140

DB2

June 2004

In this issue

update

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

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

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Nicole Thomas
E-mail: nicole@xephon.com

Subscriptions and back-issues

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

DB2 Update on-line

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

Contributions

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

Printed in England.

DB2 UDB V8 LUW – the MERGE statement

Have you ever wanted to combine the rows from one table into another? In the past you had to write one, possibly two or more, statements to do this. Well, one of the new SQL statements available with DB2 UDB V8 FP2 (LUW) is the MERGE statement. Put simply, it lets you merge data from one table into another table. The best way to show this is in an example. I ran the following SQL on a Windows 2000 machine running DB2 V8 FP2.

Let's look at a very simple ordering system. We will create two tables called 'total' and 'daily' in the SAMPLE database. As the names suggest, the daily table contains orders taken on that day, and consists of an ID number, a name, a zip code, and the amount of the order. The total table will contain the ID number, a name, zip code, the total number of orders for that ID number, and the total value of all those orders for that ID number. We will want to merge the daily table into a total table, updating records where the ID number is the same, and inserting into the total table where the ID number does not exist in that table.

We will use the MERGE statement, which works on the basis of matching and non-matching columns between two tables, namely the daily and total tables. If a column match is found then the SQL following the WHEN MATCHED THEN keywords is executed; and if no match is found then the SQL following the WHEN NOT MATCHED THEN keywords is executed. This will become clear in the example.

So let's create the total table:

```
create table total (id int, name char(10), zipcode char(10)
,num_of_trans int, amount decimal(6,2))
```

And let's create the daily table:

```
create table daily (id int, name char(10), zipcode char(10), amount int)
```

Now let's insert two rows into each table:

```
insert into total values(1,'Helen','123456',3,100.00)
insert into total values(2,'Fred' ,'234567',1,20.00)

insert into daily values(1,'Helen','123456',10.00)
insert into daily values(3,'Sue'  ,'987654',50.00)
```

And let's select from the two tables:

```
select * from total

ID          NAME       ZIPCODE    NUM_OF_TRANS AMOUNT
----------- ---------- ---------- ------------ --------
          1 Helen      123456                3   100.00
          2 Fred       234567                1    20.00

select * from daily

ID          NAME       ZIPCODE    AMOUNT
----------- ---------- ---------- -----------
          1 Helen      123456              10
          3 Sue        987654              50
```

Now let's issue the MERGE statement, and we will match on the ID column and increment the *num_of_trans* column in the total table if the ID values match (and sum up the amount values). If the ID columns don't match, then the row from the daily table will be inserted into the total table:

```
merge into total t1 using (select * from daily) t2
on (t1.id = t2.id)
when matched then
  update set t1.num_of_trans = t1.num_of_trans + 1,
  t1.amount = t1.amount + t2.amount
when not matched then
  insert (id,name,zipcode,num_of_trans,amount) values
  (t2.id, t2.name, t2.zipcode,1,t2.amount)
```

Now let's select from the total table again:

```
select * from total

ID          NAME       ZIPCODE    NUM_OF_TRANS AMOUNT
----------- ---------- ---------- ------------ --------
          1 Helen      123456                4   110.00
          2 Fred       234567                1    20.00
          3 Sue        987654                1    50.00
```

You can see that the amount column for name *Helen* has gone up to 110 (100 + 10) and the *num_of_trans* value has gone up from 3 to 4. Also, the row from daily for *Fred* has been added to the total table.

We can still select from daily (the merged rows have not been deleted):

```
select * from daily

ID          NAME       ZIPCODE    AMOUNT
----------- ---------- ---------- -----------
          1 Helen      123456              10
          3 Sue        987654              50
```

You can use the index advisor utility (**db2advis**) to advise on any indexes that are required on the tables in the MERGE statement. However, when I tried to look at the MERGE statement using Visual Explain, I got the following message: 'An access plan cannot be created because the statement you used is not in the correct format or is not a query'.

I hope I have shown how the MERGE statement works, and how it might be beneficial to you in certain circumstances.

C Leonard
Freelance Consultant (UK)

An ISPF-SQL interface

What the following REXX EXEC does is quite simple: basically it takes an SQL SELECT statement as input, executes it, and fills the result set into an ISPF table for further processing. The name of the ISPF table is given back to the calling program (plus some other information, like errormessage, sqlcode, number of rows returned, etc). I found that this is a much easier approach to reading data from a DB2 table than the usual procedure:

1 Establish DSNREXX environment.

2 Establish DSNREXX connection.

3 Establish DB2 connection.

4 DECLARE CURSOR.

5 PREPARE SQL statement.

6 OPEN CURSOR.

7 FETCH result rows.

8 CLOSE CURSOR.

9 Examine all possible error conditions for steps 1-8.

If you are a DBA or programmer who often needs to write REXX programs that read DB2 data, you will find DSN2ISPF very handy.

There is one thing you need to beware of: if your ISPF table contains one or more columns that are *not* part of the result set from the DB2 table, specify placeholder values in your SQL statement! Otherwise DSN2ISPF won't work properly.

For example, your ISPF table should consist of three columns – COL1, COL2, and COL3. You're about to read COL1 and COL2 from your DB2 table and COL3 is going to be calculated later in your program. Write an SQL SELECT similar to the following one:

```
SELECT DB2_COL1, DB2_COL2, 0
FROM your-table
WHERE where-condition
```

If you do so, the ISPF table column COL3 will be initially set to zero. Of course you can choose any initial value you want.

Note: DSN2ISPF contains two subroutines, DSN2ISPF_LOCALDB2 and DSN2ISPF_DB2LOC.DSN2ISPF_LOCALDB2 gives you the name of a DB2 subsystem that is running on the System/390 LPAR where DSN2ISPF is currently executing. That DB2 subsystem is used for the DSNREXX

connect. DSN2ISPF_DB2LOC gives you the location name of a given DB2 subsystem-identifier. This location name is used for the SQL CONNECT TO statement. At our site, we use these two subroutines as external REXX EXECs that can be used by any other program. This is a very useful standardization method too.

DSN2ISPF uses some input parameters that are documented in the source code itself. I've chosen default values for most of them, which is sufficient for about 99% of all the occasions when you are using DSN2ISPF.

Following the code for DSN2ISPF you will find a tiny sample program to show the usage of DSN2ISPF in its most basic form.

If you have any questions or comments, feel free to contact me at peter.adlersburg@t-systems.at.

DSN2ISPF

```
/* Rexx
   Proc DSN2ISPF
   Do SQL Select Stmt and Create an ISPF Table containing the Data
   Returns ISPF Tablename (DSN2nnnn)
   Who: Peter Adlersburg / T-Systems Austria

   DSN2ISPF VGETs its Parameters from the ISPF-Var-Pool

   1) DSN2SSID      : the DB2 Subsystem-ID (eg DSN)
   2) DSN2STMT      : the SQL Select Statement that will be executed
   3) DSN2VARS      : a list of Variable names that make up the ISPF
                      table (Format var1,var2,...)
   4) DSN2KEYS      : a list of Variable names that make up the Key of
                      the ISPF Table (if DSN2ISPF_KEYS contains no
                      value an unkeyed ISPF Table is created; Format
                      key1,key2,...)
   5) DSN2TRCE      : Causes a REXX Trace to be activated (normal
                      REXX Trace Values will be used; default 0 (Off))
   6) DSN2ELVL      : Error-Level:
                      Ø ... No Errors and Informations
                      1 ... Errors only (default)
                      2 ... Informations and Errors
   7) DSN2ROWS      : the maximum number of Rows that will be fetched
                      (default is 999)
```

```
   8) DSN2ISOL      : Desired Isolation level (CS,UR,RR,RS; UR is the
                      default)
   9) DSN2TABB      : Behaviour if NO-ROWS-FOUND-Condition is met
                      TAB   ... Build empty ISPF Table and return
                      NOTAB ... Do NOT build ISPF Table and return
                                (default)
  1Ø) DSN2ISPP      : Parameters that will be used for ISPF Table
                      Creation. The default is 'NOWRITE,REPLACE'

  11) DSN2LOCL      : Determines whether a Connect to DSN2SSID can
                      be made via another DB2 subsystem or not
                      This can be useful when DDF of DSN2SSID is down
                      during execution
                      1 ... Connect is allowed via another subsystem
                            (default)
                      Ø ... Connect is done via Locationname of
                            of DSN2SSID
   1), 2) and 3) are mandatory parameters !!

   All Parameters are deleted from ISPF Variable Pool after Execution
   of DSN2ISPF

   DSN2ISPF VPUTs some Variables after Execution that can be used by
   the caller

   DSN2RETC ..... ErrorCode
   DSN2ERRM ..... Errormessage
   DSN2RSLT ..... Number of Rows fetched

   DSN2RETC-Values:

        Ø  Successfull Execution
        1  Max.Number of Rows reached  (Trunc.ISPF-Table returned)
        4  No Rows returned            (Empty ISPF-Table returned)
        8  Parameter-Error
        12 ISPF- or DSNREXX-Error
    -nnnnn SQL-Error
*/
SIGNAL ON SYNTAX NAME DSN2ISPF_SYNTAX ;
Address TSO ;
DSN2RETC = '4' ;                          /* Default Returncode       */
DSN2RSLT = 'Ø' ;                          /* Default Number of Result-*/
                                          /* Rows                     */
DSN2TNAM = 'DSN2ØØØØ' ;                   /* Default ISPF-Tablename   */
"ISPEXEC VERASE (DSN2RETC,DSN2ERRM,DSN2RSLT) BOTH" ;
/************************ ErrorLevel *********************************/
"ISPEXEC VGET DSN2ELVL" ;
if rc = 8 | DSN2ELVL = '' then,           /* Set Default Error level  */
   DSN2ELVL = '1' ;
if DSN2ELVL < 'Ø' | DSN2ELVL > '2' then do ; /* Check Error level     */
```

```
   DSN2ELVL_ERR = DSN2ELVL ;
   DSN2ELVL = '1' ;
   retc = DSN2ISPF_MSG('Invalid Errorlevel: 'DSN2ELVL_ERR,'E') ;
   DSN2RETC = '8' ;
   signal DSN2ISPF_END ;
end ;
retc = DSN2ISPF_MSG('Current Errorlevel: 'DSN2ELVL,'I') ;
/************************ REXX Tracing ********************************/
"ISPEXEC VGET DSN2TRCE" ;
if rc = 8 | DSN2TRCE = '' then,           /* Set Default REXX Tracing */
   DSN2TRCE = 'O' ;
Rx_Cmd = "Trace "DSN2TRCE ;
interpret Rx_Cmd ;                        /* Activate REXX Tracing    */
retc = DSN2ISPF_MSG('Current Tracelevel: 'DSN2TRCE,'I') ;
/************************ Max.Number of Rows **************************/
"ISPEXEC VGET DSN2ROWS" ;
if rc = 8 | DSN2ROWS = '' then,           /* Set Default MaxRows      */
   DSN2ROWS = 999
if datatype(DSN2ROWS) <> 'NUM' then do ;
   retc = DSN2ISPF_MSG('Invalid DSN2ROWS-Value 'DSN2ROWS,'E') ;
   DSN2RETC = '8' ;
   signal DSN2ISPF_END ;
end ;
DSN2ROWS = trunc(DSN2ROWS) ;              /* Truncate DSN2ROWS        */
retc = DSN2ISPF_MSG('Max.Number of Rows set to: 'DSN2ROWS,'I') ;
/************************ Isolationlevel ******************************/
"ISPEXEC VGET DSN2ISOL" ;
if rc = 8 | DSN2ISOL = '' then,           /* Set Default-ISOLevel     */
   DSN2ISOL = 'UR' ;
if DSN2ISOL <> 'UR' &,
   DSN2ISOL <> 'RR' &,
   DSN2ISOL <> 'CS' &,
   DSN2ISOL <> 'RS' then do ;
   retc = DSN2ISPF_MSG('Invalid DSN2ISOL-Value 'DSN2ISOL,'E') ;
   DSN2RETC = '8' ;
   signal DSN2ISPF_END ;
end ;
retc = DSN2ISPF_MSG('DSN2ISOL Value set to: 'DSN2ISOL,'I') ;
/************************ ISPF-Table Behaviour ************************/
"ISPEXEC VGET DSN2TABB" ;
if rc = 8 | DSN2TABB = '' then,           /* Set Default              */
   DSN2TABB = 'NOTAB' ;
if DSN2TABB <> 'TAB' &,                   /* Check Values             */
   DSN2TABB <> 'NOTAB' then do ;
   retc = DSN2ISPF_MSG('Invalid DSN2TABB-Value 'DSN2TABB,'E') ;
   DSN2RETC = '8' ;
   signal DSN2ISPF_END ;
end ;
retc = DSN2ISPF_MSG('DSN2TABB-Value set to: 'DSN2TABB,'I') ;
/************************ ISPF Table Parameter ************************/
```

```
"ISPEXEC VGET DSN2ISPP" ;
if rc = 8 | DSN2ISPP = '' then,           /* Set Default              */
   DSN2ISPP = 'NOWRITE,REPLACE' ;

retc = DSN2ISPF_MSG('DSN2ISPP-Value set to: 'DSN2ISPP,'I') ;
/************************ DB2 Subsystem ID ****************************/
"ISPEXEC VGET DSN2SSID" ;
if rc = 8 | DSN2SSID = '' then do ;
   retc = DSN2ISPF_MSG('DSN2SSID-Value missing','E') ;
   DSN2RETC = '8' ;
   signal DSN2ISPF_END ;
end ;
DSN2_LOCATION = DSN2ISPF_DB2LOC(DSN2SSID) ; /* Get Location Name      */
if DSN2_LOCATION = '????' then do ;
   retc = DSN2ISPF_MSG('Invalid DSN2SSID-Value: 'DSN2SSID,'E') ;
   DSN2RETC = '8' ;
   signal DSN2ISPF_END ;
end ;

retc = DSN2ISPF_MSG('DSN2SSID set to: 'DSN2SSID,'I') ;
retc = DSN2ISPF_MSG('DSN2_LOCATION set to: 'DSN2_LOCATION,'I') ;
/************************ Select Statement ****************************/
"ISPEXEC VGET DSN2STMT" ;                 /* Determine SELECT-Stmt    */
if rc = 8 | DSN2STMT = '' then do ;
   retc = DSN2ISPF_MSG('DSN2STMT-Value missing','E') ;
   DSN2RETC = '8' ;
   signal DSN2ISPF_END ;
end ;
col_count = Ø ;                           /* Num of Cols in SEL-Stmt  */
db2_colname. = '' ;                       /* Array of Column Names    */
/************************ ISPF KEYLIST ********************************/
"ISPEXEC VGET DSN2KEYS" ;                 /* Determine ISPF Keylist   */
if rc = 8 | DSN2KEYS = '' then DSN2KEYS = '' ;
else do ;
  DSN2_KEYS = translate(DSN2KEYS,' ',',') ;
  key_count = words(DSN2_KEYS) ;
  do kc = 1 to key_count ;
     key_col = word(DSN2_KEYS,kc) ;
     if length(key_col) > 8 then,
     do;
        retc = DSN2ISPF_MSG('Invalid IspKey-Column Name: 'key_col,'E') ;
        DSN2RETC = '8' ;
        signal DSN2ISPF_END ;
     end;
     col_count = col_count + 1 ;
     db2_colname.col_count = key_col ;
  end kc ;
end ;
/************************ ISPF VAR-NAMES ******************************/
"ISPEXEC VGET DSN2VARS" ;                 /* Determine ISPF Varlist   */
```

```
if rc = 8 | DSN2VARS = '' then do ;
   retc = DSN2ISPF_MSG('DSN2VARS-Value missing','E') ;
   DSN2RETC = '8' ;
   signal DSN2ISPF_END ;
end ;
DSN2_VARS = translate(DSN2VARS,' ',',') ;
var_count = words(DSN2_VARS) ;
do vc = 1 to var_count ;
   var_col = word(DSN2_VARS,vc) ;
   if length(var_col) > 8 then,
   do;
      retc = DSN2ISPF_MSG('Invalid Isp-Column Name: 'var_col,'E') ;
      DSN2RETC = '8' ;
      signal DSN2ISPF_END ;
   end;
   col_count = col_count + 1 ;
   db2_colname.col_count = var_col ;
end vc ;
retc =,
DSN2ISPF_MSG('Number of Columns in the Result-Set: 'col_count,'I') ;
/************************ DSN2LOCL ************************************/
"ISPEXEC VGET DSN2LOCL" ;                  /* Determine DSN2LOCL-Value */
if rc = 8 | DSN2LOCL = '' then DSN2LOCL = '1' ;    /* Set default     */
else if DSN2LOCL <> 'Ø' & DSN2LOCL <> '1' then do ;
   retc = DSN2ISPF_MSG('Invalid DSN2LOCL-Value: 'DSN2LOCL,'E') ;
   DSN2RETC = '8' ;
   signal DSN2ISPF_END ;
end ;
/************************ Get local DB2-Subsystem ID ******************/
if DSN2LOCL = '1' then DSN2ISPF_LOCAL_SSID = DSN2ISPF_LOCALDB2() ;
else DSN2ISPF_LOCAL_SSID = DSN2SSID ;
if DSN2ISPF_LOCAL_SSID = '????' then do ;
   retc = DSN2ISPF_MSG('Unable to determine DSN2ISPF_LOCAL_SSID','E') ;
   DSN2RETC = '8' ;
   signal DSN2ISPF_END ;
end ;
retc = DSN2ISPF_MSG('LOCAL_SSID is: 'DSN2ISPF_LOCAL_SSID,'I') ;
/************************ Build the FETCH-Stmt ************************/
DSN2ISPF_FETCH_STMT = "FETCH C1 INTO";
do cc = 1 to col_count ;
   if cc < col_count then ,
      DSN2ISPF_FETCH_STMT = ,
      DSN2ISPF_FETCH_STMT" :"db2_colname.cc" :I"cc"," ;
   else ,
      DSN2ISPF_FETCH_STMT = ,
      DSN2ISPF_FETCH_STMT" :"db2_colname.cc" :I"cc ;
end cc ;
/************************ Generate a unique ISPF Tablename ************/
Address TSO "ISPEXEC CONTROL ERRORS RETURN";
DSN2ISPF_GENTAB:
```

```
DSN2TNAM = "DSN2"random(1,9999);
DSN2TNAM = overlay(DSN2TNAM,copies('Ø',8));
Address TSO "ISPEXEC TBQUERY "DSN2TNAM;
if rc = 12 then,
   Address TSO "ISPEXEC CONTROL ERRORS CANCEL";
else if rc = Ø then signal DSN2ISPF_GENTAB ; /* Table already exists  */
/************************ Create the ISPF Table ************************/
if DSN2KEYS = '' then,
   Address TSO "ISPEXEC TBCREATE "DSN2TNAM" NAMES ("DSN2VARS")",
               DSN2ISPP ;
else,
   Address TSO "ISPEXEC TBCREATE "DSN2TNAM" NAMES ("DSN2VARS")",
               "KEYS("DSN2KEYS")",
               DSN2ISPP ;
if rc <> Ø then do ;
   retc = DSN2ISPF_MSG('Unable to create ISPF Table:' rc,'E');
   DSN2RETC = '12' ;
   signal DSN2ISPF_END ;
end ;
/************************ Start DSNREXX Processing  *******************/
'SUBCOM DSNREXX' ;
if rc <> Ø then do;
  s_rc = RXSUBCOM('ADD','DSNREXX','DSNREXX') ;
  if s_rc <> Ø then do ;
     retc = DSN2ISPF_MSG('Unable to add DSNREXX-Environment:' s_rc,'E');
     DSN2RETC = '12' ;
     signal DSN2ISPF_END ;
  end ;
  retc = DSN2ISPF_MSG('DSNREXX-Environment added','I') ;
end;
Address DSNREXX ;
"CONNECT "DSN2ISPF_LOCAL_SSID ;
if rc <> Ø then do ;
   retc = ,
   DSN2ISPF_MSG('Error at LOCAL_SSID Attach: 'rc,'E') ;
   DSN2RETC = '12' ;
   signal DSN2ISPF_END ;
end ;
DSN2ISPF_MSG('Successfull Attach to 'DSN2ISPF_LOCAL_SSID,'I') ;
"EXECSQL CONNECT TO "DSN2_LOCATION ;
if rc <> Ø then do ;
   retc = ,
   DSN2ISPF_MSG('Error at CONNECT to' DSN2_LOCATION,'E') ;
   retc = DSN2ISPF_SQLERROR() ;
   DSN2RETC = SQLCODE ;
   signal DSN2ISPF_END ;
end;
DSN2ISPF_MSG('Successfull Connect to 'DSN2_LOCATION,'I') ;
DSN2ISPF_PKSET = 'DSNREX'||DSN2ISOL ;
"EXECSQL SET CURRENT PACKAGESET = 'DSNREXUR'" ;
```

```
if rc <> Ø then do ;
   retc = ,
   DSN2ISPF_MSG('Error at SET CURRENT PACKAGESET','E') ;
   retc = DSN2ISPF_SQLERROR() ;
   signal DSN2ISPF_END ;
end;
DSN2ISPF_MSG('Packageset 'DSN2ISPF_PKSET' will be used','I') ;
/************************ Declare Cursor ******************************/
"EXECSQL DECLARE C1 CURSOR FOR S1" ;
if rc <> Ø then do ;
   retc = ,
   DSN2ISPF_MSG('Error at DECLARE CURSOR','E') ;
   retc = DSN2ISPF_SQLERROR() ;
   signal DSN2ISPF_END ;
end;
/************************ Prepare Statement ***************************/
"EXECSQL PREPARE S1 FROM :DSN2STMT" ;
if rc <> Ø then do ;
   retc = ,
   DSN2ISPF_MSG('Error at PREPARE STATEMENT','E') ;
   retc = DSN2ISPF_SQLERROR() ;
   signal DSN2ISPF_END ;
end;
/************************ Open Cursor *********************************/
"EXECSQL OPEN C1" ;
if rc <> Ø then do ;
   retc = ,
   DSN2ISPF_MSG('Error at OPEN CURSOR','E') ;
   retc = DSN2ISPF_SQLERROR() ;
   signal DSN2ISPF_END ;
end;
/************************ Fetch Loop **********************************/
num_rows = Ø ;                          /* Number of Rows fetched        */
"EXECSQL "DSN2ISPF_FETCH_STMT ;
do while (SQLCODE = Ø & num_rows < DSN2ROWS ) ;
   num_rows = num_rows + 1 ;
   /* Substitute NULL-Values with Empty-String and strip Outputvars   */
   do cc = 1 to col_count ;
      Rx_Cmd = db2_colname.cc" = strip("db2_colname.cc",,)" ;
      interpret Rx_Cmd ;
      Rx_Cmd = "DSN2_NULLVAL = I"cc ;
      interpret Rx_Cmd ;
      if DSN2_NULLVAL < Ø then do ;
         Rx_Cmd = db2_colname.cc" = '" ;
         interpret Rx_Cmd ;
      end ;
   end cc ;
   Address TSO "ISPEXEC TBADD "DSN2TNAM ;
   if rc <> Ø then do ;
      retc = ,
```

```
      DSN2ISPF_MSG('Error at TBADD (RowNr.'num_rows')','E') ;
      signal DSN2ISPF_END ;
   end ;
   "EXECSQL "DSN2ISPF_FETCH_STMT ;
end ;
if SQLCODE = Ø then do ;
   retc = ,
   DSN2ISPF_MSG('Max. Number of Rows ('num_rows') reached','I') ;
   DSN2RETC = '1' ;
end ;
else if SQLCODE = 1ØØ then do ;
   retc = ,
   DSN2ISPF_MSG('Number of Rows fetched: 'num_rows,'I') ;
   if num_rows = Ø then do ;
      DSN2RETC = '4' ;
      if DSN2TABB = 'NOTAB' then,                 /* Erase ISPF-Table */
         Address TSO "ISPEXEC TBEND "DSN2TNAM ;
      signal DSN2ISPF_CLOSE_CURSOR ;
   end ;
   else do;
     DSN2RETC = 'Ø' ;
     DSN2ERRM = 'Normal Completion' ;
   end ;
end ;
else do ;
   retc = ,
   DSN2ISPF_MSG('Error at FETCH CURSOR','E') ;
   retc = DSN2ISPF_SQLERROR() ;
   signal DSN2ISPF_END ;
end ;
/************************ Close Cursor and Disconnect *****************/
DSN2ISPF_CLOSE_CURSOR:
"EXECSQL CLOSE C1" ;
"EXECSQL CONNECT RESET" ;
"DISCONNECT" ;
Address TSO "ISPEXEC TBTOP "DSN2TNAM ;
DSN2RSLT = num_rows ;                    /* Set Number of Rows fetched */
/************************ End Processing ******************************/
DSN2ISPF_END:
Address TSO ;
/* Clear all DSN2ISPF-Variables in ISPF-Pool                          */
"ISPEXEC VERASE (DSN2SSID,DSN2STMT,DSN2VARS,DSN2KEYS,",
"DSN2TRCE,DSN2ELVL,DSN2ROWS,DSN2ISOL,DSN2TABB,DSN2ISPP) BOTH" ;
/* Set Returncode,Errormessage and Number of Rows                     */
"ISPEXEC VPUT (DSN2RETC,DSN2ERRM,DSN2RSLT)" ;
return(DSN2TNAM) ;
/*********************************************************************/
/* DSN2ISPF_MSG: SQL-Errors                                          */
/*********************************************************************/
DSN2ISPF_SQLERROR:
```

```
DSN2RETC = SQLCODE ;
DSN2ERRM = SQLERRMC ;
retc = ,
DSN2ISPF_MSG('SQLCODE : 'DSN2RETC,'E') ;
retc = ,
DSN2ISPF_MSG(DSN2ERRM,'E') ;
return(Ø) ;
/*********************************************************************/
/* DSN2ISPF_MSG: Errors and Informations based on Errorlevel         */
/*********************************************************************/
DSN2ISPF_MSG:
parse arg Msg_String,Msg_Type ;
if DSN2ELVL = 'Ø' then return(Ø) ;         /* Return immediately      */
if DSN2ELVL = '1' & Msg_Type = 'E' then,   /* Errors only             */
   say 'DSN2ISPF => 'Msg_string ;
else if DSN2ELVL = '2' & ( Msg_Type = 'I' | Msg_Type = 'E') then,
   say 'DSN2ISPF => 'Msg_string ;
if Msg_Type = 'E' then DSN2ERRM = Msg_String ;
return(Ø) ;
/*********************************************************************/
/* DSN2ISPF_SYNTAX: REXX Errors                                      */
/*********************************************************************/
DSN2ISPF_SYNTAX:
retc = DSN2ISPF_MSG('A REXX Error has occurred ||','E') ;
retc = DSN2ISPF_MSG('Maybe wrong Tracelevel     ','E') ;
DSN2RETC = 8 ;
signal DSN2ISPF_END ;
return(Ø) ;
/*********************************************************************/
/* DSN2ISPF_DB2LOC: Maintain the following Table with your Location  */
/*                  names !!!                                        */
/*********************************************************************/
DSN2ISPF_DB2LOC:
arg conn_ssid ;
select ;
  when ssid = 'DSN'  then return('your-location-name-for-dsn') ;
  /* Specify all of your DB2-Subsystems and their location-names
     here */
  otherwise return('????') ;
end ;
return(Ø) ;
/*********************************************************************/
/* DSN2ISPF_LOCALDB2: Returns the name of a DB2 Subsystem that is    */
/*                    running on the LPAR where this REXX EXEC       */
/*                    executes                                       */
/*********************************************************************/
DSN2ISPF_LOCALDB2:
/* Retrieve SMF-ID of the executing System :                         */
cvt1    = STORAGE(1Ø,4)                    /* pointer to CVT hex     */
cvt2    = BITAND(cvt1,'7FFFFFFF'X)         /* clear high order bit   */
```

```
cvt     = C2X(cvt2)                          /* pointer to CVT char    */
cvtsmca = D2X(X2D(cvt)+X2D(C4))              /* CVTSMCA address        */
smca1   = STORAGE(cvtsmca,4)                 /* CVTSMCA value=SMCA ptr */
smca2   = BITAND(smca1,'7FFFFFFF'X)          /* clear high order bit   */
smca    = C2X(smca2)                         /* pointer to SMCA char   */
if X2D(smca) = Ø then smfid = '????' ;       /* SMF not on system      */
else do ;
   smcasid = D2X(X2D(smca)+X2D(1Ø)) ;        /* SMCASID address        */
   smfid = STORAGE(smcasid,4) ;              /* SMCASID value = SMFID  */
end ;
/* Apply one db2-subsystem for every LPAR you are running             */
select;
 when( smfid = 'SYS1' ) THEN db2sys='DSN';
 otherwise                  db2sys='????';
end ;
return(db2sys) ;
```

SAMPLE PROGRAM TO DEMONSTRATE THE USAGE OF DSN2ISPF

```
/* Rexx
*/
DSN2STMT = "SELECT DBNAME,NAME FROM SYSIBM.SYSTABLESPACE",
           "WHERE STATUS = 'A' AND DBNAME LIKE 'DBA%'",
           "ORDER BY DBNAME,NAME";

DSN2SSID =  'DSN'   ;
DSN2ROWS =  999999  ;
DSN2VARS =  "DB,TS" ;

/*  ----------------------------------------------------------------*/
/* VPUT Parameters for DSN2ISPF:                                    */
/*  ----------------------------------------------------------------*/
"ISPEXEC VPUT (DSN2SSID,DSN2STMT,DSN2ROWS,DSN2VARS)";

/*  ----------------------------------------------------------------*/
/* Call DSN2ISPF and build ISPF-Table                               */
/*  ----------------------------------------------------------------*/
tstab = DSN2ISPF() ;

/*  ----------------------------------------------------------------*/
/* VGET Return-Values / Check Return-Values                         */
/*  ----------------------------------------------------------------*/
"ISPEXEC VGET (DSN2RETC,DSN2ERRM,DSN2RSLT)";
if DSN2RETC <> Ø then do ;
    if DSN2RETC < Ø then do ;
       say _pgm time()': DSN2ISPF()-SQLSTMT IS: ' ;
       say _pgm time()': 'DSN2STMT ;
    end ;
```

```
    say 'DSN2ISPF()-RETURNCODE IS: 'DSN2RETC ;
    say 'DSN2ISPF()-MESSAGE    IS: 'DSN2ERRM ;
    if DSN2RETC > 4 | DSN2RETC < 0 then exit(12) ;
end ;

tstab_count = DSN2RSLT ;                    /* Number of Rows returned */
/*  ------------------------------------------------------------------*/
/* TBSKIP through the Result-Table                                    */
/*  ------------------------------------------------------------------*/
"ISPEXEC TBSKIP "tstab ;
do while rc = 0 ;
   say "Database: "db" Tablespace: "ts ;
   "ISPEXEC TBSKIP "tstab ;
end ;
exit(0) ;
```

Peter Adlersburg
Senior DBA
T-Systems (Austria)

Using Real Time Statistics (RTS)

This article is adapted from the forthcoming revision of Craig's book, DB2 Developer's Guide *(5th edition).*

To maintain efficient production of DB2-based systems, you must periodically monitor the DB2 objects that make up those systems. This type of monitoring is an essential component of post-implementation duties because the production environment is dynamic. Fluctuations in business activity, changes in data access patterns, or lack of attention to administrative needs can cause a system to perform inadequately. An effective strategy for monitoring DB2 objects in the production environment will catch and forestall problems before they affect performance.

One type of DB2 database object monitoring is to query the DB2 catalog tables. However, a new feature of DB2 delivers real time statistics, providing up-to-date information about DB2 database objects.

AUTONOMIC STATISTICS

Real Time Statistics (RTS) is the first step in IBM's grand plans to automate parts of DB2 database administration. Introduced after the general availability of Version 7, but before Version 8, RTS provides functionality that maintains statistics about DB2 databases on-the-fly, without having to run a utility program.

Prior to the introduction of RTS, the only way to gather statistics about DB2 database structures was by running the RUNSTATS utility. RUNSTATS collects statistical information about DB2 database objects and stores this data in the DB2 catalog. RTS, on the other hand, runs in the background and automatically updates statistics in two special tables as the data in DB2 databases is modified. Where RUNSTATS is a hands-on administrative process, RTS is hands-off.

Real Time Statistics was announced with APARs PQ48447, PQ48448, PQ46859, and PQ56256.

THE RTS TABLES

Although DB2 is always collecting RTS data, nothing is externalized until you set up the RTS database and tables to store the real time statistics. The RTS database is named DSNRTSDB and there is one table space (DSNRTSTS) with two tables:

- SYSIBM.TABLESPACESTATS – contains statistics on table spaces and table space partitions.
- SYSIBM.INDEXSPACESTATS – contains statistics on index spaces and index space partitions.

The columns in the SYSIBM.TABLESPACESTATS table are as follows:

- DBNAME – database name.
- NAME – table space name.
- PARTITION – the data set number within the table space.

For partitioned table spaces, it contains the partition number for a single partition. For non-partitioned table spaces, it contains 0.

- DBID – internal database identifier.
- PSID – internal page set identifier (for the table space).
- UPDATESTATSTIME – the timestamp when this statistics row was inserted or last updated.
- TOTALROWS – the total number of rows or LOBs in the table space or partition. Indicates the number of rows in all tables for multi-table table spaces.
- NACTIVE – the number of active pages in the table space or partition. Indicates the total number of preformatted pages in all data sets for multi-piece table spaces.
- SPACE – the amount of space (in kilobytes) that is allocated to the table space or partition. Indicates the amount of space in all data sets for multi-piece linear page sets.
- EXTENTS – the number of extents used by the table space or partition. Indicates the number of extents for the last data set for multi-piece table spaces. For a data set that is striped across multiple volumes, the value is the number of logical extents.
- LOADRLASTTIME – the timestamp when the last LOAD REPLACE was run for the table space or partition.
- REORGLASTTIME – the timestamp when the last REORG was run on the table space or partition.
- REORGINSERTS – the number of records or LOBs that have been inserted since the last REORG or LOAD REPLACE was run on the table space or partition.
- REORGDELETES – the number of records or LOBs that have been deleted since the last REORG or LOAD REPLACE on the table space or partition.
- REORGUPDATES – the number of rows that have been

updated since the last REORG or LOAD REPLACE was run on the table space or partition. Does not include LOB updates because they are implemented as deletions followed by insertions.

- REORGDISORGLOB – the number of LOBs that were inserted since the last REORG or LOAD REPLACE that are not perfectly chunked. A LOB is perfectly chunked if the allocated pages are in the minimum number of chunks.
- REORGUNCLUSTINS – the number of records that were inserted since the last REORG or LOAD REPLACE that are not well clustered with respect to the clustering index. A record is well clustered if the record is inserted into a page that is within 16 pages of the ideal candidate page.
- REORGMASSDELETE – the number of mass deletes from a segmented or LOB table space, or the number of dropped tables from a segmented table space, since the last REORG or LOAD REPLACE was run.
- REORGNEARINDREF – the number of overflow records created and relocated near the pointer record since the last REORG or LOAD REPLACE was run. For non-segmented table spaces, a page is near the present page if the two page numbers differ by 16 or less. For segmented table spaces, a page is near the present page if the two page numbers differ by SEGSIZE*2 or less.
- REORGFARINDREF – the number of overflow records created and relocated far from the pointer record since the last REORG or LOAD REPLACE was run. For non-segmented table spaces, a page is far from the present page if the two page numbers differ by more than 16. For segmented table spaces, a page is far from the present page if the two page numbers differ by more than SEGSIZE*2.
- STATSLASTTIME – the timestamp when RUNSTATS was last run on this table space or partition.
- STATSINSERTS – the number of records or LOBs that have been inserted since the last RUNSTATS was executed

on this table space or partition.

- STATSDELETES – the number of records or LOBs that have been deleted since the last RUNSTATS was executed on this table space or partition.
- STATSUPDATES – the number of records or LOBs that have been updated since the last RUNSTATS was executed on this table space or partition.
- STATSMASSDELETE – the number of mass deletes from a segmented or LOB table space, or the number of dropped tables from a segmented table space, since the last RUNSTATS was run.
- COPYLASTTIME – the timestamp of the last full or incremental image copy on the table space or partition.
- COPYUPDATEDPAGES – the number of distinct pages that have been updated since the last COPY was run.
- COPYCHANGES – the number of INSERT, UPDATE, and DELETE operations since the last COPY was run.
- COPYUPDATELRSN – the LRSN or RBA of the first update after the last COPY was run.
- COPYUPDATETIME – specifies the timestamp of the first UPDATE made after the last COPY was run.

The columns of SYSIBM.INDEXSPACESTATS are as follows:

- DBNAME – database name.
- NAME – index space name.
- PARTITION – the data set number within the index space. For partitioned index spaces, it contains the partition number for a single partition. For non-partitioned index spaces, it contains 0.
- DBID – internal database identifier.
- ISOBID – internal identifier of the index space page set descriptor.

- PSID – internal page set identifier (for the table space holding the table on which this index was created).
- UPDATESTATSTIME – the timestamp when this statistics row was inserted or last updated.
- TOTALENTRIES – the number of entries, including duplicates, in the index space or partition.
- NLEVELS – the number of levels in the index tree.
- NACTIVE – the number of active pages in the index space or partition.
- SPACE – the amount of space (in kilobytes) that is allocated to the index space or partition. Indicates the amount of space in all data sets for multi-piece linear page sets.
- EXTENTS – the number of extents used by the index space or partition. Indicates the number of extents for the last data set for multi-piece table spaces. For a data set that is striped across multiple volumes, the value is the number of logical extents.
- LOADRLASTTIME – timestamp of the last LOAD REPLACE on the index space or partition.
- REBUILDLASTTIME – timestamp of the last REBUILD INDEX on the index space or partition.
- REORGLASTTIME – timestamp of the last REORG INDEX on the index space or partition.
- REORGINSERTS – the number of index entries that have been inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE on the index space or partition.
- REORGDELETES – the number of index entries that have been deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE on the index space or partition.
- REORGAPPENDINSERT – the number of index entries that have been inserted since the last REORG, REBUILD

INDEX, or LOAD REPLACE on the index space or partition that have a key value that is greater than the maximum key value in the index or partition.

- REORGPSEUDODELETES – the number of index entries that have been pseudo-deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE on the index space or partition.
- REORGMASSDELETE – the number of times that an index or index space partition was mass deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE.
- REORGLEAFNEAR – the number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was near the location of the original page. The higher part is near the original page if the two page numbers differ by 16 or less.
- REORGLEAFFAR – the number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was far from the location of the original page. The higher part is far from the original page if the two page numbers differ by more than 16.
- REORGNUMLEVELS – the number of levels in the index tree that were added or removed since the last REORG, REBUILD INDEX, or LOAD REPLACE.
- STATSLASTTIME – timestamp of the last RUNSTATS on the index space or partition.
- STATSINSERTS – the number of index entries that have been inserted since the last RUNSTATS on the index space or partition.
- STATSDELETES – the number of index entries that have been deleted since the last RUNSTATS on the index space or partition.

- STATSMASSDELETE – the number of times that the index or index space partition was mass deleted since the last RUNSTATS.
- COPYLASTTIME – timestamp of the last full image copy on the index space or partition.
- COPYUPDATEDPAGES – the number of distinct pages that have been updated since the last COPY.
- COPYCHANGES – the number of INSERT and DELETE operations since the last COPY.
- COPYUPDATELRSN – the LRSN or RBA of the first update after the last COPY.
- COPYUPDATETIME – timestamp of the first update after the last COPY.

Each table has a unique index defined on it. Both are defined on the DBID, PSID, and PARTITION columns. The index names are:

- SYSIBM.TABLESPACESTATS_IX
- SYSIBM.INDEXSPACESTATS_IX.

WHEN ARE REAL TIME STATISTICS EXTERNALIZED?

As soon as RTS is applied (by running the proper version or maintenance level of DB2), DB2 begins to gather real time statistics. However, the RTS tables must exist in order for DB2 to externalize the real time statistics that it gathers.

Once the RTS tables have been created and started, DB2 externalizes real time statistics to the tables at the following times:

- When the RTS database is stopped, DB2 first externalizes all RTS values from memory into the RTS tables before stopping the database.
- When an individual RTS table space is stopped, DB2 first

externalizes all RTS values for that particular table space from memory into the RTS tables before stopping the database. Bear in mind, though, that the default installation uses only a single table space to store both RTS tables.

- When you issue -STOP DB2 MODE(QUIESCE), DB2 first externalizes all RTS values. Of course, if you stop using MODE(FORCE) no RTS values are externalized; instead, they are lost when DB2 comes down.
- As specified by the DSNZPARM STATSINT value. The default is every 30 minutes.
- During REORG, REBUILD INDEX, COPY, and LOAD REPLACE utility operations, DB2 externalizes the appropriate RTS values impacted by running that utility.

RTS ACCURACY

In certain situations, the RTS values may not be 100% accurate. Situations that can cause the real time statistics to be wrong include:

- A restarted utility can sometimes cause the RTS values to be wrong.
- Utility operations that leave indexes in a restrictive state, such as RECOVER pending (RECP), will cause statistics to be inaccurate.
- A DB2 subsystem failure.
- A notify failure in a data sharing environment.

To fix RTS statistics that are inaccurate, run a REORG, RUNSTATS, or COPY on the objects for which the statistics are suspect. Furthermore, if you are using DB2 utilities from a third-party vendor (ie not IBM), make sure that those utilities work with RTS. The third-party utilities should be able both to reset the RTS values and to use the RTS statistics for recommending when to run utilities.

DSNACCOR: THE RTS STORED PROCEDURE

IBM supplies a sample stored procedure called DSNACCOR, which can be used to query the RTS tables and make recommendations based on the statistics. You can use DSNACCOR to recommend when to run a REORG, take an image copy, or run RUNSTATS. Additionally, DSNACCOR can report on the data set extents of table spaces and index spaces as well as on objects in a restricted state.

You can specify parameters to indicate to DSNACCOR which table spaces and indexes to analyse, or just run it without parameters to evaluate all table spaces and index spaces in the subsystem.

Bear in mind, though, that if the RTS values are inaccurate, the recommendations made by DSNACCOR will not be correct. Also, DSNACCOR makes recommendations based on general formulas requiring user input about your maintenance policies. These recommendations might not be accurate for every installation or subsystem.

You should consider using DSNACCOR in conjunction with DB2 Control Center. Control Center provides a nice GUI interface to the parameters of DSNACCOR, making it easier to use than directly calling the procedure would be.

USING THE REAL TIME STATISTICS

The following RTS guidelines and queries can be used to help you identify maintenance and administration that needs to be carried out for database objects in your DB2 subsystems.

Checking for activity

Because real time statistics are updated in an on-going manner as DB2 operates, you can use them to see whether any activity has occurred during a specific timeframe. To determine whether any activity has happened in the past several days for a particular table space or index, use the UPDATESTATSTIME

column. Here is an example checking whether any activity has occurred in the past ten days for a table space (just supply the table space name):

```
SELECT   DBNAME, NAME, PARTITION, UPDATESTATSTIME
FROM     SYSIBM.TABLESPACESTATS
WHERE    (JULIAN_DAY(CURRENT DATE) - JULIAN_DAY(UPDATESTATSTIME)) <= 10
AND      NAME = ?;
```

Basic table space information

The RTS tables contain some good basic information about table spaces. The following query can be run to report on the number of rows, active pages, space used, number of extents, and when the COPY, REORG, LOAD REPLACE, and RUNSTATS were last run:

```
SELECT   DBNAME, NAME, PARTITION, TOTALROWS, NACTIVE,
         SPACE, EXTENTS, UPDATESTATSTIME, STATSLASTTIME,
         LOADRLASTTIME, REORGLASTTIME, COPYLASTTIME
FROM     SYSIBM.TABLESPACESTATS
ORDER BY DBNAME, NAME, PARTITION;
```

You can add a WHERE clause to this query to limit the output to only a certain database or for specific table spaces.

Pay particular attention to the timestamps indicating the last time that COPY, REORG, and RUNSTATS were run. If the date is sufficiently old, consider further investigating whether you should take an image copy, reorganize the table space, or run RUNSTATS.

Bear in mind, though, that the span of time between utility runs is not the only indicator for when to copy, reorganize, or capture statistics. For example, RUNSTATS may need to be run only once on static data; similar caveats apply to COPY and REORG when data does not change.

Reorganizing table spaces

Statistics that can help determine when to reorganize a table space include space allocated, extents, number of INSERTs, UPDATEs, and DELETEs since the last REORG or LOAD

REPLACE, number of unclustered INSERTs, number of disorganized LOBs, and number of near and far indirect references created since the last REORG.

```
SELECT   DBNAME, NAME, PARTITION, SPACE, EXTENTS,
         REORGLASTTIME, REORGINSERTS, REORGDELETES, REORGUPDATES,
         REORGINSERTS+REORGDELETES+REORGUPDATES AS TOTAL_CHANGES,
         REORGDISORGLOB, REORGUNCLUSTINS, REORGMASSDELETE,
         REORGNEARINDREF, REORGFARINDREF
FROM     SYSIBM.TABLESPACESTATS
ORDER BY DBNAME, NAME, PARTITION;
```

You might want to add a WHERE clause that limits the table spaces returned to just those that exceed a particular limit. For example:

- Table spaces having more than 20 extents:

  ```
  WHERE EXTENTS > 20
  ```

- Table spaces with more than 100K changes:

  ```
  WHERE TOT_CHANGES > 100000
  ```

- Table spaces with more than 50 far indirect references:

  ```
  WHERE REORGFARINDREF > 50
  ```

Another way to get more creative with your RTS queries is to build formulas into them to retrieve only those table spaces that need to be reorganized. For example, the following query will return only those table spaces having more than 10% of their rows as near or far indirect references:

```
SELECT   DBNAME, NAME, PARTITION, SPACE, EXTENTS
FROM     SYSIBM.TABLESPACESTATS
WHERE    (((REORGNEARINDREF + REORGFARINDREF)*100)/TOTALROWS) > 10
ORDER BY DBNAME, NAME, PARTITION;
```

Of course, you can change the percentage as you wish. After running the query you have a list of table spaces meeting your criteria for reorganization.

Examining the impact of a program

You can use the TOTALROWS column of SYSIBM.TABLESPACESTATS to determine how many rows

were affected by a particular program or process. Simply check TOTALROWS for the table space both before and after the process; the difference between the values is the number of rows impacted.

When to run RUNSTATS for a table space

There are also statistics to help in determining when RUNSTATS should be executed. Run the following query to show the number of INSERTs, UPDATEs, and DELETEs since the last RUNSTATS execution:

```
SELECT   DBNAME, NAME, PARTITION, STATSLASTTIME,
         STATSINSERTS, STATSDELETES, STATSUPDATES,
         STATSINSERTS+STATSDELETES+STATSUPDATES AS TOTAL_CHANGES,
         STATSMASSDELETE
FROM     SYSIBM.TABLESPACESTATS
ORDER BY DBNAME, NAME, PARTITION;
```

When to take an image copy for a table space

You can issue the following query to report on statistics that will help you to determine whether a COPY is required:

```
SELECT   DBNAME, NAME, PARTITION, COPYLASTTIME,
         COPYUPDATEDPAGES, COPYCHANGES, COPYUPDATELRSN,
         COPYUPDATETIME
FROM     SYSIBM.TABLESPACESTATS
ORDER BY DBNAME, NAME, PARTITION;
```

Basically, as the number of distinct updated pages and changes since the last COPY execution increases, the need to take an image copy increases. A good rule-of-thumb to follow is when the percentage of updated pages since the last COPY is more than 25% of the active pages, then it is time to COPY the table space. You can add the following WHERE clause to the above query to limit the output to only these table spaces:

```
WHERE ((COPYUPDATEDPAGES*100) / NACTIVE) > 25
```

Basic index space information

Do not forget that there are also RTS statistics gathered on indexes. The following query can be run to report on the

number of rows, active pages, space used, number of extents, and when the COPY, REORG, LOAD REPLACE, and RUNSTATS were last run:

```
SELECT   DBNAME, INDEXSPACE, PARTITION, TOTALENTRIES, NLEVELS, NACTIVE,
         SPACE, EXTENTS, UPDATESTATSTIME, LOADRLASTTIME,
REBUILDLASTTIME,
         REORGLASTTIME, STATSLASTTIME, COPYLASTTIME
FROM     SYSIBM.INDEXPACESTATS
ORDER BY DBNAME, NAME, PARTITION;
```

Reorganizing index spaces

Just like the table space stats, there are index space statistics that can be used to determine when to reorganize indexes. These statistics include the last time REBUILD, REORG, or LOAD REPLACE occurred, as well as statistics showing the number of INSERTs and DELETEs since the last REORG or REBUILD. And RTS does not skimp in the details. You get both real and pseudo DELETEs, as well as both single and mass DELETE information. RTS also tracks both the number of index levels and index page split information resulting in near and far indirect references since the last REORG, REBUILD INDEX, or LOAD REPLACE. The following query can be used to return this information:

```
SELECT   DBNAME, NAME, PARTITION,
         REORGLASTTIME, LOADRLASTTIME, REBUILDLASTTIME,
         TOTALENTRIES, NACTIVE, SPACE, EXTENTS,
         NLEVELS, REORGNUMLEVELS,
         REORGINSERTS, REORGAPPENDINSERT,
         REORGDELETES, REORGPSEUDODELETES, REORGMASSDELETE,
         REORGLEAFNAR, REORGLEAFFAR
FROM     SYSIBM.INDEXPACESTATS
ORDER BY DBNAME, NAME, PARTITION;
```

These statistics can be examined after running jobs or processes that cause heavy data modification.

Pay particular attention to the REORGAPPENDINSERT column. It contains the number of inserts into an index since the last REORG for which the index key was higher than any existing key value. If this column consistently grows, you have identified an object where data is inserted using an ascending

key sequence. Think about lowering the free space for such objects because the free space is wasted space if inserts are always done in ascending key sequence.

When to run RUNSTATS for an index space

RTS provides index space statistics to help determine when to run RUNSTATS similar to the table space statistics. Run the following query to show the number of INSERTs, UPDATEs, and DELETEs since the last RUNSTATS execution:

```
SELECT   DBNAME, NAME, PARTITION, STATSLASTTIME,
         STATSINSERTS, STATSDELETES, STATSMASSDELETE
FROM     SYSIBM.TABLESPACESTATS
ORDER BY DBNAME, NAME, PARTITION;
```

SUMMARY

Real time statistics can be used to augment your DB2 object monitoring process. Be sure to take advantage of the continuously updated RTS values to improve the administration and performance of your DB2 databases.

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

Renaming a DB2 subsystem for datasharing

OVERVIEW

If and when you start to move your DB2 subsystems to datasharing, you will need to revisit your DB2 naming standards. In a stand-alone subsystem, datasets and other objects are generally named in a way that makes it obvious they belong to the subsystem. With a datasharing group, the group and the members of the group have their own names, but the naming

should make it obvious that the members and group relate to each other. Related datasets and other objects should be named so that it's obvious whether they belong to the group (ie are shared among members) or to individual members.

All this means that moving to datasharing will involve some sort of renaming.

It may be possible to keep the name of your existing subsystem as the name of a member in the new datasharing group, or you may find that it just won't fit into any sort of naming standard and has to be changed. In either case, some renaming will need to be done to fit into a consistent naming structure.

One way round this is to create the datasharing group as a brand new empty subsystem, and to unload from the old subsystem and load into the datasharing group. There is nothing wrong with this apart from the time involved in moving large quantities of data. The renaming strategy which this article considers can process a reasonably-sized production subsystem in an hour.

Although the title of this article specifically mentions datasharing, renaming may be done for all sorts of reasons, and may not involve the whole subsystem. It may, for example, need to be done just to bring dataset names into line after a company merger, in which case a subset of the following procedures can be used.

The idea behind the process described is that virtually all of the effort required takes place in the preparation phase, so that the implementation is just a simple run-through of tasks.

WHAT NEEDS TO CHANGE?

It will be easiest to see what needs changing by considering an example. Let's say I have a DB2 subsystem called DB2P. I want to create a datasharing group called DBBG, with members called DBB1, DBB2, etc. Let's first look at DB2P and see how its name percolates through the entire subsystem.

The list below shows each item and its value, followed by any comments:

- Subsystem name – DB2P – how the subsystem is known to MVS and anything wanting to connect to it.
- IRLM subsystem name – RB2P – the MVS subsystem name for the IRLM.
- STC userids – STCDB2P – the userid for all the started tasks.
- STC proc names – DB2PMSTR, DB2PIRLM, DB2PDBM1, etc – started task names.
- DSNZPARM module – DSNZPARM – the zparm load module.
- BSDSs – DB2P.BSDSnn.
- Active logs – DB2P.LOGCOP%.Dsnn.
- Archive logs – DB2P.ARCHLOG.**.
- System VCAT – DB2P – high-level qualifier for directory and catalog tablespace datasets.
- Other VCAT DB2P – high-level qualifier for user tablespace datasets.
- Location – DB2P – the DDF location.
- VTAM node – ADB2P.

Note that the DSNZPARM module can have the same name regardless of the subsystem it belongs to. This is because, in general, each subsystem has its own SDSNEXIT load library.

The way that the other names are based on the subsystem name varies from site to site. The scheme shown has very simple relationships between the different names – given a log name, for example, it is very easy to relate it back to a subsystem.

With datasharing, there are a couple of extra names – the

group name and the group attach name and some of the names we've already looked at apply at a group level rather than at a subsystem level because the entities they describe are shared between the subsystem members.

The list below shows each item and its value, followed by any comments:

- Group Name – DBBG – name of the datasharing group.
- Group attach name – DBBG – the group name that programs connect to.
- System VCAT – DBBG – high-level qualifier for directory and catalog tablespace datasets.
- Other VCAT – DBBG – high-level qualifier for user tablespace datasets.
- Location – DBBG – the DDF location.

The other names apply at a member level. For our first group member, DBB1, we have the following items with values, and comments:

- Subsystem name – DBB1 – how the subsystem is known to MVS and anything wanting to connect to it.
- IRLM subsystem name – RBB1 – the MVS subsystem name for the IRLM.
- STC userids – STCDBB1 – the userid for all the started tasks.
- STC proc names – DBB1MSTR, DBB1IRLM, DBB1DBM1, etc – started task names.
- DSNZPARM module – DSNZPBB1 – the zparm load module.
- BSDSs – DBBG.DBB1.BSDSnn.
- Active logs – DBBG.DBB1.LOGCOP%.Dsnn.
- Archive logs – DBBG.DBB1.ARCHLOG.**.

- VTAM node – ADBB1.

Note that the BSDSs and logs have a high-level qualifier of *group-name.member-name*. This explicitly relates the group name and member name. It could equally well have just been *member-name*. Which you choose is a matter of discretion.

Calling the ZPARM module DSNZP*xxx* instead of DSNZPARM means that the different members of the group can share the SDSNEXIT library. This is generally desirable because the DSNHDECP module and authorization exits used by all the members should be the same.

THE RENAMING PROCESS

The renaming needs to be done before datasharing is enabled – the two processes can be done on different dates.

The renaming process can be split into three different stages:

- New system definition – definitions in MVS, VTAM, SMS, etc are set up as if for a brand new subsystem. This includes the DB2 installation CLIST (DSNTINST). Depending on the level of bureaucracy in your organization, definitions from departments supporting the non-DB2 side may need to be requested some time in advance.
- DB2 job preparation – this is the building of the jobs necessary to rename the DB2 datasets and make the changes within the subsystem to recognize the new names. Best done in the week preceding the rename, so that few objects will have changed.
- Implementation – what happens ‘on the night’.

I’ll cover each of these in the following sections.

NEW SYSTEM DEFINITION

Definitions need to be set up as for a new subsystem. Loosely, these are:

- MVS – subsystem names in SYS1.PARMLIB(IEFSSN*). APF authorization if any new datasets.
- RACF – RACF profiles, started task names in STARTED class, USS RACF segment.
- SMS – dataset alias for HLQ, SMS rules.
- WLM – WLM set-up.
- Network – VTAM nodes, TCP/IP ports.

Normally, when creating new definitions, the subsystem is being created in advance of its actual use, so a few glitches in the definitions can be picked up and corrected over a few days. For our process, where an existing subsystem is going to use the new definitions, there is no room for error – all the definitions must be checked and rechecked to make sure that they are all in place for the night of the rename.

You need to go through the DB2 installation CLIST (DSNTINST) as if for a new install. To cut down manual transcription of the old parameter values to the installation screens, the best way is to copy and modify the old DSNTID*xx* member (hereafter referred to as 'the TID') and use that as the input member for the CLIST.

Talking of TIDs and naming standards – the IBM standard is to call your TID DSNTID*xx*, where *xx* is chosen by you – in fact, if you don't use this naming convention, the installation CLIST will complain but will let you use your choice anyway. The DSNTID*xx* convention is woefully inadequate for any site with more than a few subsystems, especially if datasharing is being used. You really need something that can be easily matched to the subsystem it relates to. Something like TID*xxxx*, where *xxxx* is the subsystem name, does the job, and leaves you an extra character for versioning.

So, from our old system, we've got TIDDB2P. Copy this into member TIDDBB1. To save a bit of time, edit the new member, and change all instances of 'DB2P' to 'DBB1'. Then go through the installation CLIST. On the first screen, choose **INSTALL** as

the install type. For INPUT MEMBER NAME, use TIDDBB1 instead of the default, DSNTIDXA. Then just go through each panel, adjusting the names that need changing.

The names you will probably want to change are shown below giving panel name, field, and suggested value:

- DSNTIPA1 – Input Member Name – TIDDBB1
  – Output Member Name – TIDDBB1R.
- DSNTIPA2 – Catalog Alias – DBBG.
- DSNTIPH – BSDS Names – DBBG.DBB1.BSDS01/2
  – Active Log Names – DBBG.DBB1.LOGCOPY1/2
  – Archive Log Prefix – DBBG.DBB1.ARCHLOG1/2
- DSNTIPO – Parameter Module – DSNZPBB1
- DSNTIPI – IRLM Subsystem Name – RBB1
  – IRLM Proc Name – DBB1IRLM
- DSNTIPM – Subsystem Name – DBB1
  – Command prefix – -DBB1
- DSNTIPR – Location Name – DBB1
  – Network LU Name – xDBB1
- DSNTIPX – WLM proc name – DBBGWLM
  – DB2 Proc name – DBB1SPAS
  – WLM Environment – DBBGWM01.

Note that the WLM proc name has the group name as a prefix rather than the member name. The WLM environment (here DBBGWM01) may be invoked on any member, but the WLM set-up allows only one procedure name, unparameterized, to be specified for a WLM environment. Thus a group name should be used.

DB2 JOB PREPARATION

A number of jobs need to be prepared prior to implementation night so that the process itself is fairly automatic and requires little thought.

The jobs fall into groups:

- Dataset renaming – because of the large number of datasets involved, the commands have to be generated for the sake of efficiency and accuracy. Tablespaces, indexspaces, active logs, and BSDSs need to be renamed.
- DB2 Object alteration – tablespaces and indexspaces need their VCATs changed to the new qualifier. Again because of the large number of objects, most of these jobs need to be generated.
- BSDS alteration – the VCAT held within the BSDS needs changing, and the active logs need to be removed and added back in. The latter operation is labour-intensive and prone to error, so we'll look at a way to automate part of it.

The first two jobs can be done beforehand, but not too far in advance. You need to be happy that objects won't be created or dropped between your preparing the jobs and running them. Also consider that an increase in size could make a non-partitioned tablespace throw another dataset.

For the BSDS changes, most of the jobs can be prepared in advance, though one needs to be generated on the night.

DATASET RENAMING

The datasets that need renaming are the tablespaces, indexspaces, BSDSs, and active logs (archive logs do not need renaming because they will all expire over time). The high-level qualifiers will change as shown below, giving the object, the old HLQ, and the new HLQ:

- Table/index space – DB2P – DBBG
- BSDS – DB2P – DBBG.DBB1

- Active log – DB2P – DBBG.DBB1.

For each dataset to be renamed, we need an IDCAMS ALTER statement. For example, for dataset:

```
DB2P.DSNDBC.DSNDB06.SYSPKAGE.I0001.A001
```

the rename statement looks like this:

```
ALTER DB2P.DSNDBC.DSNDB06.SYSPKAGE.I0001.A001 -
  NEWNAME(DBBG.DSNDBC.DSNDB06.SYSPKAGE.I0001.A001)
```

Note that for such a rename to be possible, the DB2P and DBBG aliases must reside in the same MVS catalog.

For a reasonably-sized subsystem, there will probably be tens of thousands of datasets to be renamed. The size of these datasets doesn't matter, just the number. Obviously we need an automatic process to do it. First we need a list of datasets to process: ISPF Option 3.4 is the easiest way. The list of tablespaces for the DB2P system can be brought up by using this pattern **DB2P.DSNDBC**. This brings up a list of cluster (DSNDBC) as well as data portion (DSNDBD) datasets.

The list starts like this:

```
DSLIST - Data Sets Matching DB2P.DSNDBC                  Row 1 of 25545
Command ===>                                            Scroll ===> CSR
Command - Enter "/" to select action            Message            Volume
-------------------------------------------------------------------------
         DB2P.DSNDBC.DABRATE.SABRATE.I0001.A001                     *VSAM*
         DB2P.DSNDBC.DABRATE.SABWARM.I0001.A001                     *VSAM*
         DB2P.DSNDBC.DABRATE.XABRATE1.I0001.A001                    *VSAM*
         DB2P.DSNDBC.DABRATE.XABWARM0.I0001.A001                    *VSAM*
         DB2P.DSNDBC.DADMSDCH.SADCRGPD.I0001.A001                   *VSAM*
         DB2P.DSNDBC.DADMSDCH.SADIFCON.I0001.A001                   *VSAM*
         DB2P.DSNDBC.DADMSDCH.SADPJREG.I0001.A001                   *VSAM*
         DB2P.DSNDBC.DADMSDCH.SADSYSUB.I0001.A001                   *VSAM*
         DB2P.DSNDBC.DADMSDCH.XADCRGPD.I0001.A001                   *VSAM*
```

Note that all the cluster portions come first, followed by all the data portions. It is important that none of the datasets have been migrated by HSM. If they have, the data portions will not be listed. In addition, the rename process itself will be considerably delayed. I would expect that a production subsystem wouldn't have any migrated datasets, while a

development subsystem might.

Save the list of datasets by typing on the command line **save db2pds**. This saves the list of datasets in a dataset called userid.db2pds.datasets, which looks like this:

```
VIEW       SMITHAC.DB2PDB.DATASETS                    Columns 00001 00072
Command ===>                                             Scroll ===> CSR
****** *************************** Top of Data ***************************
000001 DB2P.DSNDBC.DABRATE.SABRATE.I0001.A001       *VSAM* VS           ?
000002 DB2P.DSNDBC.DABRATE.SABWARM.I0001.A001       *VSAM* VS           ?
000003 DB2P.DSNDBC.DABRATE.XABRATE1.I0001.A001      *VSAM* VS           ?
000004 DB2P.DSNDBC.DABRATE.XABWARM0.I0001.A001      *VSAM* VS           ?
000005 DB2P.DSNDBC.DADMSDCH.SADCRGPD.I0001.A001     *VSAM* VS           ?
000006 DB2P.DSNDBC.DADMSDCH.SADIFCON.I0001.A001     *VSAM* VS           ?
000007 DB2P.DSNDBC.DADMSDCH.SADPJREG.I0001.A001     *VSAM* VS           ?
000008 DB2P.DSNDBC.DADMSDCH.SADSYSUB.I0001.A001     *VSAM* VS           ?
000009 DB2P.DSNDBC.DADMSDCH.XADCRGPD.I0001.A001     *VSAM* VS           ?
```

To process the dataset, I wrote a REXX edit macro called ALTREN (ALTer RENname). ALTREN in turn calls two other general purpose edit macros that I had already written.

ALTREN

```
/*********************************************************************/
/*                  REXX                                             */
/* format for alter rename                                           */
/* parm1 is the from-string, parm2 is the to-string                  */
/* Alan Smith                                                        */
/*********************************************************************/
ADDRESS ISPEXEC
"ISREDIT MACRO (parm1,parm2)"
"ISREDIT (lvar,rvar) = BOUNDS"
"isredit c p'=' 45 "rvar" '' all"
lno = 1
"ISREDIT (LASTNUM) = LINENUM .ZLAST"
do while lno <= lastnum
  "ISREDIT (lStat) = XSTATUS (lno)"
  if lStat = "NX" then
    do
    "ISREDIT SHIFT ) "lno 1
    end
  lno = lno + 1
end
"isredit replines @"
"isredit x all @ 1"
```

```
"isredit c ' ' 1 ' ALTER ' all"
"isredit onend ' -'"
"isredit x all"
"isredit f @ 1 all"
"isredit c '@' 1 '  NEWNAME(' all"
"isredit onend ) "
if parm2 ¬= "" then
    do
    "isredit c '"||parm1||"' '"||parm2||"' all nx"
    "isredit reset"
    end
exit
```

ALTREN takes two parameters, the from-string and the to-string – in this case, ‘DB2P.’ and ‘DBBG.’.

Let’s consider a single tablespace consisting of a cluster with its data portion. In our dataset saved from 3.4, these entries look like this:

```
DB2P.DSNDBC.DSNDB06.SYSPKAGE.I0001.A001      *VSAM* VS             ?
DB2P.DSNDBD.DSNDB06.SYSPKAGE.I0001.A001      S2P302+VS   ?         ?
```

First of all, ALTREN changes all the characters from column 45 to the right boundary to spaces. This is because we’re interested in only the dataset names. Then the lines are shifted one column to the right to give a space in the first column. Then another edit macro, REPLINES, is called with a parameter of @. REPLINES repeats each line, and sets the first column of the repeated line to the specified parameter – @ in this case. This gives us:

```
 DB2P.DSNDBC.DSNDB06.SYSPKAGE.I0001.A001
@DB2P.DSNDBC.DSNDB06.SYSPKAGE.I0001.A001
 DB2P.DSNDBD.DSNDB06.SYSPKAGE.I0001.A001
@DB2P.DSNDBD.DSNDB06.SYSPKAGE.I0001.A001
```

Then the @ lines are excluded and the string ALTER is added to the front of the other lines. Another REXX, ONEND, is called. ONEND takes one parameter and adds this parameter to the end of all unexcluded lines. This gives us:

```
 ALTER DB2P.DSNDBC.DSNDB06.SYSPKAGE.I0001.A001 -
-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  1 Line(s) not Displayed
 ALTER DB2P.DSNDBD.DSNDB06.SYSPKAGE.I0001.A001 -
-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  1 Line(s) not Displayed
```

Then the @ lines are unexcluded and the others excluded. NEWNAME(is added to the front of these lines and ‘)’ to the end. Finally, for these lines only, **c parm1 parm2** is issued to change the old string to the new one. This gives us:

```
ALTER DB2P.DSNDBC.DSNDB06.SYSPKAGE.I0001.A001 -
  NEWNAME(DBBG.DSNDBC.DSNDB06.SYSPKAGE.I0001.A001)
ALTER DB2P.DSNDBD.DSNDB06.SYSPKAGE.I0001.A001 -
  NEWNAME(DBBG.DSNDBD.DSNDB06.SYSPKAGE.I0001.A001)
```

Which is what we wanted.

REPLINES REXX

```
/*********************************************************************/
/*                  REXX                                             */
/* Edit Macro to repeat every line in a file                         */
/* Alan Smith                                                        */
/*********************************************************************/
ADDRESS ISPEXEC
"ISREDIT MACRO (PARM)"
lno = 1
"ISREDIT (LASTNUM) = LINENUM .ZLAST"
do while lno <= lastnum
  "ISREDIT (lStat) = XSTATUS (lno)"
  if lStat = "NX" then
    do
    "ISREDIT (theLine) = LINE (lno)"
    if parm ¬= "" then
      theLine = overlay(parm,theLine)
    "ISREDIT LINE_AFTER (lno) = (theLine)"
    lno = lno + 1
    end
  lno = lno + 1
  "ISREDIT (LASTNUM) = LINENUM .ZLAST"
end
exit
```

ONEND

```
/*********************************************************************/
/*                  REXX                                             */
/* Edit macro to put data on the end of lines.                       */
/* Alan Smith                                                        */
/*********************************************************************/
ADDRESS ISPEXEC
"ISREDIT MACRO (PARM)"
```

```
lno = 1
if parm = "" then
  do
  say "ONEND needs one parameter"
  exit
  end
if left(parm,1) = "'" then
  parm = strip(parm,both,"'")
if left(parm,1) = '"' then
  parm = strip(parm,both,'"')
"ISREDIT (LASTNUM) = LINENUM .ZLAST"
do while lno <= lastnum
  "ISREDIT (lStat) = XSTATUS (lno)"
  if lStat = "NX" then
    do
    "ISREDIT (theLine) = LINE (lno)"
    pos = FindLast(substr(theLine,1,length(theLine) - 8))
    if pos > 0 then
      theLine = overlay(parm,theLine,pos)
    "ISREDIT LINE (lno) = (theLine)"
    end
  lno = lno + 1
end
exit
```

These REXXs are all general purpose. I had written them some time before for other requirements not related to DB2.

Note that when I called ALTREN, I appended a full-stop (period) to the from and to subsystem names – I used DB2P. and DBBG. instead of DB2P and DBBG. This reduces the chance of a problem where the *from* name exists within a database or tablespace name and gets changed as well as the first qualifier. It doesn't completely eliminate the problem, but if a few like this do exist, those rename statements will fail and need to be sorted out manually – this is fairly easy to do.

Follow the same process to generate jobs for the BSDS datasets and active logs.

DB2 OBJECT ALTERATION

DB2 indexes and tablespaces need to be altered. Those that are DB2-defined rather than user-defined, ie DB2 performs the DEFINE CLUSTER rather than us, need to be changed so

that DB2 can find the datasets that have had their high-level qualifier (VCAT) changed in previous steps. In fact it is the storage groups (STOGROUPs) referred to by the tablespaces and indexspaces that need to be changed. SYSIBM.SYSTABLEPART and SYSIBM.SYSINDEXPART each have a column called STORNAME, which holds the name of the storage group. The storage group details are held in SYSIBM.SYSSTOGROUP.

The significance of the primary and secondary quantities will become apparent later.

The sequence that needs to be followed is this:

- ALTER the tablespaces and indexspaces so that they do not use a storage group, ie they are user defined.
- Drop each storage group and recreate it with the new VCAT.
- ALTER the tablespaces and indexspaces to use the new storage groups.

At most sites, the number of indexspaces and tablespaces will be large, so we will need to generate the ALTERs. The number of storage groups is typically small, as object placement can generally be left up to SMS.

So we need to generate four jobs to ALTER the tablespaces and indexspaces. All the information we need is in the DB2 catalog, so we can access it using SQL. Let's look first at altering tablespaces to not use a storage group. The statement we need to generate is:

```
ALTER TABLESPACE dbname.tsname PART part USING VCAT DBBG;
```

The variable bits of the statement (italicized) can be filled in from SYSIBM.SYSTABLEPART using SQL:

```
SELECT DBNAME,TSNAME,PARTITION
FROM SYSIBM.SYSTABLEPART
WHERE STORTYPE = 'I'
```

Running this in SPUFI, we get something like:

```
--------+--------+--------+--------+------
SELECT DBNAME,TSNAME,PARTITION
    FROM SYSIBM.SYSTABLEPART
    WHERE STORTYPE = 'I'
--------+--------+--------+--------+------
DBNAME    TSNAME    PARTITION
--------+--------+--------+--------+------
DABRATE   SABRATE           0
DABRATE   SABWARM           0
DADMSDCH  SADCRGPD          0
DADMSDCH  SADIFCON          0
```

A couple of problems here, there are a lot of extra bits included with the results, column headings, etc. Also, we need to get the literals such as ALTER TABLESPACE into the results. These are general problems with generating statements, so let's digress for a moment and come up with a way round the problem.

GENERATING STATEMENTS

We've seen that SPUFI gives us a lot of headlines and general layout that we don't want. DSNTEP2 is the same. This leaves us with DSNTIAUL, the sample unload program. By running DSNTIAUL with a parameter of SQL, you can give it a general SQL statement to run, and the output from this will appear in the 'unload' dataset. Let's try the statement from the previous section and see what we get:

```
SELECT DBNAME,TSNAME,PARTITION
     FROM SYSIBM.SYSTABLEPART
      WHERE STORTYPE = 'I'
      ORDER BY DBNAME,TSNAME,PARTITION
  WITH UR
;
```

Note that I've added WITH UR to avoid taking any page locks and added an ORDER BY statement. When you browse the output dataset, you can see from the number of dots that there's a lot of hex in the output. Turning on hex display:

```
 BROWSE    SMITHAC.TSP.UNLOAD
 Command ===>
DAIACCNTSSYSLCK .............
CCCCCCDEEEEEDCD4000000000000000
```

```
4191335322823320000000000000
 ----------------------------
DAMMATCHSAMACA  .............
CCDDCECCECDCCC440000000000000
4144133821413100010000000000
 ----------------------------
DAMMATCHSAMACA  .............
CCDDCECCECDCCC440000000000000
4144133821413100020000000000
 ----------------------------
DAMMATCHSAMACA  .............
CCDDCECCECDCCC440000000000000
4144133821413100030000000000
 ----------------------------
```

I've scrolled down a few lines. You can see that the first eight characters are the database name, and the next eight are the tablespace name. Then comes the partition number, but this is not in readable format – it is a two-byte binary integer. The first row has X'00', partition zero, the next one has X'01', partition number one, etc. The column needs to be in character format. The char function will do this (char became much more flexible in DB2 Version 6). Running a slightly changed query:

```
SELECT DBNAME,TSNAME,CHAR(PARTITION)
     FROM SYSIBM.SYSTABLEPART
      WHERE STORTYPE = 'I'
      ORDER BY DBNAME,TSNAME,PARTITION
  WITH UR
;
```

gives us:

```
 BROWSE    SMITHAC.TSP.UNLOAD
 Command ===>
DAIACCNTSSYSLCK Ø     .......
CCCCCCDEEEEEDCD4F444440000000
4191335322823320000000000000
 ----------------------------
DAMMATCHSAMACA  1     .......
CCDDCECCECDCCC44F444440000000
4144133821413100100000000000
 ----------------------------
DAMMATCHSAMACA  2     .......
CCDDCECCECDCCC44F444440000000
4144133821413100200000000000
 ----------------------------
DAMMATCHSAMACA  3     .......
```

```
CCDDCECCECDCCC44F444440000000
414413382141310030000000000000
 ---------------------------
```

Now to add some of the layout. This query adds the literals we require:

```
SELECT 'ALTER TABLESPACE ',DBNAME,'.',TSNAME,' PART ',
       CHAR(PARTITION),' USING VCAT DBBG;'
     FROM SYSIBM.SYSTABLEPART
      WHERE STORTYPE = 'I'
      ORDER BY DBNAME,TSNAME,PARTITION
  WITH UR
;
```

giving the result:

```
 BROWSE    SMITHAC.TSP.UNLOAD                          Line 00000021 Col
 Command ===>                                                  Scroll =
..ALTER TABLESPACE DAIACCNT...SSYSLCK .. PART 0     .. USING VCAT DBBG;
01CDECD4ECCDCEDCCC4CCCCCCDE004EEEEDCD4004DCDE4F44444014EECDC4ECCE4CCCC5
011335903123527135041913353O1B228233200607193000000001042957O531304227E
  ---------------------------------------------------------------------
..ALTER TABLESPACE DAMMATCH...SAMACA  .. PART 1     .. USING VCAT DBBG;
01CDECD4ECCDCEDCCC4CCDDCECC004ECDCCC44004DCDE4F44444014EECDC4ECCE4CCCC5
011335903123527135041441338O1B214131000607193010000001042957O531304227E
  ---------------------------------------------------------------------
..ALTER TABLESPACE DAMMATCH...SAMACA  .. PART 2     .. USING VCAT DBBG;
01CDECD4ECCDCEDCCC4CCDDCECC004ECDCCC44004DCDE4F44444014EECDC4ECCE4CCCC5
011335903123527135041441338O1B214131000607193020000001042957O531304227E
  ---------------------------------------------------------------------
..ALTER TABLESPACE DAMMATCH...SAMACA  .. PART 3     .. USING VCAT DBBG;
01CDECD4ECCDCEDCCC4CCDDCECC004ECDCCC44004DCDE4F44444014EECDC4ECCE4CCCC5
011335903123527135041441338O1B214131000607193030000001042957O531304227E
  ---------------------------------------------------------------------
```

There are a few problems here. Each of the literals has a two-byte count on the front – this is because it's a VARCHAR. The partition number has some trailing spaces. Although it's not an obvious problem here, if the database name is shorter than eight characters, there will be one or more spaces between it and the full-stop (period) separating it from the tablespace name.

Two DB2 functions, concat and strip come to our rescue. Strip removes trailing and leading spaces by default, and concat joins strings together. For readability, a double vertical bar, ||, can be used instead of concat, although it can be difficult to

find on some keyboards. In this version of the query, I've used strip and concat to remove extra spaces and joined all the strings together:

```
SELECT 'ALTER TABLESPACE '                          ||
        STRIP(DBNAME)||'.'||STRIP(TSNAME)           ||
        ' PART ' || STRIP(CHAR(PARTITION))          ||
        ' USING VCAT DBBG;'
     FROM SYSIBM.SYSTABLEPART
      WHERE STORTYPE = 'I'
      ORDER BY DBNAME,TSNAME,PARTITION
  WITH UR
;
```

Since everything is now one long varchar, there is just one two-byte count at the start of the row:

```
 BROWSE    SMITHAC.TSP.UNLOAD                          Line 0000
 Command ===>
..ALTER TABLESPACE DAIACCNT.SSYSLCK PART Ø USING VCAT DBBG;....
Ø3CDECD4ECCDCEDCCC4CCCCCCDE4EEEEDCD4DCDE4F4EECDC4ECCE4CCCC50000
Ø913359031235271350419133538228233207193000429570531304227E0000
  ---------------------------------------------------------------
..ALTER TABLESPACE DAMMATCH.SAMACA PART 1 USING VCAT DBBG;.....
Ø3CDECD4ECCDCEDCCC4CCDDCECC4ECDCCC4DCDE4F4EECDC4ECCE4CCCC500000
Ø8133590312352713504144133 8B2141310719301042957053130422 7E00000
  ---------------------------------------------------------------
..ALTER TABLESPACE DAMMATCH.SAMACA PART 2 USING VCAT DBBG;.....
Ø3CDECD4ECCDCEDCCC4CCDDCECC4ECDCCC4DCDE4F4EECDC4ECCE4CCCC500000
Ø8133590312352713504144133 8B2141310719302042957053130422 7E00000
  ---------------------------------------------------------------
..ALTER TABLESPACE DAMMATCH.SAMACA PART 3 USING VCAT DBBG;.....
Ø3CDECD4ECCDCEDCCC4CCDDCECC4ECDCCC4DCDE4F4EECDC4ECCE4CCCC500000
Ø8133590312352713504144133 8B2141310719303042957053130422 7E00000
  ---------------------------------------------------------------
```

To get rid of the counts and the trailing X'00's, use char again, this time with a second parameter, which states the length of the resultant string:

```
SELECT CHAR(
            'ALTER TABLESPACE '                 ||
            STRIP(DBNAME)||'.'||STRIP(TSNAME)   ||
            ' PART ' || STRIP(CHAR(PARTITION))  ||
            ' USING VCAT DBBG;'
           ,80)
       FROM SYSIBM.SYSTABLEPART
       WHERE STORTYPE = 'I'
       ORDER BY DBNAME,TSNAME,PARTITION
```

```
  WITH UR
;
```

giving:

```
 BROWSE    SMITHAC.TSP.UNLOAD                          Line 0000
 Command ===>
ALTER TABLESPACE DAIACCNT.SSYSLCK PART 0 USING VCAT DBBG;
ALTER TABLESPACE DAMMATCH.SAMACA PART 1 USING VCAT DBBG;
ALTER TABLESPACE DAMMATCH.SAMACA PART 2 USING VCAT DBBG;
ALTER TABLESPACE DAMMATCH.SAMACA PART 3 USING VCAT DBBG;
```

Almost there, but there is an additional complication: the PART clause should be present only for partitions of a partitioned tablespace. So, if the partition is zero, the PART clause should simply not be there. We can achieve this fairly easily using a CASE statement:

```
SELECT CHAR(
            'ALTER TABLESPACE '                ||
            STRIP(DBNAME)||'.'||STRIP(TSNAME)  ||
            CASE WHEN PARTITION = 0
                THEN ' '
            ELSE
                ' PART ' || STRIP(CHAR(PARTITION))
            END                                ||
            ' USING VCAT DBBG;'
           ,80)
       FROM SYSIBM.SYSTABLEPART
       WHERE STORTYPE = 'I'
       ORDER BY DBNAME,TSNAME,PARTITION
  WITH UR
;
```

This puts out a space for a partition number of zero, otherwise the PART clause is as before:

```
 BROWSE    SMITHAC.TSP.UNLOAD                          Line
 Command ===>
ALTER TABLESPACE DAIACCNT.SSYSLCK  USING VCAT DBBG;
ALTER TABLESPACE DAMMATCH.SAMACA PART 1 USING VCAT DBBG;
ALTER TABLESPACE DAMMATCH.SAMACA PART 2 USING VCAT DBBG;
ALTER TABLESPACE DAMMATCH.SAMACA PART 3 USING VCAT DBBG;
```

The SQL to generate the statements for the indexes is very similar. It needs to extract the index name and qualifier from SYSIBM.SYSINDEXPART:

```
SELECT CHAR(
```

```
            'ALTER INDEX '                          ||
            STRIP(IXCREATOR)||'.'||STRIP(IXNAME)  ||
            CASE WHEN PARTITION = Ø
                THEN ' '
            ELSE
                ' PART ' || STRIP(CHAR(PARTITION))
            END                                    ||
            ' USING VCAT DBBG;'
           ,8Ø)
       FROM SYSIBM.SYSINDEXPART
       WHERE STORTYPE = 'I'
       ORDER BY IXCREATOR,IXNAME,PARTITION
  WITH UR
;
```

giving the following output:

```
ALTER INDEX NUAI.XAIITADJØ1  USING VCAT DBBG;
ALTER INDEX NUAI.XAIITADJØ2  USING VCAT DBBG;
ALTER INDEX NUAI.XAIITADJØ3  USING VCAT DBBG;
ALTER INDEX NUAM.XAMACA1 PART 1 USING VCAT DBBG;
ALTER INDEX NUAM.XAMACA1 PART 2 USING VCAT DBBG;
ALTER INDEX NUAM.XAMACA1 PART 3 USING VCAT DBBG;
```

DSNTIAUL still seems to put a trailing X'00' at the end of the line. In any case, you need to copy the statements into an FB, 80-column dataset to execute them – this will get rid of the trailing X'00's.

RECAP

That was a long section, so let's just remember where we've got to. We've just generated some SQL ALTER statements to change indexes and tablespaces from using a DB2 storage group to being user defined. At this point on the implementation night, we'd drop the storage groups and recreate them, and then change the tablespaces and indexes back to use the storage groups they were using before. So we need to generate statements to ALTER the tablespaces and indexes back to use storage groups.

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

Alan Smith
Norwich Union (UK)

Computer Associates has announced its continued support for DB2 Universal Database for z/OS with a suite of Unicenter database products that help customers reduce the cost and complexity of managing large-scale DB2 environments.

Unicenter Database Management R11 for DB2 UDB for z/OS V8, currently in beta release, features 33 tightly integrated products that help protect DB2 data, improve response time, and increase system availability. It comprises three suites – Unicenter Database Performance Management, Unicenter Database Backup and Recovery, and Unicenter Database Administration.

Similarly, NEON Systems says its Shadow mainframe integration products have been tested for compatibility with DB2 UDB for z/OS Version 8.

Shadow Version 5.1 enables customers to use new DB2 features including: support for extended limits for names, columns, and tables; 2 MB, BLOCK FETCH/INSERT SQL and INSERT within a SELECT statements; performance enhancements such as DYNAMIC SQL cacheing; 64-bit virtual storage support; and the extension of Shadow systems management to support GET DIAGNOSTICS facility.

For further information contact:
Computer Associates, One CA Plaza, Islandia, CA 11749, USA.
Tel: (631) 342 5224.
URL: http://www3.ca.com/press/PressRelease.asp?CID=57232.
NEON Systems, 14100 Southwest Freeway, Suite 500, Sugar Land, TX 77478, USA.
Tel: (281) 491 4200.
URL: http://neonsys.com/Shadow/shadow_event_publisher.asp.

* * *

UFD Solutions has announced DB2RCF (DB2 Resources Checking Facility), which monitors and checks z/OS DB2 database objects.

DB2RCF allows users to check online for the availability of specific DB2 tables. Plus it informs users and/or applications when DB2 objects become unavailable, giving the reason(s) for the unavailability.

During periodic checks of a DB2 tables availability, latent restrictions at the tablespace and indexspace level (COPYP, RECP, CHKP, etc) as well as lost connections (DDF/DRDA), stopped utilities, etc, will be detected.

For further information contact:
UFD AG Schweiz, Arnold Böcklin-Strasse 29, CH-4011 Basel, Switzerland.
Tel: +41 61 271 65 50.
URL: http://www.ufd.ch

* * *

Sites with a mixture of databases that want to standardize on DB2 will be interested in AdventNet's SwisSQL product suite. These automated database migration tools enable the migration of stored procedures implemented on one database to another database. It supports conversion of procedures, functions, triggers, cursors, SQLs, and other constructs of the stored procedure language. The tool automates up to 95% of the manual tasks associated with stored procedure migration.

Oracle, SQL Server, and Sybase can be migrated to DB2.

For further information contact:
AdventNet, 5645 Gibralter Drive, Pleasanton, CA 94588, USA.
Tel: (925) 924 9500.
URL: http://www.swissql.com/data-migration.html.