126 DB2 update

April 2003

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.

Recreating ZPARM

Although there are many ways to display the contents of the running ZPARM, eg as a stored procedure from IBM, there is no way to recreate it, should the current version be lost.

ZPARMR7 recreates the most recent Assembler version of your DSNZPARM and DSNDECP, in the form that makes it available as a load module. Reassembling this version will recreate the input file. All PTFs up to November 2002 are allowed for.

A JCL example is included in the ASM file itself.

ZPARMR7.ASM

```
* ======================================================================
*               DISPLAY DSNZPARM VALUES
*   NAME         LEVEL            RELEASE                     ASSEMBLY
*   ZPARMV7      DSN71Ø           DSN71Ø                      11/2Ø/Ø2
*   DSNZPARM     DSN71Ø           DSN71Ø                      11/2Ø/Ø2
* ----------------------------------------------------------------------
    DSN6ENV    MVS=XA                   DEFAULT VALUE
    DSN6SPRM   RESTART,                 RESTART TYPE                    X
               ALL,                                                     X
               ABEXP=NO,                EXPLAIN DURING AUTOBIND         X
               ABIND=NO,                AUTOBIND ENABLED                X
               AUTH=NO,                 AUTHORIZATION ENABLED           X
               AUTHCACH=128Ø,           AUTHORIZATION CACHE             X
               BINDNV=BIND,             BIND OR BINDADD AUTHORITY       X
               BMPTOUT=7,               IMS/BMP TIMEOUT FACTOR          X
               CACHEDYN=NO,             CACHE DYNAMIC SQL IN EDM POOL   X
               CACHEPAC=27,             CACHE FOR PACKAGE AUTHORIZATION X
               CACHERAC=67,             AUTHORIZATION CACHE FOR ROUTINESX
               CHGDC=NO,                ACTIVATE CHANGED DATA CAPTURE   X
               CATALOG=DSNC$$Ø7,        VSAM CATALOG NAME               X
               CDSSRDEF=ANY,            CURRENT DEGREE SPECIAL REGISTER X
               CONTSTOR=NO,             CONTRACT DBM1 CT STORAGE        X
               DBACRVW=NO,              DBA CREATE VIEWS/ALIASES FOR OTHX
               DBCHK=NO,                SERVICE AID - CHECK DB CONSISTENX
               DECDIV3=NO,              DECIMAL DIVIDE OPTION           X
               DEFLTID=ME§WHO§7,        SYSTEM DEFAULT USERID           X
               DESCSTAT=NO,             REMOTE DESCRIBE AS STATIC SQL   X
               DLITOUT=7,               IMS/DLI WAIT TIMEOUT FACTOR     X
               DSMAX=7767,              MAXIMUM CONCURRENT DATASETS     X
```

```
EDMBFIT=NO,             BETTER FIT FOR LARGE EDMPOOLS   X
EDMPOOL=12377,          EDMPOOL SIZE                    X
EDMDSMAX=1Ø48576,       EDMPOOL DATA SPACE SIZE MAX     X
EDMDSPAC=237767,        EDMPOOL DATA SPACE SIZE         X
EDPROP=NO,              ALLOW CHANGES TO CAPTURED TABLESX
EVALUNC=NO,             EVALUATION OF UNCOMMITTED DATA OX
HOPAUTH=RUNNER,         3RD SITE HOP REQUESTER AUTHORITYX
IRLMAUT=NO,             IRLM AUTOSTART                  X
IRLMPRC=IRLMPRO7,       IRLM PROZEDURE NAME             X
IRLMSID=IRØ7,           IRLM SUBSYSTEM ID               X
IRLMRWT=997,            DB2 MAXIMUM SECONDS WAIT FOR LOCX
IRLMSWT=7,              IRLM START COMPLETION DELAY     X
LEMAX=7,                LE ELEMENTS                     X
MAXKEEPD=17,            SYSTEM KEEPDYNAMIC SQL ALLOWED  X
MINDVSCL=NONE,          MIN SCALE FOR DECIMAL DIVISION  X
MAXRBLK=144,            RID SIZE IN KBYTES              X
MINRBLK=7,              MIN RIDLISTS IN EACH RIDMAP     X
MINSTOR=NO,             MANAGE STORAGE TO MINIMIZE SIZE X
MXQBCE=2777,            LIMIT DIFFERENT JOIN SEQUENCES  X
MXTBJOIN=15,            MAX TABLES IN A QUERY - V7      X
NPGTHRSH=Ø,             NPAGE ADJUSTMENT FOR ACCESS PATHX
NUMLKTS=2557,           LOCKS PER TABLESPACE            X
NUMLKUS=1557,           LOCKS PER USER                  X
OJPERFEH=NO,            OUTER JOIN PERFORMANCE ENHANCEMEX
OPTHINTS=NO,            ALLOW OPTIMIZER HINTS           X
PARAMDEG=177,           PARALLEL GROUP DEGREE LIMIT     X
OPTSUBQ1=NO,            OPTIMIZER SUBQ ENHANCEMENTS     X
PARTKEYU=YES,           ALLOW UPDATE OF PARTITIONING KEYX
PKGLDTOL=NO,            TOLERATE 'PACKAGE NOT FOUND' V7 X
RECALL=NO,              HSM AUTO RECALL                 X
RECALLD=667,            SECONDS WAIT HSM AUTO RECALL COMX
RELCURHL=NO,            RELEASE LOCKS FOR HELD CURSOR ATX
RETLWAIT=7,             IRLMWAIT FOR INCOMPATIBLE RETAINX
RETVLCFK=NO,            ALLOW KEY WITH VARCHAR          X
RGFCOLID=RGCO7,         DDL REGISTRATION OWNER ID       X
RGFDBNAM=RGFD7,         DDL REGISTRATION DATABASE NAME  X
RGFDEDPL=NO,            DDL ONLY BY REGISTERED APPLICATIX
RGFDEFLT=REJECT,        NOT REGISTERED DDL REACTION DEFAX
RGFESCP=:,              DDCS DEFAULT ESCAPE IN ART/ORT SX
RGFFULLQ=NO,            OBJECT LOOKUP WITH FULL LOCAL NAX
RGFINSTL=NO,            VALIDATE DDL STATEMENTS         X
RGFNMORT=REGISTER_OBJT_MY7, NAME OF OBJECT RGN TBL      X
RGFNMPRT=REGISTER_APPL_MY7, NAME OF APPL. RGN TBL       X
RRULOCK=NO,             RR U-LOCK FOR CURRENT PAGE      X
SEQCACH=BYPASS,         SET SEQ MODE BYPASS IN I/O COMMAX
SEQPRES=NO,             UTILITIES CAN CACHE SEQUENTIALDX
SITETYP=LOCALSITE,      TYPE OF RESTART                 X
SMSDCFL=SMSFL7,         SMS DATACLASS NAME FOR FILE TS  X
SJTABLES=1Ø,            TOTAL TABLES IN STARJOIN        X
```

```
           SMSDCIX=SMSXI7,         SMS DATACLASS NAME FOR INDEX TS X
           SUPERRS=YES,            SUPPRESS LOGREC SOFT ERR RECORD X
           STARJOIN=12377,         STATUS OF STARJOIN ALLOWANCE    X
           STATHIST=NONE,          STATISTICS HISTORY DEFAULT      X
           STATROLL=NO,            AGGREGATE PARTITION LEVEL STATS X
           TABLES_JOINED_THRESHOLD=225, JOIN THRESHOLD FOR OPT LIMIX
           TRKRSITE=NO,            SITE IS USED FOR TRACKER        X
           STATSINT=3Ø,            RTS STATISTICS TIMER INTERVAL   X
           XLKUPDLT=NO,            X LOCK FOR SEARCHED UPDATE/DELETX
           SRTPOOL=31732,          SIZE OF SORT POOL               X
           SYSADM=SYSADM27,        SYSADM1                         X
           SYSADM2=SYSADM37,       SYSADM2                         X
           SYSOPR1=SYSOPR77,       SYSOPR1                         X
           SYSOPR2=SYSOPR57,       SYSOPR2                         X
           UTIMOUT=1Ø7             UTILITY TIME OUT FACTOR
DSN6ARVP   ALCUNIT=CYL,            ARCHIVE ALLOCATION UNIT         X
           ARCWRTC=(1,3,7),        ARCHIVE MSG ROUTE CODE          X
           ARCWTOR=NO,             ARCHIVE WTOR REQUIRED           X
           ARCPFX1=DSNC$$17,                                       X
           ARCPFX2=DSNC$$27,                                       X
           ARCRETN=7,              ARCHIVE RETENTION PERIOD        X
           BLKSIZE=16384,          ARCHIVE BLOCKSIZE               X
           CATALOG=NO,             CATALOG ARCHIVE DATASET         X
           COMPACT=NO,             ARCHIVE TAPE COMPACT WITH IDRC  X
           MSVGP=GROUP7,           ARCHIVE MSS GROUP               X
           MSVGP2=GROUP27,         ARCHIVE MSS GROUP               X
           PRIQTY=7,               ARCHIVE ALLOCATION PRIMARY SPACEX
           PROTECT=NO,             ARCHIVE RACF PROTECTION         X
           QUIESCE=7,              ARCHIVE LOG MODE(QUIESCE) MAX PEX
           SECQTY=7,               ARCHIVE LOG SECONDARY SPACE ALLOX
           SVOLARC=NO,             DASD ARCH WITH UNIT 1,VOL 1     X
           TSTAMP=NO,              ARCHIVE LOG MIDDLE-FIX IS TIMESTX
           UNIT=NODASD,            ARCHIVE TAPE UNIT TYPE          X
           UNIT2=DASD#7            ARCHIVE TAPE UNIT TYPE
DSN6LOGP   DEALLCT=(3,7),          ARCH TAPE DEALLOCATION TIME (MINX
           MAXARCH=997,            MAX ARCHIVE ENTRIES IS BSDS     X
           MAXRTU=7,               MAXIMUM ARCHIVE READ TAPE UNITS X
           OFFLOAD=NO,         ==> NOT FIT FOR PRODUCTION <==      X
           OUTBUFF=378Ø,           OUTPUT BUFFER FOR ACTIVE LOG    X
           TWOACTV=NO,             TWO ACTIVE LOG COPIES           X
           TWOARCH=NO,             TWO ARCHIVE COPIES              X
           ARC2FRST=NO,            ALLOC SECOND ARCHIVE AT RECOVERYX
           TWOBSDS=NO              TWO BSDS DATASETS
DSN6SYSP   AUDITST=(1,2,4,5,7,9,1Ø,11,12,13,14,15,16,17,18,19,23,24X
           ,26,27,28,29,3Ø,31,32), AUDIT TRACE START               X
           BACKODUR=7,             NON DATA SHARING BACKOUT DURATIOX
           CONDBAT=7,              MAX NO. CONNECTED DBAT          X
           CTHREAD=7,              MAX NO OF CONCURRENT THREADS    X
           DBPROTCL=PRIVATE,       DATABASE PROTOCOL FOR 3-PART NAMX
           DLDFREQ=7,                CHECKPOINTS PER LEVEL ID UPDATE X
           DSSTIME=7,                TIME BETWEEN RESET OF DATASET STX
           EXTSEC=NO,                EXTENDED SECURITY               X
           IDBACK=7,                 MAX NO OF BACKGROUND IDS        X
           EXTRAREQ=7,               EXTRA DRDA QUERY BLOCKS REQUESTEX
           EXTRASRV=7,               EXTRA DRDA QUERY BLOCKS SERVER  X
           IDFORE=7,                 MAX NO OF FOREGROUND IDS        X
           CHKFREQ=5ØØØØ,            CHECKPOINT FREQUENCY            X
           MAXDBAT=7,                MAX NO OF ACTIVE REMOTE THREADS X
           MON=(1,2,4,5,6,7,8,9,1Ø,11,12,13,14,15,16,17,18,19,23,24X
           ,26,27,28,29,3Ø,31,32),   MONITOR TRACING FLAGS           X
           MONSIZE=1Ø48576,          MONITOR BUFFER SIZE             X
           PCLOSEN=7,                CHECKPOINTS FOR READ ONLY SWITCHX
           PCLOSET=7,                MINUTES TO PSEUDO-CLOSE READ ONLX
           PTASKROL=NO,              ROLL UP PARALLEL TASK ACCOUNTINGX
           RLF=NO,                   ENABLE RLF                      X
           RLFTBL=Ø7,                RESOURCE LIMIT FACILITY TABLE IDX
           RLFAUTH=SYSIBM7,          RESOURCE LIMIT FACILITY         X
           RLFERR=NORUN,             RLF SU OVERRUN ACTION           X
           ROUTCDE=(1,2,7),          SYSTEM MESSAGE ROUTING CODE     X
           SMFACCT=(1,2,4,5,7,9,1Ø,11,12,13,14,15,16,17,18,19,23,24X
           ,26,27,28,29,3Ø,31,32),   SMF ACCOUNTING FLAGS            X
           SMFSTAT=(1,2,4,5,7,9,1Ø,11,12,13,14,15,16,17,18,19,23,24X
           ,26,27,28,29,3Ø,31,32),   SMF STATISTICS FLAGS            X
           STATIME=7,                STATISTICS TIME                 X
           STORPROC=STOR7,           STORED PROCEDURE MVS NAME       X
           STORMXAB=7,               ALLOWABLE ABENDS FOR STORED PROCX
           STORTIME=5,               STORED PROCEDURE TIMEOUT VALUE  X
           SYNCVAL=NO,               SYNCHRONIZE STATISTICS RECORDINGX
           URCHKTH=7,                UR CHECKPOINT THRESHOLD         X
           URLGWTH=Ø,                UR LOG RECORD WRITTEN THRESHOLD X
           TRACLOC=7,                4K ELEMENTS IN LOCAL TRACE TRABLX
           TRACSTR=(1,2,4,5,7,9,1Ø,11,12,13,14,15,16,17,18,19,23,24X
           ,26,27,28,29,3Ø,31,32),   MONITOR TRACING FLAG            X
           IDXBPOOL=BP7,             DEFAULT BP FOR INDEXES          X
           TBSBPOOL=BP7,             DEFAULT BP FOR TABLESPACES      X
           LOGAPSTG=7,               FAST LOG APPLY STORAGE IN MB    X
           LBACKOUT=NO,              RESTART BACKOUT OPTION NON DS   X
           LOBVALA=7,                KB FOR LOB VALUES PER AGENT     X
           LOBVALS=7,                MB FOR LOB VALUES PER SYSTEM    X
           WLMENV=MASTER7,           DEFAULT WLM ENVIRONMENT NAME    X
           TRACTBL=7                 4K SEGMENTS IN LOCAL TRACETBL
DSN6FAC    DDF=NO,                   DDF STARTUP                     X
           CMTSTAT=INACTIVE,         DDF THREAD STATUS               X
           IDTHTOIN=7,               DDF IDLE THREAD TIMEOUT         X
           RESYNC=7,                 DDF RESYNC PERIOD LENGTH (MIN)  X
           POOLINAC=7,               DDF INACTIVE POOL TIME (SEC)    X
           TCPKPALV=7,               TCP/IP STACK KEEP ALIVE TIME    X
           TCPALVER=NO,              TCP/IP USERID ALREADY VERIFIED  X
```

```
               MAXTYPE1=7,             MAXIMUM NUMBER OF INACT TYP1 DDFX
               RLFERRD=5ØØØØØØ         RLF ERROR LIMIT (CPU SECONDS)
*   DSN6GRP                        DSN71Ø
    DSN6GRP    DSHARE=NO,              DATASHARING DEFINITION          X
               GRPNAME=GROUP7,         DB2 GROUPNAME                   X
               COORDNTR=NO,            DS COORD. FOR QUERY PARALLELISM X
               ASSIST=NO,              DS ASSIST FOR QUERY PARALLELISM X
               IMMEDWRI=NO,            IMMEDIATE WRITE AT COMMIT       X
               MEMBNAME=MEMBER7        DS MEMBER NAME
    AGO   .EXIT
* ======================================================================
*               DISPLAY DSNHDECP VALUES
*   NAME         LEVEL             RELEASE                     ASSEMBLY
*   ZPARMV7      DSN71Ø            DSN71Ø                      11/2Ø/Ø2
*   DSNHDECP     71Ø               V7R1MØ
* ----------------------------------------------------------------------
    DSNHDECM   CHARSET=ALPHANUM,       DEFAULT SUBSYSTEM CHARACTER SET X
               COMPAT=OFF,             SERVICEBILITY PARAMETER         X
               ASCCSID=875,            ASCII SINGLE BYTE CHARSET ID    X
               AMCCSID=65534,          ASCII MIXED BYTE CHARSET ID     X
               AGCCSID=65534,          ASCII DOUBLE BYTE CHARSET ID    X
               SCCSID=29Ø,             EBCDIC SINGLE BYTE CHARSET ID   X
               MCCSID=93Ø,             EBCDIC MIXED BYTE CHARSET ID    X
               GCCSID=3ØØ,             EBCDIC DOUBLE BYTE CHARSET ID   X
               USCCSID=875,            UNICODE SINGLE BYTE CHARSET ID  X
               UMCCSID=65534,          UNICODE MIXED BYTE CHARSET ID   X
               UGCCSID=65534,          UNICODE DOUBLE BYTE CHARSET ID  X
               APPENSCH=EBCDIC,        DEFAULT ENCODING SCHEME         X
               ENSCHEME=EBCDIC,        DEFAULT ENCODING SCHEME         X
               DATE=ISO,               DEFAULT DATE FORMAT             X
               DATELEN=15,             DEFAULT DATE LENGTH             X
               DECARTH=DEC31,          DEFAULT DECIMAL PRECISION       X
               DECIMAL=PERIOD,         DEFAULT DECIMAL PERIOD          X
               DEFLANG=CPP,            DEFAULT LANGUAGE                X
               DELIM=APOST,            DEFAULT DELIMITER               X
               MIXED=YES,              DEFAULT MIXED GRAPHIC           X
               SQLDELI=APOST,          DEFAULT SQL DELIMITER           X
               DSQLDELI=APOST,         DEFAULT DDF SQL DELIMITER       X
               SSID=DB24,              SUBSYSTEM ID                    X
               DYNRULS=YES,            DYNAMIC RULES FROM PRECOMPILER  X
,              LC_CTYPE=ANY.NAM4.YOU.NEED.FOR.XLATION.OR.MORE.THAN.YOU.X
               NED,                                                    X
               STDSQL=YES,             USE 86 STD SQL, NOT DB2 SQL     X
               TIME=EUR,               TIME FORMAT                     X
               TIMELEN=14              TIME LENGTH
.EXIT END   DSN6SPRM
```

Rolf Loeben (Germany) © Xephon 2003

DataPropagator user experiences and expectations on MVS

Our project was simply to install DataPropagator and to set up continuous replication of all tables in a database from one MVS/DB2 to another MVS/DB2 system to run as fast as possible. That sounds straightforward, and it worked OK with a small volume of changes, but when first tried with a full workload it simply couldn't cope. Then we used many tricks and techniques that are not in the IBM manual. This article explains many of those things, including some undocumented features of DataPropagator.

WHAT IS DATAPROPAGATOR?

DataPropagator (Dprop) is the relational data replication product from IBM. It runs as two main component tasks called Capture and Apply. Capture reads source table changes directly from the DB2 logs and stores a copy of them in Change Data (CD) and Unit of Work (UOW) tables on the source DB2 subsystem.

Apply makes regular subscription set cycles to read those CD and UOW tables. Then it replicates any outstanding changes as inserts, updates, or deletes to the target tables, which may be on a different DB2 subsystem.

Replication is controlled via Control tables called ASN.IBMSNAP_xxxxxx, which can be on the source, target, or a separate DB2 subsystem. These tables contain data specifying exactly what to replicate, and Capture and Apply read and update them to coordinate their actions.

MANUALS

No hardcopy manuals come with the Dprop software for MVS. Softcopy manuals are on the CDs for DB2 Connect, or you can get them from http://www.ibm.com/software/data/dpropr/library.html.

The manual for Dprop is supplied in either PDF or BookManager format. It is called *Replication Guide and Reference*. It contains all the information about Dprop for MVS (and also for Windows, OS/2, AIX, and Linux platforms). Use it as your main reference for Dprop.

For Dprop V8 there will also be a new Redbook, which is entitled *IBM Data Replication V8* in the draft (but that may change when it is released). It contains a lot of information to supplement the main manual.

There are also a few interesting Redbooks and white papers, which you can get from the Internet address above. I suggest that you copy at least the *Guide and Reference* manual to a shared folder on a hard disk.

WHAT VERSION TO USE

At the time of writing, Version 8.1 of Dprop has not been released. However, draft versions of the new reference manual and Redbook are available, and they detail many changes, some of which I will mention in this article. Because Version 8 has so many changes and is not yet thoroughly tested by customers, I would recommend waiting for a while before using it.

Use Version 7.1 of Dprop on MVS even if your DB2 is Version 5 or 6. Then you can use some of the undocumented features I describe here. The only disadvantage with Dprop V7 is that there is no ‘try and buy’, as there was for previous versions.

DB2 UDB CONNECT

You will need DB2 Connect installed on a Windows, Unix, or Linux server, which can connect to your MVS system(s) to administer Dprop, even if your propagation runs entirely on MVS.

Preferably use Version 7.2, because DB2 Connect Version 7.1 has bugs in Client Configuration Assistant and Control Center, which must be fixed by installing Fixpack 3 (or later), before you can use it. If you have DB2 Connect Version 8.1 installed, you should be able to use it (although I personally haven’t tested that).

If a migration to Dprop V8 is to be done, you can install maintenance to make Dprop V7 compatible with Dprop V8 control tables, but in that case you will need a DB2 Connect V8 too.

ADMINISTRATION

I recommend that all Dprop administration be done via the Data Joiner Replication Administration tool, which is referred to as DJRA tool in the rest of the article. You can download it from http://www6.software.ibm.com/dl/datajoiner/djra-p. It uses DB2 Connect to access the MVS DB2 systems and update control tables etc.

The only alternative is Control Center in DB2 Connect. But even with the maintenance, Control Center does not have all the functions of DJRA tool, and it doesn’t always work!

Neither of the above can be used for administration of Dprop V8. You must use Replication Center, a new user interface packaged with the DB2 Administration Client of DB2 Connect V8.

CONFIGURING MVS DB2 SUBSYSTEMS TO DB2 CONNECT

It may sound trivial, but when you are configuring MVS DB2 subsystems in Client Configuration Assistant:

- Specify the DB2 ssid name in *Database Alias*.
- Specify the MVS system name in *System name* (if you leave it blank, the IP address will be shown in Control Center).

That makes it easier to work with.

CREATING DPROP CONTROL TABLES

DB2 Connect comes with facilities for data replication, including a sample DDL file *c:\Program\SQLLIB\samples\repl\dpcntl.mvs* to create Dprop control tables. But the SQL in that file generates only one tablespace with LOCKSIZE ROW for all the Dprop control tables. The *Dprop Installation Instructions* say you should use that file, but I recommend that you don't use it.

The DJRA tool can generate a better DDL file to create the control tables. Its DDL creates a separate UOW tablespace with LOCKSIZE TABLE, and the rest of the tables together in a TSCNTL tablespace with LOCKSIZE ROW. DJRA also specifies TYPE 2 indexes, but that is hopefully your default anyway. You should also consider splitting the tables in TSCNTL into separate tablespaces. In particular, the ASN.IBMSNAP_APPLYTRAIL table can grow very quickly if your propagation is running as fast as possible. Then a large number of Apply cycles will occur and be logged in the APPLYTRAIL table. IBM has recommended preformatting the APPLYTRAIL's tablespace for better INSERT performance, but that is probably not necessary with DB2 Version 7, which does automatic asynchronous pre-formatting of the new pages. Use the DJRA file and edit it as required.

Unfortunately, the DJRA-generated DDL files are still not optimal for DB2 on MVS. I recommend that you copy the DDL file to your MVS system, modify it to your own standards, and run it using SPUFI. Consider adding the following:

- SET CURRENT SQLID
- (CREATE STOGROUP)
- (CREATE DATABASE)
- STOGROUP, PRIQTY, SECQTY, and BPOOL for tablespaces and indexes.

The control tables are required on every DB2 subsystem that will use Dprop, hence that modified DDL can run (with small changes) on each. Not all the tables are required on each subsystem, but it's easiest to create them all anyway.

The tables structures and usage are described in great detail in *Table Structures*, Chapter 14 of the *Replication Guide and Reference* manual.

Dprop V8 has extensive control table changes. Dprop V7 has 16 control tables. In Dprop V8, 12 of those tables have changes (mostly extra columns), two old tables become obsolete, and 17 completely new tables are added.

CUSTOMIZING DPROP CONTROLS

There are a few run-control values specified in the Dprop control tables. They will be given default values, or different values can be specified directly in DJRA before it generates the SQL to create the control tables, or you can update them later via normal UPDATE SQL.

Some of the control table default values should be changed for better performance. Here are some default values and recommended values (that we used) to achieve the fastest possible replication.

Table ASN.IBMSNAP_SUBS_SET (for Apply):

```
    SLEEP_MINUTES     2Ø  –>  Ø       (don't sleep)
MAX_SYNCH_MINUTES     3Ø  –>  5       (max 5 min changes/cycle)
```

Table ASN.IBMSNAP_CCPPARMS (for Capture):

```
    COMMIT_INTERVAL   3Ø  –>  1Ø      (seconds)
    PRUNE_INTERVAL   3ØØ  –>  6Ø48ØØ  (7 days)
```

The meaning of these columns is detailed in the manual in Chapter 14, *Table Structures*. In summary:

- *SLEEP MINUTES* was set to zero to get almost continuous propagation.
- *MAXIMUM SYNCH MINUTES* was set low so that Apply will not try to replicate more than five minutes of changes in a single Apply cycle. This is to limit the use of temporary work database (DSNDB07) on the DB2 source subsystem.
- *COMMIT INTERVAL* was set to the minimum time recommended by IBM for a DB2 subsystem with dynamic cacheing active – namely 10 seconds. This value affects the lag in reading the log: it normally ranges from 2 to 16 seconds when commit interval is 10 seconds. Commit interval of 5

seconds was also tried and that reduced the Capture lag to range from 2 to 8 seconds, which resulted in an average end-to-end lag time of 5 to 10 seconds.

For Dprop V8 the commit interval can be set to 1 or even 0 seconds according to the latest IBM performance advice.

- *PRUNING INTERVAL* was set very high so that Capture will not do any pruning even if it is started without the NOPRUNE parameter. For Dprop V8 the pruning is done in a separate thread so there will be no need to avoid pruning for maximum performance.

If any of these values are to be altered, it is normally best to stop Apply or Capture then restart it once the change has been made. But it is also possible to make some dynamic changes. For example, if you change the SLEEP_MINUTES for a running Apply it will see the change when the next subscription set cycle starts and then honour the new sleep time.

SPECIFYING SUBSCRIPTION SETS AND REPLICATION SOURCES

Use DJRA to generate SQL to create control table entries and Change Data tables. The generated file to define table(s) as replication source can be tailored by modifying DJRA scripts or by using an ISPF edit macro, as I described in *Tailoring DataPropagator tables* for OS/390 in *DB2 Update* (Issue 114).

The DJRA tool can also be used for much more (eg to generate SQL to create CCD tables) but it has many bugs. Therefore, you must check the output carefully (and possibly modify it) to ensure that it is valid and correct by your local standards. IBM probably will not make much effort to improve the DJRA tool, since it is not valid for future versions of Dprop.

SYSTEM SET-UP FOR MAXIMUM PERFORMANCE

For best Dprop performance, make the Control Server the same subsystem as the Target Server, and run your Apply task(s) there. Then Apply is optimized because the propagation data transfer uses DB2’s block fetch, and the reading and updating of

Dprop control tables is done from the same subsystem.

Our project called for the maximum performance from Dprop, meaning the minimum delay between changes in the source tables and the corresponding changes replicated to the target tables. Therefore, we set up our source application on one member of a (two member) DB2 data sharing group and Dprop on the other member of the group, thereby separating the load. There are no other applications running on either the source or target DB2 systems.

Because of this separation, the Dprop tables and application tables are not usually shared by the members, hence their Group Bufferpools have very little activity. That is because Dprop Capture normally reads the DB2 logs to get any changes, then writes them into Dprop's CD and UOW tables – never reading the source tables directly. Dprop Apply does regular subscription set cycles, reading the CD and UOW tables then replicating any changes to the target tables – also not reading the source tables directly. The only exception to that is when Dprop Apply must do a full refresh of a table. Then it reads the source table(s) directly and replicates all rows to the target table(s). But that is very rarely required, for example for a COLD start of Dprop.

In the set-up described above, the application tables need to have Group Bufferpools to allow for possible full refreshes. The Dprop Control and CD tables do not need Group Bufferpools unless a Dprop task is moved from one DB2 group member to another. But for added flexibility we defined Group Bufferpools for the Control and CD tables too.

If your source application has an extremely high update rate, the above arrangement may not cope with the load. Then you could try creating one more DB2 group member so that you can split up Capture and Apply (source activity) onto separate subsystems. (The communication database tables SYSIBM.IPNAMES and SYSIBM.LOCATIONS are used to direct these remote Apply activities to the desired subsystem.) However, note that the Group Bufferpools for the Dprop tables will become busy too, possibly requiring more space in the Coupling Facility.

BUFFERPOOLS

For best performance you would have bufferpools dedicated to Dprop. We are lucky enough to have a DB2 system with primarily only one application, and its data is also being propagated to another DB2. Our MVS systems have 2GB of real storage, enabling us to be generous with bufferpools to get the best performance.

The SOURCE DB2 system is the most critical for Dprop performance, so that is where bufferpool tuning makes the most difference.

The main activity of Apply on the SOURCE DB2 is the following:

```
SELECT IBMSNAP_OPERATION, IBMSNAP_INTENTSEQ, IBMSNAP_COMMITSEQ,
    .... source table columns
  FROM cd_table A , ASN.IBMSNAP_UOW
WHERE ASN.IBMSNAP_UOW.IBMSNAP_UOWID = A.IBMSNAP_UOWID AND
  ASN.IBMSNAP_UOW.IBMSNAP_COMMITSEQ > ? AND
  ASN.IBMSNAP_UOW.IBMSNAP_COMMITSEQ <= ?
  ORDER BY IBMSNAP_COMMITSEQ ASC,
  A.IBMSNAP_INTENTSEQ ASC OPTIMIZE FOR 50000 ROWS;
```

That is done for each CD table in each subscription set for each Apply subscription cycle. It does a nested loop join of the two tables (or hybrid join if you use our fake statistics), then sorts for the ORDER BY. To optimize that we have four bufferpools for Dprop:

- UOW table
- UOW table index
- All CD tables
- All CD table indexes.

Our UOW and UOW index bufferpools are large enough to hold most of the table and index permanently. Our CD and CD index bufferpools are large enough to hold 50% of the largest table and index at one time. This results in very little I/O and hit ratios close to 100%.

They do not really need to be so large provided they are separated from the rest to prevent their bufferpool pages from being ‘stolen’ for some other application. If you have only limited real storage to back extra bufferpools, it is more important to isolate the UOW table/index than the CD tables/indexes.

In Dprop V8, all of the above has been changed. The CD tables include a column (IBMSNAP_COMMITSEQ), which holds the log sequence number of the captured commit statement, which is also in the UOW table. That enables Apply to avoid the CD/ UOW join for some types of target tables. If that is your case, you will not have to optimize I/Os for the UOW table with separate bufferpools.

DSNDB07 can be heavily used by Dprop for sorts, therefore we set it up with its own dedicated bufferpools, with:

- VDWQT = 80% (default 10%)
- DWQT = 90% (default 50%).

In Dprop V8 the changes supposedly allow the ORDER BY described above to be satisfied by the index, to avoid the sorts. That should reduce the bufferpool size requirement for the temporary database.

The main application has its own bufferpools. The DB2 catalog and directory are the only things in bufferpool BP0 (of course!).

STATISTICS

Like any other DB2 application, you need appropriate catalog statistics for the optimizer to get the best access paths for Dprop. All of the Dprop control tables in tablespace TSCNTL are small and their size rarely alters. Therefore, a normal RUNSTATS should be run for them whenever significant changes are made to the Dprop definitions.

The possible exception to that would be the ASN.IBMSNAP_APPLYTRAIL table, which I recommend to be separated into its own tablespace. Dprop writes to this table but never reads it. If you have a Dprop monitor (like the one in *Monitoring DataPropagator on MVS* in *DB2 Update*, Issues 120 and 121 (October and November 2002), an index should have been added to this table for efficient access. Then you should run RUNSTATS when the table has a representative number of rows.

Additionally, the source DB2 system has CD tables and a UOW table. When first created, these tables are empty. But do not run RUNSTATS on them!

Apply does a join of CD and UOW tables, and will use a tablespace scan instead of index access. If you are replicating a large number of changes, these tables can grow very quickly (especially if you run Capture with NOPRUNE). The best statistics are those representative of the largest size that the tables normally reach. If you are not sure of the expected sizes, it is much better to have statistics for large tables than for empty tables (or the DB2 defaults). You can generate fake statistics to achieve that, for example with the following SQL, which has been used successfully with DB2 Versions 5, 6, and 7:

```
UPDATE SYSIBM.SYSTABLES
  SET CARD = 1ØØØØØ
    , CARDF = 1ØØØØØ
    , STATSTIME = CURRENT TIMESTAMP
    , NPAGES =  -1
    , PCTPAGES = -1
    , PCTROWCOMP = -1
  WHERE DBNAME = 'dbname'
    AND TSNAME ^= 'TSCNTL'
    AND CARD < 1ØØØØØ ;
UPDATE SYSIBM.SYSCOLUMNS
  SET COLCARD = 1Ø1ØØ
    , COLCARDF = 1Ø1ØØ
    , STATSTIME = CURRENT TIMESTAMP
    , LOW2KEY =  X'ØØØØØØØØØØØØØØØ1'
    , HIGH2KEY = X'FFFFFFFFFFFFFFFE'
WHERE NAME = 'IBMSNAP_INTENTSEQ'
    AND TBCREATOR = 'userid'
    AND TBNAME IN (
                   SELECT CD_TABLE
                   FROM ASN.IBMSNAP_REGISTER
                   WHERE CD_OWNER = 'userid'
                     AND SOURCE_VIEW_QUAL = Ø
                  );
UPDATE SYSIBM.SYSCOLUMNS
SET COLCARD = 1Ø1
```

```
  , COLCARDF = 1Ø1
  , STATSTIME = CURRENT TIMESTAMP
  , LOW2KEY =  X'ØØØØØØØØØØØØØØØ1'
  , HIGH2KEY = X'FFFFFFFFFFFFFFFE'
WHERE NAME = 'IBMSNAP_UOWID'
  AND TBCREATOR = 'userid'
  AND TBNAME IN (
                 SELECT CD_TABLE
                 FROM ASN.IBMSNAP_REGISTER
                 WHERE CD_OWNER = 'userid'
                   AND SOURCE_VIEW_QUAL = Ø
                );
UPDATE SYSIBM.SYSINDEXES
  SET FIRSTKEYCARD = 1Ø1
    , FIRSTKEYCARDF = 1Ø1
    , FULLKEYCARD = 1Ø1ØØ
    , FULLKEYCARDF = 1Ø1ØØ
    , CLUSTERRATIO = 99
    , NLEVELS = 3
    , NLEAF = 12Ø
    , STATSTIME = CURRENT TIMESTAMP
  WHERE TBCREATOR = 'userid'
    AND TBNAME IN (
                   SELECT CD_TABLE
                   FROM ASN.IBMSNAP_REGISTER
                   WHERE CD_OWNER = 'userid'
                     AND SOURCE_VIEW_QUAL = Ø
                  );
  DELETE FROM SYSIBM.SYSCOLDIST
    WHERE TBOWNER = 'userid'
      AND TBNAME IN (
                     SELECT CD_TABLE
                     FROM ASN.IBMSNAP_REGISTER
                     WHERE CD_OWNER = 'userid'
                       AND SOURCE_VIEW_QUAL = Ø
                    );
```

BINDING APPLY AND CAPTURE

Once the control tables exist and the statistics have been created, you can BIND the Dprop programs. And don't forget to REBIND them after each time you change the statistics. Almost all of the packages have ISOLATION(UR). Leave them as specified by IBM. If they are changed you can get occasional contention timeouts. For best performance, enable dynamic cacheing in your DB2 subsystems using the ZPARM CACHEDYN=YES. Then bind all Dprop packages with

KEEPDYNAMIC(YES) instead of the default (NO) in the sample job from IBM.

APPLY SPILL FILES

The sample Apply JCL from IBM has the parameter:

```
    PARM='.. DISK ..',REGION=1ØM
```

If you use the DISK parameter, it creates temporary spill files. It uses the ASNASPL DD statement in your Apply JCL as a model – so you can adjust their SPACE by altering that DD statement. There will be one file created for each table to propagate. They are not freed until Apply is stopped, even if the tables are removed from the subscription set. Each of them will have the same size allocation, which can become a problem if you have a large number of tables to propagate (one size fits all). Note that this can require many I/Os for your MVS system, and, if the Apply cycles are very quick, many SMF type 101 records will be written (for each propagated table one file will be written in each cycle), even if there are no changes to propagate!

If possible, it is better to use memory for the spill files (that is the default). You may need to increase the region size to hold all the data to be propagated. If there is not enough memory, Apply will fail.

The REGION=10M was too much for our MVS systems (we could use up to 9M for the private area below the line) , so that had to be changed. We decided to use the maximum (above and below the line). Our Apply uses REGION=0M and PARM='.. MEMORY'.

TRACING APPLY

The sample Apply JCL from IBM has the parameter:

```
    PARM='.... TRCFLOW'
```

It produces too much output to be used in a normal production system, but it is occasionally requested by IBM Support to help diagnose a difficult problem with Apply.

The default NOTRC produces no trace information.

Parameter TRCERR produces trace information only in the case of errors. Unfortunately, each time an error occurs you can get several thousand lines of output, but often that information includes basic diagnostic details, which are missing from the console error messages – making the reason much clearer.

A new parameter, TRCPERF, was introduced in Apply Version 7, but it is not documented in any of the manuals. It must be used in conjunction with another trace parameter (eg PARM='... TRCERR TRCPERF'). IBM says that it creates negligible overhead for Apply.

TRCPERF is now documented in the Dprop V8 manual.

The output from TRCPERF is a set of statistics at the end of each Apply cycle (corresponding to a row in the APPLYTRAIL table). It includes one line of data for each table defined for propagation. Thus you can see how many updates/inserts/deletes/reworks were done for each table in each Apply cycle. It is formatted as raw data values, separated by commas, with no headings. To interpret it, we compared the values with those in the APPLYTRAIL table to determine the meanings.

From our brief testing, it appears to contain (in this order):

- 'S' or 'M' – 'S' is first line of each cycle (has many timestamps); 'M' is for each table member line with more statistics.
- SUBSET – subscription set name.
- cycle no. – the Apply cycle number (starting from 1).
- table no. – number of the table (using the order in the subscription set) starting from 0; or it is 'S' for first line of cycle.
- timestamps – six timestamps in 'yyyy/mm/dd hh:mm:ss' format.
- inserts – number of inserts (for the table).
- updates – number of updates (for the table).

- deletes – number of deletes (for the table).
- reworks – number of reworks (for the table).

Our Apply uses PARM='.. TRCERR'.

STARTED TASK FOR APPLY

Our Apply runs as a started task with the following JCL:

```
//D2OØAPP  PROC PRM='DELAY(3) TRCERR ERRWAIT(12Ø) OPT4ONE MEMORY'
//*
//*   PARM OPTIONS: 'DELAY(Ø)'      DO NOT WAIT AFTER EACH CYCLE
//*                 'DELAY(6)'      WAIT 6 SECONDS AFTER EACH CYCLE
//*                 'TRCERR'        TRACE FOR ERROR INFORMATION ONLY
//*                 'TRCFLOW'       VERY DETAILED TRACING
//*                 'TRCPERF'       PERFORMANCE TRACING
//*                 'ERRWAIT(3ØØ)   WAIT 3ØØ SECONDS AFTER EACH ERROR
//*                 'OPT4ONE'       OPTIMISE FOR 1 SUBSCRIPTION SET
//*                 'MEMORY'        STORE FETCHED ROWS IN MEMORY
//*                 'DISK'          STORE FETCHED ROWS IN DISK FILES
//*                 'COMMIT(X)'     COMMIT AFTER EVERY X TRANSACTIONS
//*
//*  START:  S D2OØAPP  OR  S D2OØAPP,PRM='DELAY(6) TRCERR TRCPERF'
//*  STOP :  P D2OØAPP
//*
//**************************************************************/
//*        THIS STEP PREVENTS OUTPUT BEING IMMEDIATELY DELETED */
//**************************************************************/
//DUMMY   EXEC PGM=IEBGENER
//SYSPRINT  DD DUMMY
//SYSUT1    DD DSN=D2OØ.DUMMY.TEXT,DISP=SHR   (one line of '-' chars)
//SYSUT2    DD SYSOUT=T
//SYSIN     DD DUMMY
//*
//**************************************************************/
//*        FOR EXECUTION OF THE APPLY FOR MVS PROGRAM          */
//**************************************************************/
//ASNAPV76 EXEC PGM=ASNAPV76,REGION=ØM,     APPLY V7 WITH DB2 V6
//         PARM='APP1 D2O1 DR_D2OØ &PRM'
//*              !    !    !       !
//*              !    !    !       +— OTHER PARAMETERS
//*              !    !    +— CONTROL_SERVER (LOCATION NAME)
//*              !    +— DB2 SUBSYSTEM ID
//*              +— APPLY_QUALIFIER
//*
//STEPLIB  DD  DISP=SHR,DSN=D2OØ.ASNALNK
//         DD  DISP=SHR,DSN=D2OØ.DSNLOAD
//         DD  DISP=SHR,DSN=CEE.SCEERUN
```

```
//SYSTERM  DD  SYSOUT=*
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//MSGS     DD  DISP=SHR,DSN=D2OØ.ASNAMSGS
//ASNASPL  DD  DSN=&&ASNASPL,
//             DISP=(NEW,DELETE,DELETE),
//             UNIT=SYSDA,SPACE=(CYL,(5,5)),
//             DCB=(RECFM=VB,BLKSIZE=64Ø4)
```

We added a step (DUMMY) to let JES3 keep the output when the task ends.

APPLY OPT4ONE

We use the undocumented parameter OPT4ONE, which can be used for an Apply with only one subscription set. Hence our Apply task reads the details of the subscription set from the ASN.IBMSNAP_SUBS_MEMBR and ASN.IBMSNAP_ SUBS_COLS tables only once instead of re-reading the same information in each subscription cycle. The information is cached and reused, improving CPU utilization and throughput rates. But if you change the subscription set details, you must stop and start Apply to recognize the changes. OPT4ONE is now documented in the Dprop V8 manual.

APPLY DELAY(x)

We use DELAY(3) to make the gap between cycles 3 seconds (default = 6). The value for this parameter is a balancing act when Apply runs with SLEEP_MINUTES = 0. We have some Apply cycles that have nothing to replicate because Capture has not done a COMMIT since the last Apply cycle. I call these ‘zero cycles’. If this value were less we would get many more zero cycles, and if it were more we would get increased lag times. The zero cycles use system resources and can fill up the APPLYTRAIL table with zero rows, hence we don’t use DELAY(0). About two-thirds of our current Apply cycles are zero cycles, and that seems to be a good balance without sacrificing lag too much.

APPLY COMMIT(x)

COMMIT(x) is another undocumented Apply parameter. It is a foretaste of what is coming in Dprop V8. Apply works by writing all changes for a subscription cycle into spill files, one file for each table to be updated. Then it normally processes all the changes for each of the tables in turn until all tables have been updated, and does a COMMIT at the end. Alternatively, when parameter COMMIT(x) is specified, it processes the changes in chronological UOW order, updating all tables involved in each UOW, and does a COMMIT after every ‘x’ units of work have been replicated – and that process is repeated until all UOWs in the cycle are complete. COMMIT(x) probably makes each subscription set cycle marginally slower than the standard process, but some changes are committed earlier and commits done more often (with full referential integrity and with less locking). Hence, it could reduce timeouts when you have other applications simultaneously reading the target tables. COMMIT(x) is now documented in the Dprop V8 manual.

REPLICATION WITH DB2 DATASHARING SUBSYSTEMS

Note that Capture and Apply require a parameter specifying the DB2 ssid and not a group name in Data Sharing group. To make this more flexible, we created a variable in the Capture JCL for the DB2 subsystem id, and created an MVS system static symbol on each MVS system. Then the task can be started on alternative MVS systems and the parameter is always the correct local DB2 subsystem.

Here is a sample of defining the symbol in SYS1.PARMLIB(IEASYM**):

```
SYSDEF   SYSNAME(S5H1)
         SYMDEF(&DB2SSID='D2H1')            /* DB2 DATA PROPAGATOR*/
```

STARTED TASK FOR CAPTURE

```
//D2HØCAP  PROC START='WARMNS NOPRUNE'
//*
//* &DB2SSID='D2H1' OR 'D2H2', DEFINED AS A SYSTEM STATIC
//* SYMBOL IN SYS1.PARMLIB(IEASYM**)
//* //OVERIDE SET DB2SSID=D2H2  <== UNCOMMENT THIS TO USE AS OVERRIDE
//*
//****************************************************************/
```

```
//*        FOR EXECUTION OF THE CAPTURE FOR MVS PROGRAM        */
//*************************************************************/
//ASNL2RN6 EXEC PGM=ASNLRP76,REGION=8M,    CAPTURE V7 WITH DB2 V6
//            PARM='&DB2SSID &START'
//*
//STEPLIB  DD  DISP=SHR,DSN=D2HØ.ASNLLNK
//         DD  DISP=SHR,DSN=D2HØ.DSNLOAD
//         DD  DISP=SHR,DSN=CEE.SCEERUN
//MSGS     DD  DISP=SHR,DSN=D2HØ.ASNLMSGS
//SYSTERM  DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
```

We also have a DUMMY step to keep the JES3 output, but it is not shown.

CAPTURE WARMNS START

To warm start Capture I recommend using start type WARMNS. The default start type for Capture is WARM. But if any problem is found in the warm start data (in ASN.IBMSNAP_WARM_START table) it will switch automatically to a COLD start. That is not always desirable!

My recommendation is to use start type WARMNS, which will not switch, just terminate with an error message. Then you can check the error and decide what is the best action to take.

CAPTURE COLD START

COLD start invokes a full refresh of all target tables. It makes Apply read the source tables directly and replace the contents of the targets. Our application was the only one to be replicated on that DB2 subsystem and it was active only during normal business hours. Also, the source tables were backed up then emptied before the beginning of each day. Therefore, we were able to do a COLD start at the beginning of each day, via MVS command:

```
S D2HØCAP,START='COLD NOPRUNE'
```

Capture COLD start resets the warm start table, then empties the trace, UOW, and CD tables with SQL like:

```
DELETE table WHERE 1=1;
```

which can cause lots of DB2 logging. Therefore, it's better to do your own LOAD REPLACE with dummy input, for all CD and UOW tables before a COLD START; and the target tables can also be cleared the same way before Apply is started.

You must be careful starting Capture and Apply if you are doing a COLD start. Up to Dprop V7, you must first start Capture and wait for it to initialize. Capture will show the message:

```
'ASNØ1Ø3I The Capture program started ..'.
```

Only then should you start Apply; otherwise it may do an initial full refresh of the target tables, but it will not propagate any further incremental changes.

Dprop V8 has been changed to overcome this weakness – the Capture and Apply tasks can be started in any order and they will wait if necessary to achieve correct synchronization.

Up to Dprop V7, Apply does the full refresh of target tables via mass deletes of all existing rows followed by INSERTS to repopulate each table. It cannot use the LOAD utility. Therefore it could be slow with much logging if the volume of data is large. In such cases it is better to use your own external full refresh technique. This is described in the Dprop manual for Version 5 (but not for Version 6 or 7), and is still valid in principle. (But be warned that the section in the manual has 12 SQL statements with a total of 8 syntax errors!) You stop the propagation, copy the data from source to target tables externally from Dprop (eg via LOAD REPLACE utility), reset some values in Dprop control tables, then restart the Apply. DJRA also has an option for this, Replication Operations / Off-line Load. And now, after many years of user requests, a full refresh in Dprop V8 invokes the LOAD utility, through a WLM-managed stored procedure.

If you have multiple subscription sets, you can use the same technique to stop, refresh, and restart a single set, while the others continue unaffected, whereas a COLD start would refresh ALL subscription sets.

CAPTURE NOPRUNE

The Capture reads the DB2 logs and writes the data updates into the CD tables. When those updates have been propagated Capture can ‘prune’ them again, which can be very time consuming:

```
DELETE
  FROM cd_table
  WHERE cd_table.IBMSNAP_UOWID
  IN (SELECT DISTINCT B.IBMSNAP_UOWID
  FROM ASN.IBMSNAP_UOW B
  WHERE B.IBMSNAP_REJ_CODE = 'Ø' AND B.IBMSNAP_COMMITSEQ <= ?);
```

IBMSNAP_REJ_CODE can be added to the index for ASN.IBMSNAP_UOW table to change the access to index only, but IBM advised that it has tried that and not found it significantly better.

Up to Dprop V7, Capture does alternately either pruning or log reading, hence the log reading is not continuous if pruning is done and that can delay propagation. Starting with Dprop V8 the pruning will be done in parallel with log reading by a separate subtask to alleviate this ‘problem’; and IBM recommends running with pruning active for the best performance. Also note that the join of CD and UOW tables is not necessary for pruning in Dprop V8, further improving performance.

However, for maximum performance with the current releases the best solution is to run Capture with NOPRUNE and initiate pruning manually at times when the load is not too high (via the MVS command: **F capture-task,PRUNE**), or alternatively create your own external job to prune the tables and run it whenever necessary.

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

Ron Brown (Consultant) © Xephon 2003

Commit effects

Many people working with DB2 for z/OS or OS/390 believe that all commit does is apply changes and release locks – but, depending on the nature of the application, a commit can have a lot more to do.

This article explains the various implications a commit can have on an application with respect to its performance, availability, and recovery.

Let us start by focusing on one of the less understood functions of commit – releasing claims. Both locks and claims are released on commit. The only exception to this is an application that uses a cursor defined with the WITH HOLD option, where locks and claims are retained past a commit point.

CLAIMS AND DRAINS

When an application first accesses an object, it makes a CLAIM on the object. This claim is released on the next commit point. Claims prevent other applications/processes taking over the access of a particular object. The action of taking over the access to an object is called a drain. When DB2 requests a drain on an object, it allows the application to reach a commit point, but prevents the application from making a new claim. Keep in mind that making a claim does not depend upon any of the bind parameters. Even a program bound with ISOLATION (UR) doing a simple SELECT statement makes a claim. Claims play a vital role in deciding an application's availability when running concurrently with utilities (especially REORG). REORG, and some utilities, requires a drain lock on the object (in some phase or another) and will wait until it can acquire one. A drain lock is acquired when all the claims on the object are released and there are no pre-existing drain locks. Drain requestors prevent any new claims from being taken on the drained object.

To monitor CLAIMS on a particular object, issue the DB2 command:

```
-DISPLAY DB(database name) SPACE(tablespace name) CLAIMERS
```

EFFECT OF COMMIT ON PRE-FETCH

Whereas a commit has no effect on list and sequential pre-fetch because these are detected at bind time by the optimizer, commit does have a significant role to play on dynamic pre-fetch (or sequential detection). Dynamic pre-fetch is activated during run-time when DB2 detects a sequential data access pattern. This is also known as sequential detection. If five of the last eight pages accessed by an application are sequential, DB2 activates dynamic pre-fetch. A page access is considered sequential if it is within P/2 pages (left or right) of a previously retrieved page – where P is the pre-fetch quantity, usually 32.

When a program is bound with the bind parameter **RELEASE(COMMIT), COMMITS**, sequential page access tracking information is reset. In such a case, increasing the program's commit frequency can eliminate (or reduce) dynamic pre-fetch requests associated with the program's execution.

Unlike sequential and list pre-fetch, which can be detected using EXPLAIN, a plan table does not show sequential detection (dynamic pre-fetch). An accounting record (SMF 101) indicates whether dynamic pre-fetch was activated or not.

EFFECT OF COMMIT ON LOGICAL CLOSE OF A DATASET

DB2 maintains a counter for each open dataset. This counter indicates the number of active users of the dataset at a given point in time. On a commit, DB2 decrements this counter by 1 if the release parameter specified at bind time is **RELEASE(COMMIT)**.

When this counter is zero, page set is considered not in use and becomes a candidate of physical close. If 99% of DSMAX is reached, DB2 asynchronously closes 3% of the datasets that are candidates for a physical close. Tablespaces defined with **CLOSE YES** are closed first followed by the ones with **CLOSE NO** attribute (if 3% of datasets are not closed).

COMMIT IMPACTS INDEX LOOKASIDE

DB2 uses the index lookaside technique to minimize the number of getpages and lock requests by storing the keys and identifiers on the leaf and next higher non-leaf page.

For some applications, index lookaside can significantly reduce the number of getpages and lock requests, thereby reducing the CPU time. If such applications are bound with the bind parameter **RELEASE(COMMIT)** and **COMMIT** is issued, the stored keys and identifiers are lost. The higher the commit frequency, the more expensive it gets if the application is utilizing index lookaside.

IMPACT OF COMMIT ON IPROCS AND UPROCS

When DB2 detects repetitive insert or update activity against the same table within a unit of work, it builds procedures for insert or update activity. These procedures reduce the path length for performing the insert or update activity, thereby reducing CPU costs.

Frequent commits executed by programs bound with **RELEASE(COMMIT)** destroy or prevent the building of insert procedures (IPROCS) and update procedures (UPROCS). As a result, the performance of the program doing repetitive inserts or updates degrades. An untimely commit in a program that has the potential to use IPROCS or UPROCS can in fact result in the higher overhead of building the PROCS but never utilizing them.

IMPACT ON LOCK ESCALATION

When DB2 finds too many locks at row or page level it releases the low level locks and takes a lock at the table level or partition level. This is lock escalation. This helps in releasing the memory that is required for storing information for every lock (approximately 250 bytes per lock), but may result in decreased application availability. With frequent commits, lock escalation can be prevented. The value for NUMLKTS specified in DSNZPARM determines the threshold for lock escalation. If the sum of locks for a tablespace at all levels (row, page, table, table space) per user exceeds the value specified in NUMLKTS, DB2 escalates the lock at table level or partition level (for partitioned tablespace).

DB2 statistics trace indicates whether lock escalation took place. This is also flagged as an informational message in the DB2 subsystem address space (MSTR) with complete thread and lock details like LOCK STATE (Share or Exclusive), PLAN/ Package Name, Statement number, etc.

USE CAUTION WITH COMMIT PLACEMENT WHEN USING ROWID COLUMN

Because a column defined with the ROWID data type may change its value after a REORG, care should be taken to utilize the value retrieved from the ROWID column prior to committing. As commit releases the claim on an object, there is a potential risk to data integrity if the REORG job takes over after the commit. A previously retrieved, but unused, ROWID value may have been changed by the REORG utility and may no longer be of interest to the application.

WHEN ARE THE CHANGES APPLIED?

The final destination of any change (Update/Delete/Insert) is the tablespace page on the DASD, but an updated page need not be written to the tablespace immediately. In fact DB2 first logs the change to the DB2 subsystem log buffer. This buffer is externalized to a subsystem log dataset when it is filled, or an application commits, so that the data can be recovered in case of a failure. Commit frees up the log buffer space.

Updated pages may still be in the bufferpool and are externalized when any of the following happens:

- DB2 takes a checkpoint. It takes a checkpoint when a predefined number of log records have been written. This number is defined either on the installation panel by field CHECKPOINT FREQ or by DSNZPARM parameter LOGLOAD. DB2 also takes a checkpoint when there is a switch of active log dataset.

- QUIESCE is done on the table space.
- The percentage of updated pages in a buffer pool for a table space or index space (or partition) exceeds the vertical deferred write threshold (VDWQT).
- The percentage of unavailable pages exceeds the deferred write threshold (DWQT); DB2 schedules a write operation to decrease the number of unavailable pages. When this happens the dataset with the oldest updated pages are written asynchronously.

COMMIT FREQUENCY

In general, a high commit frequency has a negative impact on applications' performance and a positive impact on resource consumption by releasing storage acquired by locks and log buffers, application availability, and recovery time. Not committing at all leaves us to the mercy of successful termination.

In the event of a rollback, a long-running job doing updates without interim commits can face a tragic end by taking even longer to undo changes. If the undo logs are archived off to tape, oh dear!!!

As seen earlier, committing too frequently may add an extremely high overhead, depending on the nature of the application. Besides the impact of frequent commits, as mentioned earlier, there is an additional cost of two MVS cross-memory service calls when committing updates, deletes or inserts: one to the system services address space to write out to the system logs and the other to the IRLM address space to release locks. Having a commit logic in the program also makes it necessary to code the restart logic.

No matter what the nature of the application is, when you decide to commit, you are releasing locks and setting your application free. With careful planning it is possible to achieve good returns (application availability) with a fairly low investment.

Pranav Sampat
Consultant
Cognizant Technology Solutions (USA)

UDB – introduction to identity columns and sequences

This article looks at the identity column and sequence functions in DB2 UDB. What is an identity column and what is a sequence function? Let's first look at the identity function. In a nutshell, an identity column allows you to have a counter column in a table, which is automatically incremented every time a row is inserted into that particular table. For example, this could be the invoice number column in an order table. The identity column concept was introduced in Release 7.1 of UDB DB2. Before this release, you had to write a trigger to achieve similar functionality.

Let's look at how to set up and use an identity column. You set up an identity column in a table when you define the table using the create table statement. You can either have DB2 always generate a number for you, or you can override the generated number and insert a number of your own. The parameters you have to play with are: do you want to allow a user to generate the identity column value themselves (GENERATED BY DEFAULT AS IDENTITY), or do you want the system to always generate them (GENERATED ALWAYS AS IDENTITY)? Secondly, do the identity column values have to be sequential (NO CACHE) or can there be gaps in the numbering (CACHE *n*)? Examples of both of these operations will be shown, and the merits of each discussed.

The different parameters you can specify are GENERATED ALWAYS AS IDENTITY or GENERATED BY DEFAULT AS IDENTITY, and CACHE 20, CACHE n, or NO CACHE.

You always need a starting value and an increment value:

```
(START WITH numeric-constant )
INCREMENT BY numeric-constant
```

If you use the GENERATED ALWAYS AS IDENTITY option then you cannot override the number generated. If you use the GENERATED BY DEFAULT AS IDENTITY option, you can specify a number – if you don't, DB2 will use the next one in the sequence.

The best way to see how to use these parameters is to look at various examples. Let's look at the following four examples:

- GENERATED ALWAYS AS IDENTITY – default CACHE 20.
- GENERATED BY DEFAULT AS IDENTITY – default CACHE 20.
- GENERATED ALWAYS AS IDENTITY – NO CACHE.
- GENERATED BY DEFAULT AS IDENTITY – NO CACHE.

I ran all of the SQL below on Windows 2000 running DB2 UDB 7.2 FP7 and used the sample database.

GENERATED ALWAYS AS IDENTITY – DEFAULT CACHE 20

Create a table called FRED as follows:

```
CREATE TABLE FRED
(INV_NUM INT NOT NULL unique GENERATED ALWAYS AS IDENTITY
(START WITH 1ØØØ,INCREMENT BY 1), ITEM CHAR (1Ø))
```

Then try the following:

```
>db2 insert into fred values('videos')
```

This insert statement will fail with a 'SQL0117N The number of values assigned is not the same as the number of specified or implied columns. SQLSTATE=42802' message.

```
>db2 insert into fred values(1,'videos')
```

This insert statement will fail with a 'SQL0798N A value cannot be specified for column "INV_NUM" which is defined as GENERATED ALWAYS. SQLSTATE=428C9' message.

```
>db2 insert into fred(item) values('videos')
```

This statement will work, and if you do a:

```
>db2 select * from fred
```

it will return:

```
INV_NUM     ITEM
--------- ---------
       1000 videos
```

This shows that if you are going to use the GENERATED ALWAYS AS IDENTITY parameter, you need to specify the same number of columns in the insert statement as there are columns in the table; you cannot specify you own value, and you need to specify a column name for the other columns in the table when you do the insert.

Now consider the following scenario: what happens if two users (user1 and user2) try to insert a record into table FRED at the same time? Each one will issue the SQL to update table FRED using the **+c** option of the CLP (do not commit on statement completion). User1 then rolls back the update. What is the result? This is shown below:

```
User1 issues:                          User2 issues:
>db2 select * from fred
INV_NUM     ITEM
--------- ---------
      1000 videos
                                   >db2 select * from fred
                                   INV_NUM     ITEM
                                   --------- ---------
                                         1000 videos
>db2 +c "insert into fred(item)
values('videosa')"
                                   >db2 +c "insert into fred(item)
                                   values('videosb')"
>db2 rollback
                                   >db2 commit
                                   >db2 select * from fred
                                   INV_NUM     ITEM
                                   --------- ---------
                                         1000 videos
                                         1002 videosb
```

When user1 issues the insert command, DB2 allocates the value 1001 to INV_NUM. So, when user2 issues an insert command, DB2 allocates the next value (1002) to INV_NUM. It has no choice about this – just because user1 hasn't issued a commit when user2 issues the insert, DB2 doesn't know if user1 will commit or not, so it must use the next value for INV_NUM. When user1 rolls back and user2 commits, the value for INV_NUM is then 1002. There is a gap; this is unavoidable if you have multiple users inserting into the table (with some users rolling back).

If we now disconnect then reconnect to the database, the INV_NUM counter value will continue with 1020 (this is because the default cache size is 20, and so the next available number on a 20 boundary is 1020). If we were to disconnect and reconnect to the database again, then the next value of INV_NUM to be used will be 1040. This is shown below:

```
(a6) >db2 select * from fred

INV_NUM     ITEM
---------- ----------
      1000 videos
      1020 videosa
      1002 videosb
```

GENERATED BY DEFAULT AS IDENTITY – DEFAULT CACHE 20

Drop and create the table FRED as follows:

```
Drop table FRED;
CREATE TABLE FRED
(INV_NUM INT NOT NULL unique GENERATED BY DEFAULT AS IDENTITY
(START WITH 1000,INCREMENT BY 1),ITEM CHAR (10));
```

Then try the following:

```
>db2 insert into fred values('videos1')
```

You will get a 'SQL0117N The number of values assigned is not the same as the number of specified or implied columns. SQLSTATE=42802' message.

This is because the insert command still expects the number of columns in the insert command to match the number of columns in the table, or it expects you to explicitly specify the column names that you want to insert values for. Therefore, the insert command should look like:

```
>db2 insert into fred(item) values('videos1')

>db2 select * from fred
```

```
INV_NUM     ITEM
--------- ---------
      1000 videos1
```

The identity number generated is a DB2 generated value. Now let’s try to insert an identity value of our choosing:

```
>db2 insert into fred values(1001,'videos2')

>db2 select * from fred

INV_NUM     ITEM
--------- ---------
      1000 videos1
      1001 videos2
```

Now let’s try to let DB2 generate the next identity number:

```
>db2 insert into fred(item) values('videos3')
```

You get back a message: ‘SQL0803N One or more values in the INSERT statement, UPDATE statement, or foreign key update caused by a DELETE statement are not valid because the primary key, unique constraint or unique index identified by "1" constrains table "DB2ADMIN.FRED" from having duplicate rows for those columns. SQLSTATE=23505’. DB2 does not recognize the identity value that you entered manually (1001), it assumes that the last value was 1000 and that its next value should be 1001 – but this value already exists, hence the SQL0803N message.

What the above shows is that the GENERATED BY DEFAULT AS IDENTITY option lets you override the identity value that DB2 chooses, but DB2 does not take into account any non-DB2 generated identity values. This was shown in the above example, where we inserted a value of 1001. What this means is that if you do specify the GENERATED BY DEFAULT AS IDENTITY option you have to be extremely careful in allowing users the privilege of inserting non-DB2 identity values.

If you use the GENERATED BY DEFAULT AS IDENTITY option instead of the GENERATED ALWAYS AS IDENTITY option, you will get the same results as above if two users try to insert into the table at the same time.

GENERATED ALWAYS AS IDENTITY – NO CACHE

If you use the NO CACHE option instead of the default cache option, you will get the same results as above if two users try to insert into the table at the same time, except that if you disconnect and reconnect to the database, the next value of INV_NUM will be the next numeric value that DB2 thinks it should use (see below for a sequence of events).

Drop and create the table FRED:

```
Drop table FRED;
CREATE TABLE FRED (INV_NUM INT NOT NULL unique GENERATED BY DEFAULT AS
IDENTITY (START WITH 1000,INCREMENT BY 1, no CACHE), ITEM CHAR (10))
```

Then try the following:

```
>db2 Insert into fred(item) values('videos1')

>db2 select * from fred

INV_NUM     ITEM
---------- ----------
      1000 videos1

>db2 connect reset; >db2 connect to sample

>db2 Insert into fred(item) values('videos2')

>db2 select * from fred

INV_NUM     ITEM
---------- ----------
      1000 videos1
      1001 videos2
```

As you can see, the next value of the identity column is 1001 (as the increment is 1, and the starting point is the last inserted value of 1000).

GENERATED BY DEFAULT AS IDENTITY – NO CACHE

If you use the NO CACHE option instead of the default cache option, you will get the same results as above if two users try to insert into the table at the same time, except when you disconnect/ reconnect to the database, when the next value of INV_NUM will be the next numeric value. All the caveats that were mentioned above about users inserting their own values apply equally to the NO CACHE option as to the CACHE option.

QUESTIONS

Below is a list of questions that summarize the above examples.

Can I insert identity numbers manually?

Yes – if you specify GENERATED BY DEFAULT AS IDENTITY.

No – if you specify GENERATED ALWAYS AS IDENTITY.

Don't forget that DB2 doesn't take into account numbers that you enter manually when it decides what the next number to generate should be.

If I input an identity number manually, then will DB2 take this number into account when generating the next identity number?

No – but if you don't define the identity number as unique, then you can have duplicate values. Also, see answer above.

Can I use the ALTER TABLE command to toggle between the CACHE and NO CACHE options?

Yes, you can. The SQL is:

```
>db2 alter table fred alter column inv_num set no cache
```

I could not do the alter through the control centre.

Can I use the ALTER TABLE command to change the INCREMENT value?

Yes, you can. The SQL is:

```
>db2 alter table fred alter column inv_num set increment by 2
```

Can I use the ALTER TABLE command to change the START value?

Yes, you can. The SQL is:

```
>db2 alter table fred alter column inv_num set minvalue  20000
```

This will not change the START value in syscat.sequences, but it will be picked up the next time you disconnect/reconnect to the database.

If I must have sequential identity column numbers, is this possible?

I don’t think this is possible, unless you can guarantee that only one user will be inserting into the table at any one time. In this situation you must define the table with the NO CACHE option. If you have multiple users trying to do inserts, then, assuming that some of them will roll back their transactions, you will have gaps.

When should I use the CACHE/NO CACHE options?

The advantage of using the CACHE *n* option is that DB2 will cache the next *n* values, and will thus not have to recalculate the next value every time. This is good for performance, but bad if you want sequential numbers, as any numbers stored in memory when you disconnect from the database will be ‘lost’. If you definitely need sequential numbering, then I think your only option is to use the NO CACHE option.

In which catalog tables do I find information about identity columns?

There are three tables that you need to look at – syscat.columns, syscat.sequences, and syscat.tables. Below is a query which you might find useful. Just change the tabname value in the WHERE clause to the table you are interested in:

```
Select
create_time,
substr(seqname,1,2Ø),
start,increment From syscat.sequences;
--
Select
substr(a.tabschema,1,1Ø) as schema,
substr(a.tabname,1,1Ø)   as tabname,
substr(a.colname,1,1Ø)   as colname,
a.identity               as id,
a.generated              as gen,
b.cache,
substr(char(b.start),25)     as start,
substr(char(b.increment),25) as incre,
substr(char(b.minvalue),25)  as minv,
substr(char(b.maxvalue),25)  as maxv,
b.cycle
From syscat.columns a,syscat.sequences b, syscat.tables c
where
a.tabname = 'FRED'
and b.create_time=c.create_time
and a.identity = 'Y';
```

This will produce:

```
SCHEMA    TABNAME  COLNAME  ID GEN CACHE START    INCRE    MINV    MAXV  CYCLE
--------- -------- -------- -- --- ----- -------- -------- ------- ----- -----
DB2ADMIN  FRED     INV_NUM   Y  A     1  ØØØ1ØØØ. ØØØØØØ2. ØØ2ØØØØ. 7483647. N
```

Note that I have truncated the start/increment/min/max columns to make the output easier to read – if you have large values, then the substr value of 25 may have to be reduced.

Can I specify a maximum value for the identity column value?

Yes, you can, by using the alter table command to specify a maximum value. This is shown in the SQL below. One thing I did find was that, after you issue the alter table command, DB2 will calculate a new start value based on multiplying the increment by the cache and adding it to the start value specified in the create table statement, to give you a new start value. This is best shown in an actual example:

```
CREATE TABLE FRED
(INV_NUM INT NOT NULL unique GENERATED ALWAYS AS IDENTITY
(START WITH 1ØØØ,INCREMENT BY 5,cache 5Ø),
ITEM CHAR (1Ø))

>db2 insert into fred(item) values('videosa')
>db2 select * from fred

INV_NUM     ITEM
---------- ----------
      1ØØØ videosa

>db2 insert into fred(item) values('videosb')
>db2 select * from fred

INV_NUM     ITEM
---------- ----------
      1ØØØ videosa
      1005 videosb

>db2 insert into fred(item) values('videosc')
>db2 select * from fred

INV_NUM     ITEM
---------- ----------
      1000 videosa
      1005 videosb
      1010 videosc

>db2 insert into fred(item) values('videosd')
>db2 select * from fred

INV_NUM     ITEM
---------- ----------
      1000 videosa
      1005 videosb
      1010 videosc
      1015 videosd

>db2 insert into fred(item) values('videose')
>db2 select * from fred

INV_NUM     ITEM
---------- ----------
      1000 videosa
      1005 videosb
      1010 videosc
      1015 videosd
      1020 videose

>db2 alter table fred alter column inv_num set maxvalue  1200

>db2 insert into fred(item) values('videosf')

DB21034E The command was processed as an SQL statement because it was
not a valid Command Line Processor command. During SQL processing it
returned:
SQL0359N The range of values for the identity column or sequence is
exhausted. SQLSTATE=23522
```

Therefore, if you are going to specify a maximum value, I would issue the alter table command before you insert any records. If you do, you don't seem to hit this restriction; however, if you issue the alter command once the table is populated, you seem to hit the above problem.

What does the CYCLE/NO CYCLE option do?

You use the CYCLE/NO CYCLE option when you have used the MAXVALUE option. What it does is tell DB2 to start from the beginning once you have hit the maxvalue limit. However, if you have the identity column defined as unique, you just end up with lots of messages saying, ‘SQL0803N One or more values in the INSERT statement, UPDATE statement, or foreign key update caused by a DELETE statement are not valid because the primary key, unique constraint or unique index identified by "1" constrains table "DB2ADMIN.FRED" from having duplicate rows for those columns. SQLSTATE=23505’, because you are trying to insert an identity column value which already exists.

Now let’s look at the SEQUENCE function.

The SEQUENCE function allows you define a counter that is not dependent on any particular column in a table, but is defined for the database in which it is created. This is different from an identity column described above, which is table/column dependent. Just as with an identity column, you can specify various options when you define the sequence. You can specify a start value, an increment value, and whether you want values cached by DB2 or not (or accept the defaults). You can also specify a maximum value and a minimum value, and whether you want to cycle back to the beginning when you hit the maximum/minimum value.

Let’s look at an example. You define a sequence (let’s call it hm) as follows:

```
>db2 create sequence hm as integer start with 1Ø increment by 2
```

The default values for the options not specified in the above command are: MINVALUE (1), MAXVALUE (2147483647), CYCLE (N), CACHE (200), ORDER (N).

Now create a table – note that there is nothing in the table creation SQL which specifies that we will be using the sequence function with it.

```
>db2 CREATE TABLE FRED (INV_NUM INT NOT NULL unique, ITEM CHAR (1Ø))
```

Now insert a row into this newly created table using the sequence function:

```
>db2 INSERT INTO fred (inv_num,item) VALUES (NEXTVAL FOR hm,'video')
```

Now select from the table:

```
>db2 select * from fred

INV_NUM     ITEM
---------- ----------
        1Ø video
```

If we now create a second table (FRED2) and use the same sequence function to insert a row into that table, it will use the next value in the sequence (12), and if we then insert a record into table FRED it will use a sequence value of 14. This is shown below:

```
>db2 CREATE TABLE FRED2 (INV_NUM INT NOT NULL unique, ITEM CHAR (1Ø))

>db2 INSERT INTO fred2 (inv_num, item) VALUES (NEXTVAL FOR hm, 'videob')

>db2 select * from fred2

INV_NUM     ITEM
---------- ----------
        12 videob

>db2 INSERT INTO fred (inv_num,item) VALUES (NEXTVAL FOR hm,'videoc')

>db2 select * from fred

INV_NUM     ITEM
---------- ----------
        1Ø videoa
        14 videoc
```

What happens if two users (user1 and user2) try to use sequence numbers at the same time? Let’s use the above table FRED as an example. Just as with the identity column example, each user will issue the SQL to update table FRED using the **+c** option of the CLP (do not commit on statement completion). User1 then rolls back the update. What is the result? This is shown below:

```
User1 issues:                       User2 issues:
>db2 select * from fred

INV_NUM     ITEM
---------- ----------
        1Ø video
```

```
>db2 +c INSERT INTO fred
(inv_num,item) VALUES
(NEXTVAL FOR hm,'videoa')

                                   >db2 +c INSERT INTO fred
                                   (inv_num,item) VALUES
                                   (NEXTVAL FOR hm,'videob')
>db2 rollback
                                   >db2 commit
                                   >db2 select * from fred

                                   INV_NUM     ITEM
                                   ---------- ----------
                                           1Ø video
                                           14 videob
```

You can see that the sequence function operates in exactly the same way as the identity column function. When user1 issues the insert command the value 12 is assigned to INV_NUM, and therefore when user2 issues the insert command the value 14 is assigned to INV_NUM. When user1 rolls back its insert, the value 12 of INV_NUM is 'lost', and when user2 commits, it uses the value of 14 for INV_NUM.

Along with the NEXTVAL expression there is the PREVVAL expression. This is useful if you want to insert records into two tables with the same sequence value. For example, you might have an order table (fred1) and a dispatch table (fred2), where the primary column of each is a unique order number. You want the row in the dispatch table to have the same sequence value as the row in the order table. You would therefore use the NEXTVAL expression to insert a row into the first table (fred1) and the PREVVAL expression to insert a row into the second table (fred2). This is shown below:

```
>db2 create sequence hm as integer start with 4Ø increment by 5

>db2 CREATE TABLE FRED1 (INV_NUM INT NOT NULL unique,ITEM CHAR (1Ø))

>db2 CREATE TABLE FRED2 (INV_NUM INT NOT NULL unique,DISP CHAR (1Ø))

>db2 INSERT INTO fred1 (inv_num,item) VALUES (NEXTVAL FOR hm,'video1')

>db2 INSERT INTO fred2 (inv_num,disp) VALUES (PREVVAL  FOR hm,'Sent')

>db2 select * from fred1
```

```
INV_NUM     ITEM
--------- ---------
        4Ø video1

>db2 select * from fred2

INV_NUM     DISP
--------- ---------
        4Ø Sent
```

You can see that both tables fred1 and fred2 have an INV_NUM value of 40. You could, of course, have written a trigger to do this.

Just like the identity column, you use an alter command (ALTER SEQUENCE command to be precise) to change the option values of sequences. The command is (taken from the *SQL Reference* manual):

```
ALTER SEQUENCE <sequence-name> RESTART
WITH numeric-constant
INCREMENT BY numeric-constant
MINVALUE numeric-constant / NO MINVALUE
MAXVALUE numeric-constant/ NO MAXVALUE
CYCLE / NO CYCLE
CACHE integer-constant / NO CACHE
ORDER / NO ORDER
```

The meanings of the above options are as in the identity column description.

You drop the sequence (hm in this example) as follows:

```
>db2 drop sequence hm restrict
```

You need the restrict keyword, which stops you dropping it if there are dependencies on it. You cannot unconditionally drop a sequence, you need to break all the dependencies first (if any exist).

Can I change/delete sequences through the control centre?

I could not find a way of doing this!

Which catalog table contains details about the sequence?

The catalog table of interest is syscat.sequences, and you can use the query shown below to obtain details about a particular sequence:

```
>db2 select
substr(seqname,1,1Ø),
start, minvalue, maxvalue, cycle, cache, order
from syscat.sequences
where seqname = 'HM'
```

This will give you:

```
1       START     MINVALUE       MAXVALUE     CYCLE  CACHE  ORDER
------ --------- ------------- ----------- ------ ------ -----
HM      1Ø.        1Ø.           2147483647.   N       2Ø      N
```

When would I use the sequence function/the identity column function?

I think the only difference between the sequence function and the identity column function is that the identity column function is specific to a particular table, whereas the sequence function is defined for use by the whole database in which it was defined. Both functions have the restriction about not being able to generate strictly consecutive numbers if you have concurrent users using the functions and some of them perform roll backs.

I hope I have shown how easy it is to implement identity columns and sequences and have given you some design pointers. Which one you use depends on whether you want to restrict the counter to a particular table or whether you want the counter to be available to the whole database. Coding an identity column in the create table statement is a lot easier than coding a trigger, and coding a sequence create statement couldn't be easier, so, if you haven't used them before, give them a go!

C Leonard
Freelance Consultant (UK)

How to get DB2 entity-relationship diagrams via your Web browser

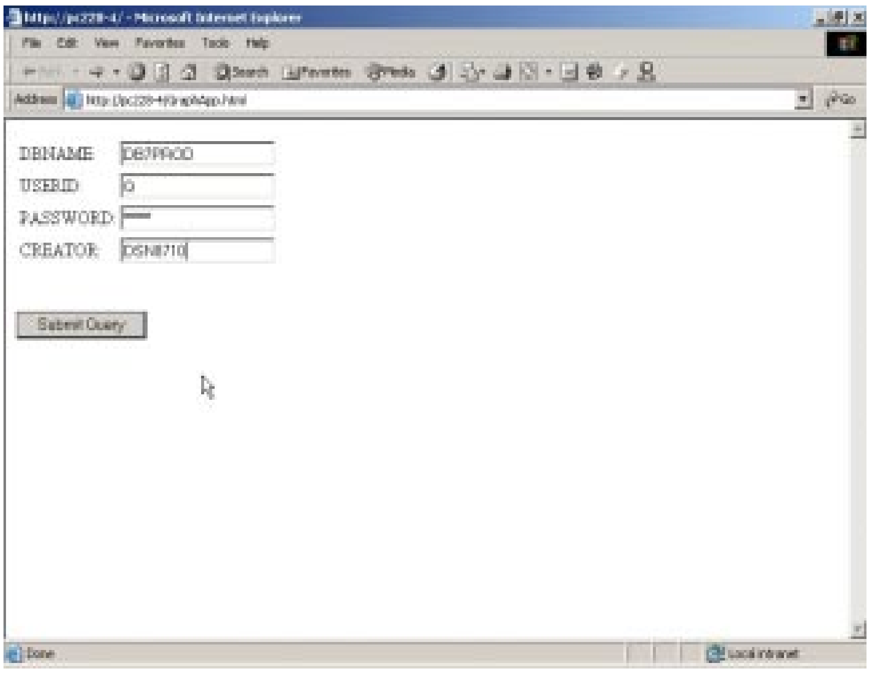
A database data model is needed by database administrators as well as the application development team in their everyday work for reasons such as the definition of system requirements, picture of referential constraints, system documentation, etc. This application is developed to satisfy some of those needs – providing an easy way of presenting a graphical view of the data model, as well as getting it in a printed form. It is designed for DB2 database management systems, but with some modifications it can be also applied for other relational DBMSs. Since, at our site, all application development staff workstations are connected to a local intranet and have some kind of Web browser installed, using it is a very simple and convenient way to provide the desired information.

So, on the client side, you need just a Java-enabled Web browser. On the server side Java 1.1.6 or higher, Web server, and DB2 server should be installed. If the server is on a PC platform, as in our case now, and a picture of the database on the mainframe is needed, then DB2 Connect, installed on the same machine as the Web server, is also required. Another prerequisite is that the DB2 JDBC applet server should be started.

The reason for using Java is obvious: it is platform-independent, so the code can be executed on the mainframe if desired. The Java applet is initiated by HTML with dynamically assigned applet parameters. Our internal standard is that linked tables have the same creator name, so this is used as the filtering criterion for presenting the data model. Obviously it can easily be changed and adjusted to different standards. Bearing in mind how much visual information the human eye can take in at one time, this application gives a good picture of approximately 7–10 entities and the relationships between them. Information on referential constraints is extracted from the DB2 catalog. In cases where the target database is on a mainframe, data is collected from SYSIBM.SYSRELS and SYSIBM.SYSFOREIGNKEYS tables, while on a PC platform all necessary information resides in the table SYSIBM.SYSRELS. So the existence of the table SYSIBM.SYSFOREIGNKEYS in the DB2 catalog is used as an indicator that it is a mainframe database.

In this application the principles of IDEFIX (Integration Definition for Information Modelling) methodology are used for presenting the physical data model. Programming is based on the Graph theory and doubly-linked adjacent lists are used to determine the relationships among parent and child tables.

The data model includes independent and dependent entities depending on the relationship that exists between them. So in contrast to a dependent entity, in the case of an independent entity its instances can be uniquely identified without determining its relationship to another entity. Entities are graphically represented by rectangles (those for dependent entities have


http://pc228-4/ - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Search Favorites Media
Address
Go
Submit Query
Done
Local intranet

http://pc228-4/GraphApp.html

DBNAME	DB7PROD
USERID	[illegible]
PASSWORD	*****
CREATOR	DSN8710

Figure 1: Main panel on Web browser

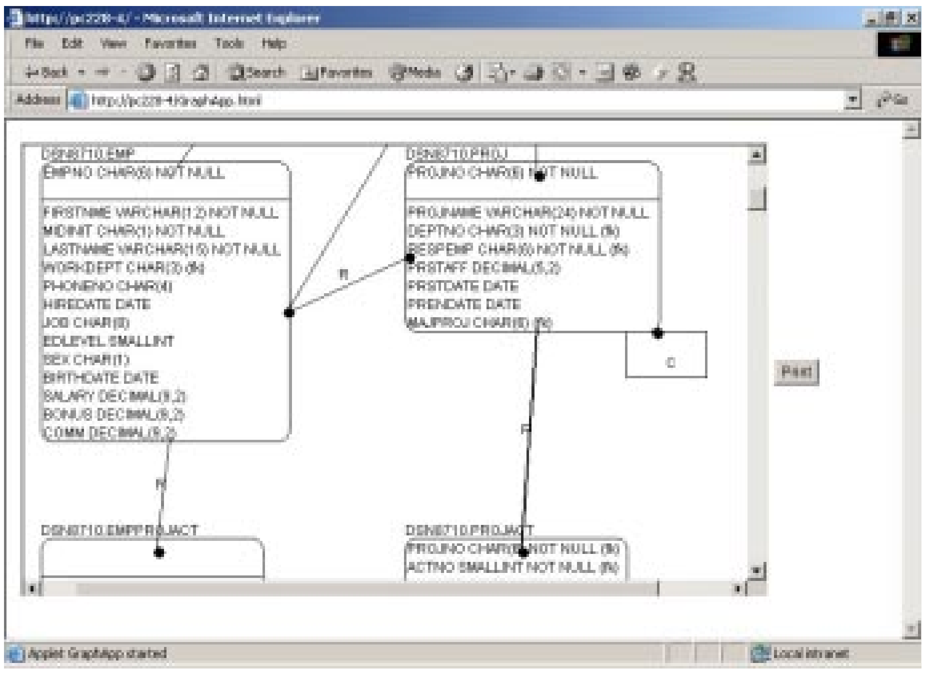


Figure 2: Example of entity-relationship diagram

rounded corners). Primary keys are positioned in the upper part of the rectangle, while corresponding foreign keys are identified by the sign (fk). Relationships are represented by lines that connect the entities: thick lines are for identifying while thin are for non-identifying relationships and self-referencing constraints. The type of delete rule for the referential constraint is shown by the corresponding letter above the line – R for restrict, C for cascade, N for set null, and A for no action.

GraphApp.html and GraphApp.class should be stored on the Web server. In the source code, GraphApp.html parameters ‘code base’ and ‘server value’ should be set according to your environment, while other parameters are optional. If DB2 JDBC applet server is not started on default port 6789, the chosen port should be given in the parameter ‘port value’. File Db2java.zip, which is in the DB2 installation directory, should be copied into the directory where the application is stored (on the Web server, for example c:\inetpub\wwwroot).

Compilation of Java source code is executed by the following command:

```
javac –deprecation GraphApp.java
```

If order to use the print function, in the java.policy file the following lines should be inserted (where *xxxx* represents the pathname where the application resides):

```
grant codeBase "http://xxxx/" {
  permission java.lang.RuntimePermission "queuePrintJob";
};
```

If you access a database on a different DB2 DBMS, obviously this database should be catalogued.

Figure 1 shows the main panel on a Web browser.

Figure 2 shows an example of an entity-relationship diagram produced.

GRAPHAPP.HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.Ø Transitional//EN">
<HTML>
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Homepage Builder V5.Ø.1
for Windows">
<TITLE></TITLE>
<SCRIPT type="text/javascript">
<!-- Begin
function buildApplet(thisform) {
  var db = thisform.DBNAME.value;
  var us = thisform.USERID.value;
  var pa = thisform.PASSWORD.value;
  var cr = thisform.CREATOR.value;
document.write("<applet code='GraphApp.class' codebase='http://pc228-4/'
width=7ØØ height=4ØØ archive='db2java.zip'>")
  document.write("<param name=server value=pc228-4>")
  document.write("<param name=port value=6789>")
  document.write("<param name=dbname value=",db,">")
  document.write("<param name=userid value=",us,">")
  document.write("<param name=password value=",pa,">")
  document.write("<param name=creator value=",cr,">")
  document.write("<\/applet>")
}
// End -->
</SCRIPT>
</HEAD>
<BODY>
```

```
<FORM method="POST">
<TABLE border="Ø">
  <TBODY>
    <TR>
      <TD>DBNAME:</TD>
    <TD><INPUT size="2Ø" type="text" name="DBNAME" value="DB7PROD"></TD>
    </TR>
    <TR>
      <TD>USERID:</TD>
      <TD><INPUT size="2Ø" type="text" name="USERID" value="Q"></TD>
    </TR>
    <TR>
      <TD>PASSWORD:</TD>
     <TD><INPUT size="2Ø" type="password" name="PASSWORD" value=""></TD>
    </TR>
    <TR>
      <TD>CREATOR:</TD>
   <TD><INPUT size="2Ø" type="text" name="CREATOR" value="DSN871Ø"></TD>
    </TR>
  </TBODY>
</TABLE>
<BR>
<BR>
<INPUT type="submit" name="SUBMIT" onclick="buildApplet(this.form);">
</FORM>
</BODY>
</HTML>
```

GRAPHAPP.JAVA

```
  // GraphApp.java
  // ER diagram for DB2 tables
   import java.applet.Applet;
   import java.awt.*;
   import java.awt.event.*;
   import java.util.*;
   import java.sql.*;
   public class GraphApp extends Applet {
     final static int maxCharHeight = 15;
     private boolean pc;
     private String message;
     private Vector queryResults = new Vector();
     private ScrollPane sp;
     private int maxdx = 64Ø;
     private int maxdy = 48Ø;
     private Font font;
  // ------------------------------------------------------------
     class Key {
       public String name;
       public String description;
```

```
      public String fklabel;
      public Key nextK;
      public Key previousK;
      public Key(String kna, String kde) {
        name = kna;
        description = kde;
        fklabel = " ";
      }
    }
    class DoublyLinkedListKey {
      public Key firstK;
      public Key lastK;
      public DoublyLinkedListKey() {
        firstK = null;
        lastK = null;
      }
    }
    class OtherAttr {
      public String nameA;
      public String descriptionA;
      public String fklabelA;
      public OtherAttr nextA;
      public OtherAttr previousA;

      public OtherAttr(String ana, String ade) {
        nameA = ana;
        descriptionA = ade;
        fklabelA = " ";
      }
    }
    class DoublyLinkedListOtherAttr {
      public OtherAttr firstA;
      public OtherAttr lastA;

      public DoublyLinkedListOtherAttr() {
        firstA = null;
        lastA = null;
      }
    }
    class adjList {
      public String tbcreatorL;
      public String tbnameL;
      public String deleteruleL;
      public boolean strongL;
      public adjList nextL;
      public adjList previousL;
      public adjList(String tbcL, String tbnL, String delR, boolean
ind) {
        tbcreatorL = tbcL;
        tbnameL = tbnL;
        deleteruleL = delR;
```

```
            strongL = ind;
          }
        }
        class DoublyLinkedListadjList {
          public adjList firstL;
          public adjList lastL;
          public DoublyLinkedListadjList() {
            firstL = null;
            lastL = null;
          }
        }
     // ----------------------------------------------------------------
        class Vertex {
          public String tbcreator;
          public String tbname;
          public boolean strong;
          public int xv;
          public int yv;
          public int dxv;
          public int dyv;
          public int rowv;
          public int colv;
          public boolean visited;
          public DoublyLinkedListKey theKey;
          public DoublyLinkedListOtherAttr theAtt;
          public DoublyLinkedListadjList theadjList;
          public Vertex next;
          public Vertex previous;
          public Vertex(String tbc, String tbn) {
            tbcreator = tbc;
            tbname = tbn;
            xv = Ø;
            yv = Ø;
            dxv = Ø;
            dyv = Ø;
            rowv = Ø;
            colv = Ø;
            visited = false;
            strong = true;
            theKey = new DoublyLinkedListKey();
            theAtt = new DoublyLinkedListOtherAttr();
            theadjList = new DoublyLinkedListadjList();
          }
        }
        class DoublyLinkedListVertex {
          private draw_area da;
          private Vertex first;
          private Vertex last;
          public DoublyLinkedListVertex() {
            first = null;
            last = null;
          }
          public boolean isEmptyVertex() {
            return first==null;
          }
          public void insertLastVertex(String tbc, String tbn) {
            Vertex newVertex = new Vertex(tbc, tbn);
            if (isEmptyVertex())
              first = newVertex;
            else {
              last.next = newVertex;
              newVertex.previous = last;
            }
            last = newVertex;
          }
     // first adjList with max number Vertexes
          public void arrangeVertex() {
            if (first != null && first.next != null) {
              Vertex current = first;
              Vertex follow;
              Vertex help;
              DoublyLinkedListKey theKey;
              DoublyLinkedListOtherAttr theAtt;
              DoublyLinkedListadjList theadjList;
              adjList currentL;
              int nadjcur;
              int nadjfol;
              while(current.next != null) {
                nadjcur = Ø;
                currentL = current.theadjList.firstL;
                while(currentL != null) {
                  nadjcur++;
                  currentL = currentL.nextL;
                }
```

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

Nikola Lazovic
DB2 System Administrator
Postal Savings Bank (Yugoslavia)

DB2 news

IBM has announced DB2 UDB Version 8, a new reengineered database for z/OS. New in this version are 64-bit virtual addressing, ‘extensive’ enhancements to SQL, and usability and portability enhancements through major catalogue changes.

There are major improvements in long object names, Unicode for worldwide support and improved SQL compatibility, DB2 family compatibility for portability of transaction applications from Unix and Windows environments, and enhanced data availability through on-line schema evolution.

Database management flexibility has been improved with indexing enhancements: variable-length indexes and up to 4,096 partitions and data partitioning for indexes.

Utility enhancements include system-level point-in-time back-up and recovery, automatic restart of utilities, and greater DB2 family compatibility through support for load and unload with delimited data.

Specifically, virtual storage addressing has been expanded from 31-bit to full 64-bit addressing, table name sizes expanded from 18 to 128 characters, VIEW and ALIAS names expanded from 18 to 128 characters, column name sizes expanded from 18 to 30 characters, maximum number of partitions expanded from 254 to 4096, SQL statement length expanded from 32KB to 2MB, index key size expanded from 255 to 2,000 characters, character literals expanded from 255 to 32,704 characters, tables in a join expanded from 15 to 225, active logs expanded from 31 to 93, and archive logs expanded from 1,000 to 10,000.

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

* * *

BMC is to fully support DB2 Universal Database (UDB) V8 on zSeries. The company is taking part in the testing of DB2 V8 in its early release stage. It’s also supporting customer participants in their V8 Early Support Program (ESP) activities, including planned delivery and support during the ESP of specific BMC products for validation in DB2 UDB for z/OS V8 customer environments.

For further information contact:
BMC Software, 2101 City West Blvd, Houston, TX 77042-2827, USA.
Tel: (800) 841 2031.
URL: http://www.bmc.com/solutions/database.

* * *

Princeton Softech has begun shipping Archive for DB2 5.1, its software for active archiving in mainframe environments, which promises better processing performance and faster access to archived data.

Version 5.1 has new options to speed the removal of large volumes of rarely-accessed data from overloaded databases once it’s been safely archived. The enhanced archive indexing and search capabilities are said to provide faster methods for accessing archived data.

Archive indexing is designed to be more efficient for searching archived data stored on tape or near-line devices, or searching data that has been migrated using DFHSM or similar software, without directly accessing the data.

For further information contact:
Princeton Softech, 111 Campus Drive, Princeton, NJ 08540-6400, USA.
Tel: (609) 627 5500.
URL: http://www.princetonsoftech.com