114

DB2 update

April 2002

In this issue

3 DB2 UDB Version 7.2 enhancements
8 Tailoring DataPropagator tables for OS/390
17 Utility lists and templates in DB2 V7
35 Editing UDB tables with DbVisualizer
42 Summary tables
48 DB2 news

DB2 Update

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

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

Subscriptions and back-issues
A year's subscription to *DB2 Update*, comprising twelve monthly issues, costs £255.00 in the UK; $380.00 in the USA and Canada; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1997 issue, are available separately to subscribers for £22.50 ($33.75) 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.

DB2 UDB Version 7.2 enhancements

Version 7.2 includes some noteworthy enhancements to e-business, business intelligence, data management, and the DB2 family. This article details specific enhancements by principal functional tool.

DB2 WAREHOUSE MANAGER (WM)

WM enhancements include:

- Visual Warehouse and the DB2 Control Center have been merged to provide a single user interface for business intelligence users. Launchpad's Wizard provides guidance for populating warehouses.
- Process Modeler lets users graphically link the steps needed to build and maintain data warehouses and dependent data marts. It can be one-time, repeated, or triggered by internal/external processes.
- Data Warehouse Center (DWC) makes it easier to create and change data warehouse target tables. DWC can automatically create new columns mapping results to those new columns.
- DWC has added Trillium tools to Vality and Evolutionary Technologies for performing name and address cleansing, matching, merging, and demographic augmentation.
- I certainly appreciate the seemingly minor enhancement of displaying red borders on required fields, such as database names, user IDs, passwords, etc. The border disappears on entry.
- Business Intelligence Tutorial is an online HTML document accessible from Information Center, DWC, and OLAP. Successful participants learn how to:
  - Define a data warehouse to contain the star schema.
  - Define security by specifying a warehouse user and a warehouse group.
  - Define a star dimension table by using warehouse sources of the tutorial, by moving them into the data warehouse.

  - Copy other dimension tables and fact tables from the tutorial.
  - Test and schedule data warehouse steps.
  - Define primary and foreign keys.
  - Perform maintenance to improve performance.
  - Authorize data warehouse users.
  - Publish metadata to the Information Catalog, enhancing it by associating a program with an object.
  - Define the star schema to the Data Warehouse Center, exporting it to the OLAP Integration Schema.

- DB2 Warehouse Manager Connectors (WMC) extend access to SAP R/3, i2 TradeMatrix BPI, web clickstream, MS OLE DB objects and Data Transactions Services targets, and MQSeries message queue data.
- Microsoft OLE DB data can be accessed using a wizard to create an MS OLE DB table function with its DB2 view.
- Common Warehouse Metadata Interchange (CWMI) metadata can be imported/exported. The standard is sponsored by the Object Management Group (OMG) backed by IBM, Hyperion, NCR, and Oracle. Metadata includes SAP WebSphere Site Analyzer and IBM ERwin. Interim commits are allowed.
- New metadata templates for primarykey.tag, primarykeyadditional.tag, foreignkey.tag, foreignkeyadditional .tag, and commit.tag.

E-VIDEO CENTRAL TUTORIAL

The new DB2 tutorial can be downloaded from http://www.ibm.com/ software/data/developer/samples/evideo for example solutions for B2B applications.

QMF

QMF for Windows provides a multipurpose query tool for business reporting, data sharing, server resource protection, robust application development, and native connectivity to DB2 workstation platforms.

QUERY PATROLLER (QP)

QP can trap SQL going to a DB2 Server by integration into client code, allowing dynamic SQL to be managed, scheduled, and governed. A query retry mechanism provides for the resubmission of aborted jobs. (Note: it requires Extended or Enterprise Edition.)

SQL

SQL enhancements include:

- SQL Assist helps users define over a hundred transforms using the improved DB2 rich SQL. It keeps a notebook for organizing information required to build an SQL statement including SELECT, INSERT, UPDATE, and DELETE.
- Warehouse Schema Modeler is a wizard for generating star schemas (see *Using a relational database for data warehousing*, Issue 83, *DB2 Update,* September 1999*)* for use within OLAP as a relational database.
- New SQL functions for moving aggregates of moving averages (calculating student grade point average) and moving sums (calculating bank statement balance column).
- Single SQL distributed query statement can access multiple heterogeneous data sources including Microsoft SQL Server, Oracle, and Sybase. SQL can use nicknames for heterogeneous tables.
- Frequently-used dynamic SQL can be converted to static for improved performance.
- Single connection temporary tables will be used for queries that can utilize intermediary tables for performance improvements.
- SQL now has column-type mechanisms for structured data, such as geometric shapes or hierarchical data (student-id, name, address, etc). Referential integrity constraints and triggers are available. Methods can be defined permitting behaviour encapsulation. SQL-bodied functions containing simple SQL procedure

statements can be embedded into a calling SQL. This lets the Query Compiler optimize entire SQL statements. Transform functions convert structured data types into ordered sets of their base SQL types and *vice versa*.

- SQL functions can now use control statements of FOR, GET DIAGNOSTICS, IF, ITERATE, LEAVE, and WHILE, atomic compound statements, and local variables.
- User-defined extended index types allow for user logic to determine how an index works. CREATE INDEX EXTENSION permits control of index maintenance, search, and exploration.
- I especially like the ability to update a partition key.
- SQL has three new scalar functions for decimal data types of:
  - ABS or ABSVAL, which returns an argument absolute value.
  - MULTIPLY_ALT, which returns the product of two arguments as a decimal value. Useful if precision is > 31 decimal places.
  - ROUND, which returns expression1 rounded to expression2.

DATABASE WIZARDS

V7.2 has added many database wizards including:

- Add Database – click *Add in Client Configuration Assistant*.
- Back up Database – right-click database selecting *Backup/ Database Using Wizard*.
- Configure Multisite Update – right-click Databases folder selecting *Multisite Update*.
- Create Database – right-click Databases folder selecting *Create/ Database Using Wizard*.
- Create Table – right-click Tables icon selecting *Create/Table Using Wizard*.
- Create Table Space – right-click Table Spaces icon selecting *Create/Table Space Using Wizard*.
- Create Index – right-click Index icon selecting *Create/Index Using Wizard*.

- Performance Configuration – right-click database selecting *Configure Performance Using Wizard*.
- Restore Database – right-click database selecting *Restore/Database Using Wizard*.

REFERENCES

- *DB2 Release Notes,* softcopy.
- *Administration Guide,* SBOF-8934. Includes three guides of *Planning, Implementation,* and *Performance.*
- *Data Warehouse Center Administration Guide,* SC26-9993.
- *Data Warehouse Center Application Integration Guide,* SC26-9994.
- *DB2 Warehouse Manager Installation Guide,* GC26-9998.
- *DB2 Connect User's Guide,* SC09-2945.
- *DB2 Connect Release Notes,* softcopy.
- *Glossary,* softcopy.

SUMMARY

Most V7.2 enhancements have been detailed in this article. Others include DB2 and DB2 Connect access to Microsoft SQL Server on AIX and Windows NT, Oracle on Solaris and Linux, Sybase on AIX and Solaris; global snapshots that let DBAs monitor their entire data warehouse; improved replication using DB2 Data Propagator, DB2 DataJoiner, data links, or ASNSAT command (Capture and Apply) on Windows 32-bit operating systems; Native OLE DB Support allowing DB2 to be both an OLE DB provider and consumer; and parallel creation or resizing of table space containers. These enhancements provide important new functionality. I wonder what IBM has in store when it announces V8!

Eric Garrigue Vesely
Principal/Analyst
Workbench Consulting (Malaysia)

Tailoring DataPropagator tables for OS/390

DataPropagator is the cornerstone of IBM's data replication solutions for relational databases.

When you create DB2 tables for DataPropagator you use either Control Center or the DataJoiner Replication Administration tool. They generate SQL to create control tables and 'Change Data' tables, but the SQL is not optimized for OS/390 DB2, so you have no control over the size, the STOGROUP, or BUFFERPOOLs to be used. They also create generic names for the tablespaces, tables, and indexes, which makes it hard to identify which CD (Change Data) table corresponds to which real table.

Here is a solution.

OVERVIEW

Even if you want to run DataPropagator only between DB2 subsystems on OS/390, you *must* install DB2 UDB and be able to connect to your OS/390 system to use either DJRA or Control Center. There is no other realistic way to administer DataPropagator.

Up to Version 7 of DataPropagator, the general advice is to use DJRA to generate the SQL because it offers more facilities and because it works (which is not always the case with Control Center). It can be downloaded from the Internet at no charge. It runs with DB2 UDB/EEE on NT, connecting to your OS/390 DB2 systems, generating SQL, and running it directly.

The solution is one of:

A – Transfer some generated SQL to OS/390.

- Use ISPF Edit and Edit macro(s) to tailor the SQL for your site.
- Run the SQL via SPUFI (or in batch).

B – Modify script(s) that generate SQL (on NT).

- Possibly rename scripts immediately before running DJRA.

– Run the SQL directly from DJRA.

C A combination of methods A and B.

Then you should have all your Dprop tables with sensible names, the correct size, on the right disks, and using the correct buffer pools.

RUNNING THE GENERATED SQL

You must use DJRA to generate some SQL into a file. That file can then be run either directly from DJRA on NT to create the tables on your DB2 OS/390 systems, or transferred to OS/390 to be run on the host system. If you run it directly, DJRA often overwrites existing files from the previous time you used it. If you run it on OS/390 you will need to modify the SQL.

Some of the SQL files CONNECT to more than one DB2 subsystem, and therefore it is best to run them directly from DJRA. There are, however, two which are good for running on OS/390, and they are the files for:

- Creating replication control tables.
- Defining replication sources.

The files are created in the *c:\Program\SQLLIB\DPRTOOLS* folder with a filename (like CRTABLES1.SQL or TABLEREG2.SQL), which DJRA shows you.

You can transfer them to OS/390 as you would transfer any text file.

MODIFYING SQL ON OS/390

The two SQL files must be modified in an ISPF Edit before they are run. As a minimum you must remove statements like:

```
CONNECT TO DB2X USER MYUSERID;
```

When there are many changes to make, an ISPF Edit macro can help.

Create replication control tables SQL

SQL must be used to create tables on the source, control, and target DB2 systems, although not all tables are used on each system.

Note that DJRA creates two tablespaces with different lock sizes as recommended by IBM, unlike the sample DDL which is supplied with DB2 UDB installation.

It makes sense to generate this file only once, transfer it to OS/390, and manually edit it to your standards, because there are only a small number of tables and indexes, and it never changes. Then you can copy it to all other required OS/390 systems and reuse it.

Define table(s) as replication sources SQL

SQL creates one or many tables/indexes on the DB2 source system where Dprop Capture runs. Each time you do this you will be creating CD tables for different source tables. Here is a sample ISPF Edit macro to modify the default DJRA file to local standards (and to clean out a lot of unnecessary comments etc):

```
/********************** REXX EDIT MACRO ******************************
 ** EDTREG:  macro for EDIT of DJRA SQL to create Dprop CD tables.   **
**********************************************************************/
    Trace O
    "ISPEXEC CONTROL ERRORS RETURN"        /* no macro error panel */
    Address ISREDIT
    "MACRO"
 /*-----------------------------------------------------------------*/
 /* New naming standards */
 /*-----------------------------------------------------------------*/
    sysadm = 'SYSADMØ1'                           /* userid to use */
    dbname = 'DPROPR'                       /* Dprop database name */
    ts_mask = "'TS'substr(srctable,2,6)"        /* new space name */
    tb_mask = "'CD_'srctable"                   /* new table name */
    ix_mask = "'CDI_'srctable"                  /* new index name */
 /*----------------------------------------------------------*/
 /* Comments to go at start of the SQL                       */
 /* Note: this includes the actual statements to be inserted */
 /*----------------------------------------------------------*/
    titl = ' Create DataPropagator Change Data table(s)/index(es)'
    com1 = ' Generated by DJRA then modified by EDTREG edit macro.'
    com2 = '   a) All unnecessary comments & blank lines were removed'
    com3 = '   b) All CONNECT & COMMIT statements were removed'
    com4 = '   c) Added "SET CURRENT SQLID = '''sysadm'''"'
    com5 = '   d) The database name was changed to:' dbname
    com6 = '   e) The names of all CD tablespaces, tables & indexes'
    com7 = '      were changed to reflect the source table name:'
    mas1 = '        space name = "'ts_mask'"'
    mas2 = '        table name = "'tb_mask'"'
    mas3 = '        index name = "'ix_mask'"'
    com8 = '   f) The following was added to each CREATE TABLESPACE:'
    add1 = '  USING STOGROUP SG2       '
    add2 = '    PRIQTY 10000           '
    add3 = '    SECQTY 2000            '
    add4 = '  BUFFERPOOL BP5;'
    com9 = '   g) The following was added to each CREATE INDEX:'
    ind1 = '  USING STOGROUP SG2       '
    ind2 = '    PRIQTY 5000            '
    ind3 = '    SECQTY 1000            '
    ind4 = '  BUFFERPOOL BP6;'
    not1 = ' Before running this SQL check the following carefully:'
    not2 = ' stogroup, sizes and bufferpools (some large tablespaces'
    not3 = ' may need a 32K bufferpool).'
/*-----------------------------------------------*/
/* Delete unwanted comments, CONNECTs, & COMMITs */
/*-----------------------------------------------*/
    "EXCLUDE ALL"
    "FIND 'CONNECT TO' 1  ALL"
    "FIND 'COMMIT;' 1 ALL"
    "FIND '–' 1 ALL"
    "EXCLUDE '– enable change data capture' ALL"
    "EXCLUDE '– create the cd/ccd table' ALL"
    "EXCLUDE '– create the index' ALL"
    "EXCLUDE '– insert a registration record' ALL"
    "DELETE ALL NX"
    "RESET"
/*---------------------------------------------------------*/
/* Ensure there are no more than 2 consecutive blank lines */
/*---------------------------------------------------------*/
    "(lline) = LINENUM .ZLAST"          /* get no. of the last line */
    Do linenum = lline to 1 By -1       /* check from the bottom up */
       "(linetext) = LINE" linenum
       If linetext <> '' Then Do
          one_blank = 0
          two_blank = 0
          Iterate linenum
          End
       If two_blank = 1 Then Do      /* two blank lines already ... */
          "DELETE" linenum           /* delete the extra blank line */
          Iterate linenum
          End
       If one_blank = 1
          Then two_blank = 1
          Else one_blank = 1
       End
/*-------------------------*/
/* Insert heading comments */
/*-------------------------*/
    "LOCATE .ZFIRST"      /* go to top */
    "LINE_AFTER .ZCSR = DATALINE ' '"
    'LINE_AFTER .ZCSR = DATALINE "SET CURRENT SQLID = ''sysadm'';"'
    "LINE_AFTER .ZCSR = DATALINE ' '"
    "LINE_AFTER .ZCSR = DATALINE '"Copies('-',60)"'"
    "LINE_AFTER .ZCSR = DATALINE '—"not3"'"
    "LINE_AFTER .ZCSR = DATALINE '—"not2"'"
    "LINE_AFTER .ZCSR = DATALINE '—"not1"'"
    "LINE_AFTER .ZCSR = DATALINE '—'"
    "LINE_AFTER .ZCSR = DATALINE '—      "ind4"'"
    "LINE_AFTER .ZCSR = DATALINE '—      "ind3"'"
    "LINE_AFTER .ZCSR = DATALINE '—      "ind2"'"
    "LINE_AFTER .ZCSR = DATALINE '—      "ind1"'"
    "LINE_AFTER .ZCSR = DATALINE '—"com9"'"
    "LINE_AFTER .ZCSR = DATALINE '—      "add4"'"
    "LINE_AFTER .ZCSR = DATALINE '—      "add3"'"
    "LINE_AFTER .ZCSR = DATALINE '—      "add2"'"
    "LINE_AFTER .ZCSR = DATALINE '—      "add1"'"
    "LINE_AFTER .ZCSR = DATALINE '—"com8"'"
    "LINE_AFTER .ZCSR = DATALINE '—"mas3"'"
    "LINE_AFTER .ZCSR = DATALINE '—"mas2"'"
    "LINE_AFTER .ZCSR = DATALINE '—"mas1"'"
    "LINE_AFTER .ZCSR = DATALINE '—"com7"'"
    "LINE_AFTER .ZCSR = DATALINE '—"com6"'"
    "LINE_AFTER .ZCSR = DATALINE '—"com5"'"
    "LINE_AFTER .ZCSR = DATALINE '—"com4"'"
    "LINE_AFTER .ZCSR = DATALINE '—"com3"'"
    "LINE_AFTER .ZCSR = DATALINE '—"com2"'"
    "LINE_AFTER .ZCSR = DATALINE '—"com1"'"
    "LINE_AFTER .ZCSR = DATALINE '—'"
    "LINE_AFTER .ZCSR = DATALINE '"Copies('-',60)"'"
    "LINE_AFTER .ZCSR = DATALINE '—"titl"'"
    "LINE_AFTER .ZCSR = DATALINE '"Copies('-',60)"'"
/*-------------------------------------------------------------*/
/* Update names of CD database, tablespaces, tables & indexes */
/*-------------------------------------------------------------*/
    "CHANGE DSNDB04" dbname "ALL"
    "SEEK ' ' FIRST"      /* go to top */
    Do i = 1 To 99999
       "SEEK C'CREATE TABLESPACE' 1"
       If rc <> 0 Then Leave i
       "(curline) = LINENUM .ZCSR"
       "(linetext) = LINE" curline
       Parse Var linetext ' TABLESPACE ' spname ' ' .

       "SEEK '— create the cd/ccd table for'"
       If rc <> 0 Then Leave i
       "(curline) = LINENUM .ZCSR"
       "(linetext) = LINE" curline
       Parse Var linetext '.' srctable ' ' .         /* source name */
       Interpret 'newspname = 'ts_mask            /* new space name */
       Interpret 'newtbname = 'tb_mask            /* new table name */
```

```
        Interpret 'newixname = 'ix_mask            /* new index name */
        curline = curline + 1
        "(linetext) = LINE" curline
        Parse Var linetext '.' cdtable '(' .      /* default CD name */
        cdindex = Overlay('CDI',cdtable)    /* default CD index name */
        "CHANGE" spname  newspname "ALL"
        "CHANGE" cdtable newtbname "ALL"
        "CHANGE" cdindex newixname "ALL"
        End
 /*-------------------------------------------------------*/
 /* Insert lines for BUFFERPOOL, PRIQTY, SECQTY, STOGROUP */
 /*-------------------------------------------------------*/
     "LOCATE .ZFIRST"      /* go to top */
     Do i = 1 To 999999
        /* TABLESPACE */
        "SEEK 'LOCKSIZE TABLE CLOSE NO;'"/* end of CREATE TABLESPACE
*/
        If rc = Ø Then Do
           "(linetext) = LINE .ZCSR"
           Parse Var linetext newtext ';' .
           "LINE .ZCSR = '"newtext"'"   /* same text but with no ';' */
           "LINE_AFTER .ZCSR = DATALINE '"add4"'"     /* BUFFERPOOL  */
           "LINE_AFTER .ZCSR = DATALINE '"add3"'"     /* SECQTY      */
           "LINE_AFTER .ZCSR = DATALINE '"add2"'"     /* PRIQTY      */
           "LINE_AFTER .ZCSR = DATALINE '"add1"'"     /* STOGROUP    */
           "(curline) = LINENUM .ZCSR"
           nextline = curline + 4
           "LOCATE" nextline      /* put cursor after inserted lines */
        /* INDEX */
           "SEEK 'IBMSNAP_INTENTSEQ ASC);'"  /* end of CREATE INDEX  */
           If rc = Ø Then Do
              "(linetext) = LINE .ZCSR"
              Parse Var linetext newtext ';' .
              "LINE .ZCSR = '"newtext"'"    /* same text without ';' */
              "LINE_AFTER .ZCSR = DATALINE '"ind4"'"  /* BUFFERPOOL  */
              "LINE_AFTER .ZCSR = DATALINE '"ind3"'"  /* SECQTY      */
              "LINE_AFTER .ZCSR = DATALINE '"ind2"'"  /* PRIQTY      */
              "LINE_AFTER .ZCSR = DATALINE '"ind1"'"  /* STOGROUP    */
              "(curline) = LINENUM .ZCSR"
              nextline = curline + 4
              "LOCATE" nextline   /* put cursor after inserted lines */
              End
           End
        Else Leave i
        End
 /*--------------------------------*/
 /* Finished, now redisplay the SQL */
 /*--------------------------------*/
     "LOCATE .ZFIRST"      /* go to top (to display the altered SQL) */
     "RESET"               /* clean it up ! */
     zedsmsg = 'SQL updated'
     Address ISPEXEC "SETMSG MSG(ISRZØØ1)"
     Return
```

The edit macro is put in your ISPEXEC or SYSPROC allocations, and it is invoked from an Edit of the SQL by the command EDTREG.

The result is much ‘cleaner’ than the original and there is a block of comments inserted at the beginning, detailing exactly what has been changed by the macro. If you (accidentally) run it twice, it will do no harm to the SQL.

To customize this macro to your local standards, update the code in the ‘New naming standards’ section (for tablespace, table, or index names) and/or the ‘Comments to go ...’ section (for stogroup, sizes, or bufferpools). If your standards are different for each ‘subscription set’ you can simply create a different edit macro for each.

TAILORING DJRA SCRIPTS

The scripts to generate the SQL are written in Object REXX, which is syntactically the same as mainframe REXX plus many object extensions.

There are two scripts which you could modify.

Create replication control tables script

Enter the DJRA application and select the *Create Replication Control Tables* button and then the *Edit Tablespace Logic* button. You then edit file *c:\Program\SQLLIB\DPRTOOLS\tblspace.rex*.

Save the original script under a different name (as a back-up). Then make the following changes.

Near the start of the code you will see these two lines:

```
CNTL_TS_NAME="TSCNTL"
UOW_TS_NAME="TSUOW"
```

These specify two tablespace names. Leave them as they are or change them to your local standards.

After a few comments you will see two lines to create these tablespaces and you can modify them as in the following example, where I have specified my database name (DPROPR) and added lines with details for the size, STOGROUP, and BUFFERPOOL:

```
SAY "CREATE TABLESPACE "||CNTL_TS_NAME;
SAY " IN DPROPR SEGSIZE 4 LOCKSIZE ROW CLOSE NO";
SAY "    USING STOGROUP SG1";
SAY "      PRIQTY 75ØØØ";
SAY "      SECQTY 15ØØØ";
SAY "    BUFFERPOOL BP3;";

SAY "CREATE TABLESPACE "||UOW_TS_NAME;
SAY " IN DPROPR SEGSIZE 4 LOCKSIZE TABLE CLOSE NO";
SAY "    USING STOGROUP SG1";
SAY "      PRIQTY 75ØØØ";
SAY "      SECQTY 15ØØØ";
SAY "    BUFFERPOOL BP8;";
```

Then, after a few more comments:

```
OUT.CNTL_TABLESPACE="DPROPR."||CNTL_TS_NAME;
OUT.UOW_TABLESPACE="DPROPR."||UOW_TS_NAME;
```

Unfortunately, that is all you can change in the script on NT. You must add size, STOGROUP, and BUFFERPOOL details for all indexes after the SQL has been generated. The SQL file can be edited in DJRA in the Notepad Editor before DJRA runs it. Alternatively, transfer it to OS/390 and modify it there before you run it.

As explained above, you could decide to leave this script as is, generate a file, edit it to your local standards, and reuse that SQL file for each DB2 system.

Define table(s) as replication sources script

Enter the DJRA application, select the *Define Tables as Replication Sources* button and then the *Edit Logic* button. You will then edit file *c:\Program\SQLLIB\DPRTOOLS\srcesvr.rex*.

Make changes like the following in the section for ‘DB2 FOR MVS’:

```
TSNAME = SUBSTR(IN.SOURCE_TABLE,2,6)
SAY "CREATE TABLESPACE TS"||TSNAME;
SAY " IN DPROPR SEGSIZE 4 LOCKSIZE TABLE CLOSE NO";
SAY "    USING STOGROUP SGØ1 PRIQTY 75ØØ SECQTY 15ØØ";
SAY " BUFFERPOOL BP5;";

OUT.CD_TABLESPACE="DPROPR.TS"||TSNAME
OUT.CD_TABLE="CD_"||IN.SOURCE_TABLE;
OUT.CD_INDEX="CDI_"||IN.SOURCE_TABLE;  /*INDEX NAME */
OUT.CD_INDEXPARMS="USING STOGROUP SGØ1 "||,
                  "PRIQTY 5ØØØ SECQTY 1ØØØ "||,
                  "BUFFERPOOL BP6";
```

This creates tablespace names starting with ‘TS’ and then 6 bytes of the table name. The Change Data table names are the same as the source table with ‘CD_’ added at the front, and their index names start with ‘CDI_’ followed by the source table name. It has specified a new database name (‘DPROPR’ instead of ‘DSNDB04’), and it has also added values to specify size, STOGROUP, and BUFFERPOOL instead of using the DB2 defaults.

You could create such a script for a particular ‘subscription set’ to your own standards and give it a different name. Then you can simply rename it to srcesvr.rex whenever you want to use it. That way you could keep a separate script for each subscription set (with different naming conventions etc) and use each of them when needed by renaming files.

CONCLUSION

I have explained two different strategies, and there are arguments for and against each approach, for example:

- Running a modified script directly from DJRA is faster and easier than copying the SQL to OS/390 and running it there.
- Copying the SQL to OS/390 and running it there allows you to keep it in a library, whereas DJRA reuses the filenames and overwrites previous SQL.
- DJRA must be installed on an individual PC and it uses the local folders, so a tailored script must be modified for each copy of DJRA.

I’ll leave you to decide which suits you best.

Meanwhile IBM is working on DataPropagator Version 8 with a new interface for this, possibly to be released some time in 2002 as part of DB2 UDB Version 8 for NT/AIX/etc. It will be a new drop-down from Control Center called ‘Replication Center’. Maybe IBM’s third attempt will be better!

Ron Brown (Consultant) © Xephon 2002

Utility lists and templates in DB2 V7

IBM made several improvements to the general usability of DB2 utilities in Version 7. Most of these improvements have been available from third-party ISVs for a number of years and IBM is just now starting to catch up.

DATABASE OBJECT LISTS

A database object list is created using the LISTDEF command. The purpose of LISTDEF is to allow DB2 utilities to execute against multiple database objects. Traditionally (prior to V7) a DB2 utility is run against a single database object. Each utility requires that a specific control card be set up to execute that utility against each database object. Or, alternatively, a separate utility job could be constructed for each database object. With the LISTDEF statement, the DBA can create a list of database objects that can be submitted to a single utility invocation for execution. So, as of V7, it is very simple to set up one utility execution to run against multiple database objects.

For example, the following statement creates a list named CUSTALL that includes all the database objects in the CUSTOMER database:

```
LISTDEF CUSTALL INCLUDE TABLESPACES DATABASE CUSTOMER
                INCLUDE INDEXSPACES DATABASE CUSTOMER
```

LISTDEF is not a utility; it is a control statement that can be used within other DB2 utilities. The LISTDEF statement can be used to assign a name to a list of DB2 database objects and previously defined lists of database objects – so, a list can consist of other lists. Once the list is defined, it can be used when executing other utilities. Each list must be named, and that name can be up to 18 characters in length.

To run a DB2 utility against a list, you have the option of putting the LISTDEF statements in a separate library dataset or coding it directly before the DB2 utility control statement that refers to the list. The default DDname for a LISTDEF dataset is SYSLISTD, but this can be changed using the OPTIONS LISTDEFDD control statement.

To run LISTDEF as part of a utility execution, the process or user running the utility must have SELECT authority on SYSIBM.SYSINDEXES, SYSIBM.SYSTABLES, and SYSIBM .SYSTABLESPACE. Of course, the process or user running the utility must also have the requisite authority to execute the utility being used to process the list.

Because LISTDEF is a control statement, not an individual utility, LISTDEF conforms to the concurrency rules for the utility to which the list is being applied. The LISTDEF list is stored until it is referenced by a specific utility, at which time the list is expanded. At that time, the concurrency and compatibility restrictions of the utility being executed apply, with the additional restriction that the catalog tables necessary to expand the list must be available for read-only access.

CREATING LISTS WITH LISTDEF

The INCLUDE and EXCLUDE keywords are used to build the list of database objects:

- INCLUDE – specifies database objects to add to the list.
- EXCLUDE – specifies database objects to remove from the list.

The LISTDEF statement can consist of multiple INCLUDE and EXCLUDE clauses. At least one INCLUDE clause must be specified. Furthermore, an INCLUDE clause must be coded before any subsequent EXCLUDE clauses. For most utilities, the INCLUDE and EXCLUDE clauses will be processed in the order they are coded in the LISTDEF statement. Certain specific utilities will modify the order of the list to optimize its processing:

- The CHECK INDEX, REBUILD INDEX, and RUNSTATS INDEX utilities will process all index spaces that are related to a given table space at once, regardless of the order in which the indexes appear in the list.
- The UNLOAD utility will process all specified partitions of a given table space at one time regardless of the order in which the indexes appear in the list.

The list will be built a clause at a time, by adding database objects to the list or removing database objects from the list. Any EXCLUDE statements that try to remove a database object from the list that has not yet been added to the list will cause DB2 to ignore that database object and proceed to the next INCLUDE or EXCLUDE clause. Be aware that a subsequent INCLUDE can return the excluded object to the list.

Furthermore, INCLUDE and EXCLUDE statements are typed – meaning that they can be set to include or exclude only table spaces or only index spaces. Of course, typing INCLUDE and EXCLUDE statements is optional. If the type is not specified the statement will default to a particular type of object based on the subsequent keywords that are coded. The available keywords for including and excluding database objects from a list include the following:

- DATABASE – to specify table spaces, index spaces, or both from a particular database or databases.
- TABLESPACE – to specify table spaces, index spaces, or both for a particular table space or table spaces.
- TABLE – to specify table spaces, index spaces, or both from a particular table space or table spaces.
- INDEXSPACE – to specify index spaces.
- INDEX – to specify index spaces.
- LIST – to specify a list of table spaces, index spaces, or both, that were previously defined using LISTDEF.

Additionally, the PARTLEVEL keyword can be used to specify only certain partitions for inclusion. The PARTLEVEL keyword applies to both table spaces and index spaces, and is ignored for non-partitioned database objects. The PARTLEVEL keyword takes an integer parameter that specifies the partition to be included. Failure to specify the partition number causes a list to be generated that contains one entry for every existing partition of the table space or index space.

Be aware that the only way to exclude a partition is if it was first included explicitly as a partition. For example, the following LISTDEF statement will exclude partition 7 from the list because both statements are coded at the partition level:

```
LISTDEF LST1 INCLUDE TABLESPACE DB.TS1 PARTLEVEL
             EXCLUDE TABLESPACE DB.TS1 PARTLEVEL(7)
```

But the next LISTDEF statement will not exclude partition 7 from the list because the INCLUDE statement was specified at the table space level, but the EXCLUDE statement is specified at the partition level:

```
LISTDEF LST2 INCLUDE TABLESPACE DB.TS1
             EXCLUDE TABLESPACE DB.TS1 PARTLEVEL(7)
```

WILDCARDING

The LISTDEF statement can use wildcarding to rapidly specify multiple database objects without having to explicitly name each of the objects. For example, you might specify:

```
LISTDEF LST3 INCLUDE TABLESPACE DBXN.*
             EXCLUDE TABLESPACE DBXN.TS2

REORG LIST LST3 . . .
```

This sequence of statements will reorganize all table spaces in the database named DBXN except for the one table space exempted, namely TS2. Furthermore, if a table space is subsequently added to DBXN, the REORG job does not need to be changed. The next time it runs, REORG will query the DB2 Catalog to determine the table spaces that exist in the list named DB1. Since it specifies all table spaces in DBXN, any new table space added to DBXN will automatically be picked up for processing.

The valid wildcard options supported by LISTDEF are as follows:

- The question mark (?), which can be used to indicate any single character.
- The underscore (_), which can be used to indicate any single character.
- The per cent sign (%), which can be used to indicate a string of 0, 1, or many characters.
- The asterisk (*), which can be used to indicate a string of 0, 1, or many characters.

These options were chosen by IBM to mimic the wildcarding capability of the SQL LIKE clause. However, the underscore and per cent sign characters are commonly occurring characters within database object names, so additional wildcard character options were provided.

Be sure to use the question mark (?) wildcard character instead of the underscore (_) character for pattern-matching in table and index names. For table and index names the underscore character represents a single occurrence of itself. The underscore can be used as an alternative to the question mark for database, table space, and index space names.

Although wildcarding is a very powerful capability, DB2 does place some limits on its use. For example, it is not permissible to create all-inclusive lists, such as DATABASE * or TABLESPACE *.*.

The bottom line, though, is the ability to create lists using wildcards, which greatly eases the DBA's utility creation and management tasks.

LIST EXPANSION

Lists created using the LISTDEF statement are expanded each time the utility job in which the LISTDEF is included is executed. This is important because it simplifies the task of creating utility jobs for new database objects. For example, suppose you have a utility job set up to make image copies for every table space within a given database, such as:

```
LISTDEF LSTX INCLUDE TABLESPACE DBGLØØX1.*

COPY LIST LSTX …
```

This sequence of commands is set up to copy every table space in the DBGL00X1 database. If you add a new table space to that database you do not need to create a new image copy job or modify the existing job. This is so because the next time the existing job runs the list will be expanded and will then include the new table space – which will cause it to be copied. Lists (and templates), therefore, can greatly simplify the DBA task of creating utility jobs for new database objects.

REFERENTIAL INTEGRITY AND LISTS

The RI parameter can be included on the INCLUDE statement to cause all tables that are referentially connected to tables specified in the list to also be included. This relieves the DBA of the burden of manually determining which tables are connected to which. Be advised, however, that the RI parameter cannot be used in conjunction with the PARTLEVEL parameter.

LOBS AND LISTS

Database objects containing LOBs are special and may need to be treated differently from other database objects for utility processing. The LISTDEF statement provides options that enable the DBA to handle database objects that use LOBs differently.

LISTDEF offers three options that can be specified on an INCLUDE or EXCLUDE specification, to indicate how to process the LOB-related objects. The auxiliary LOB relationship can be traversed in either direction:

- From LOB auxiliary table to base table.
- From base table to LOB auxiliary table.

LOB objects include LOB table spaces, auxiliary tables, indexes on auxiliary tables, and their associated index spaces. Failure to specify one of the following three keyword options will cause DB2 not to follow the auxiliary relationships and it will not filter LOB from BASE objects in the enumerated list.

- ALL – indicates that both BASE and LOB objects are to be included in the list. Auxiliary relationships will be followed from all objects resulting from the initial object look-up and both BASE and LOB objects will remain in the final enumerated list.
- BASE – indicates that only base table spaces and index spaces will be included in the list; that is, no LOB objects will be included for this specific INCLUDE or EXCLUDE. Of course, you can specify a LOB object to begin with, causing LISTDEF to search for the related base objects and only the base objects will be included in the list.

- LOB – indicates that only LOB table spaces and related index spaces containing indexes on auxiliary tables are to be included in the list. Once again, you can specify a base object or a LOB object to begin with, but only LOB objects will be included in the list.

INDEXES, LISTS, AND THE COPY UTILITY

An additional consideration for lists built using the LISTDEF statement is the treatment of indexes with regard to the COPY utility. As of DB2 V6 it has been possible to take image copy back-ups of index spaces to be used for subsequent recovery; previously, indexes had to be rebuilt from table data after table spaces were recovered.

LISTDEF offers the COPY keyword, which can be used to specify whether indexes that are defined as COPY YES (or COPY NO) are to be included or excluded in the list. If the COPY keyword is not coded, all index spaces that satisfy the INCLUDE or EXCLUDE expression, regardless of their COPY attribute, will be included or excluded in the list. If specified, this keyword must immediately follow the INDEXSPACES keyword. If specified elsewhere, the keyword COPY is interpreted as the start of the COPY utility control statement. The COPY keyword can take on one of two values:

- YES – indicates that index spaces defined with or altered to COPY YES are the only ones to be included in the list.
- NO – indicates that index spaces defined with or altered to COPY NO are the only ones to be included in the list.

You can use INCLUDE with COPY YES to create a list of index spaces that can be processed by the COPY utility. Or, you can use EXCLUDE with COPY NO to remove index spaces that cannot be processed by the COPY utility from a list of index spaces.

LIST USAGE GUIDELINES

The whole purpose of using LISTDEF to create lists of database objects is to use those lists when executing DB2 utilities. Once created, the LIST keyword can be specified by utilities to process the objects on the list. The LIST keyword can be used with the following utilities:

- CHECK INDEX – during execution, the database objects in the list will be grouped by related table space for processing.
- COPY – during execution, the database objects in the list will be processed in the order they were specified.
- MERGECOPY – during execution, the database objects in the list will be processed in the order they were specified.
- MODIFY RECOVERY – during execution, the database objects in the list will be processed in the order they were specified.
- MODIFY STATISTICS – during execution, the database objects in the list will be processed in the order they were specified.
- QUIESCE – during execution, the database objects in the list will be processed in the order they were specified.
- REBUILD – during execution, the database objects in the list will be grouped by related table space for processing.
- RECOVER – during execution, the database objects in the list will be processed in the order they were specified.
- REORG – during execution, the database objects in the list will be processed in the order they were specified.
- REPORT – during execution, the database objects in the list will be processed in the order they were specified.
- RUNSTATS INDEX – during execution, the database objects in the list will be grouped by related table space for processing.
- RUNSTATS TABLESPACE – during execution, the database objects in the list will be processed in the order they were specified.

Certain utilities, such as RECOVER and COPY, can process a LIST without a specified database object type. These utilities will derive the database object type to process based on the contents of the list. But other utilities, such as QUIESCE and REORG INDEX, must have the database object type specified before the utility can be executed. For these utilities you must specify an object type in addition to the LIST keyword, for example:

```
QUIESCE TABLESPACE LIST list-name.
```

Additionally, most utilities require output datasets in order to process. You can use the TEMPLATE statement to specify the naming convention and, optionally, the allocation parameters for each output dataset type. Templates, like lists, can be reused as long as the naming convention can prohibit the specification of duplicate dataset names. Although it may be possible to use traditional JCL with certain LISTDEF lists, it is not recommended because the JCL quickly becomes unwieldy and impractical to maintain (unless the list of database objects to be processed is quite small).

When used together, LISTDEF and TEMPLATE statements make it easier to develop DB2 utility jobs. Job creation is faster, lists can be coded to capture and process new database objects automatically, and the resulting JCL is easier to maintain because fewer modifications are required when database objects are added to the list.

PREVIEWING THE CONTENTS OF A LIST

You can use the OPTIONS PREVIEW control statement to expand the list and see the database objects that will be included in the list before actual processing. The OPTIONS ITEMERROR control statement can be used to alter the handling of errors that may occur during list processing. The OPTIONS LISTDEFDD control statement may be used to switch LISTDEF library datasets between control statements within a job step. The default is LISTDEFDD SYSLISTD.

LISTDEF EXAMPLE

The following example shows the JCL and control statements used to build a list of table spaces to QUIESCE. The list created is named QLIST. This list includes all of the table spaces in database DB1 that begin with the characters ‘TSL’ except for TSLY028A, TSLY066V, and TSLU071X. Also, one table space is included from database DB7, namely TSLU077T.

```
 //QUIESCE JOB 'USER=NAME',CLASS=A,...
//******************************************************
//*QUIESCE LISTDEF DD LILSTDEF datasets
//******************************************************
```

```
//STEP1   EXEC  DSNUPROC,UID='DBAPSCM.QUIESCL',
//           UTPROC='',SYSTEM='DB2A'
//LISTDSN DD DSN=JULTU1Ø3.TCASE.DATA2,DISP=SHR
//        DD DSN=JULTU1Ø3.TCASE.DATA3(MEM1),DISP=SHR
//SYSUT1  DD DSN=JULTU1Ø3.QUIESC2.STEP1.SYSUT1,
//        DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//        SPACE=(4ØØØ,(2Ø,2Ø),,,ROUND)
//SORTOUT DD DSN=JULTU1Ø3.QUIESC2.STEP1.SORTOUT,
//        DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,
//        SPACE=(4ØØØ,(2Ø,2Ø),,,ROUND)
//SYSIN   DD *

    LISTDEF QLIST INCLUDE TABLESPACE DB1.TSL*
                  EXCLUDE TABLESPACE DB1.TSLYØ28A
                  EXCLUDE TABLESPACE DB1.TSLYØ66V
                  EXCLUDE TABLESPACE DB1.TSLUØ71X
                  INCLUDE TABLESPACE DB7.TSLUØ77T

    QUIESCE LIST QLIST

/*
```

TEMPLATES

In order to support database object lists and wildcarding, IBM also had to come up with a way of allocating datasets to support utility processing on a large number of unknown database objects – well, at least unknown at the time the utility JCL is being built. The new TEMPLATE statement accomplishes this.

The purpose of the TEMPLATE statement is to provide DB2 utilities with the basic allocation information necessary to automatically generate and allocate valid datasets that are required for use as DB2 utilities are executed. Traditionally a DB2 utility is run against a single database object and the JCL is manually prepared with each required dataset hard-coded into the JCL jobstream. However, with the advent of LISTDEF, DB2 utility jobs can be set up to operate on multiple database objects with a single run. Furthermore, the person developing the JCL has no way of knowing how many database objects will be processed, so it is not practical to manually allocate datasets to such utility jobs. And because of the nature of the LISTDEF statement, it is possible that a different number of database objects will be processed each time a utility is run using the same list. So, once again, allocating datasets to the JCL in such an environment is impractical, if it is even possible at all.

Like LISTDEF, TEMPLATE is not a new utility, but is a new control statement. The TEMPLATE statement lets you allocate datasets, without using JCL DD cards, during the processing of a LISTDEF list. Using TEMPLATE the developer can define the dataset naming convention, and specify further allocation parameters such as dataset sizing, location, and attributes. The TEMPLATE statement is flexible enough to allow different characteristics for tape and disk datasets. Each TEMPLATE statement generates a named template to be referenced as needed.

To perform the dynamic dataset allocation during the utility execution, the TEMPLATE control statement deploys the MVS DYNALLOC macro (SVC 99).

To use a template with a DB2 utility, you have the option of putting the TEMPLATE statements in a separate library dataset or directly before the DB2 utility control statements. The default DD name for a TEMPLATE dataset is SYSTEMPL, but this can be changed using the OPTIONS TEMPLATEDD control statement.

No additional privileges are required to run TEMPLATE as part of a utility execution. Of course, the process or user running the utility also must have the requisite authority to execute the utility for which the template is being used.

USING TEMPLATE

Each TEMPLATE is named, with the name limited to eight alphanumeric characters, the first of which must be an alphabetic character (A to Z). After the template name, you can specify keywords to control the allocation of tape and disk datasets. Each TEMPLATE statement can apply to either disk or tape, but not both. The UNIT keyword specifies a generic unit name that must be defined on your system. Additional keywords specified for the TEMPLATE statement must be consistent with the unit type specified; that is, valid for tape or disk.

There are three groupings of options that can be specified for a TEMPLATE statement:

- Common options – those that apply to both disk and tape templates.
- Disk options – those that apply only to a disk template.
- Tape options – those that apply only to a tape template.

A single TEMPLATE statement can have a set of common options, and then either a set of disk options or a set of tape options.

COMMON TEMPLATE OPTIONS

The following keywords are common options that can be applied to the definition of any TEMPLATE statement:

- UNIT – indicates the device-number, the generic device-type, or the group-name to use for allocating the dataset.
- DSN – indicates the template for the MVS dataset name. The dataset name can be created by using any combination of symbolic variables, alphanumeric constants, or national characters. When creating the DSN template, follow the same basic rules for using symbolic variables within JCL. Special DB2 symbolic variables are provided that can be used to insert characteristics of the database object, utility, or job that is running. These symbolic variables are summarized below. Each symbolic variable is substituted by its related value at execution time to form an explicit dataset name. When used in a DSN expression, the symbolic variable begins with the ampersand sign (&) and ends with a full stop (period). For example:

  ```
  DSN &DB..&TS..D&JDATE..COPY&ICTYPE.
  ```

- This DSN parameter creates a dataset named using:
  - The database name as the first part.
  - The table space name as the second part.
  - The character ‘D’ followed by the actual date as the third part.

    – Lastly the word ‘COPY’ followed by the letter ‘F’ (for a full image copy), the letter ‘C’ (for a CHANGELIMIT image copy), or the letter ‘I’ (for an incremental image copy) as the fourth and final part of the name.

- DISP – indicates the dataset disposition using standard JCL parameters for status, normal termination, and abnormal termination. The legal values for each are as follows:
    - status – NEW, OLD, SHR, MOD.
    - normal termination – DELETE, KEEP, CATLG, UNCATLG.
    - abnormal termination – DELETE, KEEP, CATLG, UNCATLG.
- MODELDCB – indicate the DSN of a model dataset that will be used to get the DCB information.
- BUFNO – indicates the number of BSAM buffers to use for dataset buffering.
- DATACLAS – indicates the name of the SMS data class to use. If specified, this value must be a valid data class defined to SMS.
- MGMTCLAS – indicates the name of the SMS management class to use. If specified, this value must be a valid management class defined to SMS.
- STORCLAS – indicates the name of the SMS storage class to use. If specified, this value must be a valid storage class defined to SMS.
- RETPD – indicates the retention period for the dataset in days. The value must be between 1 and 9999 if specified.
- EXPDL – indicates the expiration date for the dataset. The date must be enclosed in single quotes and specified in Julian data format, that is YYYYDDD.
- VOLUMES – to provide a list of volume serial numbers for this allocation. The volume serial numbers are provided in a comma-delimited list and enclosed within parentheses. The first volume on the list must contain sufficient space for the primary allocation of space for the dataset.

- VOLCNT – indicates the maximum number of volumes that an output dataset might require.
- GDGLIMIT – indicates the number of entries to be created in a GDG base if the DSN specifies a GDG and the GDG base does not already exist. The value of GDGLIMIT can range from 0 to 255, where 0 is a special value to specify that a GDG base should be created if one does not already exist.

DB2 TEMPLATE symbolics are described below:

- &JOBNAME (or &JO) – the name of the OS/390 (or z/OS) job.
- &STEPNAME (or &ST) – the name of the OS/390 (or z/OS) job step.
- &UTILID (or &UT) – the utility ID for this utility job (truncated to eight characters or the first full stop in the utility ID).
- &USERID (or &…) – the userid of the OS/390 (or z/OS) job.
- &SSID (or &SS) – the DB2 subsystem identifier.
- &UTILNAME (or &UN) – the utility name for this utility job (truncated to eight characters).
- &LIST (or &LI) – the name of the list.
- &SEQ (or &SQ) – the sequence number of the item in the list being processed.
- &JDATE (or &JU) – the Julian date, formatted as YYYYDDD. Must be prefixed with a character number when used (eg D&JDATE).
- &JDAY (or &JD) – the day portion of the Julian date, formatted as DDD. Must be prefixed with a character number when used (eg D&JDAY).
- &TIME – the system time, formatted as HHMMSS. Must be prefixed with a character number when used (eg T&TIME).
- &HOUR (or &HO) – the hour component of the system time, formatted as HH. Must be prefixed with a character number when used (eg T&HOUR).

- &MINUTE (or &MI) – the minute component of the system time, formatted as MM. Must be prefixed with a character number when used (eg T&MINUTE).
- &SECOND (or &SC) – the second component of the system time, formatted as SS. Must be prefixed with a character number when used (eg T&SECOND).
- &YEAR – the year, formatted as YYYY. Must be prefixed with a character number when used (eg Y&YEAR).
- &MONTH – the month, formatted as MM. Must be prefixed with a character number when used (eg M&MONTH).
- &DAY – the day of the month, formatted as DD. Must be prefixed with a character number when used (eg D&DAY).
- &DB – the database name.
- &TS – the table space name.
- &IS – the index space name.
- &SN – the space name, either table space or index space depending on which is being processed.
- &PART (or &PA) – the partition number, formatted as five digits, left-padded.
- &ICTYPE (or &IC) – the image copy type – F for full copy, C for CHANGELIMIT copy, or I for incremental copy.
- &LOCREM (or &LR) – the location of the image copy – either L for local copy or R for remote copy.
- &PRIBAC (or &PB) – the type of copy – either P for primary copy or B for back-up copy.

DISK OPTIONS

The following keywords are disk options that can be applied only to the definition of a statement defining a disk TEMPLATE:

- SPACE – indicates the primary and secondary disk space allocation values for the DSN template. The option takes two values, placed between parentheses and separated by a comma: the first is the primary allocation and the second is the secondary allocation.

  - CYL – indicates that allocation quantities are specified in terms of cylinders and allocation will occur in cylinders.
  - TRK – indicates that allocation quantities are specified in terms of tracks and allocation will occur in tracks.
  - MB – indicates that allocation quantities are specified in terms of megabytes and allocation will occur in records.

- PCTPRIME – indicates the percentage of space to be obtained as the primary quantity.
- MAXPRIME – specifies an upper limit on the primary quantity specified using the SPACE parameter (expressed in the units specified on the SPACE parameter).
- NBRSECOND – indicates the division of secondary space allocations. After the primary space is allocated, the remaining space is divided into the specified number of secondary allocations. The NBRSECOND value can range from 1 to 10.

TAPE OPTIONS

The following keywords are disk options that can only be applied to the definition of a statement defining a disk TEMPLATE:

- UNCNT – the number of devices to be allocated. If a specific device number is coded on the UNIT common option, then UNCNT must be 1 (or not coded).
- STACK – indicates whether output datasets are to be stacked contiguously on tape. Valid values are YES or NO.
- JES3DD – indicates the name of the JCL DD to be used at job initialization time for the tape unit. (JES3 requires that all necessary tape units are pre-allocated by DD statement.)
- TRTCH – indicates the track recording technique (for tape drives

with improved data recording capabilities). Valid values are:

- NONE – indicates that TRTCH is eliminated from dynamic allocation.
- COMP – indicates that data is to be written in compacted format.
- NOCOMP – indicates that data is to be written in standard format (that is, not compacted).

TEMPLATE EXAMPLES

The following statement shows a disk template with an explicit specification of space. The dataset name will consist of the database name, the table space name, the name of the utility it was used with, and the time of allocation. Allocation is by cylinder with a primary space of 115 and a secondary of 15:

```
TEMPLATE DISKTMP2 DSN(&DB..&TS..&UTILNAME..T&TIME.)
     SPACE(115,15) CYL
```

This next example shows how to use TEMPLATE in conjunction with LISTDEF. The LISTDEF creates a list of table spaces to COPY. Two TEMPLATEs are specified, one for a disk local copy and one for a tape remote copy. The tape and disk template each specify different UNIT values, the disk template specifies the dataset disposition, and the tape template specifies an expiration date of 2 January 2005 (in Julian date format):

```
LISTDEF  COPYLIST INCLUDE TABLESPACE DBGLØ1.*
                  EXCLUDE TABLESPACE DBGLØ1.TSNAME
TEMPLATE DISKTMPL UNIT SYSDA
                  DSN(&DB..&TS..COPY&IC.&LR.&PB..D&DATE..T&TIME.)
                  DISP (MOD,CATLG,CATLG)
TEMPLATE TAPETMPL UNIT T359Ø1
                  DSN(&DB..&TS..COPY&IC.&LR.&PB..D&DATE..T&TIME.)
                  EXPDL '2ØØ5ØØ2'
COPY LIST COPYLIST COPYDDN (DISKTMPL) RECOVERYDDN (TAPETMPL)
     SHRLEVEL REFERENCE
```

TESTING OPTIONS FOR LISTS AND TEMPLATES

Finally, the OPTIONS utility control statement is new and can be used to indicate processing options that apply across many utility executions. By specifying various options you can:

- Preview utility control statements.
- Override library names for LISTDEF lists or TEMPLATEs.
- Specify how to handle errors during list processing.
- Alter the return code for warning messages.
- Restore all default options.

The processing options set using the OPTIONS statement remain in effect for the duration of the job step or until they are over-ridden by another OPTIONS statement in the same job step.

The OPTIONS control statement requires no specific privileges to be specified. Of course, the process or user running the utility must have the requisite authority to execute the actual utility that is being executed with the OPTIONS statement.

OPTIONS is not a utility itself, but a control statement used to set up the environment for other utilities to run. The OPTIONS statement is simply stored until it is referenced by a DB2 utility. When the OPTIONS statement is referenced, the list specified for that utility will be expanded and the concurrency and compatibility restrictions of that utility will apply. In addition, the catalog tables necessary to expand the list also must be available for read-only access.

USING OPTIONS

The PREVIEW keyword is similar to the JCL PREVIEW parameter. It is used to check for syntax errors in subsequent utility control cards and will not execute the utilities. For example, PREVIEW can be used in conjunction with LISTDEF to expand a list in order to verify that the proper database objects have been included in the list.

The OFF keyword can be used to restore to the default options. Using OPTIONS OFF will not override the PREVIEW JCL parameter which, if specified, remains in effect for the entire job step. When specifying OPTIONS OFF, no other OPTIONS keywords may be used.

SUMMARY

DB2 Version 7 provides new capabilities for utility execution that make running and managing IBM's DB2 utilities easier. Many organizations using third-party ISV DB2 utilities have experienced these capabilities (and many others) already. But now that IBM offers support for lists and templates the functionality is available to all.

Craig S Mullins
Director, DB2 Technology Planning
BMC Software (USA)

Editing UDB tables with DbVisualizer

INTRODUCTION

When developing or maintaining database applications, a common requirement of the application developer is to directly edit the DB2 UDB tables. There are several reasons why a developer would want to do such a thing. A primary reason is that it enables the developer to change data quickly and easily during unit testing. A table editor removes the need to write long and complex SQL queries. It may also remove the need to reload or restore an entire tablespace, when perhaps only a few rows need to be reset for the purpose of a test.

DbVisualizer, from Minq Software, is a shareware product that provides an SQL interface to relational databases, such as UDB, and also contains a table editor function. Minq Software claims that this tool is a 100% pure Java product that can handle multiple database connections. The product enables the user to issue SQL requests to the connected databases, and its colourful SQL editor aids the user in this process. It provides a graphical view of relationships, including referential integrity (RI) constraints. Result tables can be saved in HTML format or in ASCII file format. DbVisualizer contains a table editor function that allows basic table editing. It interfaces to the database using JDBC.

The supplier claims that the product has been tested on a number of database management systems, including DB2 UDB, using IBM's

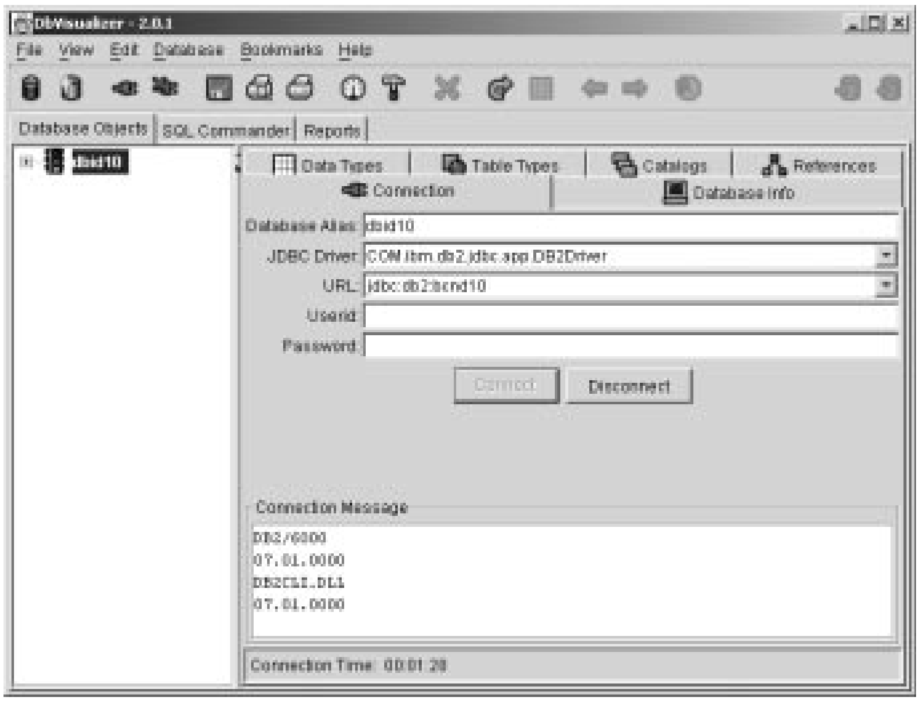
DB2 JDBC driver. Because the product is free to use, even for commercial use, no guaranteed support for the product is provided, nor is any documentation available.

The product can be downloaded free of charge from http://www.minq.se/innovations/dbvis/.

PRODUCT OVERVIEW

DbVisualizer incurs no direct cost to deploy. For the purposes of testing and development, the DbVisualizer product could prove to be a useful tool while saving your company unnecessary expenditure.

DbVisualizer has an MS Office look and feel with the familiar navigation bar (nav bar) window permanently on the left of the screen. On entry, the user is able to select from the nav bar a UDB instance that was previously defined by the user. On selecting the instance, the user is prompted to connect to the instance. The user can enter a userid and password or these fields can be left blank if the user's network userid and password are to default. Once connection is made to the UDB



Database Alias: dbid10
JDBC Driver: COM.ibm.db2.jdbc.app.DB2Driver
URL: jdbc:db2:bcnd10
Userid:
Password:

```
DB2/6000
07.01.0000
DB2CLI.DLL
07.01.0000
```

Connection Time: 00:01:20

Figure 1: Connection prompt

instance, further navigation can take place. Figure 1 shows the connection prompt.

When the UDB instance is double-clicked, the nav bar expands to show the databases within the selected UDB instance. Selecting a database by single-clicking the database icon prompts DbVisualizer to collect object information from the database, which enables an entity relationship diagram to be drawn. Figure 2 shows a typical entity relationship diagram that is displayed when a database icon is selected.

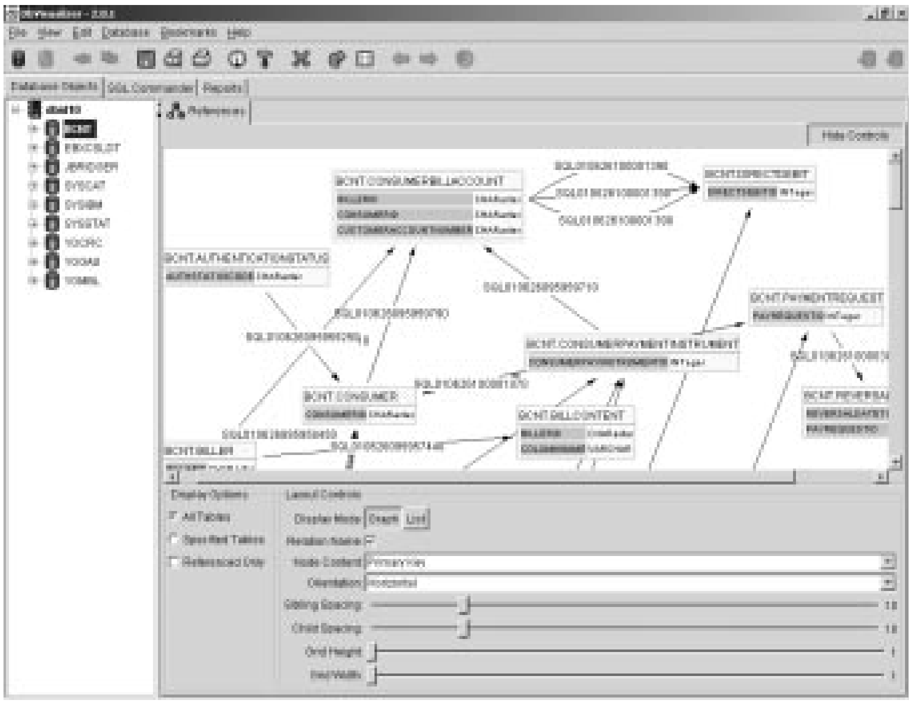


Figure 2: Entity relationship diagram

This interesting feature of DbVisualizer gives each application developer an accurate and up-to-date entity relationship diagram at a very low cost. The information to build the diagram is obtained from the UDB system catalog tables.

The user is able to move the objects around the screen in order to ensure a clearer diagram, and to focus on the area of interest. Scroll bars, both up and down and left and right, assist in viewing the diagram. The user is able to select only a subset of tables or views to display if required.

A display mode of *List* as well as *Graph* (as shown in Figure 2) enables the user to select the most appropriate display mode.

Double-clicking the database icon on the nav bar, and further expanding the database tree, displays the tables and views available within the selected database – shown in Figure 3.

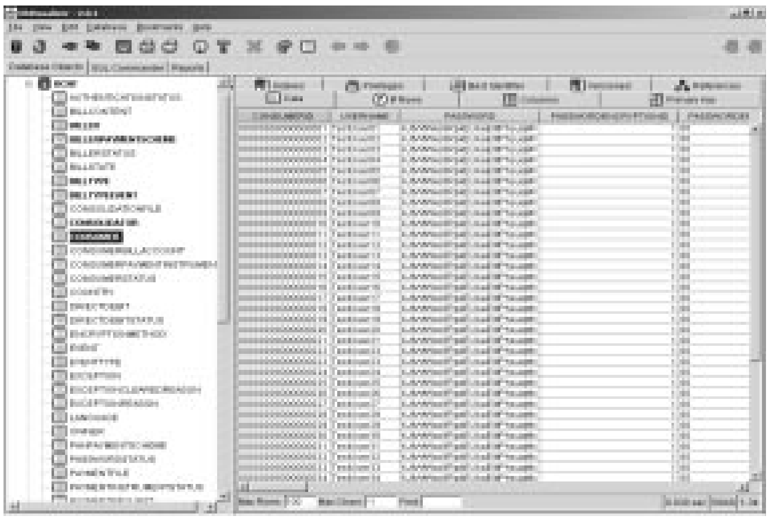


Figure 3: Table and view display

Selecting a table or view within the nav bar pulls up a number of tabs, allowing the user to select different information from the object. Tabs include raw data, number of rows, column information, details of the primary key, index column information, privileges, and some other object data.

By selecting the data tab and right-clicking a data cell, the user is able to automatically generate SQL code that can be manipulated to perform an SQL function on the data. It is by using this method that a developer is able to update or edit data within a UDB table or view – see Figure 4.

By selecting the *update where* option on the pull-down menu, DbVisualizer will generate the appropriate SQL statement required to update this row and this column. The SQL Commander tab is then

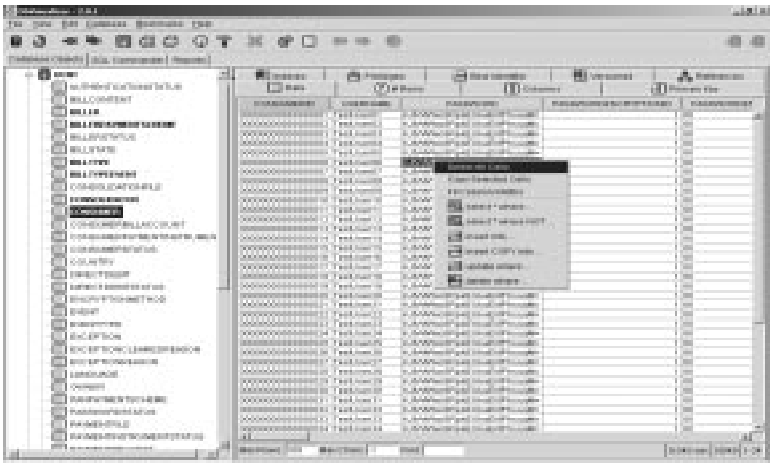


Figure 4: Right-click pull-down menu to generate SQL code

automatically displayed, and the user is able to modify the SQL code as required. Figure 5 shows the SQL Commander screen.

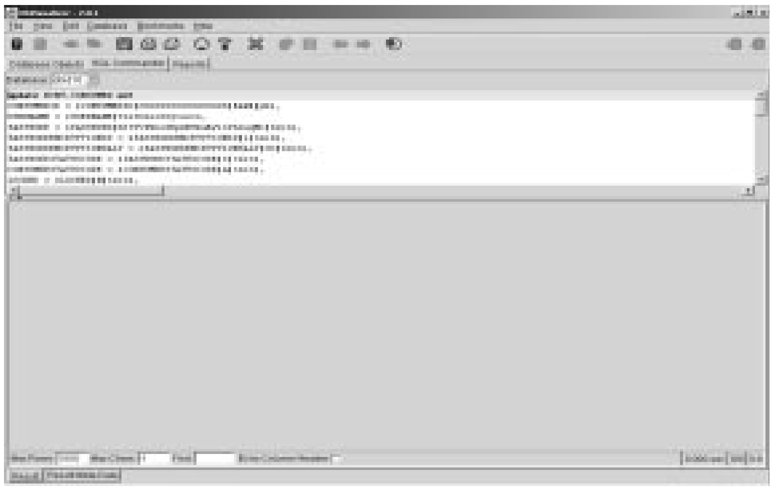
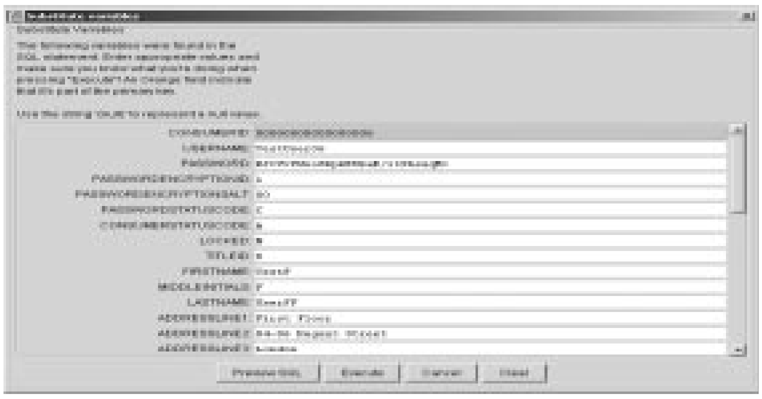


Figure 5: SQL Commander screen

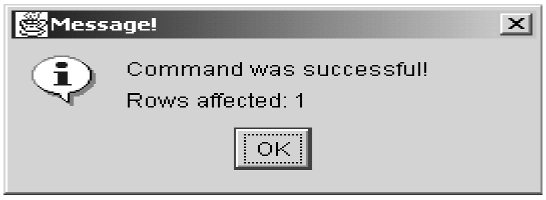
Clicking the execute icon pulls up a further window enabling the user to modify any fields within the row in a form type display. Figure 6 shows the substitute variables form.



CONSUMERID:	[illegible]
USERNAME:	[illegible]
PASSWORD:	[illegible]
PASSWORDENCRYPTIONID:	[illegible]
PASSWORDENCRYPTIONSALT:	[illegible]
PASSWORDSTATUSCODE:	[illegible]
CONSUMERSTATUSCODE:	[illegible]
LOCKED:	[illegible]
TITLEID:	[illegible]
FIRSTNAME:	[illegible]
MIDDLEINITIALS:	[illegible]
LASTNAME:	[illegible]
ADDRESSLINE1:	[illegible]
ADDRESSLINE2:	[illegible]
ADDRESSLINE3:	[illegible]

Figure 6: Substitute variables form

At this point the user can opt to preview the generated SQL statement or execute the statement. Once the statement is executed a report of the execution is displayed on a pop-up message window – see Figure 7.



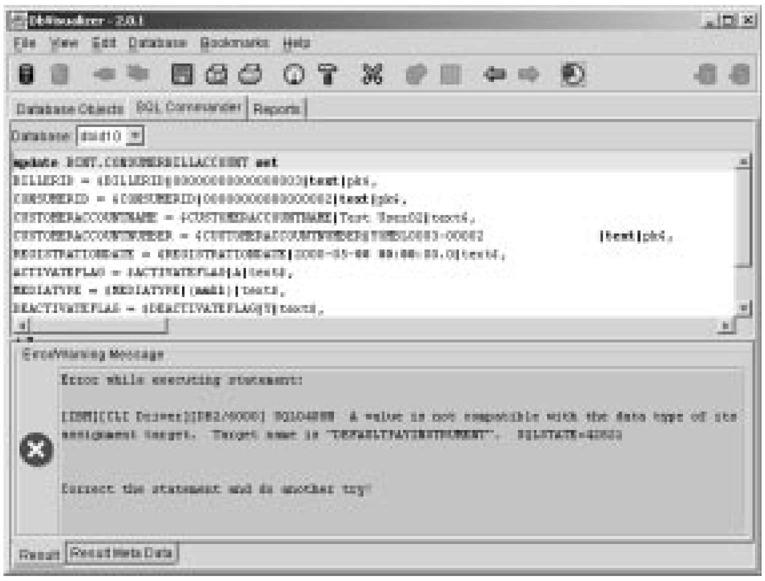
Command was successful!
Rows affected: 1

Figure 7: Pop-up message window

In the event of an error being encountered, a separate display is shown, which displays error diagnostic information – see Figure 8.

SUMMARY AND CONCLUSION

To meet the requirement to enable application developers to easily manipulate DB2 UDB data within tables, DbVisualizer is an ideal yet inexpensive solution. DbVisualizer is simple to install and deploy. Its



```
update BCNT.CONSUMERBILLACCOUNT set
BILLERID = ${BILLERID|0000000000000003|text|pk},
CONSUMERID = ${CONSUMERID|0000000000000002|text|pk},
CUSTOMERACCOUNTNAME = ${CUSTOMERACCOUNTNAME|Test User02|text},
CUSTOMERACCOUNTNUMBER = ${CUSTOMERACCOUNTNUMBER|TUBL0003-00002          |text|pk},
REGISTRATIONDATE = ${REGISTRATIONDATE|2000-05-08 00:00:00.0|text},
ACTIVATEFLAG = ${ACTIVATEFLAG|A|text},
MEDIATYPE = ${MEDIATYPE|(null)|text},
DEACTIVATEFLAG = ${DEACTIVATEFLAG|N|text},
```

```
Error while executing statement:

[IBM][CLI Driver][DB2/6000] SQL0408N  A value is not compatible with the data type of its
assignment target.  Target name is "DEFAULTPAYINSTRUMENT".  SQLSTATE=42821

Correct the statement and do another try!
```

Figure 8. Error diagnostic display

look and feel is familiar to people who use the Microsoft Office suite of programs. DbVisualizer is also a stable product.

The tool is function rich, although the requirement to be able to directly edit tables could be simpler to use. Modification of data is done via editing a form and then SQL code is automatically generated from this. Diagnostic support is good and should provide a good insight into the most typical problems encountered.

The entity relationship diagram is excellent and well worthy of a mention. It gives a real-time overview of the database design.

The product is free of charge and no maintenance or licence fees are payable. As must be expected under these conditions, no guaranteed support or documentation exists.

The product is worthy of deployment in an application development environment.

Stephen Forward
Consultant
Coveford Data Services (Switzerland)

Summary tables

With more and more sites setting up data warehousing projects, the need to reduce query execution times has never been greater. This is where summary tables come in. The concept of summary tables has been around since V6. The example below was run on UDB 7.2 FP4 on Windows 2000.

Let’s start by discussing what a summary table is. The SQL reference manual states that “*A summary table is a table whose definition is based on the result of a query*”.

Let’s look at an example to illustrate this, and highlight the benefit of using a summary table. Suppose you have a simple table, consisting of customer id number and amount paid, and you wanted to get the sums paid by each customer.

The table structure might look like this:

```
create table tab14 (cust_id int, amt int);
```

And the query you would use might look like this:

```
>db2 select distinct(cust_id),sum(amt) from tab14 group by cust_id
```

If your table size was in the order of hundreds of rows, then the cost of the above query would be about 25 timerons. If your table contained 10,000 rows, then the cost would be about 250 timerons. If your table contained millions of rows, then this timeron figure would obviously be orders of magnitude greater. Now, if your query were more complicated (and it would be for any data warehouse application), then it could take tens of minutes to hours to execute. What happens if this is a query that many users execute, perhaps several times? Obviously, you have a huge overhead of repeatedly recalculating the sums for each user. This is where summary tables come in. As the manual says, “*A summary table is a table whose definition is based on the result of a query*”. So why don’t we just calculate the sum once (and take the hit of calculating it once), and then, when users need this value, it is already calculated for them. But how would they know that we have already calculated the value for them? Would they have to re-write their

SQL to point to these summary tables? No, the optimizer takes care of all that – the user does not have to rewrite the SQL in any way. When you create a summary table, the optimizer knows about it and what values it contains. Part of the optimization process for user SQL is an SQL re-write, and part of this process is to check whether any summary tables exist, and if they could reduce the cost of the user query. Re-iterating, as a user you do not have to re-write your SQL to account for summary tables – the optimizer does that automatically for you.

Don’t forget, that any summary tables you create will have to be populated (which could take many hours), so you shouldn’t just create a summary table for each base table in your system. You have to plan the use of summary tables very carefully (by analysing the SQL throughput of your system, and seeing whether there are any repetitive queries that perform aggregates).

There are restrictions to using summary tables. These are detailed in the SQL reference manual (check out the manual for a complete list), but the key restrictions are (paraphrasing the manual):

- Summary tables can only be used with dynamic SQL.
- The optimization class in the database configuration file (DFT_QUERYOPT) must be 5 or higher (or it must be set higher locally).
- You cannot alter a summary table.
- You cannot alter the length of a column for a base table if that table has a summary table.
- You cannot import data into a summary table.
- You cannot create a unique index on a summary table.

An important decision you have to make when creating a summary table is when the values in the table should be refreshed. Do you want to control this process manually, or do you want the summary table values updated every time the base table is updated? Clearly, if it takes a long time to refresh the summary table, then you don’t want to be doing the refresh after every update to the base table. This would defeat the point of having summary tables, because your system would probably grind to a halt with the continual updates. However, if your base table is changing throughout the day, then do you really want to use sum values from the overnight run? This is why summary tables are not really suited to OLTP systems, and are more suited to decision support/OLAP systems.

You have the choice of determining when the summary table is updated when you create it, by specifying that the REFRESH can be IMMEDIATE or DEFERRED. As the names suggest, IMMEDIATE means that the summary table is updated every time the base table is updated, and DEFERRED means that summary table is updated only when you issue a command to perform this task.

Having decided to use summary tables, how do you set them up? Assuming that we will be using DEFERRED REFRESH summary tables (and that the optimization level is 5 or higher), there are two ways of telling the optimizer about them. You can either update the database configuration value DFT_REFRESH_AGE (to ANY), or you can set the special register CURRENT REFRESH AGE (to ANY). When you are testing summary tables, you could just set the special register value, but if you are running in production, then you may want to update the database configuration.

You can check the optimization level value in the database configuration file as follows.

On Windows:

```
>db2 get db cfg for <database-alias> | find "QUERYOPT"
```

On Unix:

```
$db2 get db cfg for <database-alias> | grep "QUERYOPT"
```

To check the default refresh age value, issue the following.

On Windows:

```
>db2 get db cfg for <database-alias> | find "REFRESH"
```

On Unix:

```
$db2 get db cfg for <database-alias> | grep "REFRESH"
```

To find the value of the special register, use:

```
>db2 "select (current refresh age) from <any-table>"
```

The <*any-table*>must have at least one row in it, otherwise the above query won't return the current refresh age setting. A good one to use is SYSIBM.SYSDUMMY1, eg:

```
>db2 select (current refresh age) from sysibm.sysdummy1
```

To set the DFT_REFRESH_AGE parameter in the database configuration file, use:

```
>db2 update db cfg for <database-alias> using dft_refresh_age any
```

To change the value of the CURRENT REFRESH AGE special register, use the following command:

```
>db2 set current refresh age any
```

(The value of ANY is equivalent to 99999999999999. This is all explained in the manual, but all we need to know is that a value of ANY is the correct one.)

Let's look at a simple example of how to create a summary table. I did this using the db2admin userid.

Create a test table (tab14) as below:

```
create table tab14 (id int, cust char(2), amt int)
```

Insert rows into the table (I used a REXX program to populate the table with 10,000 rows – use whatever you are comfortable with).

Run runstats on the base table (tab14):

```
>db2 runstats on table db2admin.tab14
```

We will be using the DB2 **explain** command, so if you haven't created all the EXPLAIN tables, then you can do so from the misc directory as follows:

```
C:\PROGRA~1\SQLLIB\misc>db2 -tvf explain.ddl
```

If the tables already exist, then no harm is done because you just get the "SQL0601N The name of the object to be created is identical to the existing name" message.

Run the following test query:

```
>db2 explain all with snapshot for "select distinct(cust),sum(amt) from
tab14 group by cust"
```

To see the statement that has been explained, use:

```
select
EXPLAIN_TIME TIMESTAMP,
TOTAL_COST,
substr(STATEMENT_TEXT,1,150)
from explain_statement
order by explain_time desc;
db2 select EXPLAIN_TIME TIMESTAMP, TOTAL_COST,
substr(STATEMENT_TEXT,1,150) from explain_statement order by
explain_time desc
```

The output looks like:

```
TIMESTAMP TOTAL_COST 3
2001-11-25-19.51.14.463001 +0.00000000000000E+000 select
distinct(cust),sum(amt) from tab14 group by cust
2001-11-25-19.51.14.463001 +2.64761383056641E+002 SELECT Q3.$C0 AS
"CUST", Q3.$C1 FROM (SELECT Q2.$C0, SUM(Q2.$C1) FROM (SELECT Q1.CUST,
Q1.AMT FROM DB2ADMIN.TAB14
AS Q1) AS Q2 GROUP BY Q2.$C0) AS Q3
```

Create a REFRESH DEFFERRED summary table (sum14):

```
create summary table sum14
as
(select distinct(cust) as cust,sum(amt) as mamt
from tab14
group by cust)
data initially deferred
refresh deferred;
>db2 refresh table sum14
```

Update the special register value for current refresh age, as follows:

```
>db2 set current refresh age any
```

This will be set for just your session (no one else's).

Run the same query against the base table tab14:

```
>db2 explain all with snapshot for "select distinct(cust),sum(amt) from
tab14 group by cust"
```

Look at the EXPLAIN output. This shows:

```
TIMESTAMP TOTAL_COST 3
2001-11-25-20.53.29.518000 +0.00000000000000E+000 select
distinct(cust),sum(amt) from tab14 group by cust
```

```
2001-11-25-20.53.29.518000 +2.51831054687500E+001 SELECT Q1.CUST AS
"CUST", Q1.MAMT FROM DB2ADMIN.SUM14 AS Q1


2001-11-25-19.51.14.463001 +0.00000000000000E+000 select
distinct(cust),sum(amt) from tab14 group by cust
2001-11-25-19.51.14.463001 +2.64761383056641E+002 SELECT Q3.$C0 AS
"CUST", Q3.$C1 FROM (SELECT Q2.$C0, SUM(Q2.$C1) FROM (SELECT Q1.CUST,
Q1.AMT FROM DB2ADMIN.TAB14
AS Q1) AS Q2 GROUP BY Q2.$C0) AS Q3
```

The optimized SQL (top 2 lines) is using the SUM14 table, which proves that the optimizer is using the summary table. Also compare the total_cost figures – the cost has been reduced from 264 to 25 for running exactly the same query.

I hope I have shown that there is potentially a tremendous benefit to be obtained from using summary tables (but it requires very careful planning). For a complete list of restrictions when using summary tables, and for information on how to set up summary tables with REFRESH IMMEDIATE, please see the SQL Reference manual.

C Leonard (UK)

DB2 news

Legato Systems has released its NetWorker Modules for IBM's database software running on Linux operating system. The modules are available for DB2 and Informix databases and enable on-line back-up and restore. Should data be lost, databases, tablespaces, objects, or transaction log files can be recovered individually.

Database protection is achieved using NetWorker's back-up scheduling, autochanger support, tape cloning, and advanced media handling, including library and dynamic drive sharing.

Providing back-up server, storage node, and client support for Linux, NetWorker uses parallelism and interleaving to protect data and it allows administrators to perform back-ups locally or over a network to a central tape library.

For further information contact:
Legato Systems, 2350 West El Camino Real Mountain View, CA 94040, USA.
Tel: (650) 210 7000.
URL: http://portal2.legato.com/products/networker/modules/.

* * *

Time Machine Software has announced Version 3.4.0 of its SmartProduction performance enhancement tool and its DB2 optional feature, now with over 30 new cases added to the knowledge base, allowing users to detect even more inefficiencies during system analysis.

The addition of the new cases allows SmartProduction users to detect new inefficiencies in areas such as VSAM optimization, VIO usage, DB2 sorting, and DB2 parallelism.

The DB2 optional feature now supports DB2 V7 and has been updated to display DB2 thread information in the SmartProduction Global Reports.

Specifically, Version 3.4.0 selects all jobs for specified job name prefixes, allowing users to see all matching jobs that require modification. It also provides the ability to limit the view to cases of operations inefficiency.

For further information contact:
Axios Products, 1373-10 Veterans Highway, Hauppauge, NY 11788, USA.
Tel: (631) 979 0100.
URL: http://www.tmachine.com/smart.htm.

* * *

Embarcadero has announced the release of ER/Studio 5.1, the latest update to Embarcadero's data modelling application. ER/Studio 5.1 gives users greater extensibility and the ability to model reference data, which helps users enhance data model clarity and usability. The latest release also adds enhanced physical database support through storage objects, and support for IBM DB2 and Oracle 9*i* AS/400 databases.

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/erstudio/erdatasheet.asp.