122 DB2 update

December 2002

In this issue

DB2 Update

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

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

Subscriptions and back-issues
A year's subscription to *DB2 Update*, comprising twelve monthly issues, costs £255.00 in the UK; $380.00 in the USA and Canada; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1999 issue, are available separately to subscribers for £22.50 ($33.75) each including postage.

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

Editor
Trevor Eddolls

Contributions
When Xephon is given copyright, articles published in *DB2 Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

Script for creating insert statements for all records in a table

The procedure usp_CreateInserts in the file usp_CreateInserts_DB2.sql is used to generate insert scripts for all the records in a table. Very often we are required to attach demo data/system table scripts in the ER diagram. Using this stored procedure, we can fetch data from the development server and attach it in the ER diagram for further distribution or load it onto the demo server/production server.

This stored procedure expects three parameters – schema name, table name, and p_called_from. (This parameter should be ‘T’ to return a result set. If this parameter is ‘D’ it will insert the INSERT statements in a global temporary table temp_inserts. This is used by another procedure to generate inserts for all tables in a database.)

The procedure internally calls another procedure, usp_GenerateInserts, that generates a string containing a select query, which, when executed, will produce insert statements containing data for the table.

USP_CREATEINSERTS_DB2.SQL

```
CREATE PROCEDURE usp_CreateInserts (  IN p_schema_name VARCHAR(128),
                       IN p_table_name  VARCHAR(4Ø),
                       IN p_called_from CHAR(1)
                          )
RESULT SETS 1
LANGUAGE SQL
/************************************************************************
     NAME:         usp_CreateInserts.sql
     DESCRIPTION:  This script internally calls usp_GenerateInserts
                   stored procedure which
                   creates a SELECT statement. This Select query when
                   executed generates INSERT
                   statements for the existing data in the given table.
                   The parameter p_called_from
                   should be 'D' when called from an SP which expects
                   output in temp table. It
                   should be any other value, such as 'T', when called
                   individually or when the expected
```

```
                    output is a resultset in a cursor.

     CALLS:        usp_GenerateInserts
*************************************************************************/

P1: BEGIN
     DECLARE strSelect varchar(8000);
        DECLARE v_str   VARCHAR(12000);
     DECLARE c1 CURSOR WITH RETURN TO CALLER FOR s1;

     IF (1 = 0) THEN
          DECLARE GLOBAL TEMPORARY TABLE temp_inserts
          (
            col_id  INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY(START
WITH 1, INCREMENT BY 1),
            schema_name VARCHAR(128),
            table_name  VARCHAR(128),
            strInsert   VARCHAR(8000)
          ) WITH REPLACE NOT LOGGED;
     END IF;

     /* Call the proc which generates the Select stmt. */
     CALL usp_GenerateInserts (p_schema_name, p_table_name, strSelect);

     /* If called from the proc which generates scripts for all
        tables then p_called_from is D.
        Hence we insert into the temp table. Otherwise if it is
        called from anywhere else we
        return a resultset. */
     IF (p_called_from = 'D') THEN
—         SET v_str = 'INSERT INTO SESSION.temp_inserts (schema_name,
table_name, strInsert) SELECT ' || '''' || p_schema_name || '''' || ','
|| '''' || p_table_name || '''' || ',' || strSelect ;
                 SET v_str = 'INSERT INTO SESSION.temp_inserts
(strInsert) ' || strSelect ;
          PREPARE s1 FROM v_str;
          EXECUTE s1;

                 UPDATE SESSION.temp_inserts
                 SET schema_name = p_schema_name,
                     table_name = p_table_name
                 WHERE schema_name IS NULL;

/*                 SET v_str = 'SELECT * FROM SESSION.temp_inserts';
          PREPARE s1 FROM v_str;
          OPEN c1;
*/
     ELSE
          PREPARE s1 FROM strSelect;
          OPEN c1;
     END IF;
END P1

/*
drop table p1

create table p1
(col1 INTEGER NOT NULL,
col2 VARCHAR(15),
col3 INTEGER)

describe table p1

insert into p1 values (1, 'ones story', null)
insert into p1 values (2, null , null)
insert into p1 values (3, 'one more story', null)

SELECT * FROM P1

DELETE FROM P1
*/
```

USP_GENERATEINSERTS_DB2.SQL

```
CREATE PROCEDURE usp_GenerateInserts
(    IN p_schema_name   VARCHAR(128),
     IN p_table_name    VARCHAR(128),
     OUT str            VARCHAR(8ØØØ)

)
LANGUAGE SQL
/**********************************************************************
     NAME:          usp_GenerateInserts.sql
     DESCRIPTION:   This script creates a SELECT statement for the
                    calling proc. This Select stmt when executed by
                    the calling proc generates INSERT statements
                    for the existing data in the given table.

     CALLS:         None
***********************************************************************/
P1: BEGIN

     DECLARE columnname VARCHAR(4Ø);
     DECLARE columntype INT;
     DECLARE fetchsts   INT;
–    DECLARE str        VARCHAR(8ØØØ);
     DECLARE strValuePart    VARCHAR(8ØØØ);
        DECLARE strInsertPart   VARCHAR(4ØØØ);
     DECLARE not_found CONDITION FOR SQLSTATE 'Ø2ØØØ';

        DECLARE cur_t1 CURSOR FOR
```

```
         SELECT A.colname, B.typeid
         FROM syscat.columns A, syscat.datatypes B
         WHERE A.tabname = UCASE(p_table_name)
         AND   A.tabschema = UCASE(p_schema_name)
         AND   B.typeschema = A.typeschema
         AND   B.typename = A.typename;

         DECLARE CONTINUE HANDLER FOR not_found SET fetchsts = 1;

     SET fetchsts = Ø;

—    SET NOCOUNT ON

     SET  strInsertPart = '''INSERT INTO ' || p_schema_name || '.' ||
p_table_name || '(';
         SET strValuePart = '';

     OPEN cur_t1;
     FETCH cur_t1 INTO columnname, columntype;

     WHILE fetchsts  = Ø
         DO
          /* If the datatype is of type char, Varchar, Long Varchar  */
                SET strInsertPart = strInsertPart || columnname;
          IF columntype in (52, 56, 6Ø) THEN
               SET strValuePart = strValuePart || ''' || case when '
||columnname || ' is null then '|| '''NULL''' || ' else ' || '''' ||
'''' || '''' || '''' || ' || ' || columnname ||  ' || ' || '''' || ''''
|| '''' || '''' || ' end || ' || '''';

          /* If datatype is of type Date, Time or TimeStamp */
          ELSEIF columntype in (1ØØ, 1Ø4, 1Ø8) THEN
               SET strValuePart = strValuePart || ''' || case when '
||columnname || ' is null then '|| '''NULL''' || ' else ' || '''' ||
'''' || '''' || '''' || ' || CAST(' || columnname ||  ' AS VARCHAR(2Ø))
|| ' || '''' || '''' || '''' || '''' || ' end || ' || '''';

          /* If datatype is of type Double */
          ELSEIF columntype in (8) THEN
               SET strValuePart = strValuePart || ''' || case when ' ||
columnname || ' is null then ''NULL'' else CHAR(' || columnname || ' )
end || '  || '''';

          /* If datatype is of any other type than the above mentioned
types  */
          ELSE
               SET strValuePart = strValuePart || ''' || case when ' ||
columnname || ' is null then ''NULL'' else CHAR(' || columnname || ' )
end || '  || '''';
          END IF;
          FETCH cur_t1 INTO columnname, columntype;

          IF fetchsts = Ø THEN
               SET strValuePart = strValuePart || ', ';
                         SET strInsertPart = strInsertPart || ',';
          END IF;
     END WHILE;

     CLOSE cur_t1;

     SET strValuePart = strValuePart || ')'' ';
         SET strInsertPart = strInsertPart || ') VALUES (';
         SET str = strInsertPart || strValuePart;
     SET str = 'SELECT ' || str || ' FROM ' || p_schema_name || '.' ||
p_table_name;

END P1

/*
to run this alone, comment the output parameter line and uncomment print
@str
and DECLARE @str statements
exec usp_GenerateInserts 'P1'
*/
```

Ramanath N Shanbhag (India) © Xephon 2002

Utilities to extract and update access path statistics

REQUIREMENT

There was a requirement at our site to analyse the effect of running RUNSTATS on about 50% of the tables for which RUNSTATS had not been run for over a year. The constraint was that the catalog statistics should not be updated by running RUNSTATS. Since the number of tables and programs was significant, special techniques were required to achieve this in an efficient manner.

The approach to the problem and the utilities used to resolve it will be presented here.

APPROACH

The approach was to utilize a test database system identical to the production database and populate that with two sets of statistics, bind the programs with each set, and populate the PLAN_TABLE. The first set should reflect the statistics as present in the production system (and is similar to migrating existing statistics). The second set would have to reflect the statistics that would be generated by running RUNSTATS (and is similar to generating future statistics). The RUNSTATS dbname.tsname TABLE(ALL) INDEX(ALL) SHRLEVEL CHANGE UPDATE NONE REPORT ONLY options can be utilized for this purpose.

ASSUMPTIONS

The test system resembles production in terms of DDL structure, and RUNSTATS has been run at least once at the table level for all the tables.

TABLES AND COLUMNS THAT ARE USED FOR ACCESS PATHS

In every table updated by RUNSTATS, the STATSTIME is not used for access path analysis. However, we update that in our utilities in order to establish a baseline and for verification. According to the Version 5.1 *Administration Guide* the following are the tables and the columns that are used for access path analysis.

SYSCOLDIST

Columns: COLVALUE, FREQUENCYF, TYPE, CARDF, COLGROUPCOLNO, NUMCOLUMNS.

We generate DELETE and INSERT statements to change the catalog statistics for this table alone, whereas we generate UPDATE statements for the rest of the tables.

Further, the FREQUENCYF and COLVALUE are the only values that we are really concerned with because all other values will take defaults as indicated in the *Administration Guide*.

SYSCOLUMNS

Columns: COLCARDF, HIGH2KEY, LOW2KEY

SYSINDEXES

Columns: CLUSTERING, CLUSTERRATIO, FIRSTKEYCARDF, FULLKEYCARDF, NLEAF, NLEVELS

Of these, CLUSTERING is not updated because it indicates whether the index is defined as a clustering index or not. Since the production and test systems are expected to be identical, we expect this to be identically defined.

SYSINDEXPART

Column: LIMITKEY

The same reasoning holds for this column as for CLUSTERING under *SYSINDEXES* above.

(If the definitions are not identical, the objects may be recreated with the same production definition and a RUNSTATS must be run at the complete tablespace level before the statistics are updated.)

SYSTABLES

Columns: CARDF, EDPROC, NPAGES, PCTROWCOMP

We are not concerned about EDPROC because it is expected to be the same as test.

SYSTABLESPACE

Column: NACTIVE

SYSTABSTATS

Columns: CARD, NPAGES.

Migrating existing statistics

The easiest option would be to use a product like RC-Migrator in CA-DB2 (formerly Platinum) and migrate the statistics to the test system from the production catalog. If such tools were not available, we could use the following SQLs and extract the production system catalog statistics and generate update statements for the test system catalog.

The SQLs are to be used as input to a DSNTIAUL or an UNLOAD job to generate the UPDATE statements. The generated statements are parsed to issue appropriate break points so as to get an easily readable SQL. Only one parser will be shown here and that will be for parsing the SYSTABLES UPDATE statements. Other parsers may be built easily as appropriate.

A sample DSNTIAUL JCL is shown in OUTPUT1.

The parser is shown in OUTPUT3. The input to the parser is the output from the unload job for SYSTABLES. Before running the parser against the output file, it is necessary to edit the file and remove any unprintable characters. Since each unload step will produce a different output structure, we need to have different parsers for each of them.

The SQL to be used is shown in OUTPUT2.

Generating future statistics

This utilizes the REPORT feature of the RUNSTATS utility. The sample JCL shown in OUTPUT4 has two job steps. The first step is a regular RUNSTATS for two tablespace objects with the control cards as shown below:

```
RUNSTATS TABLESPACE    dbname.tsname
  TABLE(ALL INDEX(ALL) SHRLEVEL CHANGE
  UPDATE NONE REPORT ONLY.
```

The output of the RUNSTATS utility is fed into a REXX utility, which parses the same and generates the necessary UPDATE statements for the statistics update. The statements for

SYSCOLDIST are a combination of DELETE and INSERT statements.

CONCLUSION

Once the UPDATE statements have been generated, it is a matter of executing them and re-binding the necessary package collections. After binding them, the differences in access path may be identified using any suitable approach.

The utilities shown here were developed for Version 5.1. In Version 7.0, there is one more table, SYSLOBSTATS, which has columns that are used for access path determination. If you do not have LOBs then it is not required. Also, the SYSTABLE column NPAGES has a sibling in NPAGESF; the SYSTABLESPACE column NACTIVE has been replaced by NACTIVEF; the SYSINDEXES column CLUSTERRATIO is replaced by CLUSTERRATIOF.

The utilities shown here may also be used in situations where we need to populate test databases with production database statistics to study performance. Also, after refreshing a test database with production data, it may be preferable just to update the test database system catalog with production database catalog statistics.

OUTPUT1 – SAMPLE UNLOAD JOB

```
//ACCPUNLD JOB (Account info),'ACCESSPATH UNLD',
//*     rest of job card
//*       rest of job card
//*
//*****************************************************************
//*     ACCPUNLD  -  UNLOAD AND CREATE SPUFI FOR UPDATING ACCESS
//*                  PATHS. JOB HAS 7 STEPS.
//*     QUERIES THE DB2 CATALOG TABLESPACES AND UNLOADS DATA
//*  AS SQL STATEMENTS
//*****************************************************************
//*
//***********************************
//** UNLOAD FOR SYSTABLES          **
//***********************************
//S2ØUL EXEC PGM=IKJEFTØ1,DYNAMNBR=2Ø,COND=(4,LT),REGION=4ØM
```

```
//STEPLIB  DD  DISP=SHR,DSN=XXXX.DBP5.DSNEXIT
//         DD  DISP=SHR,DSN=XXXX.DBP5.DSNLOAD
//DSNTRACE DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSRECØØ DD  STORCLAS=SCSTD,SPACE=(CYL,(5,5),RLSE),DISP=(NEW,CATLG),
//             DSN=PREFIX.ACCPTHU2.DATA
//SYSPUNCH DD  UNIT=SYSDA,SPACE=(8ØØ,(15,15),RLSE),DISP=(NEW,CATLG),
//             DSN=PREFIX.ACCPTHU2.CNTL
//SYSTSIN  DD  *            z
 DSN SYSTEM(DBP5)
 RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB41) PARM('SQL')
 END
/*
//SYSIN  DD  *
   SELECT 'UPDATE SYSIBM.SYSTABLES SET CARDF='
     CONCAT  DIGITS(DECIMAL(CARDF,31,Ø))
     CONCAT',NPAGES='CONCAT DIGITS(NPAGES)
     CONCAT',PCTROWCOMP='CONCAT DIGITS(PCTROWCOMP)
     CONCAT',STATSTIME=''' CONCAT CHAR(STATSTIME)
     CONCAT ' WHERE NAME='''CONCAT NAME
     CONCAT ''' AND CREATOR ='''CONCAT CREATOR CONCAT''';'
     FROM SYSIBM.SYSTABLES
     WHERE DBNAME LIKE 'ABC%'
        ;
/*
//
```

OUTPUT2 – SQL TO GENERATE UPDATE STATEMENTS

The punctuation is very critical and must be followed precisely. Note that there are only single quotes and no double quotes; what appears to be a double quote is really two single quotation marks. Three successive single quotes are valid and are correct usage.

```
SYSTABLESPACE:
SELECT DISTINCT 'UPDATE SYSIBM.SYSTABLESPACE SET NACTIVE='
   CONCAT DIGITS(NACTIVE)
   CONCAT', STATSTIME ='''CONCAT CHAR(TS.STATSTIME)
   CONCAT ''' WHERE NAME= '''CONCAT TS.NAME
   CONCAT ''' AND DBNAME = '''CONCAT TS.DBNAME
   CONCAT ''' AND CREATOR = '''CONCAT TS.CREATOR CONCAT''';'
   FROM SYSIBM.SYSTABLESPACE TS, SYSIBM.SYSTABLES TBL
   WHERE TS.NAME = TSNAME
     AND TBL.NAME LIKE '%'
     AND TS.NAME LIKE 'TS%'
     AND TS.DBNAME LIKE 'ABC%'  ;

SYSTABLES:
SELECT 'UPDATE SYSIBM.SYSTABLES SET CARDF='
  CONCAT  DIGITS(DECIMAL(CARDF,31,Ø))
  CONCAT',NPAGES='CONCAT DIGITS(NPAGES)
  CONCAT',PCTROWCOMP='CONCAT DIGITS(PCTROWCOMP)
  CONCAT',STATSTIME=''' CONCAT CHAR(STATSTIME)
  CONCAT ''' WHERE NAME='''CONCAT NAME
  CONCAT ''' AND CREATOR ='''CONCAT CREATOR CONCAT''';'
  FROM SYSIBM.SYSTABLES
  WHERE DBNAME LIKE 'ABC%' ;

SYSINDEXES:
SELECT 'UPDATE SYSIBM.SYSINDEXES SET FIRSTKEYCARDF='
  CONCAT DIGITS(DECIMAL(FIRSTKEYCARDF,31,Ø))
  CONCAT ',FULLKEYCARDF='CONCAT DIGITS(DECIMAL(FULLKEYCARDF,31,Ø))
  CONCAT',NLEAF='CONCAT DIGITS(NLEAF)
  CONCAT',NLEVELS='CONCAT DIGITS(NLEVELS)
  CONCAT',CLUSTERRATIO='CONCAT DIGITS(CLUSTERRATIO)
  CONCAT' WHERE NAME='''CONCAT NAME
  CONCAT ''' AND CREATOR ='''CONCAT CREATOR CONCAT''';'
  FROM SYSIBM.SYSINDEXES
  WHERE DBNAME LIKE 'ABC%'
    AND CREATOR LIKE 'IJK%' ;

SYSCOLUMNS:
SELECT 'UPDATE SYSIBM.SYSCOLUMNS SET COLCARDF='
  CONCAT DIGITS(DECIMAL(COLCARDF,31,Ø))
  CONCAT',HIGH2KEY=X''' CONCAT HEX(HIGH2KEY)
  CONCAT''',LOW2KEY=X''' CONCAT HEX(LOW2KEY)
  CONCAT''',STATSTIME=''' CONCAT CHAR(STATSTIME)
  CONCAT''' WHERE TBNAME='''CONCAT TBNAME
  CONCAT  ''' AND COLNO=' CONCAT DIGITS(COLNO)
  CONCAT ' AND NAME ='''CONCAT NAME
  CONCAT ''' AND TBCREATOR ='''CONCAT TBCREATOR CONCAT''';'
  FROM SYSIBM.SYSCOLUMNS
  WHERE TBNAME LIKE '%'
    AND TBCREATOR LIKE 'IJK%';

SYSCOLDIST:  (Generating Deletes followed by Inserts)
SELECT 'DELETE FROM SYSIBM.SYSCOLDIST '
  CONCAT 'WHERE TBOWNER = '''CONCAT TBOWNER
  CONCAT ''' AND TBNAME = '''CONCAT TBNAME
  CONCAT ''' AND NAME = '''CONCAT NAME CONCAT ''';'
  FROM SYSIBM.SYSCOLDIST
  WHERE TBNAME LIKE '%'
    AND TBOWNER LIKE  'IJK%' ;

SELECT 'INSERT INTO SYSIBM.SYSCOLDIST '
```

```
  CONCAT '(FREQUENCY, STATSTIME, IBMREQD, TBOWNER, '
  CONCAT 'TBNAME, NAME, COLVALUE'
  CONCAT ', TYPE, CARDF, COLGROUPCOLNO, NUMCOLUMNS, FREQUENCYF)'
  CONCAT ' VALUES(Ø,'''CONCAT CHAR(STATSTIME)
  CONCAT ''',''N'',''' CONCAT TBOWNER
  CONCAT ''',''' CONCAT TBNAME
  CONCAT ''',''' CONCAT NAME
  CONCAT ''',X''' CONCAT HEX(COLVALUE)
  CONCAT ''',''F'','
  CONCAT '-Ø.1E+Ø1'
  CONCAT ','' '''
  CONCAT ',1,'
  CONCAT DIGITS(DECIMAL(FREQUENCYF,31,Ø))
  CONCAT ');'
  FROM SYSIBM.SYSCOLDIST
  WHERE TBNAME LIKE '%'
    AND TBOWNER LIKE  'ABC%' ;

SYSTABSTATS:
SELECT 'UPDATE SYSIBM.SYSTABSTATS SET CARD='
  CONCAT DIGITS(CARD)
  CONCAT ',NPAGES='CONCAT DIGITS(NPAGES)
  CONCAT' WHERE DBNAME='''CONCAT DBNAME
  CONCAT''' AND TSNAME='''CONCAT TSNAME
  CONCAT''' AND PARTITION='CONCAT DIGITS(PARTITION)
  CONCAT' AND NAME ='''CONCAT NAME CONCAT''';'
  FROM SYSIBM.SYSTABSTATS
  WHERE DBNAME LIKE 'ABC%'
    AND TSNAME LIKE 'TS%'
                   AND NAME LIKE '%' ;
```

OUTPUT3 – A PARSER FOR THE SYSTABLES STATEMENTS

```
/*********************************************************************/
/*     rexx                                                          */
/*  Split a file into several small files while parsing each line    */
/*  into several small lines. Each line is split based on the        */
/*  positions of some keyword into several lines.                    */
/*  Variable maxlin is used to split the output into smaller         */
/*  members. Setting a high value for this will put all output in    */
/*  one member.                                                      */
/*  Author: Jaiwant Jonathan                                         */
/*********************************************************************/
trace o
clear
pref =strip(sysvar(syspref))
PARSE UPPER ARG P_dsname
if strip(P_dsname)='' then
do
```

```
   Call GETDBLST
end
else
do
   I_lstdsn = strip(P_dsname)
   I_lstdsn = strip(P_dsname,B,"'")
end

cts= time()
octs= substr(cts,1,2)||substr(cts,4,2)
cd = date(U)
us_date = substr(cd,7,2)||substr(cd,1,2)||substr(cd,4,2)
ts_date = substr(us_date,2)

Call ALLOCDSN
Call ALLOCODS
MAINØØØ:
maxlin = 8ØØØ
filcnt = 1
lincnt = 1
Call NEWFIL

do forever
   "execio 1 DISKR INDD1 "
   if rc=2 then leave
   pull inrec1
   inrec1 = strip(inrec1)
   acpos = pos('UPDATE',inrec1,1)
   inrec1 = substr(inrec1,acpos)
   loc1 = pos(' SET ',inrec1,1)
   if loc1=Ø then
      iterate
   loc2 = pos(',NPAGES',inrec1,1)
   if loc2=Ø then
      iterate
   len2 = loc2-(loc1)
   loc3 = pos('AND CREATOR',inrec1,1)
   if loc3=Ø then
      iterate
   len3 = loc3-(loc2)
   lin.1 = substr(inrec1,1,(loc1-1))
   lin.2 = substr(inrec1,loc1,len2)
   lin.3 = substr(inrec1,loc2,len3)
   lin.4 = substr(inrec1,loc3)
   if lincnt <= maxlin then
  do
    "execio * diskw mpds (stem lin. "
     lincnt = lincnt+1
     drop lin
  end
  else
  do
     say 'Closing file 'memnam
     "execio Ø diskw mpds (FINIS "
     address tso "free f(mpds)"
     filcnt=filcnt+1
     Call NEWFIL
     "execio * diskw mpds (stem lin. "
     lincnt = 2
     drop lin.
  end
end
say 'completed processing...' memnam
"execio Ø diskw mpds (FINIS "
"execio Ø diskr INDD1 (FINIS "
address tso "free f(mpds)"
say 'Results generated into 'ods_name
address tso "free f(opds)"
address tso "free f(INDD1)"
exit

GETDBLST:
Say  'Give the input dataset ...'
Say  '(It must be a PS )'
pull I_lstdsn
I_lstdsn = strip(I_lstdsn)
I_lstdsn = strip(I_lstdsn,Both,"'")

x = SYSDSN("'"I_lstdsn"'")
if x ¬= 'OK' then
do
   say; say '*** ERROR ' x ; say
   SIGNAL  GETDBLST
end
return

ALLOCDSN:
"ALLOCATE DD(INDD1) DSN('"I_lstdsn"') REUSE SHR"
if  rc > Ø then
do
   say  'Failed during allocation of 'I_lstdsn
   exit(8)
end
return

ALLOCODS:
ods_name = pref||"."||USERID()||".PARSFIL2.D"||us_date
xx = outtrap("zap.","*")
address tso "delete '"ods_name"'"
xx = outtrap("OFF")
```

```
address tso "alloc f(opds) new unit(sysda) space(1,1)",
          "cyl reuse dsname('"ods_name"')",
          "dsorg(po) dir(2) blksize(312Ø) lrecl(8Ø) recfm(f b)"
say 'Allocated output file 'ods_name
return

NEWFIL:
filnam = filcnt
do while length(filnam) < 4
   filnam = 'Ø'||filnam
end
trace o
memnam = 'M'||filnam
memnam = strip(memnam)
I_mdsname = ods_name||"("||memnam||")"
I_mdsname = strip(I_mdsname)
address tso "alloc f(mpds) mod dsname('"I_mdsname"')"
return
```

OUTPUT4 – SAMPLE JCL FOR RUNNING THE UTILITY

```
/* JOBCARD
//* ---------------------------------------------
//UTILØØØ1  EXEC PGM=DSNUTILB,REGION=2ØM,COND=(4,LT),
//             PARM='DBT2,RUNSTATSDBØ1'
//STEPLIB   DD   DSN=XXXX.DBT2.DSNLOAD,DISP=SHR
//          DD   DSN=XXXX.DBT2.DSNEXIT,DISP=SHR
//*SYSPRINT  DD   SYSOUT=*
//SYSPRINT DD DSN=&&TEMPFILE,DISP=(MOD,PASS),
//            UNIT=SYSDA,SPACE=(TRK,(15,5)),
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=133Ø)
//UTPRINT   DD   SYSOUT=*
//SYSIN DD *
 RUNSTATS TABLESPACE XXXXDBØ1.YYYYTSØ1
   TABLE(ALL) INDEX(ALL) SHRLEVEL CHANGE UPDATE NONE REPORT YES
 RUNSTATS TABLESPACE XXXXDBØ1.YYYYTSØ1
   TABLE(ALL) INDEX(ALL) SHRLEVEL CHANGE UPDATE NONE REPORT YES
//*
//STATSGEN   EXEC PGM=IKJEFTØ1,COND=(4,LT),
//            PARM=STATS2
//SYSEXEC  DD DISP=SHR,DSN=PREFIX.USERID.REXX
//OUTDD    DD DISP=(NEW,KEEP),DSN=PREFIX.USERID.STATS.DØ2Ø8Ø2.A,
//            UNIT=SYSDA,SPACE=(TRK,(15,5)),
//            DCB=(RECFM=FB,LRECL=8Ø,BLKSIZE=312Ø)
//INDD     DD DISP=(OLD,PASS),DSN=&&TEMPFILE
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DUMMY
```

```
//SYSUDUMP DD SYSOUT=*
//*
```

STATS2 – THE REXX EXEC

```
/* rexx  */
MAINØØ:
do forever
"execio 1 diskr INDD "
if rc = 2 then
do
   leave
end
pull inrec
parse var inrec w1 dummy
  SELECT
    When pos('DSNUØ1ØI',w1) > Ø  then iterate
    When pos('DSNUØØØI',w1) > Ø  then iterate
    When pos('DSNUØ5ØI',w1) > Ø  then
    do
       parse var inrec w1 w2 w3 w4 w5 w6 w7 rest
       out.1 = '-- STATISTICS UPDATE FOR '||strip(w6)
       "execio * DISKW OUTDD (STEM out. "
       drop out.
       iterate
    end
    When w1 = 'DSNU613I' then curproc  = 'TABPART'
    When w1 = 'DSNU614I' then
    do
      curproc  = 'TABLES'
      parse var inrec w1 w2 w3 w4 w5 w6 w7 w8 w9 w1Ø w11 rest
      parse var w9 creator '.' tbn
      creator = strip(creator)
      tbn = strip(tbn)
    end

    When w1 = 'DSNU615I' then
    do
      curproc  = 'COLUMNS'
      parse var inrec w1 w2 w3 w4 w5 w6 w7 w8 w9 w1Ø w11 rest
      colname = strip(w9)
    end

    When w1 = 'DSNU612I' then
    do
      curproc  = 'TS'
      parse var  inrec w1 w2 w3 w4 w5 w6 w7 w8 w9 w1Ø w11 rest
      parse var w9 dbn '.' tsn
      dbn = strip(dbn)
      tsn = strip(tsn)
    end

    When w1 = 'DSNU618I' then curproc  = 'IXPART'

    When w1 = 'DSNU617I' then
    do
      curproc  = 'IX'
      parse var inrec w1 w2 w3 w4 w5 w6 w7 w8 w9 w1Ø w11 rest
      parse var w9 creator '.' ixn
      creator = strip(creator)
      tbn = strip(tbn)
    end

    When w1 = 'DSNU624I' then
    do
      curproc  = 'TABSTATS'
      parse var inrec w1 w2 w3 w4 w5 w6 w7 w8 w9 w1Ø w11 rest
      parse var w9 cre '.' tbn
      creator = strip(cre)
      tbn = strip(tbn)
      prt = strip(w11)
    end

    When w1 = 'DSNU625I' then curproc  = 'COLSTATS'
    When w1 = 'DSNU626I' then curproc  = 'COLDISTSTATS'
    When w1 = 'DSNU627I' then curproc  = 'IXSTAT'
    When w1 = 'DSNU616I' then
    do
       curproc  = 'COLDIST'
       parse var inrec w1 w2 w3 w4 w5 w6 w7 w8 w9 w1Ø w11 rest
       colname = strip(w9)
    end
    Otherwise Call BRANCH
  END     /* SELECT */
  oldrec = inrec
end   /* do forever  */
exit

BRANCH:
Select
   When curproc = 'TABSTATS' then Call TABSTATUPD
   When curproc = 'TABLES'   then Call TABLESUPD
   When curproc = 'TABPART'  then return
   When curproc = 'COLUMNS'  then Call COLUMNSUPD
   When curproc = 'TS'       then Call TSUPD
   When curproc = 'IXPART'   then return
   When curproc = 'IXSTAT'   then return
   When curproc = 'IX'       then Call IXUPD
   When curproc = 'COLSTATS' then return
   When curproc = 'COLDIST'  then Call COLDISTUPD
   When curproc = 'COLDISTSTATS'  then return
   Otherwise return
end
return

TABSTATUPD:
   parse var inrec w1 w2 w3 rest
   if strip(w1) = 'CARD' then
      out.2 = '   SET CARD = '||strip(w3)

   if strip(w1) = 'NPAGES' then
   do
      out.2 = out.2||', NPAGES = '||strip(w3)
      out.1 = "UPDATE SYSIBM.SYSTABSTATS "
      out.3 = " WHERE NAME = '"||tbn||"'  AND OWNER = '"||creator||"'"
      out.4 = "   AND PARTITION = "||prt||" ; "
      address tso "execio * DISKW OUTDD (STEM out. "
      drop out.
   end
return

TABLESUPD:
   parse var inrec w1 w2 w3 rest
   if strip(w1) = 'CARDF' then
      out.2 = '   SET CARDF = '||strip(w3)
   if strip(w1) = 'NPAGES' then
      out.2 = out.2||' ,NPAGES = '||strip(w3)
   if strip(w1) = 'PCTROWCOMP' then
   do
      out.2 = out.2||',PCTROWCOMP = '||strip(w3)
      out.1 = 'UPDATE SYSIBM.SYSTABLES'
   out.3 = " WHERE CREATOR ='"||creator||"' AND NAME = '"||tbn||"' ;"
      "execio * DISKW OUTDD (STEM out. "
      drop out.
   end
return

COLUMNSUPD:
   parse var inrec w1 w2 w3 rest
   if strip(w1) = 'COLCARDF' then
   do
      out.2 = '   SET COLCARDF = '||strip(w3)
      coldist_cardf = strip(w3)  /* save this for updating COLDIST */
   end
   if strip(w1) = 'HIGH2KEY' then
      out.3 = "     , HIGH2KEY = "||strip(w3)
   if strip(w1) = 'LOW2KEY' then
   do
      out.4 = "     , LOW2KEY = "||strip(w3)
      out.1 = "UPDATE SYSIBM.SYSCOLUMNS"
      out.5 = " WHERE TBCREATOR ='"||creator||"'"
      out.6 = "   AND TBNAME = '"||tbn||"'"
      out.7 = "   AND NAME = '"||colname||"' ;"
      "execio * DISKW OUTDD (STEM out. "
      drop out.
   end
return

TSUPD:
   parse var inrec w1 w2 w3 rest
   if strip(w1) = 'NACTIVE' then
   do
      out.2 = "   SET NACTIVE = "||strip(w3)
      out.1 = "UPDATE SYSIBM.SYSTABLESPACE "
      out.3 = " WHERE DBNAME ='"||dbn||"'"
      out.4 = "   AND NAME = '"||tsn||"' ;"
      "execio * DISKW OUTDD (STEM out. "
      drop out.
   end
return

IXUPD:
   parse var inrec w1 w2 w3 rest
   if strip(w1) = 'CLUSTERRATIO' then
      out.2 = '   SET CLUSTERRATIO = '||strip(w3)
   if strip(w1) = 'FIRSTKEYCARDF=' then
      out.3 = ' ,FIRSTKEYCARDF = '||strip(w2)
   if strip(w1) = 'FULLKEYCARDF' then
      out.3 = out.3||' , FULLKEYCARDF = '||strip(w3)
   if strip(w1) = 'NLEAF' then
      out.4 = ' ,NLEAF = '||strip(w3)
   if strip(w1) = 'NLEVELS' then
   do
      out.4 = out.4||',NLEVELS = '||strip(w3)||'  '
      out.1 = 'UPDATE SYSIBM.SYSINDEXES'
      out.5 = " WHERE CREATOR ='"||creator||"'"
      out.6 = " AND NAME = '"||ixn||"' ;"
      "execio * DISKW OUTDD (STEM out. "
      drop out.
   end
return

COLDISTUPD:
   parse var inrec w1 w2 rest
   if strip(w1) = 'FREQUENCY' then
   do
      out.1 = 'DELETE FROM SYSIBM.SYSCOLDIST '
      out.2 = " WHERE TBOWNER ='"||creator||"' AND TBNAME = '"||tbn||"'"
      out.3 = "   AND NAME = '"||colname||"' ;"
      "execio * DISKW OUTDD (STEM out. "
      drop out.
      return
   end
   if strip(w1) = '--------' then
      return
   out.1 = "INSERT INTO SYSIBM.SYSCOLDIST"
   out.2 = " VALUES (Ø,CURRENT_TIMESTAMP,'N'"
   out.3 = "         ,'"||creator||"','"||tbn||"'"
   out.4 = "         ,'"||colname||"',"||strip(w2)
   out.5 = "         ,'F',-Ø.1E+Ø1, ' ', 1,"||strip(w1)||");"
   "execio * DISKW OUTDD (STEM out. "
   drop out.
return
```

Jaiwant K Jonathan
DB2 DBA
QSS Inc (USA)

DB2 Everyplace: a mobile DB2

It was a rare day off for Jill, a DBA for one of the local banks. But she was not at home or on vacation. No, she was in the hospital for some out- patient surgery. The lab technician took the blood pressure cuff from around her arm and placed it back into the receptacle on the wall. Then he grabbed what looked like a Palm PDA and started fiddling around with it. Jill loves gadgets and her curiosity got the better of her, causing her to ask “What are you doing with that?”

“I’m entering your blood pressure readings into this gadget, here,” replied the technician. “We’ve had to do this for the past few weeks or so. It’s all about some new procedure for storing patients’ vital statistics.”

“But isn’t it only useful for you? I mean, if you’re just entering it into your gadget, then no-one else can use the data, can they?”

“No, it doesn’t work like that. When I’m through entering your temperature, height, and weight, I can send your information to the doctor’s gadget. I carry this with me all day, recording patient information. But before I go home for the day I have to put this gadget in a little gizmo and press a button. The next day doctors, nurses, and I can pull your information up on our central computer system.”

“Sounds cool to me,” Jill said. “Do you have any idea how it works?”

“Nope. But it makes life easier for me. I don’t have to worry about using clumsy PCs, reading poor handwriting, or losing files any more. I love it.”

“Does everyone use them?”

“We all use them now. Even the nursing staff use them when they make home visits to the elderly and disabled. And we have outfitted the emergency crew in our ambulances with the devices too.”

MANAGING DATA ON MOBILE INFORMATION SYSTEMS

The medical scenario I just described is just one of the ways organizations across many different industries are implementing mobile information systems. A plethora of different names are used to describe this type of system – pervasive computing, wireless computing, handheld systems, and palmtop applications. The hallmark of the system is portability, because a device can be carried wherever a user needs to use it. Data is input into a small, occasionally connected, handheld device. (An occasionally connected device doesn’t have to be part of a network, plugged into a wall, or otherwise attached to anything to run.) Then, the device is synchronized to a central data store. The most common application that spawned this new computing paradigm is PDA software that stores appointment calendars, contacts, and notes. The PalmPilot from Palm was the first truly popular PDA device. However, there are many more applications that are viable for handheld computers – if you have the right tools.

One of the tools is a system to store, retrieve, organize, and manage handheld data, as well as synchronize the data with a traditional server-based RDBMS. IBM's DB2 Everyplace V7.1 is one such system. The product was introduced in May 1999 as DB2 Everywhere. It was first available from IBM as a download from their Web site in August 1999. The initial version supported only Windows CE and PalmOS. With this latest version, IBM has changed the name to DB2 Everyplace and the version number to 7.1 to align it with the DB2 version number on other platforms. DB2 Everyplace is designed for low-cost, low-power, small form-factor devices such as PDAs, handheld personal computers (HPCs) or Pocket PCs, and embedded devices (more information below).

One characteristic of handheld database systems is their small size. Instead of a footprint, IBM refers to DB2 Everyplace's *fingerprint* because its size is too small to be labelled a footprint. DB2 Everyplace is a relational database system with a tiny fingerprint of about 100KB to 150KB specifically designed for small handheld devices. The general idea is to store a small amount of critical data on the handheld device that is later synchronized to other, more complete, and long-term data stores. DB2 Everyplace provides a local data store on the handheld device. There is also a mechanism for synchronizing the relational data on the handheld device to and from other DB2 data sources such as DB2 UDB running on Unix, Windows 2000, OS/390, or z/OS platforms.

DB2 Everyplace runs on PalmOS, Windows CE, EPOC, QNX Neutrino, and embedded Linux:

- Palm Inc's (http://www.palm.com) PalmOS operating system is designed for the Palm series of devices made by Palm Inc, including the Palm II, V, and VII. Other devices, such as the Handspring Visor, also use the PalmOS.
- Microsoft's Windows CE operating system powers the PocketPC. Numerous companies supply handheld devices that run Windows CE, including Hewlett Packard and Casio.
- Symbian's (http://www.symbian.com) EPOC platform is designed for optimal flexibility, giving consumer electronics

manufacturers a broad scope for differentiation and innovation in user interfaces, hardware designs, and connectivity. EPOC is divided into two types of device family – communicators and smartphones. Companies such as Ericsson, Psion, Sony, and Texas Instruments manufacture and market devices that use the EPOC operating system.

- QNX Software Systems Ltd's (http://www.qnx.com) QNX Neutrino is a real-time, extensible, POSIX-certified operating system. Many partner vendors supply embedded systems using the QNX Neutrino operating system.

DB2 EVERYPLACE COMPONENTS

There are three basic DB2 Everyplace components – the handheld database engine, the Synchronization Server, and the Personal Application Builder (PAB).

Handheld database engine

The first component of DB2 Everyplace is its handheld database engine. The engine is a true relational database engine delivering persistent storage for sets of records and the ability to modify and retrieve records. It also provides the integrity mechanism to guarantee that data is not lost or corrupted if a handheld device is powered off or dropped during processing. Note that DB2 Everyplace's database engine is not nearly as complex as the DB2 engine that runs on OS/390, Unix, or Windows NT. DB2 Everyplace is scaled to fit into about 100 to 150KB of memory. DB2 Everyplace is sharable by multiple applications, meaning each new application does not require a new instance of the DB2 Everyplace database engine.

Data in a DB2 Everyplace database can be accessed using several different methods. One simple way that doesn't require you to know SQL is to use the Query By Example (QBE) interface to issue queries. QBE is provided with the base DB2 Everyplace product. If you are a little more sophisticated, a second way to access data is to use the QBE Command Line Processor (CLP) to issue SQL statements. The CLP is a window accessed using QBE. You simply type a SQL statement in the query field, tap the *Run SQL* button, the SQL is executed, and the results returned. A third method is to write your own applications using ODBC call-level interface functions.

DB2 Everyplace uses SQL to modify and access data. SQL is the API through which applications access relational data. The SQL version supported by DB2 Everyplace is a subset of the SQL supported by DB2 on other platforms. A full SQL implementation is not required on handheld devices because most data accesses are simple data entry and retrieval requests. All basic SQL DML statements (INSERT, UPDATE, DELETE, SELECT) are supported, as well as many other common features including DBCS support, joins, and cursors. DB2 Everyplace V7.1 even delivers scrollable cursors. However, UNION, a standard relational feature, is not supported by DB2 Everyplace.

An intriguing new feature in DB2 Everyplace is indexing support. One of the most important factors for DB2 performance is proper index design, creation, and management. However, DB2 Everyplace databases are very small, so indexing was not IBM's first priority in earlier versions. In DB2 Everyplace V7.1, indexes provide a welcome performance boost for medium and large DB2 Everyplace tables. Keep in mind that a medium-sized table for DB2 Everyplace is smaller than traditional DB2 tables residing on mid-range and mainframe servers. However, DB2 Everyplace has been used to manage databases of up to 120MB.

There are some additional limitations to DB2 Everyplace. For example, it doesn't support subqueries and you can't create views. Also, some object/relational features are not available, such as triggers, stored procedures, LOBs, and user-defined functions. DB2 Everyplace is designed to access data from small databases, so currently there are no compelling reasons for such advanced features. Additionally, keep in mind that locking is not required for DB2 Everyplace because a handheld device is intrinsically designed for a single user. There's no reason to lock the data from other users.

Synchronization Server

The second component of DB2 Everyplace is the Synchronization Server, or Sync Server, which is new to DB2 Everyplace V7.1. Sync Server takes the place of IBM Mobile Connect, an additional product that used to be required to synchronize handheld device data with a central server.

Sync Server is a client/server program that manages the data synchronization process from the handheld DB2 Everyplace database to the source DB2 database. The source DB2 database can be any DB2 UDB server platform. Sync Server requires a client to be installed on the handheld device and a server component to be installed on the platform to which data is to be synchronized. The Sync Server engine requires a mid-tier DB2 UDB for Windows NT server, regardless of the host server to which the data is to be synchronized.

Sync Server enables two-way data synchronization from the handheld database to a DB2 UDB database, as well as from the DB2 UDB database to the handheld database. To synchronize data, Sync Server initiates a synchronization session. This session is a two-way process during which:

- Mobile users submit changes that have been made to local copies of source data.
- Mobile users receive changes to source data residing on the enterprise server that have been made since the last time the data was synchronized.

Personal Application Builder

The third component of DB2 Everyplace is the Personal Application Builder (PAB). PAB is an integrated toolkit for developing DB2 Everyplace applications running on handheld devices. It supports building applications for small handheld devices that access DB2. PAB makes it easy to write robust applications on a more powerful development platform (such as a Windows PC) for deployment to handheld devices. It supports visual forms construction for different devices, and it provides scripting capabilities for user-defined logic. PAB also integrates with other tools for application testing and debugging.

IBM will support the creation of PalmOS applications with the initial release of the PAB for DB2 Everyplace. Support for other handheld platforms will be added later.

To begin developing applications using PAB for PalmOS, you need to download the supporting GNU Palm tools from IBM. PAB generates GNU-compatible C code. GNU is a self-referencing acronym that stands for GNU's Not Unix. The GNU Project was started in 1983 with the philosophy of producing non-proprietary software. Many systems, most notably Linux, rely heavily on GNU software. In the past, GNU systems used the Linux kernel. For more information, check out http://www.gnu.org.

Another useful tool for testing Palm applications is the Palm OS Emulator (POSE). POSE is not a part of PAB; it can be downloaded from the Palm Web site. POSE emulates the Palm handheld device hardware. Using POSE, you can create a *virtual* handheld device running on Windows, Mac OS, or Unix machines. To run the emulator, you will need to download the appropriate Palm ROM for the device for which you are developing applications. Palm provides debug ROMs through licensing, or you can download a ROM from your Palm computing device. Details are available on the Palm Web site.

PAB lets you develop applications and test them using POSE without ever having to move the application to the handheld device. Only when the development and testing process is complete will you need to move the code to the Palm device. DB2 Everyplace includes a sample project for the PalmOS that features sample code. You can use the sample project as a template for applications and to learn coding techniques using PAB.

PRODUCTION APPLICATION

There are a myriad of potential DB2 Everyplace applications, such as eScholar, developed by eScholar LLC (http://www.escholar.com/), a wholly owned subsidiary of IBM Business Partner Vision Associates. The eScholar application uses DB2 Everyplace to make student performance, class attendance, and other educational profile data accessible on handheld devices for teachers, administrators, and counsellors. The application is innovative because most teachers lack the time to sit at a computer and access data while they are teaching students. Teachers can use eScholar and a Palm device to quickly obtain important student information without having to disrupt student interaction.

DATABASE ADMINISTRATION CHALLENGES

So far, so good, but how will DB2 Everyplace impact your IT organization when it's implemented on handheld devices? Although DB2 Everyplace doesn't require the extensive tuning and administration necessary for enterprise databases, it is still relational. Databases should be developed using sound logical and physical design techniques, including data modelling and normalization.

The biggest impact of DB2 Everyplace is planning for and managing data synchronization from hundreds or thousands of handheld devices. When should synchronization be scheduled? How will it impact applications that use large production databases that are involved in the synchronization? How can you ensure that a mobile user will synchronize data reliably and on schedule? Potential problems that could arise from failing to synchronize include:

- Outdated information on the centralized database.
- Outdated information on the handheld device.
- Large files on the handheld device that could cause slower synchronization when the files are eventually synched.
- Slow handheld application performance.

These are not minor issues. Make sure your DBA staff are prepared for the impact before implementing a large battalion of handheld database users who must synchronize their data. It is important to use forethought to determine which existing application systems in your organization will be the first ported to handheld devices. Possible targets include sales or delivery tracking systems used by remote workers. Consider how system infrastructure will be affected by a large influx of remote connections.

KEEPING UP-TO-DATE WITH DB2 EVERYPLACE

To keep up-to-date on DB2 Everyplace and to share information and experiences with other DB2 Everyplace users, be sure to regularly visit the DB2 Everyplace Forum, which can be accessed via the DB2 Everyplace home page at http://www.ibm.com/software/data/db2/everyplace/.

CONCLUSION

Jill arrived back at her desk the day after her medical appointment only to find a project request for a new application requiring DB2 Everyplace running on handheld Palm devices. The bank's ATM division issued a mandate calling for all bank technicians who care for ATMs to carry Palm computing devices. The project will require a new application for entering details about the maintenance, stocking, and status of each ATM. The information will be collected each day and then synchronized with the enterprise ATM application running on the mainframe under CICS and DB2 for OS/390. Jill was glad she had asked questions at the hospital. She considered telling her boss that she'd been conducting research for the new project so that she wouldn't have to use up a sick day after all. At least Jill knew her project would be in good hands with DB2 Everyplace!

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

Automatic placement of user-managed datasets

For large databases that have several partitioned tablespaces, proper placement of the underlying DB2 VSAM linear datasets in a production environment is one of the most critical activities for disk I/O performance.

Large tables are generally designed with partitioned tablespaces, each having one partitioning index and supported by one or more secondary indexes (for performance reasons). These secondary indexes may have a single underlying dataset or multiple pieces (using the PIECESIZE option).

For user-managed datasets, placing them manually over different volumes, ensuring that they are evenly distributed across the volumes, is a very tedious and iterative task.

The job involves:

- Distributing datasets of all tablespace partitions across volumes and minimizing any possibility of two or more datasets getting placed on the same volume.
- Distributing datasets of all index partitions across volumes and minimizing any possibility of two or more datasets getting placed on the same volume.
- Minimizing any possibility of a tablespace partition dataset and its corresponding index partition dataset being placed on the same volume.
- Minimizing any possibility of, for a partitioned tablespace, any tablespace datasets or their corresponding partitioned index datasets overlapping any of their corresponding secondary index datasets.
- Ensuring that the sum of the sizes for all the datasets placed on a volume does not exceed the maximum allowable space limit assigned for each volume.

This REXX EXEC has been developed to automate the process of assigning individual datasets to different volumes and, as a first cut, it creates a reasonably good distribution of datasets. This tool is very useful when the number of tablespaces is very large and each tablespace has more than 32 partitions with one or more secondary indexes having multiple pieces.

This REXX EXEC works equally well for non-partitioned tablespaces and indexes as well as for a combination of partitioned and non-partitioned tablespaces/indexes.

HOW DOES THIS WORK?

This utility has a panel, PDSDTL, which allows a user to enter specific storage-related details (primary and secondary allocation quantities) for a partition or a group of partitions of a tablespace/ index and temporarily store them in an ISPF table.

The details entered are:

- Database name – database name for which these entries are made.
- Tablespace/index name – tablespace or index for which VSAM datasets are to be created.
- Group number – a number, starting from 1 (to a maximum of 254), assigned to a group of consecutive partitions of a tablespace / index having the same allocation sizes.
- Tablespace/index – type of object. Possible values are TS for tablespace, PI for partitioned index, and SI for secondary index.
- Reference tablespace – tablespace name containing the table for this index object. For an index entry, the value is the name of the tablespace containing the table corresponding to this index. For a tablespace entry, this value is blank.
- Partitions in the group – the number of contiguous partitions of tablespace/index for that group. Possible values are 1 to 254.

- Primary quantity – primary allocation size for tablespace/ index partition.
- Secondary quantity – secondary allocation size for tablespace/index partition.
- Cylinders/tracks – unit of allocation size. Possible values are C for cylinders and T for tracks.

An example

The following is an example of a tablespace TS1 having 150 partitions with partitioning index I1 and secondary index I2 with 40 pieces (using PIECESIZE option) with sizes as follows:

```
TS1  partitions       1- 5Ø ,  Primary qty = 1ØØØ, secondary qty = 5Ø
TS1  partitions      51- 1ØØ , Primary qty = 6ØØ,  secondary qty = 1Ø
TS1  partitions     1Ø1- 15Ø , Primary qty = 1ØØ,  secondary qty = 5

I1     partitions     1- 5Ø ,  Primary qty = 1ØØ,  secondary qty = 2Ø
I1     partitions    51- 1ØØ,  Primary qty = 8Ø,   secondary qty = 15
I1     partitions   1Ø1- 15Ø , Primary qty = 6Ø,   secondary qty = 1Ø

I2     pieces         1- 4Ø ,  Primary qty = 15ØØ, secondary qty = 15Ø
```

All these quantities are in cylinders.

Panel entries are as follows:

```
Tablespace TS1, Group 1, partitions = 5Ø, primary qty = 1ØØØ,
                secondary qty = 5Ø
Tablespace TS1, Group 2, partitions = 5Ø, primary qty = 6ØØ,
                secondary qty = 1Ø
Tablespace TS1, Group 3, partitions = 5Ø, primary qty = 1ØØ,
                secondary qty = 5
Index I1,       Group 1, reference tablespace =TS1, partitions = 5Ø,
                primary qty = 1ØØ, secondary qty = 2Ø
Index I1,       Group 2, reference tablespace =TS1, partitions = 5Ø,
                primary qty = 8Ø, secondary qty = 15
Index I1,       Group 3, reference tablespace =TS1, partitions = 5Ø,
                primary qty = 6Ø, secondary qty = 1Ø
Index I2,       Group 1, reference tablespace =S1, partitions = 4Ø,
                primary qty = 15ØØ secondary qty = 15Ø
```

Once all entries are done and finalized, action GEN allocates the volumes for each dataset and generates an output dataset with VSAM dataset definitions. If there is an insufficient number of volumes or individual datasets for a partitioned tablespace/ index, they cannot be distributed across the volumes, and the utility stops with a message, 'ALL VOLUMES FULL' and the tablespace/index name for which the allocation failed.

In such a case, the required action is either to increase the number of volumes or to handle placement for that object manually.

The different actions available on the panel are:

- ADD – creates new entries and stores them in an ISPF table. Please note that all contiguous partitions of the same size for a tablespace/index are assigned to one group and the minimum number of groups is one, where all partitions are the same size.
- DIS – displays an existing entry for a given database name, tablespace/index name, and group number.
- MOD – modifies an existing entry on keys (database name, tablespace/index name, group number). Fields which can be changed with this action are: type of object (tablespace, partitioning index, or secondary index); reference tablespace name (for index entry – the name of the tablespace containing table corresponding to this index: for a tablespace entry it is blank); number of contiguous partitions in that group; primary and secondary quantities (allocation sizes) for the object; and unit of allocation (C for cylinders and T for tracks).
- DEL – deletes an existing entry on keys (database name, tablespace/index name, group number).
- NXT and PRV – browse through existing entries in forward and backward direction.
- SAV – saves all existing entries in a dataset under the DDname fields. These entries can again be retrieved and populated in the ISPF table using GET action.
- GET – retrieves entries saved in a dataset, populates the ISPF table, and displays the entries on the panel.

- GEN – once all the entries are made and finalized on the panel, this action reads each entry, assigns datasets to volumes as per the entry definitions, and generates VSAM linear dataset definitions in a dataset.
- END – ends the utility.

OPTIONS USED

This REXX uses the following parameters as inputs controlling different options under which this tool can run:

- vol_name_pfx – this is the prefix for all volumes' names that this REXX uses. For example, if the prefix value is VOL, then the volume names assigned will be VOL0001, VOL0002, and so on. These names can later be substituted with the actual volume names available at the installation.
- empty_vol – this parameter is used to determine whether all volumes available are empty and they do not have some space already allocated to other datasets. Normally, the value for this is YES, which indicates that all volumes assigned are empty and this is the first run. The 'YES' value works with vol_name_pfx value.

  A value of 'NO' indicates that some or all volumes already have some spaces allocated to other datasets. This option requires another dataset in input mode (DDname fvolin). This dataset provides information on space available on each volume to start with. This dataset has entries for each volume on a separate line. Each line will have the volume name and the space available in cylinders on that volume for further dataset allocation. For example for three volumes, VOL1, VOL2, and VOL3, if the available number of cylinders for allocation are 200, 300, and 500 respectively, this dataset will have the following three entries, each on a different line:

  ```
  VOL1     2ØØ
  VOL2     3ØØ
  VOL3     5ØØ
  ```

  The REXX EXEC reads this dataset and stores this information at the beginning. Later on it uses this data at the time of allocation of datasets.

This option is used for allocating datasets in subsequent runs.

- max_vol – this parameter determines the maximum number of empty volumes available in the first run and it is valid for the empty_vol = ‘YES’ option. As with the ‘NO’ option, all volume details are available in a dataset.
- vol_cyl_limit – this parameter defines the total number of cylinders that can be allocated on each volume. For example for a 3390 device, if 3300 cylinders is the capacity, then a value of 2600 cylinders for this parameter is good enough to take care of future growth of datasets. Once this limit is reached for any volume, the REXX EXEC does not use that volume for any further allocation.
- si_ind – this indicator tells whether the secondary index datasets can be placed along with the corresponding tablespace partitions datasets or corresponding partitioning index partition datasets on the same volume.

  A value of ‘Y’ indicates that secondary index datasets cannot be placed with any of the corresponding tablespace datasets (partitioned/non-partitioned) or corresponding partitioned/ other secondary datasets (partitioned /non-partitioned) for that tablespace. This option yields a better dataset distribution, but it may require more volumes for allocation.

  A value of ‘N’ indicates that secondary index datasets can be placed on the same volumes that have the corresponding tablespace datasets or other datasets for other indexes that correspond to that tablespace
- vcat – this parameter is the vcat name and is used as the HLQ for generated VSAM datasets.

DATASETS USED

The datasets used have DDnames of fds, fileo, fvolin, and fvolout. All these four datasets need to be pre-allocated before this EXEC is run.

fds

Dataset fds (DDname) is used to save all ISPF table entries made through the panel. The action SAV on the panel, stores all entries in this dataset. Action GET on the panel restores these entries back to an ISPF table which can then be browsed, modified, or deleted on the panel. Please note that, if these entries have to be retrieved from the dataset, the GET action should be the first action on the panel, followed by any subsequent ADD, MOD, or DEL actions.

fileo

Dataset fileo (DDname) is the output dataset and it contains the required VSAM dataset definitions that can used by IDCAMS to create VSAM datasets.

fvolin

Dataset fvolin (DDname) is used in input mode with the empty_vol parameter value equal to 'NO'. This dataset is used when volumes are not empty and are already pre-allocated with some other data. In this dataset, each row contains information specific to a volume where the first field contains the volume name and the second field contains the number of cylinders available on that volume for subsequent allocation of datasets during the next run. The REXX EXEC reads this dataset at the start, stores the information in memory, and uses the information during dataset allocation. If there is an entry for a volume which has 0 cylinders available, that volume will not be allocated to any new dataset.

This dataset may be empty where empty_vol = 'YES'.

fvolout

Dataset fvolout (DDname) is the output dataset for each run. This contains volume information for each volume after the allocation is done. Each row contains the volume name and number of cylinders available on that volume at the end of the run. At the end, it also gives certain audit messages regarding the total number of cylinders available before run and after run, and how many cylinders were allocated in that run. Each run is identified by a running Run number. After the first run, this dataset can be used as fvolin (DDname) for the next run.

```
/*REXX*/

/**************** Datasets allocated for EXEC **********/
fileo='TEST.VSAMD.OUTPUT'   /* for generated VSAM datasets definitions*/
fvolin = ‘TEST.VOLINF.INPUT’         /* for Volume information input  */
fvolout = ‘TEST.VOLINF.OUTPUT’       /* for Volume information output */
fds = ‘TEST.SAVEINF.ENTRIES’         /* for saving all screen entries */
/*************** Parameters used  **********************/
vol_name_pfx  =  'VOL'           /* Prefix for Volume names generated */
start_vol  =  1         /* Starting position for volumes, normally 1 */
empty_vol  =  'YES'      /* Indication for space available on volumes */
                        /* YES - all volumes have space available as */
                        /*       per maximum cylinders available     */
                        /* NO - volumes are partially filled up      */
max_vol  = 1000              /* Maximum numbers of volumes available */
vol_cyl_limit  =  2600        /* Maximum number of Cylinders Limit on */
                                                      /* each volume */
si_ind = 'Y'   /* indicator for placement of Sec. Index datasets(Y/N) */
vcat  =  'vcatname'       /* vcat name for Linear datasets generated */
/*******************************************************/

if sysdsn("'"fds"'") ¬=  "OK" then
do
  say 'OUTPUT FILE ' fds ' DOES NOT EXIST.'
  say
  exit
end
"ALLOC DA('"||fds||"')    F(DATADS) shr"

tot_ptns  =  0
eof  =  'NO'
nxt_ind  =  0
prv_ind  =  0
tbl_entries  =  0
tot_tbl_entries  =  0
dbarr.0  =  0
dbarr.  =  ''
tsarr.0  =  0
tsarr.  =  ''
dbtsidx  =  0
get_first_ind  =  ''
```

```
ADDRESS "ISPEXEC"
"LIBDEF ISPPLIB DATASET ID ( 'TEST.ISPPLIB' )"
"LIBDEF ISPTLIB DATASET ID ( 'TEST.ISPTLIB' )"
ADDRESS "ISPEXEC" "TBCREATE TBGRPS"||,
      " KEYS (TDBNAME TTSNAME TGRP)"||,
      " NAMES (TTSREF TTIX TNUMPRTS TSIZPRTS TSECQ TCYLS)"||,
      " NOWRITE REPLACE"
ADDRESS "ISPEXEC" "TBCREATE TBGRPA"||,
      " KEYS (TGRPTXT1)"||,
      " NOWRITE REPLACE"
ADDRESS "ISPEXEC" "TBCREATE TBGRPB"||,
      " KEYS (TGRPTXT2)"||,
      " NOWRITE REPLACE"

call first_phase
if acn  =  'GEN' then
   do
      call second_phase
      call third_phase
      call fourth_phase
   end

ADDRESS "ISPEXEC"
"LIBDEF ISPPLIB "
"LIBDEF ISPTLIB "
ADDRESS TSO
"FREE F(DATADS)"
exit

 first_phase:
/***********/
do while eof = 'NO'
  "DISPLAY PANEL (PDSDTL)"
  msg = ''
  if acn  =  'END' then
      do
        eof = 'YES'
        leave
      end
  if acn  =  'GEN' then
     if tbl_entries  =  Ø then
        do
           msg = 'No Entries are made'
           iterate
        end
     else
        do
           eof = 'YES'
           leave
        end
  if acn  =  'SAV' then
      do
         call process_put
         iterate
      end
  if acn  =  'GET' then
      if get_first_ind  =  '' then
         do
            call  process_get
            acn  =  'NXT'
            call process_rtn
            iterate
         end
      else
         do
            msg = "GET should be done only at the beginning "
            iterate
         end
  if ( acn  =  'ADD' | acn  =  'MOD' | acn  =  'DEL' ) then
     do
        if dbname  =  ' ' then
           do
              msg = "Database name Invalid"
              iterate
           end
        if tsname  =  ' ' then
           do
             msg = "Tablespace name Invalid"
             iterate
           end
        if tix  =  'TS' | tix  =  'PI' | tix  =  'SI' then
           nop
        else
           do
              msg = "Tablespace /Index Indication Invalid"
              iterate
           end
        if tix  =  'PI' | tix  =  'SI' then
           if ( acn  =  'ADD' | acn  =  'MOD' ) then
              if tsref  =  ' ' then
                 do
                    msg = "Reference Tablespace must be entered"
                    iterate
                 end
        if tix  =  'PI' | tix  =  'SI' then
           if ( acn  =  'ADD' | acn  =  'MOD' ) then
              do
                 call fnd_dbtsarr
                 if fnd_dbts_ind = 'N' then
```

```
                    do
                       msg = "Reference Tablespace not Entered"
                       iterate
                    end
              end
        if tix  =  'TS' then
           if ( acn  =  'ADD' | acn  =  'MOD' ) then
              if tsref <> ' '  then
                 do
                    msg = "Reference Tablespace must be blanks"
                    iterate
                 end
        if grp < '1'  | grp > '254' then
           do
              msg = "Group Number Invalid"
              iterate
           end
        if numparts < '1' | numparts > '254' then
           do
              msg = "Number of Partitions Invalid"
              iterate
           end
     end
  call process_rtn
end /* do while */
return

 process_rtn:
/***********/
  get_first_ind  =  'N'
  if acn <> 'NXT' then
     nxt_ind = Ø
  if acn <> 'PRV' then
     prv_ind = Ø
  select
     when acn = 'DIS' then call grp_dis
     when acn = 'ADD' then call grp_add
     when acn = 'MOD' then call grp_mod
     when acn = 'DEL' then call grp_del
     when acn = 'NXT' then call grp_nxt
     when acn = 'PRV' then call grp_prv
     otherwise msg = "Wrong Action Code"
  end
return

 grp_add:
/********/
  TDBNAME  =  dbname
  TTSNAME  =  tsname
  TTIX  =  tix
  if tix  =  'PI' | tix  =  'SI' then
     TTSREF  =  tsref
  else
     TTSREF  =  tsname
  TGRP  =  grp
  TNUMPRTS  =  numparts
  TSIZPRTS  =  szparts
  TSECQ  =  secprts
  TCYLS  =  cyls
  ADDRESS "ISPEXEC" "TBADD TBGRPS"
  if rc > 4  then
     msg = 'Duplicate Entry'
  else
     do
        tbl_entries = tbl_entries + 1
        tot_tbl_entries = tot_tbl_entries + 1
        dbarr.tot_tbl_entries = dbname
        tsarr.tot_tbl_entries = tsname
     end
return

 grp_dis:
/********/
  ADDRESS "ISPEXEC" "TBSORT TBGRPS "||,
         "FIELDS(TDBNAME,C,A,TTSNAME,C,A,TGRP,C,A)"
  ADDRESS "ISPEXEC" "TBTOP TBGRPS"
  TDBNAME  =  dbname
  TTSNAME  =  tsname
  TGRP  =  grp
  ADDRESS "ISPEXEC" "TBGET TBGRPS"
  if rc <> Ø  then
     do
        msg = 'Entry Not Found'
        return
     end
  tix  =  TTIX
  if tix = 'TS' then
     tsref = ''
  else
     tsref = TTSREF
  numparts  =  TNUMPRTS
  szparts  =  TSIZPRTS
  secprts  =  TSECQ
  cyls  =  TCYLS
return

 grp_del:
/********/
  ADDRESS "ISPEXEC" "TBSORT TBGRPS "||,
         "FIELDS(TDBNAME,C,A,TTSNAME,C,A,TGRP,C,A)"
  ADDRESS "ISPEXEC" "TBTOP TBGRPS"
  TDBNAME  =  dbname
  TTSNAME  =  tsname
  TGRP  =  grp
  ADDRESS "ISPEXEC" "TBGET TBGRPS"
  if rc <> Ø  then
     do
        msg = 'Entry Not Found'
        return
     end
  tix  =  TTIX
  if tix = 'TS' then
     tsref = ''
  else
     tsref = TTSREF
  numparts  =  TNUMPRTS
  szparts  =  TSIZPRTS
  secprts  =  TSECQ
  cyls  =  TCYLS
  ADDRESS "ISPEXEC" "TBDELETE TBGRPS"
  if rc > 4  then
     msg = 'Entry Not Deleted'
  else
     do
        tbl_entries = tbl_entries - 1
        call del_dbtsarr
     end
return

 grp_mod:
/********/
  ADDRESS "ISPEXEC" "TBSORT TBGRPS "||,
         "FIELDS(TDBNAME,C,A,TTSNAME,C,A,TGRP,C,A)"
  ADDRESS "ISPEXEC" "TBTOP TBGRPS"
  TDBNAME  =  dbname
  TTSNAME  =  tsname
  TGRP  =  grp
  ADDRESS "ISPEXEC" "TBGET TBGRPS"
  if rc <> Ø  then
     do
        msg = 'Entry Not Found'
        return
     end
  TTIX  =  tix
  if tix  =  'PI' | tix  =  'SI'       then
     TTSREF  =  tsref
  else
     TTSREF  =  tsname
  TNUMPRTS  =  numparts
  TSIZPRTS  =  szparts
  TSECQ  =  secprts
  TCYLS  =  cyls
  ADDRESS "ISPEXEC" "TBMOD TBGRPS"
  if rc > 4  then
     msg = 'Entry Not Changed'
return

 grp_nxt:
/********/
  if nxt_ind  =  Ø then
     do
        ADDRESS "ISPEXEC" "TBSORT TBGRPS "||,
               "FIELDS(TDBNAME,C,A,TTSNAME,C,A,TGRP,C,A)"
        ADDRESS "ISPEXEC" "TBTOP TBGRPS"
        ADDRESS "ISPEXEC" "TBSKIP TBGRPS"
        if rc > 4  then
           do
              msg = "First Entry Not Found"
              return
           end
        nxt_ind = 1
        dbname  =  TDBNAME
        tsname  =  TTSNAME
        grp  =  TGRP
        tix  =  TTIX
        if tix = 'TS' then
           tsref = ''
        else
           tsref = TTSREF
        numparts  =  TNUMPRTS
        szparts  =  TSIZPRTS
        secprts  =  TSECQ
        cyls  =  TCYLS
     end
  else
     do
        ADDRESS "ISPEXEC" "TBSKIP TBGRPS"
        if rc > 4  then
           do
              msg = "No More Entries Found"
              return
           end
        dbname  =  TDBNAME
        tsname  =  TTSNAME
        tix  =  TTIX
        if tix = 'TS' then
           tsref = ''
        else
           tsref = TTSREF
        grp  =  TGRP
        numparts  =  TNUMPRTS
        szparts  =  TSIZPRTS
        secprts  =  TSECQ
        cyls  =  TCYLS
        return
     end
return

 grp_prv:
/********/
  if prv_ind  =  Ø then
     do
        ADDRESS "ISPEXEC" "TBSORT TBGRPS "||,
               "FIELDS(TDBNAME,C,D,TTSNAME,C,D,TGRP,C,D)"
        ADDRESS "ISPEXEC" "TBTOP TBGRPS"
        ADDRESS "ISPEXEC" "TBSKIP TBGRPS"
        if rc > 4  then
           do
              msg = "Last Entry Not Found"
              return
           end
        prv_ind = 1
        dbname  =  TDBNAME
        tsname  =  TTSNAME
        tix  =  TTIX
        if tix = 'TS' then
           tsref = ''
        else
           tsref = TTSREF
        grp  =  TGRP
        numparts  =  TNUMPRTS
        szparts  =  TSIZPRTS
     end
  else
     do
        ADDRESS "ISPEXEC" "TBSKIP TBGRPS"
        if rc > 4  then
           do
              msg = "No More Entries Found"
              return
           end
        dbname  =  TDBNAME
        tsname  =  TTSNAME
        tix  =  TTIX
        if tix = 'TS' then
           tsref = ''
        else
           tsref = TTSREF
        grp  =  TGRP
        numparts  =  TNUMPRTS
        szparts  =  TSIZPRTS
        return
     end
return

 del_dbtsarr:
/************/
fnd_dbts_ind  =  'N'
do dbtsidx   =  1 to tot_tbl_entries
   if ( dbname  =  dbarr.dbtsidx & tsname  =  tsarr.dbtsidx ) then
      do
         fnd_dbts_ind = 'Y'
         dbarr.dbtsidx  = ''
         tsarr.dbtsidx  = ''
         leave
      end
end
return

 fnd_dbtsarr:
/************/
fnd_dbts_ind = 'N'
do dbtsidx  =  1 to tot_tbl_entries
   if ( dbname = dbarr.dbtsidx & tsref = tsarr.dbtsidx ) then
      do
         fnd_dbts_ind = 'Y'
         leave
      end
end
return

 second_phase:
/*************/
  ADDRESS "ISPEXEC" "TBSORT TBGRPS "||,
          "FIELDS(TDBNAME,C,A,TTSREF,C,A,TTSNAME,C,A,TGRP,C,A)"
  ADDRESS "ISPEXEC" "TBTOP TBGRPS"
  ADDRESS TSO
  dbname = ''
  w_dbname = ''
  w_tsref = ''
  w_sp_name = ''
  w_tsp_name = ''
  sp_name = ''
  tsp_name = ''
  tix_ctr = Ø
  rec1 = ''
  rec2 = ''
  ptns = Ø
  max_ptns = Ø
  inp_rows = Ø
  oup_rows = Ø
  tot_cyls = Ø
  i = Ø
  j = Ø
  k = Ø
  call sec_rtn
  rec2 = w_dbname||' '||w_tsref||' '||substr(max_cyl,1,6)||'  '||rec2
  oup_rows = oup_rows + 1
  if ptns > max_ptns then
      max_ptns = ptns
  call write_tbgrpa
  say 'Total number of input rows processed  =  '||inp_rows
  say 'Total number of Cylinders  =  '||tot_cyls
  say 'Maximum number of Partitions =  '||max_ptns
return

 sec_rtn:
/********/
  eof = 'NO'
  rec1 = ' '
  rec2 = ' '
  max_cyl = Ø
  first_rec = Ø
  j = Ø
  k = Ø
  INLIST.    =  ""
  INLIST.Ø   =  Ø
  OUTLIST.    =  ""
  OUTLIST.Ø   =  Ø
  do while eof = 'NO'
      ADDRESS "ISPEXEC" "TBSKIP TBGRPS"
      if rc > 4  then
         do
            ADDRESS TSO
            eof = 'YES'
         end
      else
         do
            count_ptn  =  value(TNUMPRTS)
            dbname  =  TDBNAME
            tsname  =  TTSNAME
            szparts  =  TSIZPRTS
            secprts  =  TSECQ
            cyl   =  TCYLS
            tix  =  TTIX
            tsref  =  TTSREF
            do  ptn_ctr  =  1 to count_ptn
                call process_rtn_sec
            end
         end
  end
  ADDRESS "ISPEXEC" "TBCLOSE TBGRPS"
return

 process_rtn_sec:
/*****************/
  ADDRESS TSO
  pqty = value(szparts)
  sqty = value(secprts)
  if cyl = 'T' then
    do
         tot_cyls = tot_cyls +  (trunc(pqty/15)  + 1)
         pqtyc = trunc(pqty/15)  + 1
    end
  else
    do
         tot_cyls = tot_cyls + pqty
         pqtyc = pqty
    end
  inp_rows = inp_rows + 1
  sp_name = substr(tsname,1,8)
  if first_rec = Ø then
    do
      first_rec = 1
      rec2 = ''
      rec2 = rec2||'  '||sp_name||'  '||cyl||'  '||tix||'  '
      rec2 = rec2||' '||substr((pqty||'.'||secprts||'.'||cyl),1,11)
      tix_ctr  =  1
      w_dbname = dbname
      w_sp_name = sp_name
      w_tsref = tsref
      max_cyl = pqtyc
      ptns = 1
    end
  else if ( sp_name = w_sp_name & dbname = w_dbname ) then
     do
         rec2 = rec2||' '||substr((pqty||'.'||secprts||'.'||cyl),1,11)
         ptns = ptns + 1
         if pqtyc > max_cyl then
            max_cyl = pqtyc
     end
  if ( sp_name <> w_sp_name  | dbname <> w_dbname ) then
   do
  rec2 = w_dbname||' '||w_tsref||' '||substr(max_cyl,1,6)||'  '||rec2
    oup_rows  =  oup_rows  +  1
    if ptns > max_ptns then
         max_ptns  =  ptns
    call write_tbgrpa
    rec2 = ''
    rec2 = rec2||'  '||sp_name||'  '||cyl||'  '||tix||'  '
    rec2 = rec2||' '||substr((pqty||'.'||secprts||'.'||cyl),1,11)
    w_dbname = dbname
    w_tsref = tsref
    w_sp_name = sp_name
    max_cyl = pqtyc
    ptns = 1
   end
return Ø

 write_tbgrpa:
/*************/

  TGRPTXT1 = rec2
  ADDRESS "ISPEXEC" "TBADD TBGRPA"
  if rc > 4  then
     say 'Write Error in Table TBGRPA'
  ADDRESS TSO
return
```

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

Sharad K Pande
Senior DBA (USA)

DB2 news

Embarcadero Technologies has announced DBArtisan Version 7.0 for building, managing, and trouble-shooting enterprise database infrastructures. DBArtisan 7.0 includes support enhancements for DB2 UDB. With the addition of Embarcadero SQL Debugger for UDB, DBAs and developers can now diagnose and fix problematic server-side code and *ad hoc* SQL.

With more than 30 enhancements, DBArtisan 7.0 allows for the administration of more complex database environments by delivering the most up-to-date support for the current release of DB2. It also provides cross-platform functionality.

New features include DB2 system-level navigation and DB2 UDB debugging.

For further information contact:
Embarcadero Technologies, 425 Market Street, Suite 425, San Francisco, CA 94105, USA.
Tel: (415) 834 3131.
URL: http://www.embarcadero.com/products/dbartisan/index.asp.

* * *

IBM has announced 11 new data management tools targeting administration and utilities, performance management, recovery, and replication and application management.

The name of the DB2 Recovery Manager for z/OS program has been changed to IBM Application Recovery Tool for IMS and DB2 Databases V1.2, which is out now. There are also two IMS and one DB2 new programs, and six new program releases: four IMS, one DB2, and one supporting IMS and DB2.

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

* * *

IBM has released Tivoli System Automation for OS/390 (SA OS/390) under the Tivoli Environment-Managed Licensing Model, which means pricing and licensing are based on what is managed rather than how the software is implemented.

The software is designed to automate I/O, processor, and system operations and includes canned automation for IMS, CICS, IBM Tivoli Workload Scheduler, and DB2. Key functions include Parallel Sysplex application automation, policy-based self-healing, integration, processor operations (ProcOps) and I/O operations, and SAP R/3 high-availability automation.

Other features include cluster-wide policy to help reduce complexity, implementation time, coding, and support plus Parallel Sysplex management and automation functions, including single system image, single point of control, and Parallel Sysplex application automation.

It also provides policy-based e-business automation that can start, stop, monitor, and recover z/OS Unix applications and resources.

For further information contact your local IBM representative.
URL: http://www.tivoli.com/products.