copy at all if less than 15% have changed.

Unfortunately the implementation of CHANGELIMIT is flawed to say the least. If no copy is taken, the image copy dataset has still been allocated in the JCL. After a few such utility runs, all of your valid copies will have rolled off the end of your GDG (and this is the same in Version 7 even when the dataset is dynamically allocated with TEMPLATE – a major disappointment). Similarly, the utility could decide to take an incremental copy every day until your last full copy disappears. CHANGELIMIT is stupid.

To attempt to get round these limitations, COPY CHANGELIMIT can be run with the REPORTONLY option – no copy is done, but the return code from the job step shows whether a full, incremental, or no copy would be taken. The return code can be used to avoid a later COPY step. This requires a read through the tablespace even if no copy is to be taken. For our large tablespaces, this just takes too much time.

REAL-TIME STATISTICS

The Real-Time Statistics (RTS) facility was introduced some way into DB2 Version 7. You can find some information in later versions of the *DB2 Administration Guide* (Appendix G in my copy). Also see Craig Mullins' article in the June 2004 edition of *DB2 Update (Using Real-Time Statistics)*.

In brief, the statistics are held in two tables, SYSIBM.TABLESPACESTATS and SYSIBM.INDEXSPACESTATS. A row in TABLESPACESTATS corresponds to a tablespace or tablespace partition (similar to SYSIBM.SYSTABLEPART). Most of the statistical columns relate to the inserts, updates, or deletes since the last REORG, LOAD, RUNSTATS, or COPY. For the purpose of this article, we're interested in the COPY statistics. They are:

- COPYLASTTIME – timestamp of last copy.
- COPYUPDATEDPAGES – number of distinct pages changed since the last COPY.

- COPYCHANGES – number of INSERT, UPDATE, and DELETE operations since the last COPY.
- COPYUPDATELRSN – RBA or LRSN of the first update after the last COPY.
- COPYUPDATETIME – timestamp of the first update after the last COPY.

There is also a NACTIVE column, which holds the number of active pages, and TOTALROWS, which holds the total number of rows. These can be used in conjunction with COPYUPDATEDPAGES and COPYCHANGES to give a percentage view of the amount of change that has taken place.

When RTS is turned on for a subsystem, rows appear for objects as they are used. At this point, the NACTIVE and EXTENTS (giving number of extents for the object) columns are populated immediately, while most of the other columns are NULL. TOTALROWS is not populated until a REORG or LOAD is done (even RUNSTATS does not populate it). The COPY-related columns above are populated after a COPY utility is run. Taking all this together, if you want to come up with an idea of the percentage of changes since the last COPY, you should use COPYUPDATEDPAGES / NACTIVE * 100 rather than COPYCHANGES / TOTALROWS * 100 because you can’t rely on TOTALROWS being non-null.

Once a column is populated, it is kept up-to-date continuously by DB2 (the values are externalized to the tables periodically).

LISTDEF AND TEMPLATE

The final components that we need for our automated image copy process are LISTDEF and TEMPLATE, introduced in DB2 Version 7.

In previous versions of DB2, the step to copy several partitions of a tablespace might look like this:

```
//UTIL    EXEC DSNUPROC,
//             SYSTEM=DB2T
//SYSIN    DD  *
   COPY TABLESPACE DDBIVP.SDBACCT DSNUM 3 COPYDDN(COPØØØ3)
        TABLESPACE DDBIVP.SDBACCT DSNUM 5 COPYDDN(COPØØØ5)
        TABLESPACE DDBIVP.SDBACCT DSNUM 7 COPYDDN(COPØØØ7)
//COPØØØ3  DD  DSN=NUDBS.DB2T.DDBIVP.SDBACCT.FØØØ3(+1),
//             DISP=(NEW,CATLG),
//             SPACE=(TRK,(1ØØ,1Ø),RLSE)
//COPØØØ5  DD  DSN=NUDBS.DB2T.DDBIVP.SDBACCT.FØØØ5(+1),
//             DISP=(NEW,CATLG),
//             SPACE=(TRK,(8Ø,8),RLSE)
//COPØØØ7  DD  DSN=NUDBS.DB2T.DDBIVP.SDBACCT.FØØØ7(+1),
//             DISP=(NEW,CATLG),
//             SPACE=(TRK,(2ØØ,2Ø),RLSE)
```

Using TEMPLATE and LISTDEF, it looks like this:

```
//UTIL    EXEC DSNUPROC,
//             SYSTEM=DB2T
//SYSTEMPL DD  *
  TEMPLATE TMPL1 DSN('NUDBS.&SS..&DB..&TS..F&PA(2,4).(+1)')
    GDGLIMIT(9)
//SYSLISTD DD  *
   LISTDEF LISTØ1
       INCLUDE TABLESPACES TABLESPACE DDBIVP.SDBACCT PARTLEVEL 3
       INCLUDE TABLESPACES TABLESPACE DDBIVP.SDBACCT PARTLEVEL 5
       INCLUDE TABLESPACES TABLESPACE DDBIVP.SDBACCT PARTLEVEL 7
//SYSIN    DD  *
   COPY LIST LISTØ1
     COPYDDN(TMPL1) SHRLEVEL CHANGE
```

The major improvement here is that only the SYSLISTD SYSIN needs to change from day to day as the partitions copied and their sizes change – and SYSLISTD can be put into a dataset. The JCL is static. This means that the same job can be run every day with different inputs. Previously, the JCL itself would have had to be regenerated every day because of the DD statements for the image copy datasets. The other improvement is that we don’t have to work out the sizes of the image copy datasets – TEMPLATE does it for us. So under Version 7, all we need to create for the COPY utility is a list of partitions to be copied.

OUR IMAGE COPY STRATEGY

For each set of objects to be copied, we will have a pair of jobs

– a generating job and a copy job – the latter running the copy utility based on the output from the former. The generating job may cover a single database or a number of databases, depending on the number of objects contained in the databases.

We need to decide what level of changes should trigger a COPY – an absolute number of changes or percentage change, for example – also, whether we should always take full copies, or take incremental copies when the amount of change is small.

Here, we're taking a simple approach – if there have been any changes since the last image copy, take a full image copy. From the RTS point of view, this means take a copy if the COPYCHANGES value is greater than zero.

ACCURACY OF REAL-TIME STATISTICS

There is a possibility of statistics not being entirely accurate – in particular, the possibility of some updates not being recorded. This can happen because the statistics are only externalized to the TABLESPACESTATS and INDEXSPACESTATS tables periodically – by default, every 30 minutes. If DB2 abends, any statistics in memory that have not yet been written will be lost.

To avoid any risk caused by inaccurate statistics, the image copy strategy requires a catch-all condition – that a partition be copied at least every 30 days. Archive logs are kept for 35 days, so if some updates have not been recorded in the statistics, and a subsequent image copy has not been taken, the log records still exist for recovery purposes and there is no risk of loss of data. The only risk is of a slightly extended recovery.

THE GENERATING JOB

The generating job will produce a list of INCLUDE statements as input to the COPY job, which will actually run the COPY utility. Each statement will look something like this:

```
    INCLUDE TABLESPACES TABLESPACE dbname.tsname PARTLEVEL partno
```

The generating job will take as input the rows from SYSIBM.TABLESPACESTATS. All it's interested in is whether the partition has been updated at all. This SQL will do it for database DIGOAM97:

```
SELECT DBNAME,NAME,PARTITION
    FROM SYSIBM.TABLESPACESTATS
 WHERE DBNAME = 'DIGOAM97'
   AND COPYCHANGES > Ø
  WITH UR
;
--------+--------+--------+--------+--------+---------
DBNAME    NAME      PARTITION
--------+--------+--------+--------+--------+---------
DIGOAM97  SIGOSMD           Ø
DIGOAM97  SIGOSMØ4          Ø
DIGOAM97  SIGOSM32         57
DSNE61ØI NUMBER OF ROWS DISPLAYED IS 3
```

We also need the 'INCLUDE…' and other literals in the output, and want to concatenate the database name and tablespace name in the query:

```
SELECT 'INCLUDE TABLESPACES TABLESPACE',
    STRIP(DBNAME)||'.'||STRIP(NAME),
    'PARTLEVEL',PARTITION
    FROM SYSIBM.TABLESPACESTATS
 WHERE DBNAME = 'DIGOAM97'
   AND COPYCHANGES > Ø
  WITH UR
;
--------+--------+--------+--------+--------+--------+--------+--
                                                              PARTITION
--------+--------+--------+--------+--------+--------+--------+--
INCLUDE TABLESPACES TABLESPACE  DIGOAM97.SIGOSMD   PARTLEVEL         Ø
INCLUDE TABLESPACES TABLESPACE  DIGOAM97.SIGOSMØ4  PARTLEVEL         Ø
INCLUDE TABLESPACES TABLESPACE  DIGOAM97.SIGOSM32  PARTLEVEL        57
DSNE61ØI NUMBER OF ROWS DISPLAYED IS 3
```

You can see a slight problem – the non-partitioned tablespaces will have 'PARTLEVEL 0' in the include statement, which we don't want. A CASE statement will improve things:

```
SELECT 'INCLUDE TABLESPACES TABLESPACE',
    STRIP(DBNAME)||'.'||STRIP(NAME),
    CASE
```

```
       WHEN PARTITION = Ø THEN
         ''
       ELSE
         'PARTLEVEL '|| CHAR(PARTITION)
     END
     FROM SYSIBM.TABLESPACESTATS
  WHERE DBNAME = 'DIGOAM97'
    AND COPYCHANGES > Ø
   WITH UR
;
--------+--------+--------+--------+--------+--------+----

--------+--------+--------+--------+--------+--------+----
INCLUDE TABLESPACES TABLESPACE  DIGOAM97.SIGOSMD
INCLUDE TABLESPACES TABLESPACE  DIGOAM97.SIGOSMØ4
INCLUDE TABLESPACES TABLESPACE  DIGOAM97.SIGOSM32  PARTLEVEL 57
```

Note that I had to convert the PARTITION column to a CHAR to concatenate it with ‘PARTLEVEL’.

As you can see, if you use SPUFI or DSNTEP2 to produce the query, you get headings and column delimiters. We can get around this by using DSNTIAUL, but we end up with a lot of non-printable characters in the output:

```
******************************** Top of Data****************************
..INCLUDE TABLESPACES
TABLESPACE..DIGOAM97.SIGOSMD......................
..INCLUDE TABLESPACES
TABLESPACE..DIGOAM97.SIGOSMØ4.....................
..INCLUDE TABLESPACES TABLESPACE..DIGOAM97.SIGOSM32..PARTLEVEL
57........
******************************* Bottom of Data**************************
```

Turning on hex display shows that they are either the length bytes of VARCHARs, or trailing zero bytes:

```
******************************** Top of Data ***************************


  --------------------------------------------------------------------------
..INCLUDE TABLESPACES
TABLESPACE..DIGOAM97.SIGOSMD......................
Ø1CDCDECC4ECCDCEDCCCE4ECCDCEDCCCØ1CCCDCDFF4ECCDEDCØØØØØØØØØØØØØØØØØØØØØØØ
ØE95334450312352713520312352713500497614 97B2976244ØØØØØØØØØØØØØØØØØØØØØØØ
  --------------------------------------------------------------------------
..INCLUDE TABLESPACES
TABLESPACE..DIGOAM97.SIGOSMØ4.....................
```

```
Ø1CDCDECC4ECCDCEDCCCE4ECCDCEDCCCØ1CCCDCDFF4ECCDEDFFØØØØØØØØØØØØØØØØØØØØØØ
ØE9533445Ø31235271352Ø3123527135Ø149761497B297624Ø4ØØØØØØØØØØØØØØØØØØØØØØ
  -----------------------------------------------------------------------
..INCLUDE TABLESPACES TABLESPACE..DIGOAM97.SIGOSM32..PARTLEVEL 57
....
Ø1CDCDECC4ECCDCEDCCCE4ECCDCEDCCCØ1CCCDCDFF4ECCDEDFFØ1DCDEDCECD4FF4444ØØØØ
ØE9533445Ø31235271352Ø3123527135Ø149761497B29762432ØØ719335553Ø57ØØØØØØØ
  -----------------------------------------------------------------------
****************************** Bottom of Data ************************
```

This can be solved by concatenating all the fields together, and applying the CHAR function to the whole lot:

```
SELECT CHAR('INCLUDE TABLESPACES TABLESPACE '||
    STRIP(DBNAME)||'.'||STRIP(NAME)||
    CASE
      WHEN PARTITION = Ø THEN
        ''
      ELSE
        ' PARTLEVEL '|| CHAR(PARTITION)
    END,8Ø)
    FROM SYSIBM.TABLESPACESTATS
 WHERE DBNAME = 'DIGOAM97'
   AND COPYCHANGES > Ø
  WITH UR
;
```

The second parameter to the CHAR function, 80, expands each row to 80 columns.

The simplicity of our query – just checking for a non-zero COPYCHANGES column – means that there are a number of objects that we might miss (and therefore will not get copied):

- Partitions that have a zero entry in COPYCHANGES, but haven't been copied for over 30 days.
- Partitions that have a null entry in COPYCHANGES because the partition was last copied before RTS was turned on.
- Partitions that have no entry in SYSIBM.TABLESPACESTATS because they never get used.

For the first case, we can easily list objects that were copied over 30 days ago:

```
SELECT DBNAME,NAME,PARTITION
    FROM SYSIBM.TABLESPACESTATS
  WHERE COPYLASTTIME < CURRENT TIMESTAMP - 30 DAYS
;
```

We can also test for objects that haven't been copied since RTS was turned on:

```
SELECT DBNAME,NAME,PARTITION
    FROM SYSIBM.TABLESPACESTATS
  WHERE COPYLASTTIME < CURRENT TIMESTAMP - 30 DAYS
     OR COPYCHANGES IS NULL
;
```

For the third case, objects that haven't been used won't have an entry in the RTS table. We can find them by listing objects that exist in catalog table SYSIBM.SYSTABLEPART, but not in SYSIBM.SYSTABLESPACESTATS. An efficient way to test for this is with an outer join:

```
SELECT B.DBNAME,B.NAME,B.PARTITION
    FROM SYSIBM.SYSTABLEPART A
    LEFT JOIN SYSIBM.TABLESPACESTATS B
       ON A.DBNAME = B.DBNAME
      AND A.TSNAME = B.NAME
      AND A.PARTITION = B.PARTITION
 WHERE COPYLASTTIME < CURRENT TIMESTAMP - 30 DAYS
  OR COPYCHANGES IS NULL
;
```

If there is no row in SYSTABLESPACESTATS, COPYLASTTIME is returned as NULL. It's also set to NULL if there is a row but the object has never been copied. This means that by testing COPYLASTTIME for NULL, we're testing the second and third conditions in one go – we're not bothered which condition is satisfied, either will do.

It's probably best to check for these 'missed' objects in a separate sweep job.

THE COPY JOB

The copy job consists of a utility step that accepts the INCLUDE statements from the generating job. This is pretty simple:

```
//UTIL    EXEC DSNUPROC,
//             SYSTEM=DIP1
//SYSTEMPL DD  *
  TEMPLATE TMPL1 DSN('NUDBS.&SS..&DB..&TS..F&PA(2,4).(+1)')
    GDGLIMIT(9)
//SYSLISTD DD  *
   LISTDEF LISTØ1
//         DD  DSN=SMITHAC.COPY.INC,DISP=SHR
//SYSIN    DD  *
   COPY LIST LISTØ1
     COPYDDN(TMPL1) SHRLEVEL CHANGE
```

The INCLUDE statements are in the dataset that is concatenated to the SYSLISTD DD statement.

But we've forgotten something. What if no objects qualify? Using a LISTDEF with no INCLUDES causes an error:

```
  DSNUGUTC -  COPY LIST LISTØ1 COPYDDN(TMPL1) SHRLEVEL CHANGE
-DIP1 DSNUILSA - LISTDEF LISTØ1 CONTAINS NO OBJECTS
  DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

The best way around this is to add a job step that tests for an empty file and then avoid the COPY step if the file is empty. This can be done easily with the IDCAMS REPRO command:

```
//STEST   EXEC PGM=IDCAMS
//INF      DD  DISP=SHR,DSN=SMITHAC.COPY.INC
//OUT      DD  DUMMY,DCB=SMITHAC.COPY.INC
//SYSIN    DD  *
 REPRO INFILE(INF),OUTFILE(OUT) COUNT(1)
//SYSPRINT DD  SYSOUT=*
```

REPRO returns 4 if there is fewer than one line in the file (ie it is empty). The utility step then needs a COND so that it won't run if STEST finishes with RC 4:

```
//UTIL    EXEC DSNUPROC,
//             SYSTEM=DIP1,
//             COND=((4,LT),(4,EQ,STEST))
```

SUMMARY

In this article, we've looked at using information from the Real-Time Statistics tables to schedule image copies for tablespaces and partitions depending on whether they've been updated.

Coupled with the new LISTDEF and TEMPLATE facilities in DB2 V7, we can create jobs with just a bit of SQL – no REXX tailoring is required.

This is just the start – soon I'll be looking at scheduling RUNSTATS and REORG utilities in the same way.

Alan Smith
Norwich Union (UK)

Modify column attributes

Some changes to a table cannot be made with an ALTER TABLE statement (changes of CHAR(50) to CHAR(15) or SMALLINT to INTEGER, or changing a column defined with NOT NULL to allow null values etc). To make such changes, you need to perform the following steps:

- Unload the table
- Drop the table
- Commit the changes
- Re-create the table
- Reload the table.

The unload and reload steps are not involved in this article (routines including these steps were published in the September 2003 issue of *DB2 Update* in the article 'Transfer data utility'). This article processes all the other steps, including the definitions of all related objects as needed.

The drop table statement deletes a table. The statement deletes the rows in the catalog tables containing information about deleted tables, and it also drops any other objects that depend on the deleted table. As a result:

- The column names of the table are dropped from SYSIBM.SYSCOLUMNS.
- If the dropped table has an identity column, all information regarding the identity column is removed from SYSIBM.SYSSEQUENCES.
- If triggers are defined on the table, they are dropped, and the corresponding rows are removed from SYSIBM.SYSTRIGGERS and SYSIBM.SYSPACKAGES.
- Any views based on the table are dropped.
- Application plans or packages that involve the use of the table are invalidated.
- Synonyms for the table are dropped from SYSIBM.SYSSYNONYMS.
- Indexes created on any columns of the table are dropped.
- Referential constraints that involve the table are dropped.
- Authorization information that is kept in the DB2 catalog authorization tables is updated to reflect the dropping of the table.
- Access path statistics and space statistics for the table are deleted from the catalog.
- If the table contains a LOB column, the auxiliary table and the index on the auxiliary table are dropped. The LOB table space is dropped if it was created with SQLRULES(STD).
- If a table has a partitioning index, you must drop the table space or use LOAD REPLACE when loading the redefined table.

Taking into account all of these remarks, this article generates the job for the re-creation of all necessary objects.

Figure 1 shows all catalog tables that this article consults to drop and re-create objects.

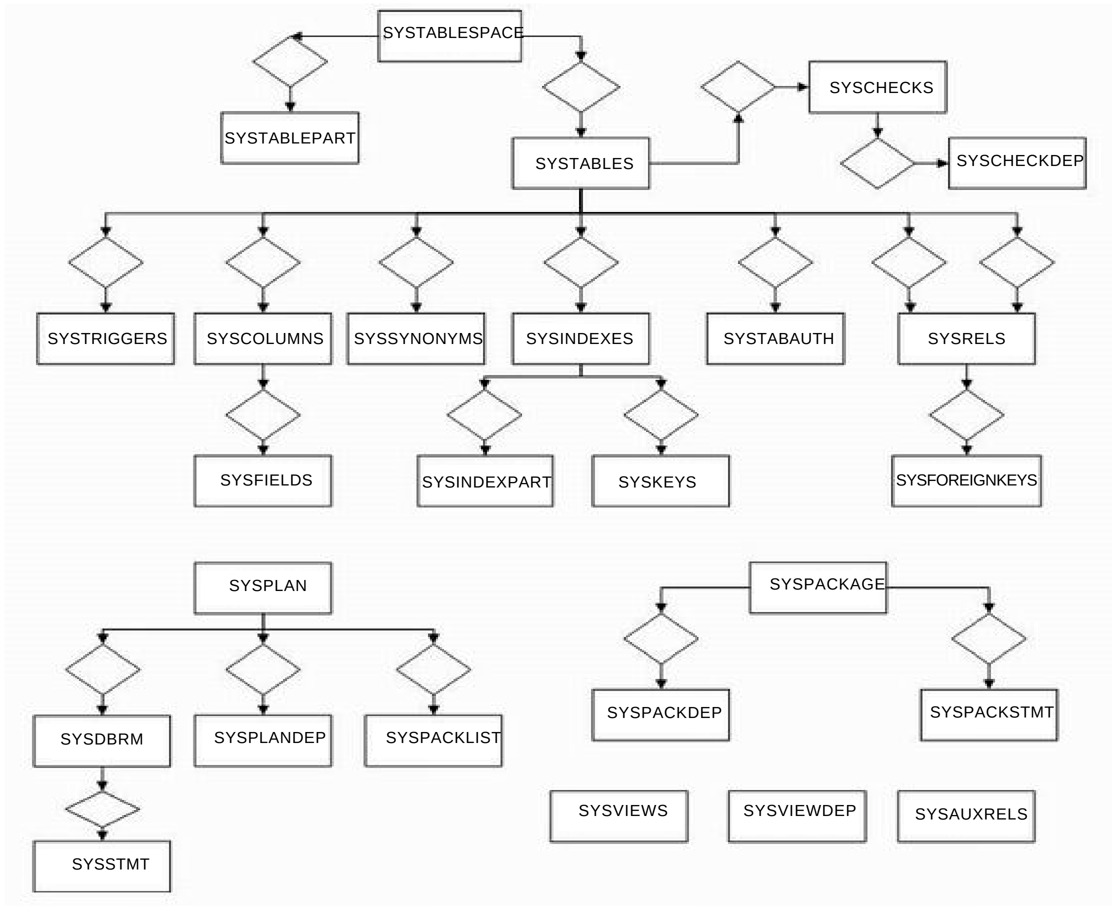

Figure 1: Catalog tables involved in the program

In this article I use a second REXX EXEC (SQLISPF, which was published in the July and August 2001 issues of *DB2 Update* in ‘Simplifying occasional, regular, and periodic tasks of the DBA’).

Note:

- A change to certain attributes may mean that the old relations are impossible to sustain unless relevant attributes in other tables are changed. These relations are placed under the comments for any subsequent use, whereas a change to the pertinent attributes may be accomplished by calling the very same program, in the process of which these tables are being used as an input parameter.

- If the job needs to prepare statements for generating a trigger, before executing the create trigger statement, the program includes the --#SET TERMINATOR # control statement to change the SQL terminator to the character #.
- A template for generating create view statements includes recursion. Recursion is applied in order to find all the views dependent on the relevant table.
- Instructions for starting the application. After changing any data on the first map you must type M in the command line, and the job will be generated.

MODCOLR0

```
/* rexx */
address ispexec 'select panel(modcolp1)'
```

MODCOLP1

```
)ATTR
%  TYPE(TEXT)
[  TYPE(TEXT) INTENS(LOW)
<  TYPE(INPUT) CAPS(ON)
+  TYPE(TEXT) INTENS(LOW)
!  TYPE(OUTPUT) INTENS(LOW) CAPS(OFF)
)BODY DEFAULT(]*;)EXPAND($$)
%-$-$- MODIFY COLUMN(s) -$-$-[

                                                                        [
+DB2 subsystem:    ===><Z    [
[

+Table creator:    ===><Z        [
[

+Table name:       ===><Z                  [
<Z[

)INIT
  .ZVARS = '(DSN8SSID Tbcre Tbname ZCMD)'
  .CURSOR = DSN8SSID
  VGET (DSN8SSID Tbcre Tbname) SHARED
  IF (&DSN8SSID = &Z)
    &DSN8SSID = DSN
)PROC
  VER  (&DSN8SSID,NB,LIST,DSN,DBT)
  VER  (&Tbcre,NB)
  VER  (&Tbname,NB)
  VPUT (DSN8SSID Tbcre Tbname) SHARED
  &ZSEL = TRANS(TRUNC(&ZCMD,'.')
               ' ','CMD(MODCOLR1)'
               '*','?')
)END
```

MODCOLP2

```
)ATTR
%  TYPE(TEXT)
[  TYPE(TEXT) INTENS(LOW)
+  TYPE(TEXT) INTENS(LOW)
*  TYPE(INPUT) CAPS(ON) COLOR(WHITE)
<  TYPE(INPUT) CAPS(OFF) COLOR(WHITE)
]  TYPE(OUTPUT) INTENS(LOW) CAPS(OFF)
)BODY DEFAULT(/,_)EXPAND($$)
%-$-$- MODIFY COLUMN(s) -$-$-[

%Command ===><Z[ (M - Job for Modify Table)                    %Scroll
===><Z   [

+DB2 subsystem ==>]Z   [
+Creator:         ]Z       [
+Table:           ]Z                 [

                  +-------------------------------------------------[
                  +COL NAME            COL TYPE  LENGTH  SCALE  NULLS
                  +-------------------------------------------------[
)MODEL
                  <Z                 +*Z       +<Z    + <Z    +  <Z+
)INIT
   .ZVARS = '(ZCMD ZSCR DSN8SSID Tbcre Tbname +
              COLNAME COLTYPE COLLEN COLSCALE COLNULLS)'
)END
```

MODCOLR1

```
/* REXX - MODCOLR1 *************************************************/
/* TITLE    :  MODIFY COLUMN(s)                                     */
/* ***************************************************************** */
/* TRACE I                                                          */
  Address ISPEXEC 'CONTROL ERRORS RETURN '
  Address ISPEXEC 'VGET (DSN8SSID Tbcre Tbname) SHARED'
  DB2V = DSN8SSID
  panel = 'MODCOLP2'
  tbnam = 'MODCOLT1'
  tvars = 'COLNAME COLTYPE COLLEN COLSCALE COLNULLS INDMODT OLDCOLN'||,
          ' KEYSEQ'
  msg  = ' '
  indmod  = Ø
  csrrow = 1
  cursor  = 'COLNAME'
  Address ISPEXEC 'TBERASE 'tbnam
  Address ISPEXEC 'TBOPEN 'tbnam' WRITE SHARE '
  if rc > Ø then do
    Address ISPEXEC 'TBCREATE 'tbnam' NAMES('tvars') NOWRITE SHARE '
    If rc ¬= Ø Then say rc
    SQLQUERY = "SELECT NAME, COLTYPE, ",
                     "CASE WHEN LENGTH2 > Ø THEN LENGTH2 ",
                                           "ELSE LENGTH ",
                     "END AS LENGTH, ",
                     "SCALE, NULLS, COLNO, KEYSEQ ",
               "FROM SYSIBM.SYSCOLUMNS ",
               "WHERE TBCREATOR = '" || Tbcre || "' AND ",
                     "TBNAME = '" || Tbname || "' ",
               "ORDER BY COLNO"
    ADDRESS ISPEXEC "SELECT PGM(SQLISPF)";
    Do i = 1 to _nrows
      COLNAME   = strip(value(_vn.1"."strip(i,l,'Ø')))
      COLTYPE   = strip(value(_vn.2"."strip(i,l,'Ø')))
      COLLEN    = strip(value(_vn.3"."strip(i,l,'Ø')))
      COLSCALE  = strip(value(_vn.4"."strip(i,l,'Ø')))
      COLNULLS  = strip(value(_vn.5"."strip(i,l,'Ø')))
      INDMODT   = Ø
      OLDCOLN   = strip(value(_vn.6"."strip(i,l,'Ø')))
      KEYSEQ    = strip(value(_vn.7"."strip(i,l,'Ø')))
      Address ISPEXEC 'TBADD 'tbnam
    End
    Address ISPEXEC 'TBTOP 'tbnam
  end
  disprc = Ø
  Do While (disprc < 8)
    Address ISPEXEC 'TBQUERY 'tbnam' ROWNUM(rowcnt)'
    If csrrow <= Ø Then csrrow = 1
    Address ISPEXEC 'TBDISPL 'tbnam' PANEL('panel') ,
                    CSRROW('csrrow') MSG('msg') ,
                    CURSOR('cursor') AUTOSEL(NO)'
    disprc = rc
    if disprc < 8 Then Do
      Do While (ZTDSELS > Ø)
        indmod  = 1
        INDMODT = 1
        Address ISPEXEC 'TBPUT 'tbnam
        Address ISPEXEC 'TBDISPL 'tbnam
        if rc >= 8 Then Do
          Address ISPEXEC 'TBCLOSE 'tbnam
          Exit 2Ø
        End
      End
      If word(strip(ZCMD), 1) = 'M' | word(strip(ZCMD), 1) = 'm' Then
        If indmod = 1 Then call process_s
          Else say "First, you must change some column(s)"
      disprc = Ø
    End
  End
  Address ISPEXEC 'TBCLOSE 'tbnam
Exit

/* *** PROCEDURES ************************************************** */

process_s:
  Address ISPEXEC 'TBTOP 'tbnam
  Address ISPEXEC 'TBVCLEAR 'tbnam
  INDMODT = 1
  Address ISPEXEC 'TBSARG 'tbnam' NEXT NAMECOND(INDMODT,EQ)'
  Address ISPEXEC 'TBSCAN 'tbnam' NOREAD POSITION('crpname')'
  RepeatP = Ø
  RepeatT = Ø
  RepeatR = Ø
  RepeatA = Ø
  RepeatI = Ø
  RepeatTg = Ø
  RepeatV = Ø
  RepeatV1 = Ø
  RepeatAu = Ø
  RepeatPP = Ø
  RepeatDR = Ø
  RepeatAR = Ø
  Do While rc = Ø
    Address ISPEXEC 'TBGET ' tbnam
    Call process_rel Tbcre Tbname OLDCOLN KEYSEQ
    csrrow = crpname
    drop crpname
    Address ISPEXEC 'TBSCAN 'tbnam' NOREAD POSITION('crpname')'
  End
```

```
drop crpname
Call process_tab Tbcre Tbname
RepeatT = RepeatT + 1
LineForRepeatT.Ø = RepeatT
LineForRepeatT.RepeatT = "CREATE TABLE "||Tbcre||"."||Tbname||" ("
pkey = ""
Address ISPEXEC 'TBTOP 'tbnam
Do i = 1 To rowcnt
  Address ISPEXEC 'TBSKIP 'tbnam
  Address ISPEXEC 'TBGET ' tbnam
  ss = value(WORD(tvars, 1)) || " " || value(WORD(tvars, 2))
  If value(WORD(tvars, 2)) = "DECIMAL" Then
    ss = ss || "(" || value(WORD(tvars, 3)) || "," ||,
         value(WORD(tvars, 4)) || ")"
  Else Do
    If value(WORD(tvars, 2)) = "CHAR" |,
       value(WORD(tvars, 2)) = "VARCHAR" |,
       value(WORD(tvars, 2)) = "GRAPHIC" |,
       value(WORD(tvars, 2)) = "VARGRAPHIC" |,
       value(WORD(tvars, 2)) = "CLOB" |,
       value(WORD(tvars, 2)) = "BLOB" Then
      ss = ss || "(" || value(WORD(tvars, 3)) || ")"
  End
  If value(WORD(tvars, 5)) = "N" Then ss = ss || " NOT NULL"
  If value(WORD(tvars, 2)) = "ROWID" Then ss=ss||" GENERATED ALWAYS"
  If value(WORD(tvars, 2)) ¬= "ROWID" &,
     value(WORD(tvars, 2)) ¬= "BLOB" &,
     value(WORD(tvars, 2)) ¬= "CLOB"
  Then call process_col Tbcre Tbname value(WORD(tvars, 7)),
                        value(WORD(tvars, 1))
  Else If (value(WORD(tvars, 2)) = "BLOB" |,
           value(WORD(tvars, 2)) = "CLOB")
       Then call process_aux Tbcre Tbname value(WORD(tvars, 1)),
                             value(WORD(tvars, 7))
  If value(WORD(tvars, 8)) > Ø Then
    pkey = pkey || value(WORD(tvars, 1)) || ","
  RepeatT = RepeatT + 1
  LineForRepeatT.Ø = RepeatT
  If i = rowcnt Then
    If pkey ¬= "" Then LineForRepeatT.RepeatT = ss || ","
    Else LineForRepeatT.RepeatT = ss || ")"
  Else LineForRepeatT.RepeatT = ss || ","
End
If pkey ¬= "" Then Do
  pkey = "PRIMARY KEY (" || Left(pkey, Length(pkey) - 1) || "))"
  RepeatT = RepeatT + 1
  LineForRepeatT.Ø = RepeatT
  LineForRepeatT.RepeatT = pkey
End
Address ISPEXEC 'TBTOP 'tbnam
```

```
  call process_tb Tbcre Tbname
  call process_idx Tbcre Tbname
  call process_trig Tbcre Tbname
  call process_view Tbcre Tbname 1
  If RepeatV > Ø Then Do
    Do k = 1 To RepeatV
      call process_view1 value(WORD(LineForRepeatV.k, 1)),
                         value(WORD(LineForRepeatV.k, 2))
    End
  End
  call process_auth Tbcre Tbname
  call process_plpkg Tbcre Tbname
  Address ISPEXEC 'TBTOP 'tbnam
  Do i = 1 To rowcnt
    Address ISPEXEC 'TBSKIP 'tbnam
    Address ISPEXEC 'TBGET ' tbnam
    If value(WORD(tvars, 6)) = 1
    Then Do
      call process_trans Tbcre Tbname value(WORD(tvars, 1)),
                                      value(WORD(tvars, 7))
      call process_transt Tbcre Tbname value(WORD(tvars, 1)),
                                       value(WORD(tvars, 7))
      call process_transv Tbcre Tbname value(WORD(tvars, 1)),
                                       value(WORD(tvars, 7))
      call process_tranrel Tbcre Tbname value(WORD(tvars, 1)),
                                        value(WORD(tvars, 7))
    End
  End
  Address ISPEXEC 'TBTOP 'tbnam
  call process_crejob
Return

/* SYSRELS */
process_rel: PROCEDURE EXPOSE RepeatP LineForRepeatP. ,
                              RepeatDR LineForRepeatDR. ,
                              RepeatAR LineForRepeatAR. DB2V
  parse arg Tbc Tbn Coln Keys
  If Keys > Ø Then Do
    SQLQUERY = "SELECT CREATOR, TBNAME, RELNAME ",
               "FROM SYSIBM.SYSRELS ",
               "WHERE REFTBCREATOR = '" || Tbc || "' AND ",
                     "REFTBNAME = '" || Tbn || "' ",
               "ORDER BY 1, 2, 3"
    ADDRESS ISPEXEC "SELECT PGM(SQLISPF)";
    Do i = 1 To _nrows
      fnd1 = Ø
      Do j = 1 To RepeatP
        If LineForRepeatP.j = CREATOR.i||" "||TBNAME.i||" "||,
                              RELNAME.i||" 1"
        Then fnd1 = 1
      End
      if fnd1 = Ø Then Do
        RepeatP = RepeatP + 1
        LineForRepeatP.Ø = RepeatP
        LineForRepeatP.RepeatP = CREATOR.i||" "||TBNAME.i||" "||,
                                 RELNAME.i||" 1"
      End
    End
  End
  SQLQUERY = "SELECT a.CREATOR, a.TBNAME, a.RELNAME ",
             "FROM SYSIBM.SYSRELS a, SYSIBM.SYSFOREIGNKEYS b ",
             "WHERE a.CREATOR = b.CREATOR AND ",
                   "a.TBNAME  = b.TBNAME AND ",
                   "a.RELNAME = b.RELNAME AND ",
                   "a.CREATOR = '" || Tbc || "' AND ",
                   "a.TBNAME  = '" || Tbn || "' AND ",
                   "b.COLNO   = " || Coln ||,
            " ORDER BY 1, 2, 3"
  ADDRESS ISPEXEC "SELECT PGM(SQLISPF)";
  Do i = 1 To _nrows
    fnd1 = Ø
    Do j = 1 To RepeatP
      If LineForRepeatP.j = CREATOR.i||" "||TBNAME.i||" "||,
                            RELNAME.i||" 1"
      Then fnd1 = 1
    End
    if fnd1 = Ø Then Do
      RepeatP = RepeatP + 1
      LineForRepeatP.Ø = RepeatP
      LineForRepeatP.RepeatP = CREATOR.i||" "||TBNAME.i||" "||,
                               RELNAME.i||" 1"
    End
  End
  SQLQUERY = "SELECT CREATOR, TBNAME, RELNAME ",
             "FROM SYSIBM.SYSRELS ",
             "WHERE (CREATOR = '" || Tbc || "' AND ",
                    "TBNAME = '" || Tbn || "') OR ",
                   "(REFTBCREATOR = '" || Tbc || "' AND ",
                    "REFTBNAME = '" || Tbn || "') ",
             "ORDER BY 1, 2, 3"
  ADDRESS ISPEXEC "SELECT PGM(SQLISPF)";
  Do i = 1 To _nrows
    fnd1 = Ø
    Do j = 1 To RepeatP
      If Left(LineForRepeatP.j, Length(LineForRepeatP.j) - 2) = ,
         CREATOR.i||" "||TBNAME.i||" "||RELNAME.i
      Then fnd1 = 1
    End
    if fnd1 = Ø Then Do
      RepeatP = RepeatP + 1
      LineForRepeatP.Ø = RepeatP
      LineForRepeatP.RepeatP = CREATOR.i||" "||TBNAME.i||" "||,
                               RELNAME.i||" Ø"
    End
  End
  Do i = 1 To RepeatP
    Tc = WORD(LineForRepeatP.i, 1)
    Tb = WORD(LineForRepeatP.i, 2)
    Tr = WORD(LineForRepeatP.i, 3)
    RepeatDR = RepeatDR + 1
    LineForRepeatDR.Ø = RepeatDR
    LineForRepeatDR.RepeatDR = "ALTER TABLE "||Tc||"."||Tb||,
                              " DROP FOREIGN KEY "||Tr||";"
    SQLQUERY = "SELECT A.REFTBCREATOR, A.REFTBNAME, A.DELETERULE, ",
                      "B.COLNAME, B.COLNO, B.COLSEQ ",
               "FROM SYSIBM.SYSRELS A, SYSIBM.SYSFOREIGNKEYS B ",
               "WHERE A.CREATOR = B.CREATOR AND ",
                     "A.TBNAME  = B.TBNAME AND ",
                     "A.RELNAME = B.RELNAME AND ",
                     "A.CREATOR = '" || Tc || "' AND ",
                     "A.TBNAME  = '" || Tb || "' AND ",
                     "A.RELNAME = '" || Tr || "' ",
               "ORDER BY 1, 2, 6"
    ADDRESS ISPEXEC "SELECT PGM(SQLISPF)";
    col = "("
    Do r = 1 To _nrows
      If r > 1 Then col = col || ", " || STRIP(COLNAME.r)
      Else Do
        col = col || STRIP(COLNAME.r)
        ref = STRIP(REFTBCREATOR.r)||"."||STRIP(REFTBNAME.r)
        Select
          When DELETERULE.r = "A" Then refr = "NO ACTION"
          When DELETERULE.r = "C" Then refr = "CASCADE"
          When DELETERULE.r = "N" Then refr = "SET NULL"
          When DELETERULE.r = "R" Then refr = "RESTRICT"
          Otherwise
        End
      End
    End
    col = col || ")"
    If WORD(LineForRepeatP.i, 4) = "1" Then comm = "-- "
    Else comm = ""
    RepeatAR = RepeatAR + 1
    LineForRepeatAR.Ø = RepeatAR
    LineForRepeatAR.RepeatAR = comm||"ALTER TABLE "||Tc||"."||Tb||,
                              " ADD FOREIGN KEY "||Tr
    RepeatAR = RepeatAR + 1
    LineForRepeatAR.Ø = RepeatAR
    LineForRepeatAR.RepeatAR = comm||"  "||col
    RepeatAR = RepeatAR + 1
```

```
    LineForRepeatAR.Ø = RepeatAR
    LineForRepeatAR.RepeatAR = comm||"  REFERENCES "||ref
    RepeatAR = RepeatAR + 1
    LineForRepeatAR.Ø = RepeatAR
    LineForRepeatAR.RepeatAR = comm||"  ON DELETE "||refr||";"
  End
Return

/* SYSTABLES/SYSTABLESPACE */
process_tab: PROCEDURE EXPOSE RepeatT LineForRepeatT. DB2V
  parse arg Tbc Tbn
  SQLQUERY = "SELECT a.PARTITION, a.TSNAME, a.DBNAME, a.PQTY, ",
                   "a.SQTY, a.STORNAME, a.PCTFREE, a.COMPRESS, ",
                   "b.BPOOL, b.PARTITIONS, ",
                   "b.LOCKRULE, b.PGSIZE, b.ERASERULE, ",
                   "b.CLOSERULE, b.LOCKMAX, ",
                   "b.LOCKPART, b.ENCODING_SCHEME ",
             "FROM SYSIBM.SYSTABLEPART a, ",
                  "SYSIBM.SYSTABLESPACE b, ",
                  "SYSIBM.SYSTABLES c ",
             "WHERE a.PARTITION > Ø AND ",
                   "a.DBNAME = b.DBNAME AND ",
                   "a.TSNAME = b.NAME AND ",
                   "b.DBNAME = c.DBNAME AND ",
                   "b.NAME = c.TSNAME AND ",
                   "c.CREATOR = '" || Tbc || "' AND ",
                   "c.NAME = '" || Tbn || "' ",
             "ORDER BY 3, 2, 1"
  ADDRESS ISPEXEC "SELECT PGM(SQLISPF)";
  If _nrows = Ø Then Do
    RepeatT = RepeatT + 1
    LineForRepeatT.Ø = RepeatT
    LineForRepeatT.RepeatT = "DROP TABLE "||Tbc||"."||Tbn||";"
    RepeatT = RepeatT + 1
    LineForRepeatT.Ø = RepeatT
    LineForRepeatT.RepeatT = "COMMIT;"
  End
  Else Do
    Do i = 1 To _nrows
      If i = 1 Then Do
        RepeatT = RepeatT + 1
        LineForRepeatT.Ø = RepeatT
        LineForRepeatT.RepeatT = "DROP TABLESPACE "||,
                                 STRIP(DBNAME.i)||"."||,
                                 STRIP(TSNAME.i)||";"
        RepeatT = RepeatT + 1
        LineForRepeatT.Ø = RepeatT
        LineForRepeatT.RepeatT = "COMMIT;"
        RepeatT = RepeatT + 1
        LineForRepeatT.Ø = RepeatT
        LineForRepeatT.RepeatT = "CREATE TABLESPACE "||,
                                   STRIP(TSNAME.i)||,
                                 " IN "||STRIP(DBNAME.i)||,
                                 " NUMPARTS "||PARTITIONS.i||"("
        If LOCKMAX.i = -1 Then LMAX = " LOCKMAX SYSTEM"
        Else LMAX = ""
        If LOCKPART.i = "Y" Then LPAR = "YES"
        Else LPAR = "NO"
        If CLOSERULE.i = "Y" Then CLO = "YES"
        Else CLO = "NO"
        Select
          When LOCKRULE.i = "A" Then LSZ = "ANY"
          When LOCKRULE.i = "L" Then LSZ = "LOB"
          When LOCKRULE.i = "P" Then LSZ = "PAGE"
          When LOCKRULE.i = "R" Then LSZ = "ROW"
          When LOCKRULE.i = "S" Then LSZ = "TABLESPACE"
          When LOCKRULE.i = "T" Then LSZ = "TABLE"
          Otherwise
        End
        Select
          When ENCODING_SCHEME.i = "A" Then CCS = "ASCII"
          When ENCODING_SCHEME.i = "U" Then CCS = "UNICODE"
          Otherwise CCS = "EBCDIC"
        End
        BPL = STRIP(BPOOL.i)
      End
      If COMPRESS.i = "Y" Then COMPR = "YES"
      Else COMPR = "NO"
      If i < _nrows Then COMPR = COMPR || ","
      Else COMPR = COMPR || ")"
      RepeatT = RepeatT + 1
      LineForRepeatT.Ø = RepeatT
      LineForRepeatT.RepeatT = "PART "||STRIP(PARTITION.i)||,
                               " USING STOGROUP "||,
                                 STRIP(STORNAME.i)||,
                               " PRIQTY "||(PQTY.i * PGSIZE.i)||,
                               " SECQTY "||(SQTY.i * PGSIZE.i)||,
                               " PCTFREE "||STRIP(PCTFREE.i)||,
                               " COMPRESS "||COMPR
    End
    RepeatT = RepeatT + 1
    LineForRepeatT.Ø = RepeatT
    LineForRepeatT.RepeatT = "BUFFERPOOL "||BPL||,
                             " LOCKSIZE "||LSZ||,
                             LMAX||,
                             " CLOSE "||CLO||,
                             " CCSID "||CCS||,
                             " LOCKPART "||LPAR||";"
  End
Return
```

```
/* SYSCOLUMNS/SYSFIELDS/SYSCHECKS */
process_col: PROCEDURE EXPOSE ss RepeatR LineForRepeatR. DB2V
  parse arg Tbc Tbn ColOld ColNew
    SQLQUERY = "SELECT DEFAULT, DEFAULTVALUE, FLDPROC, ",
                     "REMARKS, NAME ",
               "FROM SYSIBM.SYSCOLUMNS ",
               "WHERE TBCREATOR = '" || Tbc || "' AND ",
                     "TBNAME = '" || Tbn || "' AND ",
                     "COLNO = " || ColOld
    ADDRESS ISPEXEC "SELECT PGM(SQLISPF)";
    If _nrows = 1 Then Do
      If DEFAULT.1 = "1" | DEFAULT.1 = "2" | DEFAULT.1 = "3" |,
         DEFAULT.1 = "4" | DEFAULT.1 = "5" Then Do
        ss = ss || " WITH DEFAULT"
        If DEFAULT.1 = "1" | DEFAULT.1 = "5"
        Then ss = ss||" '"||STRIP(SUBSTR(DEFAULTVALUE.1,1,254))||"'"
        Else If DEFAULT.1 = "2" | DEFAULT.1 = "3" | DEFAULT.1 = "4"
             Then ss = ss||" "||STRIP(SUBSTR(DEFAULTVALUE.1,1,254))
      End
      If FLDPROC.1 = "Y" Then Do
        SQLQUERY = "SELECT FLDPROC, PARMLIST ",
               "FROM SYSIBM.SYSFIELDS ",
               "WHERE TBCREATOR = '" || Tbc || "' AND ",
                     "TBNAME = '" || Tbn || "' AND ",
                     "COLNO = " || ColOld
        ADDRESS ISPEXEC "SELECT PGM(SQLISPF)";
        If _nrows = 1 Then Do
          ss = ss || " FIELDPROC " || STRIP(FLDPROC.1)
          If SUBSTR(PARMLIST.1, 1, 1) ¬= " "
          Then ss = ss || "(" || STRIP(PARMLIST.1) || ")"
        End
      End
      If SUBSTR(STRIP(REMARKS.1), 1, 1) ¬= " " Then Do
        RepeatR = RepeatR + 1
        LineForRepeatR.Ø = RepeatR
        LineForRepeatR.RepeatR = "COMMENT ON COLUMN " ||,
                                  Tbc||"."||Tbn||"."||ColNew||,
                                 " IS '"||STRIP(REMARKS.1)||"';"
      End
      SQLQUERY = "SELECT A.CHECKNAME, A.CHECKCONDITION ",
                 "FROM SYSIBM.SYSCHECKS A, SYSIBM.SYSCHECKDEP B ",
                 "WHERE A.TBOWNER = '" || Tbc || "' AND ",
                       "A.TBNAME = '" || Tbn || "' AND ",
                       "B.COLNAME = '" || STRIP(NAME.1) || "' AND ",
                       "A.TBOWNER = B.TBOWNER AND ",
                       "A.TBNAME = B.TBNAME AND ",
                       "A.CHECKNAME = B.CHECKNAME"
      ADDRESS ISPEXEC "SELECT PGM(SQLISPF)";
      If _nrows = 1 Then Do
        COND = STRIP(CHECKCONDITION.1)
        If ColNew ¬= STRIP(NAME.1)
        Then COND = TRANSLATE(COND, ColNew, NAME.1)
        ss = ss||" CONSTRAINT "||STRIP(CHECKNAME.1)||" CHECK("||,
             COND||")"
      End
    End
Return

/* AUX TABLES */
process_aux: PROCEDURE EXPOSE RepeatA LineForRepeatA. DB2V
  parse arg Tbc Tbn ColNew ColOld
  SQLQUERY = "SELECT A.COLNAME, A.AUXTBOWNER, A.AUXTBNAME, ",
                   "B.DBNAME, B.TSNAME, C.CREATOR, C.NAME ",
             "FROM SYSIBM.SYSAUXRELS A, ",
                  "SYSIBM.SYSTABLES B, ",
                  "SYSIBM.SYSINDEXES C, ",
                  "SYSIBM.SYSCOLUMNS D ",
             "WHERE A.TBOWNER = '" || Tbc || "' AND ",
                   "A.TBNAME = '" || Tbn || "' AND ",
                   "A.AUXTBOWNER = B.CREATOR AND ",
                   "A.AUXTBNAME = B.NAME AND ",
                   "A.AUXTBOWNER = C.TBCREATOR AND ",
                   "A.AUXTBNAME = C.TBNAME AND ",
                   "A.TBOWNER  = D.TBCREATOR AND ",
                   "A.TBNAME = D.TBNAME AND ",
                   "D.COLNO = " || ColOld || " AND ",
                   "A.COLNAME = D.NAME"
  ADDRESS ISPEXEC "SELECT PGM(SQLISPF)";
  Do i = 1 To _nrows
    RepeatA = RepeatA + 1
    LineForRepeatA.Ø = RepeatA
    LineForRepeatA.RepeatA = "CREATE LOB TABLESPACE "||STRIP(TSNAME.i)
    RepeatA = RepeatA + 1
    LineForRepeatA.Ø = RepeatA
    LineForRepeatA.RepeatA = "IN "||STRIP(DBNAME.i)
    RepeatA = RepeatA + 1
    LineForRepeatA.Ø = RepeatA
    LineForRepeatA.RepeatA = "LOG NO;"
    RepeatA = RepeatA + 1
    LineForRepeatA.Ø = RepeatA
    LineForRepeatA.RepeatA = "CREATE AUX TABLE "||,
                    STRIP(AUXTBOWNER.i)||"."||STRIP(AUXTBNAME.i)
    RepeatA = RepeatA + 1
    LineForRepeatA.Ø = RepeatA
    LineForRepeatA.RepeatA = "IN "||STRIP(DBNAME.i)||"."||,
                                  STRIP(TSNAME.i)
    RepeatA = RepeatA + 1
    LineForRepeatA.Ø = RepeatA
    LineForRepeatA.RepeatA = "STORES "||Tbc||"."||Tbn
    RepeatA = RepeatA + 1
    LineForRepeatA.Ø = RepeatA
    LineForRepeatA.RepeatA = "COLUMN "||ColNew||";"
    RepeatA = RepeatA + 1
    LineForRepeatA.Ø = RepeatA
    LineForRepeatA.RepeatA = "CREATE UNIQUE INDEX "||,
                              STRIP(CREATOR.i)||"."||STRIP(NAME.i)
    RepeatA = RepeatA + 1
    LineForRepeatA.Ø = RepeatA
    LineForRepeatA.RepeatA = "ON "||,
                        STRIP(AUXTBOWNER.i)||"."||STRIP(AUXTBNAME.i)||";"
  End
Return

/* SYSTABLES/SYSSYNONYMS */
process_tb: PROCEDURE EXPOSE RepeatT LineForRepeatT. DB2V ,
                             RepeatR LineForRepeatR.
  parse arg Tbc Tbn
  SQLQUERY = "SELECT DBNAME, TSNAME, EDPROC, VALPROC, REMARKS, ",
                     "CLUSTERTYPE, AUDITING, DATACAPTURE, ",
                     "ENCODING_SCHEME ",
             "FROM SYSIBM.SYSTABLES ",
             "WHERE CREATOR = '" || Tbc || "' AND ",
                   "NAME = '" || Tbn || "'"
  ADDRESS ISPEXEC "SELECT PGM(SQLISPF)";
  If _nrows = 1 Then Do
    RepeatT = RepeatT + 1
    LineForRepeatT.Ø = RepeatT
    LineForRepeatT.RepeatT = "IN "|| STRIP(DBNAME.1)||"."||,
                                     STRIP(TSNAME.1)
    RepeatT = RepeatT + 1
    LineForRepeatT.Ø = RepeatT
    Select
      When ENCODING_SCHEME.1 = "A" Then CCS = "ASCII"
      When ENCODING_SCHEME.1 = "U" Then CCS = "UNICODE"
      Otherwise CCS = "EBCDIC"
    End
    LineForRepeatT.RepeatT = "CCSID " || CCS
    If EDPROC.1 ¬= " " Then Do
      RepeatT = RepeatT + 1
      LineForRepeatT.Ø = RepeatT
      LineForRepeatT.RepeatT = "EDITPROC " || STRIP(EDPROC.1)
    End
    If VALPROC.1 ¬= " " Then Do
      RepeatT = RepeatT + 1
      LineForRepeatT.Ø = RepeatT
      LineForRepeatT.RepeatT = "VALIDPROC " || STRIP(VALPROC.1)
    End
    If CLUSTERTYPE.1 ¬= " " Then Do
      RepeatT = RepeatT + 1
      LineForRepeatT.Ø = RepeatT
      LineForRepeatT.RepeatT = "WITH RESTRICT ON DROP"
    End
    Select
      When AUDITING.1 = "A" Then AUD = "AUDIT ALL"
      When AUDITING.1 = "C" Then AUD = "AUDIT CHANGE"
      Otherwise AUD = " "
    End
    If AUD ¬= " " Then Do
      RepeatT = RepeatT + 1
      LineForRepeatT.Ø = RepeatT
      LineForRepeatT.RepeatT = AUD
    End
    If SUBSTR(STRIP(REMARKS.1), 1, 1) ¬= " " Then Do
      RepeatR = RepeatR + 1
      LineForRepeatR.Ø = RepeatR
      LineForRepeatR.RepeatR = "COMMENT ON TABLE "||,
                               Tbc||"."||Tbn||" IS '"||,
                               STRIP(REMARKS.1)||"';"
    End
    RepeatT = RepeatT + 1
    LineForRepeatT.Ø = RepeatT
    LineForRepeatT.RepeatT = ";"
    SQLQUERY = "SELECT CREATOR, NAME ",
               "FROM SYSIBM.SYSSYNONYMS ",
               "WHERE TBCREATOR = '" || Tbc || "' AND ",
                     "TBNAME = '" || Tbn || "'"
    ADDRESS ISPEXEC "SELECT PGM(SQLISPF)";
    Do i = 1 To _nrows
      RepeatT = RepeatT + 1
      LineForRepeatT.Ø = RepeatT
      LineForRepeatT.RepeatT = "CREATE SYNONYM " ||,
                                STRIP(CREATOR.i)||"."||,
                                STRIP(NAME.i)||" FOR "||,
                                tbc||"."||Tbn||";"
    End
  End
Return
```

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

Nikola Lazovic
DB2 System Administrator
Postal Savings Bank (Serbia and Montenegro)

DB2 UDB LUW – High Performance Unload

This article looks at the High Performance Unload tool, which is part of the DB2 toolset for UDB (from the start of 2004, there are five tools available: Web Query Tool, Table Editor, Performance Expert, Recovery Expert, and High Performance Unload).

The High Performance Unload (HPU) tool allows you to unload DB2 tables faster than by using the DB2 EXPORT utility. The tool is available for Windows and pSeries platforms (not iSeries).

How does the tool work? Well, it bypasses the DB2 engine and directly accesses the underlying files, which hold the DB2 data. This type of accessing imposes some restrictions. The data that you want to extract is selected by issuing a standard type of SQL SELECT query. However, the type of SQL that you can use with HPU must be ‘simple’. And what does that mean? It’s easier to look at what makes a ‘complex’ query. Well, examples of a complex SQL query are when you join two tables, when you try to use an ORDER BY clause, or if you try to select from a view. I hope you can see the pattern here – anything that involves manipulating the underlying data will be complex SQL. This will become more evident in the examples that we will look at.

Installing the code is relatively easy, and if you install it on a Windows machine, you need to reboot the box before being able to use the tool. I installed the HPU tool (V2.1) on a Windows XP machine running DB2 UDB V8.1 FP2 and used the db2admin userid throughout.

When you have installed the code (and rebooted on Windows), you will see another option at the bottom of the list when you right click on a table name from the control centre. This option is the **Unload Table** option, and it is a wizard that will take you through a series of panels to determine what you want to unload and any SQL that you want to execute, and then execute the command for you. I will be showing you how to use the tool by issuing commands (you don’t need to be in a CLP screen to issue these commands, you can issue them from an ordinary C:> prompt). Although HPU bypasses the DB2 engine, the engine must still be active for the tool to work.

At installation time a configuration file is created (C:\Program Files\IBM\hpu\V2.1\cfg\db2hpu.cfg), which contains the default database and instance names, as well as parameters affecting the tool’s performance. Check the manual for these parameters.

You can issue HPU commands from a command line in two modes – direct command mode and control file mode. The advantage of using the direct mode is that the command is simple. The disadvantage is that you cannot specify any SQL processing. If you want to specify SQL processing, then you need to use a control file and specify the SQL in that file.

You can see all the options available for the direct mode simply by issuing the HPU command from the prompt:

```
C:\>HPU
```

HPU ensures that the data it unloads is consistent by quiescing the table before commencing the unload. This action is controlled by the QUIESCE and LOCK control file parameters. The default for both parameters is YES, which means that HPU tells DB2 that it is doing an unload of the table, and DB2 takes a quiesce share lock on the table for the duration of the unload. If you don’t want to take these types of locks, you could unload the table from a DB2 back-up. This is discussed in more detail in the examples.

So let’s look at some examples. I will start by showing some simple examples and then show some complex SQL query examples (ie non-simple SQL). I will use the tables in the SAMPLE database so that you can try the examples as well.

As a first example, let’s just try to extract all the rows from the EMPLOYEE table and write the rows to a file called e:\temp\out.txt. So, we can issue the direct command as follows:

```
C:\>db2hpu -d sample -t db2admin.employee -o e:\temp\out.txt
```

We have to specify the database that contains the EMPLOYEE table using the **-d** parameter (SAMPLE in this case), and the fully qualified table name we want to unload using the **-t** parameter (db2admin.employee in this case). The output file is specified using the **-o** parameter. It's as simple as that!

We could have used a control file instead (by specifying the **-f** parameter on the HPU command), and the command would then have been:

```
C:\>db2hpu -f hpu01.ctl
```

And the file hpu01.txt would contain:

```
global connect to sample;
unload tablespace
select * from db2admin.employee;
output ("c:\temp\out.txt" replace) format del ;
```

We need to specify the database that we want to connect to, the SQL that we want to execute (in our case select * from db2admin.employee), and the output file. Here we have also specified that we want any existing output file overwritten and the format of the output. We could have specified IXF as the format.

The above example illustrates a simple SQL query. What would happen if we had specified **select * from db2admin.employee ORDER by 1**? This would be considered a complex query and if you tried to run the HPU command it would return:

```
INZU063W Unsupported SELECT, will attempt DB2 processing
SQL3104N  The Export utility is beginning to export data to file
"c:\temp\out.txt".

SQL3105N  The Export utility has finished exporting "32" rows.

INZU413I HPU successfully ended: Real time -> 0m0.000000s
```

What has happened here is that HPU has decided that it cannot satisfy our SQL request by going directly to the underlying data files, and has therefore sent it to DB2 to handle. DB2 has invoked the EXPORT utility. I think it is also possible for DB2 to perform just a straight SELECT rather than use the EXPORT utility. This option of whether to send a complex query to the DB2 engine or not is controlled by a control file parameter called DB2 and its default value is YES (which means do send it to DB2). You can override this value in your control file as follows:

```
global connect to sample;
unload tablespace
DB2 NO
select * from db2admin.employee;
output ("c:\temp\out.txt" replace) format del ;
```

This control file tells HPU that if it cannot handle the SQL query, the command should abend (this is what the DB2 NO line does), then you would get the prompt back without any error messages(!):

```
C:\>db2hpu -f hpu01.ctl
      ----+----1--------+---
000001 global connect
000001 global connect to sample;
000002 unload tablespace
000003 DB2 NO
000004 select * from db2admin.employee order by 1;
000005 output ("c:\temp\out.txt" replace) format del ;


C:>
```

If you try to syntax check the above control file (using the **–n** parameter), you get back just the command prompt . If you delete the DB2 NO line, you get what I would expect, which is a message saying, ‘INZU426I Control file systax is correct’. This is just something to be aware of (and note the spelling of ‘syntax’)!

As mentioned previously, you can also unload data from DB2 back-ups (full on-line and off-line back-ups only, not incremental or delta ones). This means that you do not need to put a lock on your production table to extract rows from it. The control file to do this is shown below:

```
unload tablespace
backup
"c:\db2_backups\SAMPLE.0\DB2\NODE0000\CATN0000\yyyymmdd\hhmmss.001"
select * from db2admin.employee;
output ("e:\temp\out.txt" replace) format del ;
```

The above assumes that our back-up is in the c:\db2_backups directory. You obviously need to put in the correct date/timestamp.

Now let's look at some other functions that the HPU tool has. You can also use the tool to sample data in a table. For example, say we wanted to extract only one row in every 1,000 from the EMPLOYEE table. We would then add an INTERVAL 1,000 line before the SELECT line, as shown below:

```
interval 1000
select * from db2admin.employee ;
```

(The EMPLOYEE table doesn't have 1,000 rows in it – I have just shown the control file contents as an illustration.)

We could also skip the first 2,000 rows in the table by using the SKIP parameter, as shown below:

```
skip 2000
select * from db2admin.employee ;
```

You can also unload LOBs to different files. This is shown in the control file example sample04.ctl in the C:\Program Files\IBM\hpu\V2.1\sample directory.

Finally, it is possible to unload data from one table to one or many output files with just one pass through the data. As example of such a control file is shown below. We are extracting data from the EMPLOYEE table in two different formats, DEL and IXF, with one pass through the data.

```
global connect to sample;
unload tablespace
select * from db2admin.employee ;
output ("c:\TEMP\OUT_EX04.TXT" replace) format del
select * from db2admin.employee ;
output ("c:\TEMP\OUT_EX04A.TXT" replace) format ixf;
```

Running this control file will generate the following output:

```
INZU410I HPU utility has unloaded 32 rows on xxx host for
DB2ADMIN.EMPLOYEE in c:\TEMP\OUT_EX04.TXT.
INZU410I HPU utility has unloaded 32 rows on xxx host for
DB2ADMIN.EMPLOYEE in c:\TEMP\OUT_EX04A.TXT.
INZU413I HPU successfully ended: Real time -> 0m0.000000s
```

In the paragraphs below I present some answers to common questions asked about the tool.

Can I use HPU on a multi-node database? Yes you can, but you need to install the HPU code on each physical node. If you haven't installed the code on each node, you must use the EXPORT utility.

Are there any HPU manuals? Yes. The manual you want is *IBM DB2 High Performance Unload for Multiplatforms and Workgroups User's Guide Version 2 Release 1* (SC27-1623-03).

Where can I find control file examples? The manual is a good source of examples, as are the Installation Verification Program samples located in C:\Program Files\IBM\hpu\V2.1\sample.

I hope I have shown what the High Performance Unload tool for LUW is and how to use it. It has some powerful features, especially unloading data from a back-up and sampling the data in a table, and is a welcome addition to the DBA's arsenal.

C Leonard
Freelance Consultant (UK)

DB2 news

Alpha Software has announced Version 6 of Alpha 5, its database and application development tool.

Version 6 brings point-and-click development to other databases, including DB2, Oracle, SQL Server and MySQL, or any database that has an ADO component.

The product can be used to simplify the process of creating database-driven Web sites.

For further information contact:
Alpha Software, 83 Cambridge St, Burlington, MA 01803-4483, USA.
Tel: (781) 229 4500.
URL: http://www.alphasoftware.com/products/a5v5/overview.asp.

* * *

Bus-Tech has announced the next release of Mainframe Appliance for Storage (MAS) with EMC Centera Support. This release of Bus-Tech's MAS, a tape-on-disk controller, now includes support for DB2 as a repository for MAS metadata.

Mainframe Appliance for Storage with EMC Centera Support provides direct mainframe connection to EMC's Centera, allowing mainframe tape volumes to be stored as objects on EMC's content addressable storage. With this latest version of the MAS, metadata maintained by MAS is stored directly on the IBM or compatible mainframe using DB2. This allows customers to incorporate back-up and redundancy of the metadata into their existing mainframe protection policies.

When storing objects on EMC Centera the MAS maintains metadata, which cross-references mainframe tape VOLSERs to Centera objects. The MAS later references this metadata in order to retrieve tape volumes from Centera in response to mount requests received from the mainframe. Using DB2 Connect software, MAS now stores its metadata directly on the mainframe. With this latest addition, protection of the metadata can then easily be incorporated into normal mainframe back-up processes of the DB2 database.

For further information contact:
Bus-Tech, 129 Middlesex Turnpike, Burlington, MA 01803, USA.
Tel: (781) 272 8200.

URL: http://www.bustech.com/products/mainframe-appliance-storage.asp.

* * *

CipherSoft has announced Version 5.0 of Exodus, its automated solution for customers who want to migrate their complex Oracle business logic into Java source code and DB2-compliant SQL, ready for deployment with DB2.

Exodus 5.0 can automatically migrate Oracle PL/SQL packages, procedures, functions, and triggers to DB2 Stored Procedure language.

It migrates Oracle SQL code and server side PL/SQL into Java source code, ready to be deployed together with DB2. Exodus also supports dynamic Oracle SQL within the DB2 environment. Exodus has the ability to induce automatic generation of the entire static interface components required to deploy Java code within the DB2 environment.

For further information contact:
CipherSoft, #205, 279 MidPark Way, South East, Calgary, Alberta, T2X 1M2, Canada.
Tel: (403) 256 5699.
URL: http://www.ciphersoftinc.com.

xephon