83

DB2

September 1999

In this issue

update

DB2 Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38030
From USA: 01144 1635 38030
E-mail: info@xephon.com

North American office

Xephon/QNA
1301 West Highway 407, Suite 201-405
Lewisville, TX 75077-2150
USA
Telephone: 940 455 7050

Contributions

Articles published in *DB2 Update* are paid for at the rate of £170 ($250) per 1000 words and £90 ($140) per 100 lines of code for original material. To find out more about contributing an article, without any obligation, please contact us at any of the addresses above and we will send you a copy of our *Notes for Contributors*.

DB2 Update on-line

Code from *DB2 Update* can be downloaded from our Web site at http://www.xephon.com/db2update.html; you will need the user-id shown on your address label.

Editor

Robert Burgess

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, and other contents of this journal before making any use of it.

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 1994 issue, are available separately to subscribers for £22.50 ($33.50) each including postage.

Opening ‘in limbo’ tablespaces

Occasionally, a utility on one of our testing subsystems is stopped by the operators, Technical Support Group (TSG), or OPC. More often than not, this happens on a Sunday morning – when we use BMC-type utilities to image copy, reorg, runstats, or whatever, in a brief ‘window of opportunity’, ie without scheduling the jobs. TSG then require the machine for maintenance and don’t check whether it is in use, or the Auto-Operator schedules a STOP DB2.

As a result, the tablespaces find themselves open RO not RW, and so, on Monday morning, no one can update, insert, etc. There is nothing inherently wrong with the tablespaces, they are simply ‘in limbo’.

OPFORCE is a simple REXX that extracts the names it needs from SYSIBM.SYSTABLESPACE to do an OPEN ACCESS(FORCE). It uses REXXTOOLS to access the catalog.

It then submits a batch job using DSNTEP2 and off it goes.

It can, of course, be amended to do whatever OPEN ACCESS( ) you want, on whichever tablespaces you want, but it has got us out of a few scrapes in the past.

OPFORCE

```
/********************************************** REXX ***************/
/*          PRODUCES A TABLE FROM SYSIBM.SYSTABLESPACE              */
/*                FOR A SKELETON JOB TO DO OPEN FORCE               */
/*********************************************************************/
/* TRACE R */
ARG SUBSYS    /* GET DB2 SUBSYSTEM VALUE */
CALL RXSUBCOM 'ADD', 'SQL', 'RXTASQL'
IF RC > 4 THEN DO
  SAY "SQL HOST ENVIRONMENT NOT ADDED. RC="RC
  EXIT
END
IF DSNALI("OPEN",SUBSYS,"RXTOOLCS") <> Ø
   THEN DO
        SAY "OPEN FOR PLAN FAILED. RC="RC "REASON="REASON
        EXIT RC
        END
IF SUBSYS = 'DB2U'
```

```
   THEN
      ADDRESS SQL "SELECT DBNAME, NAME             ",
                  "FROM SYSIBM.SYSTABLESPACE       ",
                  "WHERE   (DBNAME LIKE 'UAT___DB' ",
                  "      OR DBNAME LIKE 'DBUWK00')",
                  "ORDER BY 1, 2                   "
   ELSE IF SUBSYS = 'DB2P'
   THEN
      ADDRESS SQL "SELECT DBNAME, NAME             ",
                  "FROM SYSIBM.SYSTABLESPACE       ",
                  "WHERE   (DBNAME LIKE 'PRD___DB' ",
                  "      OR DBNAME LIKE 'DBPWK00')",
                  "ORDER BY 1, 2                   "
   ELSE IF SUBSYS = 'DB2D'
   THEN
      ADDRESS SQL "SELECT DBNAME, NAME             ",
                  "FROM SYSIBM.SYSTABLESPACE       ",
                  "WHERE   (DBNAME LIKE 'DEV___DB' ",
                  "      OR DBNAME LIKE 'DBDWK00')",
                  "ORDER BY 1, 2                   "
   ELSE IF SUBSYS = 'DB2T'
   THEN
      ADDRESS SQL "SELECT DBNAME, NAME             ",
                  "FROM SYSIBM.SYSTABLESPACE       ",
                  "WHERE   (DBNAME LIKE 'TST___DB' ",
                  "      OR DBNAME LIKE 'DBTWK00')",
                  "ORDER BY 1, 2                   "
SAY 'SQLCODE = ' SQLCA.SQLCODE
"ISPEXEC TBCREATE OPFORCE NAMES(DBNAME NAME COUNT) NOWRITE"
DBNAME1 = STRIP(DBNAME.1) /* SET UP FIRST COPY */
NAME1 = STRIP(NAME.1)     /* SET UP FIRST COPY */
DO COUNT = 2 TO SQLCA.SQLROWS
   DBNAME = STRIP(DBNAME.COUNT)
   NAME = STRIP(NAME.COUNT)
   "ISPEXEC TBADD OPFORCE"
END
"ISPEXEC FTOPEN TEMP"
TSTAMP = 'D'DATE('J')'.T'TIME('M')
"ISPEXEC FTINCL OPFORCE"
"ISPEXEC FTCLOSE"
"ISPEXEC TBEND OPFORCE"
"ISPEXEC VGET ZTEMPF"
"SUBMIT '"ZTEMPF"'"
EXIT
```

DSNTEP2

```
//DMGIEMP2 JOB (DMGAC),'DSNTEP2',CLASS=A,MSGCLASS=X,NOTIFY=DMGIEM
//*
```

```
//*************************************************************
//*                                                           *
//*   PROGRAM DSNTEP2                                         *
//*                                                           *
//*************************************************************
//*
/*JOBPARM SYSAFF=OSØ2
//*
//BATCHTSO EXEC  PGM=IKJEFTØ1,
//             DYNAMNBR=2Ø
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(&SUBSYS)
 -START DATABASE(&DBNAME) SPACENAM(&NAME) ACCESS(FORCE)
)DOT OPFORCE
 -START DATABASE(&DBNAME) SPACENAM(&NAME) ACCESS(FORCE)
)ENDDOT
//*
//SYSIN    DD  DUMMY
/*
```

Ian McDonald
DB2 DBA (UK)

Identifying modified tablespaces

We often need to identify which tablespaces were modified during a particular period of time. This information can be useful to identify tablespaces to recover, frequently updated tablespaces, etc. Although there are products on the market that give extensive information based on the log records, such detail is not always required. Such products also require the tape archives of the log datasets to be mounted and the process can run from a few minutes to several hours depending on the time-range specified for analysis.

This utility was developed to overcome this handicap.

One of DB2's directory files, the SYSLGRNX dataset, has information about all the tablespaces that were opened for modification. This utility processes the records from SYSLGRNX to find the matching

DBIDs and OBIDs (PSIDs) in the time-range specified by the user. It also counts the matches for each DBID and OBID combination. It then queries the relevant database to find the database names and the tablespace names. The result is written to an output dataset, along with the counts noted earlier and other information.

The utility calls two subprograms which are also provided:

- CPIDCAMS – to copy the VSAM file to a sequential file.
- RUNSQL – to run a SQL through REXX.

CHKLGRNX

```
/* REXX */
trace o
clear
/*****************************************************************/
/*  CHKLGRNX -  A quick way to identify modified tables.         */
/*  Calls CPIDCAM and RUNSQL.                                    */
/*                                                               */
/*  This gives the details of the tablespaces that were          */
/*  modified during a given date and time range.                 */
/*  The count, which is the number of times that it was opened   */
/*  for update, is also indicated. This is not to be confused    */
/*  with the number of records modified.                         */
/*  The result is written to an output dataset.                  */
/*                                                               */
/*  This differentiates between a Production & Test system.      */
/*                                                               */
/*  This receives the SUBSYSTEM name and the date and time       */
/*  ranges to check for, as the inputs.                          */
/*                                                               */
/*  Uses: We use this to quickly check what tablespaces were     */
/*        modified during a given time-range. This information   */
/*        is useful to us in several ways such as identifying    */
/*        tablespaces for recovery, determining heavily used     */
/*        tables, etc.                                           */
/*                                                               */
/*  The subsystem names, dataset naming conventions, and the     */
/*  VCAT names must be modified to suit your installation.       */
/*                                                               */
/*****************************************************************/

DEF_CONSTS:
ctnue = 'Press Enter to continue ...'
C_obid = x2c('00D1')    /*             range record identifier  */
```

```
cd = date(U)
cur_date = substr(cd,1,2)||substr(cd,4,2)||substr(cd,7)
us_date = substr(cd,7,2)||substr(cd,1,2)||substr(cd,4,2)
address TSO "ispexec  vget (zsysid)"
p_sysid=zsysid
curtm = time()
cur_time=substr(curtm,1,2)||substr(curtm,4,2)
cur_time=cur_time||substr(curtm,7,2)||'00'
maxdate = 000000
mindate = 999999
fnd=0
match=0
nomatch=0
/* begin main routine  */
MAIN_PROC:
Call GETSSID
Call COPYVCAT

/*********************************************************************/
/* This is the dataset that contains the copy of SYSLGRNX         */
/*********************************************************************/
I_lgrdsn = strip(I_lgrdsn,B,"'")
"ALLOCATE DD(INDD) DSN('"I_lgrdsn"') REUSE SHR"
clear
say '  SPECIFY INPUTS CAREFULLY ...'
typ_date = ' *BEGIN* '
Call GETDATE
I_bdate = I_date
typ_date = '  *END*  '
Call GETDATE
I_edate = I_date
typ_time = ' *BEGIN* '
def_time = '00000001'
Call GETTIME
I_btime = I_time
typ_time = '  *END*  '
def_time = cur_time
Call GETTIME
I_etime = I_time
Call GETODSN
Call WRITEPS
Call BLDLST
Call PROCFND
if fnd <> 0 then
do
  Call PROCQUERY
  exit(0)
end
else
```

```
  exit(8)
/*  end of MAIN_PROC   */
/*  Receive  sub-system ID ...   */
GETSSID:
Do forever until  sid<>''
   Say 'Please give sub-system ID : '
   parse upper pull sid
   sid=strip(sid)
end
  /*  TEST is the test MVS machine  */
  /*  PROD is the prod MVS machine  */
 if p_sysid  = 'PROD' then
 do
   say
   say '  ***  Running on PROD machine  ***  '
   say
   select
   when sid = 'DBPR'
      then do
         nop
      end
   otherwise
      say 'incorrect subsystem id'
      exit(0)
    end
 end
 if p_sysid  = 'TEST' then
 do
   say
   say '  ***  Running on TEST machine  ***  '
   say
   if sid = 'DBPR' then
   do
      say 'You cannot run DBPR on TEST machine...'
         exit(0)
   end
 end
P_ssid = strip(sid)
return
/*  end  of GETSSID   */

COPYVCAT:
/*  modify this to reflect the proper HLQs of the VCAT names    */
hlq ='XX'||substr(sid,3,2)
vcat = hlq||'.DSNDBC.DSNDB01.SYSLGRNX.I0001.A001'
I_lgrdsn = 'HRDBA.'||userid()||'.'||hlq||'.TEST.FILE'
say 'Copying master ...'
say ; say 'Please wait ..this may take a while ...'
say
```

```
/*  call the REXX routine CPIDCAM to copy the SYSLGRNX dataset  */
/*  to a work dataset                                            */

address TSO "CPIDCAM" vcat I_lgrdsn
if  RC > Ø then
do
   say '             ***                      '
   say 'Return code during copy of master is...'rc
   say 'Aborting utility ...'
   exit(8)
end
return rc
/*  end  of COPYVCAT  */

GETDATE:
say
Say 'Please give 'typ_date' date in mmddyy format '
Say '  or press Enter for default 'typ_date' date ('cur_date')...'
pull I_date
If strip(I_date) = '' then
   I_date = strip(cur_date)
else
   I_date = strip(I_date)

if length(I_date) ¬= 6 then
do
   say
   say '*********** Incorrect date specified *********** '
   signal GETDATE
end
return
/* end of GETDATE routine  */

GETTIME:
say
Say 'Please input 'typ_time' time in hhmmssth format '
Say '  or press Enter for default 'typ_time' time 'def_time' ...'
pull I_time
if strip(I_time)='' then
   I_time = def_time
I_time = strip(I_time)
If I_time > '24000000' | length(I_time) ¬= 8 then
  do
    say
    say '********** Incorrect time specified **********'
    signal GETTIME
  end
return
/* end of GETTIME routine  */
```

```
GETODSN:
/*  This routine gets the name of the result dataset  */
ods_name = "PREFX."||userid()||".RESULT."||P_ssid||".D"||us_date
say
Say 'Give the name of a new sequential dataset to write result '
say '  or Press Enter for default dataset 'ods_name
pull I_odsn
if I_odsn='' then
   ods_name = strip(ods_name)
else
   ods_name = strip(I_odsn)
return
/* end of GETODSN routine  */

BLDLST:
/*********************************************************************/
/*  This routine builds the list of DBIDs and OBIDs from the master */
/*  This checks each record in SYSLGRNS for the record identifier   */
/*  between positions 24 and 3872. For each record found, it gets   */
/*  the dbid, psid, date, and time, and analyses it for a fit       */
/*  within the specified date and time range by calling PROCREQ     */
/*********************************************************************/
clear
say 'Processing master ...'
cnt=Ø
/*********************************************************************/
/*         The first two lines are not relevant to us                */
/*         do a dummy read of these two lines                        */
/*********************************************************************/
do i = 1 to 2
   "execio 1 DISKR INDD"
   pull lgr
end
do forever
   "execio 1 DISKR INDD"
   if RC=2 then leave
   pull lgr
   i = i+1
   xstrng=lgr
   cur_pos = 24
   k =1;
   loc = 1
   do until cur_pos > 3872
      strobid = (substr(xstrng,cur_pos,2))
      if  strobid ¬= C_obid then
      do
        cur_pos = cur_pos + 52
      end
      if  strobid = C_obid then
      do
```

```
         loc=cur_pos
         cnt=cnt+1
         dbidc = c2d(substr(xstrng,loc+3,2))
         obidc = c2d(substr(xstrng,loc+5,2))
         tdat = (substr(xstrng,loc+7,6))
         mdatc = substr(tdat,5,2)||substr(tdat,1,2)||substr(tdat,3,2)
         mtimc = (substr(xstrng,loc+13,8))
         Call PROCREQ
         k=k+1
         cur_pos = loc+52
         if  cnt//20000 = 0 then
            say 'Scanned 'cnt' records so far ...'
      end
   end                /*  end of do until cur_pos > 3872    */
end  /*  do  forever  */
say 'Completed processing  '
say 'Found total of 'cnt' instances in master '

/*********************************************************************/
/*    Do a dummy read and close the input file and delete it         */
/*********************************************************************/
"execio 0   DISKR INDD(FINIS"
x = outtrap("q1out.","*")
address TSO "FREE DDNAME(INDD)"
address TSO "delete '"I_lgrdsn"'"
x = outtrap("OFF")
return
/* end of  BLDLST  */

PROCREQ:
/*********************************************************************/
/*   This checks if the dbid and obid identified are within          */
/*   the range specified. It also increments a relevant counter if   */
/*   the object had been found before                                */
/*********************************************************************/
C_bd = substr(I_bdate,5,2)||substr(I_bdate,1,2)||substr(I_bdate,3,2)
C_ed = substr(I_edate,5,2)||substr(I_edate,1,2)||substr(I_edate,3,2)
if mdatc > maxdate then
   maxdate = mdatc
if mdatc < mindate  then
   mindate = mdatc
if  strip(mdatc) < strip(C_bd) | strip(mdatc) > strip(C_ed) then
   return
if  (mtimc >= I_btime) & (mtimc <= I_etime)  then
do
   match=match+1
   if fnd = 0 then
   do
      fnd=fnd+1
      O_dbid.fnd = dbidc
```

```
      O_obid.fnd = obidc
      O_count.fnd = 1
   end
   else                 /* already some records were found  */
   do
      in_fnd = Ø
      Do j = 1  to fnd
         if O_dbid.j = dbidc  then
         do
            if O_obid.j = obidc then
            do
               O_count.j = O_count.j + 1
               in_fnd = 1
            end                    /* if O_obid.j = obidc       */
         end                       /* O_dbid.j = dbidc          */
      end                          /* Do j=1 to fnd             */
      if in_fnd = Ø  then
      do
         fnd=fnd+1
         O_dbid.fnd = dbidc
         O_obid.fnd = obidc
         O_count.fnd = 1
      end                           /*  if  in_fnd = Ø           */
   end                              /*  else  of if fnd=Ø        */
end
nomatch=nomatch+1
return
/*  end of PROCREQ    */

PROCFND:
say  ' Found 'fnd' matches for your range. 'ctnue
do s = 1 to fnd
   O_dbid.s = strip(O_dbid.s)
   O_obid.s = strip(O_obid.s)
   O_count.s = strip(O_count.s)
end
if fnd = Ø then
do
  clear
  say  '  **  NO DATA FOUND **  ';say
  say 'no tablespace changes were found in the date and time '
  say 'boundries shown below:'
  say; say 'From ..' I_bdate ' - ' I_btime ' hrs '
       say 'To   ..' I_edate ' - ' I_etime ' hrs '
  say 'Terminating REXX ...'
  e.1=  "  **  NO DATA FOUND **  "
  e.2=  "No tablespace changes were found in the date and time "
  e.3=  "boundaries shown below:"
  e.4=  "From .." I_bdate " - " I_btime " hrs "
  e.5=  "  To .." I_edate " - " I_etime " hrs "
```

```
  e.6=  "  "
  e.7=  " Total of " cnt " records found in master "
  e.8=  " Found  NO  matches in your range   "
  e.9=  "  "
  e.10= " Date of earliest record analysed was (yymmdd) "mindate
  e.11= " Date of latest   record analysed was (yymmdd) "maxdate
  e.12= "  "
  "execio * diskw opds (stem e. FINIS "
  return
end

/*  records were found within specified range  */
/*  write details to result dataset           */

"makebuf"
queue "  "
queue   " Total of " cnt " records found in master "
queue   " Found    " fnd " matches in your range   "
queue "  "
queue " Date of earliest record analysed was (yymmdd) "mindate
queue " Date of latest   record analysed was (yymmdd) "maxdate
queue "  "
queue   "Details of matched DBIDs and OBIDs and COUNTS in MASTER "
queue   "--------------------"
queue   "DBID OBID    COUNT "
queue   "--------------------"
do zz = 1 to  fnd
   do while length(O_dbid.zz) < 4
      O_dbid.zz = ' '||O_dbid.zz
   end
   do while length(O_obid.zz) < 4
      O_obid.zz = ' '||O_obid.zz
   end
   do while length(O_count.zz) < 8
      O_count.zz = ' '||O_count.zz
   end
   queue O_dbid.zz||" "||O_obid.zz||" "||O_count.zz
end
queue   " ------------------------------------------------------ "
queue   "  "
"execio * diskw opds "
"dropbuf"
return
/*   end of PROCFND  */

PROCQUERY:
/*********************************************************************/
/*     The records found are used to build queries for identifying  */
/*     the database and tablespace names. The queries are built in  */
/*     sets of 250 to avoid exceeding certain size limits           */
```

```
/*********************************************************************/
P_ssid = strip(sid)
P_Dsname = "PREFX."||userid()||".sysin"
nqry = trunc(fnd/25Ø)
p = Ø
if nqry = Ø then
   SIGNAL  TEMPJUMP
do iØ = 1 to nqry
   strt = 25Ø*(iØ-1) + 1
   lstr = strt + 249
   Call QUERYDB
   if qry_err = 1  then
      return
   if  tsin.Ø > Ø then
   do
      do kk = 1 to tsin.Ø
         p = p+1
         F_out.p = tsin.kk
      end
   end
end
TEMPJUMP:
if (nqry*25Ø) ¬= fnd then
do
   strt = (nqry*25Ø) + 1
   lstr = fnd
   Call QUERYDB
   if qry_err = 1  then
      return
   if  tsin.Ø > Ø then
   do
      do kk = 1 to tsin.Ø
         p = p+1
         F_out.p = tsin.kk
      end
   end
end

Call REPORT_RESULT
return
/*  end of PROCQUERY   */

QUERYDB:
/*********************************************************************/
/*  This builds a SQL with the available obids and dbids and calls  */
/*  RUNSQL with sub-system ID and SYSIN dataset name to query DB2   */
/*  It then matches the results with earlier counts of OBIDs and    */
/*  writes the results to the result dataset                        */
/*********************************************************************/
x = outtrap("q1out.","*")
```

```
   address TSO "delete 'PREFX."||userid()||".sysin'"
   address TSO "delete 'PREFX."||userid()||".sysrec00'"
x = outtrap("OFF")
   address TSO "alloc f(tempdd) new unit(sysda) space(1,10)",
               "cyl reuse dsname('PREFX."||userid()||".sysin')"
if  rc ¬= 0 then
do
   say 'Unable to allocate sysin  dataset ...'
   say 'Terminating *** '
   exit(8)
end
say 'Building SQL for objects 'strt' to 'lstr'. 'ctnue
address TSO
"makebuf"
queue "                                                          "
QUEUE "   SELECT NAME, DBNAME, DBID, PSID                        "
QUEUE "   FROM SYSIBM.SYSTABLESPACE                              "
QUEUE "   WHERE ( "
compl = lstr - 1
do i2 = strt to compl
QUEUE "(DBID = "||O_dbid.i2||" AND PSID= "||O_obid.i2||" )  OR "
end
QUEUE "  ( DBID = "||O_dbid.lstr||" AND  PSID = "||O_obid.lstr||" ) ) "
QUEUE " ORDER BY DBNAME, NAME                                    "
QUEUE " ;                                                        "
QUEUE "                                                          "
"execio * diskw tempdd ( finis"
"dropbuf"
/*********************************************************************/
/*   uncomment the next 4 lines to print the SQL                     */
/*                                                                   */
/* "execio * diskr tempdd (finis stem jSQL. "                        */
/* do w = 1 to jSQL.0                                                */
/*    say  jSQL.w                                                    */
/* end                                                               */
/*********************************************************************/
x = outtrap("q1out.","*")
say
say 'Querying database 'P_ssid' for TSnames. '
qry_err = 0
ADDRESS TSO  "RUNSQL" P_ssid P_Dsname
if RC ¬= 0 then
   qry_err = 1
address TSO "free f(tempdd)"
address TSO "delete 'PREFX."||userid()||".sysin'"
x = outtrap("OFF")
/*********************************************************************/
/*                                                                   */
/*         Read results back from sysin to stem                      */
/*                                                                   */
```

```
/*********************************************************************/
address TSO "FREE F(SYSREC00)"
address TSO "ALLOC F(SYSREC00) OLD",
            "DSNAME('PREFX."||USERID()||".SYSREC00')"
"EXECIO * DISKR SYSREC00 (FINIS STEM tsin."
if  qry_err = 1 then
do
   say  'REXX failed in Query processing... Aborted..'
   err.1 = 'REXX failed in query processing.. Aborted'
   "execio * diskw opds (FINIS stem err. "
   address TSO "free f(opds)"
end
return

REPORT_RESULT:
if  p = 0 then
do
   say
   say 'No records found for selected DBIDs and OBIDs. 'ctnue
   pull temp
   e2.1= " "
   e2.2= "No records found for selected DBIDs and OBIDs.."
   e2.3= " "
   "execio * diskw opds (stem e2. "
   return
end
do m=1 to p
   F_out.m = strip(F_out.m)
   parse var F_out.m R_tsn.m +8 R_dbn.m +8 R_db +2 R_ob +2 Zap
   R_dbid.m = c2d(R_db)
   if R_dbid.m = 0 then R_dbid.m = 64 /*  because 64 is x'40'  */
                                      /*  and c2d returns 0    */
   R_obid.m = c2d(R_ob)
   if R_obid.m = 0 then R_obid.m = 64
   R_count.m = 0
   do i3=1 to fnd
      if (O_dbid.i3 = R_dbid.m) & (O_obid.i3 = R_obid.m) then
      do
         R_count.m = O_count.i3
      end
   end
   do while length(R_dbid.m) < 4
      R_dbid.m = ' '||R_dbid.m
   end
   do while length(R_obid.m) < 4
      R_obid.m = ' '||R_obid.m
   end
   do while length(R_count.m) < 8
      R_count.m = ' '||R_count.m
   end
```

```
   R_smry.m = R_dbn.m||' '||R_tsn.m||' '||R_dbid.m||'  '
   R_smry.m = R_smry.m||R_obid.m||' '||R_count.m
end
r1.1= "                                "
r1.2= "                                "
r1.3= "Results of Database query on  "P_ssid":"
r1.4= "                                "
r1.5= "Total objects queried for: "fnd
r1.6= "Total objects found in DB: "p
r1.7= "                                "
r1.8= "------------------------------------ "
r1.9= " DBNAME   TSNAME  DBID  OBID    COUNT "
r1.10="------------------------------------ "
"execio * diskw  opds (stem r1. "
"execio * diskw opds ( stem R_smry. "
say 'Successful completion of utility';say
say 'Results written into ..'ods_name
say; say 'Terminating utility'
fini.1 = "------------------------------------ "
fini.2 = "    "
fini.3 = ' *******  End of report  *********'
"execio * diskw opds (FINIS stem fini. "
address TSO "free f(opds)"
return

WRITEPS:
/*********************************************************************/
/*   This deletes existing result dataset and allocates a fresh     */
/*   dataset.                                                        */
/*   It writes some header information and statistics                */
/*********************************************************************/
say 'Deleting existing result dataset ..'ods_name
x = outtrap("q1out.","*")
address TSO "delete '"ods_name"'"
x = outtrap("OFF")
say 'Allocating and writing into ...'ods_name
address TSO "alloc f(opds) new unit(sysda) space(5,10)",
          "cyl reuse dsname('"ods_name"')",
          "dsorg(ps) blksize(13300) lrecl(133) recfm(f b)"
if  RC ¬= 0 then
do
   say 'Unable to allocate result dataset ...'ods_name
   say 'Terminating *** '
   exit(8)
end
f.1=  " This REXX was run on " date(U) " at " cur_time " on "sid
f.2=  " "
f.3=  " Your start date was       " I_bdate " and time " I_btime
f.4=  " Your end   date was       " I_edate " and time " I_etime
f.5=  "  "
```

```
"execio * diskw opds (stem f.
return
```

CPIDCAM

```
/* REXX */
/*********************************************************************/
/*  CPIDCAM subroutine                                               */
/*  Called from CHKLOG                                               */
/*  This copies a  VSAM dataset into a SEQUENTIAL file               */
/*  This takes the VSAM and the SEQUENTIAL file names as arguments   */
/*********************************************************************/
TRACE o
PARSE UPPER ARG P_vsam P_seql
say 'In copy routine...'
xx = outtrap("junk.","*")
CALL P1000_Allocate_Seqfil
Call P3000_Execute_Repro
EXIT (RC)
/*                                                                   */
/*                                                                   */
P1000_Allocate_Seqfil:
   P_vsam = strip(P_vsam)
   P_seql = strip(P_seql)
   address TSO "delete '"P_seql"'"
   address TSO "alloc f(OUTFILDD) new unit(sysda) space(50,50)",
           "cyl release  dsname('"P_seql"')",
           "dsorg(ps) blksize(4096) lrecl(4096) recfm(f b)"
   address TSO "alloc f(INFILDD) ",
           "shr reuse dsname('"P_vsam"')"
RETURN

P3000_Execute_Repro:
/*********************************************************************/
/* Execute the REPRO command                                         */
/*********************************************************************/

ADDRESS TSO
 "REPRO ",
 "   INFILE(INFILDD)",
 "   OUTFILE(OUTFILDD) "
xx=outtrap("OFF")
if  rc = 0 then
  say 'Successfully copied Master  '
else
  say 'Unable to copy master file ..RC is ' RC
address TSO "FREE F(INFILDD)"
address TSO "FREE F(OUTFILDD)"
RETURN RC
```

RUNSQL

```
/* REXX  */
/*********************************************************************/
/*   RUNSQL subroutine                                               */
/*                                                                   */
/*  Invocation:  TSO RUNSQL DB2SSID SYSINDSN                         */
/*  This REXX takes the DB2 SSID and the SYSIN dataset as argument   */
/*  It then runs DSNTIAUL and stores the result in a file            */
/*  named as PREFX.USERID.SYSREC00                                   */
/*  The SYSPRINT dataset is also present and the code can be         */
/*  turned on or off to browse the same                              */
/*  Use proper values for PREFX as allowed by your installation      */
/*********************************************************************/
TRACE o
PARSE UPPER ARG P_db2sub P_dsname .
say 'Running RUNSQL'
if strip(P_db2sub)='' | strip(P_dsname) = '' then
do
   say 'Proper execution is RUNSQL DB2SSID SYSIN-DSNAME  ...'
   exit(8)
end
x = outtrap("zap.","*")
CALL P1000_Allocate_Sysin
CALL P2000_Allocate_Output
Call P3000_Execute_DSNTIAUL
Call P4000_Clean_up
x = outtrap("OFF")
EXIT

P1000_Allocate_Sysin:
   P_pds_in = strip(P_dsname)
   Sysin = SYSDSN("'"P_pds_in||"'")
   if Sysin = "OK" then do
     address TSO "ALLOCATE DDNAME(SYSIN) SHR reu ",
      "DSNAME('"P_pds_in||"')"
   end
   if Sysin ¬= "OK"  then do
     SAY  '*** Error *** 'P_pds_in Sysin
     exit(8)
   end
   address TSO "delete 'PREFX."||USERID()||".SYSPRINT'"
   address TSO "delete 'PREFX."||USERID()||".SYSPUNCH'"
RETURN

P2000_Allocate_Output:
address TSO "ALLOC F(SYSREC00) NEW UNIT(SYSDA) SPACE(1,10)",
            "UNIT(sysda)",
            "CYL    REUSE DSNAME('PREFX."||USERID()||".SYSREC00')"
address TSO "ALLOC F(SYSPRINT) NEW UNIT(SYSDA) SPACE(8,2)",
```

```
            "UNIT(sysda)",
            "TRACKS REUSE DSNAME('PREFX."||USERID()||".SYSPRINT')"
address TSO "ALLOC F(SYSPUNCH) NEW UNIT(SYSDA) SPACE(8,2)",
            "UNIT(sysda)",
            "TRACKS REUSE DSNAME('PREFX."||USERID()||".SYSPUNCH')"
Return

P3000_Execute_DSNTIAUL:
/*********************************************************************/
/*  Specify the proper DSNLOAD and DSNEXIT  Dataset library names  */
/*********************************************************************/
  "STEPLIB DSN('"PROD.||P_db2sub||".DSNLOAD') SHR"
  "STEPLIB DSN('"PROD.||P_db2sub||".DSNEXIT') SHR"
/*    PLACE THE RUN AND END CMDS ON THE DATA STACK AND EXECUTE DSN */
/*                                                                 */
"NEWSTACK"
QUEUE "RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB51) PARM('SQL')"
QUEUE "END"
QUEUE
address TSO "DSN SYSTEM("P_db2sub")"
"DELSTACK"

/*  comment the signal code below to browse SYSPRINT dataset       */
signal  TEMPSTEP

/*     BROWSE THE SYSPRINT FILE                                    */
ADDRESS ISPEXEC "LMINIT DATAID(DSID) DDNAME(SYSPRINT)"
ADDRESS ISPEXEC "BROWSE DATAID("DSID")"
ADDRESS ISPEXEC "LMFREE DATAID("DSID")"
TEMPSTEP:
RETURN
 P4000_Clean_up:
address TSO "delete 'PREFX."||USERID()||".SYSPRINT'"
address TSO "delete 'PREFX."||USERID()||".SYSPUNCH'"
 RETURN
```

SAMPLE OUTPUT

```
This REXX was run on  04/16/99  at  18141100  on DBT3

 Your start date was        041699  and time  00000001
 Your end   date was        041699  and time  12000000

 Total of  93734  records found in master
 Found     45  matches in your range

 Date of earliest record analysed was (yymmdd) 900321
 Date of latest   record analysed was (yymmdd) 990416

Details of matched DBIDs and OBIDs and COUNTS in MASTER
```

DBID	OBID	COUNT
339	28	1
303	27	1
303	2	1
303	7	1
303	12	1
283	112	1
450	88	1
363	2	1
450	12	5
450	20	2
281	269	2
365	275	2
365	114	2
365	112	2
365	96	2
384	92	3
6	10	3
6	9	8
257	2	5
6	121	5
365	20	1
322	4	1
352	40	1
349	84	2
296	4	1
342	24	1
352	42	1
349	86	2
349	80	1
349	72	1
352	36	1
349	50	1
352	8	1
352	6	1
399	34	2
450	90	4
450	86	4
450	72	5
450	128	5
450	82	2
450	194	2
349	24	1
351	34	1
365	98	1
349	64	1

```
Results of Database query on  DBT3:

Total objects queried for: 45
Total objects found in DB: 36
```

DBNAME	TSNAME	DBID	OBID	COUNT
DMAUNTDB	DMASTSTS	352	42	1
DMAUNTDB	DMAUIDTS	352	40	1
DMAUNTDB	DMELRLTS	352	36	1
DMAUNTDB	DMRADRTS	352	8	1
DMAUNTDB	DMRRPDTS	352	6	1
DMBDCLDB	DMINRRTS	351	34	1
DMCLNTDB	DMCLDMTS	349	86	2
DMCLNTDB	DMCLIDTS	349	84	2
DMCLNTDB	DMCLSTTS	349	80	1
DMCLNTDB	DMDPRTTS	349	72	1
DMCLNTDB	DMEINCTS	349	0	0
DMCLNTDB	DMLIVATS	349	50	1
DMCLNTDB	DMSHLETS	349	24	1
DMMISCDB	DMDCOMTS	342	24	1
DMPARMDB	DMOTHKTS	339	28	1
DMPLAND2	DMPLANTS	363	2	1
DMRSVDDB	DMRSVDTS	322	4	1
DMSPARDB	DMSPARTS	296	4	1
DSNDB06	SYSDBASE	6	9	8
DSNDB06	SYSPKAGE	6	121	5
DSNDB06	SYSPLAN	6	10	3
DSQDBCTL	DSQTSCT1	257	2	5
HRKADB02	CACHLDTS	303	2	1
HRKADB02	CLCASETS	303	7	1
HRKADB02	CLCHLDTS	303	12	1
HRKADB02	CONSEDTS	303	27	1
HRRVDB01	TSERMSG	384	92	3
PTDB	PTG300UH	365	20	1
PTDB	PTG400TS	365	275	2
PTDB	PTITSRAA	365	96	2
PTDB	PTITSRAB	365	98	1
PTDB	PTITSRA1	365	112	2
PTDB	PTITSRA2	365	114	2
RTVTSTD1	RTWPCOTA	399	34	2
WISFEADB	SCFEDB03	281	269	2
WISPOADB	SCGAU033	283	112	1

```
 *******  End of report  *********
```

Jaiwant K Jonathan
DB2 DBA (USA)

Java meets DB2: get there from here – JDBC

You've all heard the hype – Java's the greatest thing since sliced bread, it will solve the world's computing problems, and quite possibly find the cure for the common cold. Hype aside, Java's a wonderful programming language that can be used to build mission-critical software solutions, partly because of its two primary techniques for database access, JDBC and SQLJ. In this series of two articles, both data access technologies will be covered in depth with this first part covering JDBC.

BACKGROUND

Java, the object-oriented programming language developed by Sun Microsystems, was officially unleashed upon the world in January 1996. Almost immediately there came a hue and a cry from developers all over the world concerning its lack of native database connectivity.

The engineers at Sun responded to this legitimate complaint by beginning development of JDBC almost immediately. With input from major players in the database industry (including IBM), JDBC was finalized and became a part of the core Java API with the release of Java 1.1 in February 1997.

Contrary to popular opinion, JDBC is not an acronym and does not stand for 'Java Database Connectivity' – it is simply a trademarked term loosely named after Microsoft's very popular Open Database Connectivity (ODBC) API. JDBC and ODBC share much in common: both are database independent APIs, both are predicated upon the X/Open's CLI specification, and both require some type of driver to translate the calls into a language understood by the specific database product being used on the back end (see Figure 1).

JDBC DRIVERS

Unlike ODBC, JDBC works with four distinct types of driver, officially known as Type 1 to Type 4 drivers, as described below, and depicted in Figure 2:

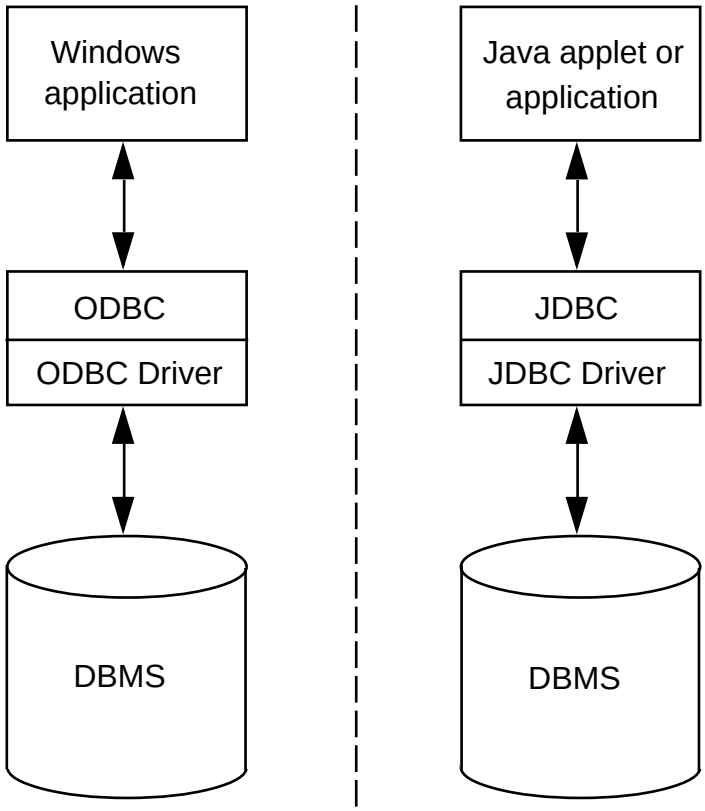

Figure 1: JDBC versus ODBC

- Type 1 – the JDBC-ODBC bridge provides JDBC access via most ODBC drivers. This bridge comes bundled free of charge with the JDK; its purpose is to convert JDBC method calls into ODBC function calls. This allows off-the-shelf ODBC drivers to be used. Note that some ODBC binary code must be loaded on each client machine that uses this driver, making this solution inappropriate for untrusted applets. Performance is also an issue. However, the price can't be beaten!
- Type 2 – a native-API-partly-Java driver converts JDBC calls into calls on the client API for the specific DBMS to which you are connecting. Like the JDBC-ODBC bridge driver, this type also requires that some binary code be loaded onto each client machine, so it, too, is inappropriate for untrusted Java applets. However, it functions well in a typical thick-client two-tier

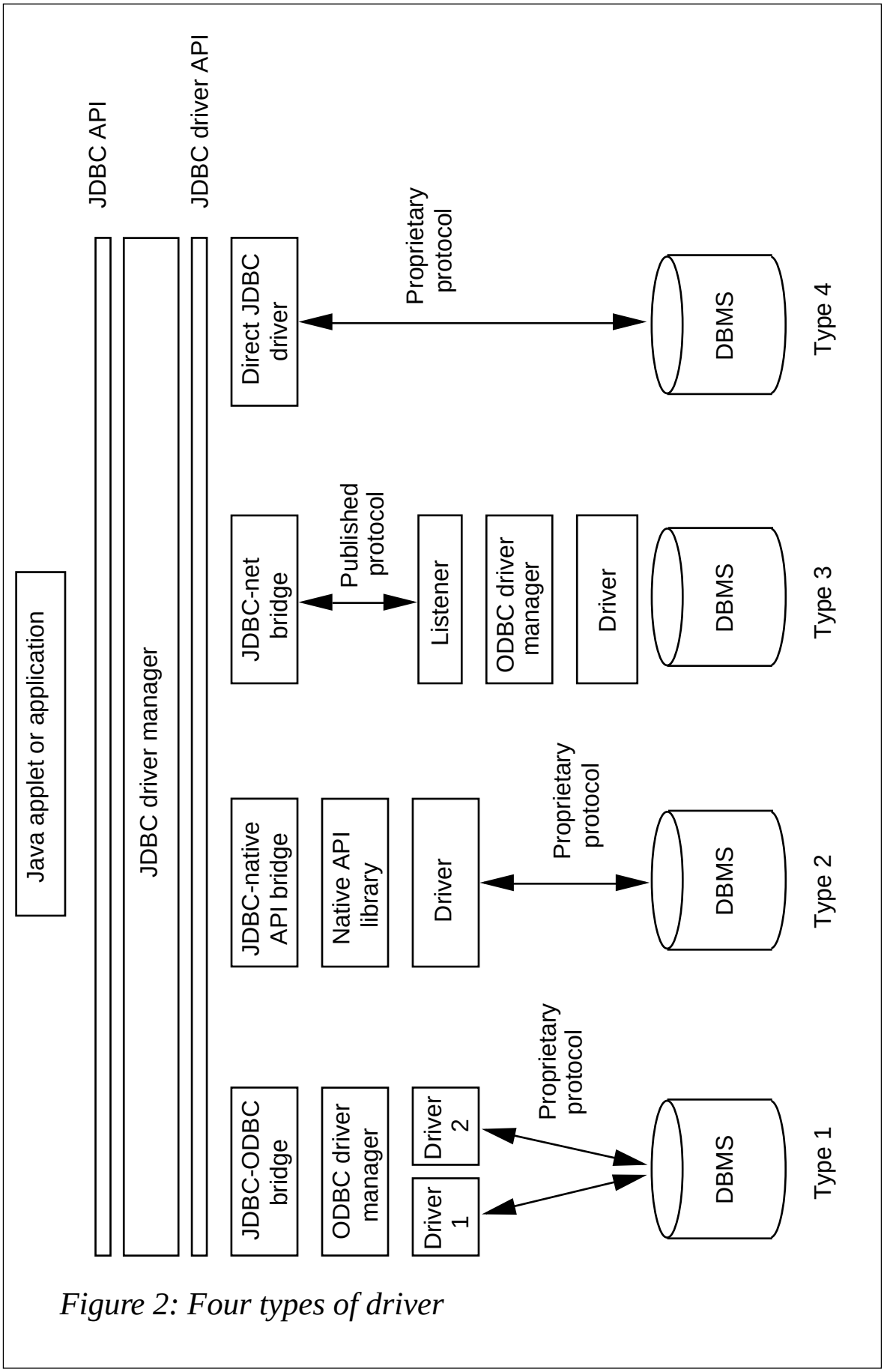

Figure 2: Four types of driver

architecture as well as on the middle tier of a three-tier solution. Performance of Type 2 drivers is typically quite good.

- Type 3 – a net-protocol all-Java driver translates JDBC calls into a DBMS-independent net protocol, which is then translated to a DBMS protocol by a server. The specific protocol used depends on the vendor. Several vendors are adding JDBC drivers to their existing database middleware products. In general, this is the most flexible driver architecture. It adds a technical third tier to the architecture; this extra tier can be used for performance enhancers (eg connection pooling and results cacheing) and for security enhancers (eg user ID mapping and HTTP tunnelling). But unless the extra tier is properly exploited, the net result is a protocol conversion bottleneck. These drivers are often written in 100% pure Java, making them appropriate for untrusted applets.
- Type 4 – a native-protocol all-Java driver converts JDBC calls into the network protocol used by the DBMSs directly. This allows for a direct call from the client machine to the DBMS server. The database vendors will be the primary source of these drivers. Sometimes called thin drivers, they are often written in 100% pure Java. Performance is typically very good.

Since JDBC is well over two years old, many software companies – particularly database vendors – have had plenty of time to bring JDBC drivers to the marketplace (see Figure 3).

Notice that database products from IBM are well-represented. This list was gleaned from a much larger master list maintained by Sun Microsystems. The master list can be found on the Web at http://java.sun.com/products/jdbc/jdbc.drivers.html.

The JDBC driver class required by your applet or application must be loaded prior to establishing a connection to a database. This will guarantee that any initialization that the JDBC driver must do will be handled before your code actually uses the driver. The simplest, and most common, way to do this is to explicitly load the class using the forName() method of the java.lang.Class class, as follows:

```
try {
     Class.forName("ibm.sql.DB2Driver");
}
```

Vendor	*Type*	*DBMS/ Data store*
Agave Software Design	3	Oracle, Informix, Sybase
BEA / Web Logic	2	Oracle, Sybase, SQL Server
Caribou Lake Software	3	CA-Ingres
Cloudscape	4	JBMS
Hit Software	4	DB2, DB2/400
HOB Electronic	4	DB2, VSAM under CICS, IMS-DB under CICS or IMS-DC, DL/I
IBM	2/3	DB2 CS and UDB
IBM	4	DB2/400
IBM	4	DB2 for OS/390
Intersolv	3	DB2, Ingres, Informix, Oracle, etc.
Recital Corporation	3	DB2/6000, Informix, Ingres, Oracle
StarQuest Software	1	DB2 for MVS, DB2 UDB, DB2 CS
Trifox	3	Adabas, DB2, Informix, Rdb, legacy systems via GENESIS

Figure 3: Some commercially available drivers

```
catch (ClassNotFoundException e) {
     System.err.print("Error loading driver class: ");
     System.err.println(e.getMessage());
}
```

The forName() method is passed the name of the class that implements the JDBC driver. Note that the forName() method can throw a ClassNotFoundException, which must be accounted for using standard Java try-catch syntax. In the code example shown, we are using IBM's JDBC driver for DB2 for OS/390, namely ibm.sql.DB2Driver.

Explicitly loading the driver's class this way will cause the Java interpreter's class loader to load the class, if it has not already done so. If this is the case, Java will look to find out if the class has a static initializer section. If it does, Java will invoke that static code immediately after the class is loaded. That allows the driver to implement any required initialization code (eg instantiating itself, registering itself with the DriverManager, loading a dynamic link library if the driver uses native methods, etc) in such a way that the code is guaranteed to get called before any other method. There is no limit on the number of drivers that can be loaded by a single applet or application.

USING THE JDBC API

Once a viable driver has been loaded the API can be used to first connect to a data source and then execute statements against that data. The API defines six Java classes and eight Java interfaces that can be used in a Java applet or application. All fourteen are bundled into a Java package (Java-speak for class library) called java.sql. The six classes are listed here:

- DriverManager provides a basic service for managing a set of JDBC drivers. As part of its initialization, this class will attempt to load the driver classes referenced in the 'jdbc.drivers' system property.
- DriverPropertyInfo is of interest only to advanced programmers who need to interact with a driver via getDriverProperties() to discover or supply properties for connections.
- Types defines no methods, only constants that are used to identify SQL types. The actual constant values are equivalent to those defined by the X/Open consortium.
- Date is a thin wrapper around java.util.Date that allows JDBC to identify this as a SQL DATE value.
- Time is a thin wrapper around java.util.Date that allows JDBC to identify this as a SQL TIME value.
- Timestamp is a thin wrapper around java.util.Date that allows JDBC to identify this as a SQL TIMESTAMP value. It adds the

ability to hold the SQL TIMESTAMP nanosecond value.

In addition to those six classes, JDBC defines the eight interfaces listed here:

- Statement – a Statement object is used for executing an SQL statement and obtaining the results produced by it. Only one result set per statement can be open at any one time.
- PreparedStatement – an SQL statement is pre-compiled and stored in a PreparedStatement object. This object can be used to efficiently execute this statement multiple times. The statement may have one or more IN parameters.
- CallableStatement – used to execute stored procedures.
- Connection – a Connection represents a session with a specific database. Within the context of a Connection, SQL statements are executed and result sets returned.
- DatabaseMetaData – provides information about the database as a whole.
- Driver – the Java SQL framework allows for multiple database drivers. Each driver should supply a class that implements the Driver interface. This interface is not typically used by the Java application programmer.
- ResultSet – provides access to a table of data generated by executing a Statement.
- ResultSetMetaData – can be used to find out about the types and properties of the columns in a ResultSet.

The journey into the use of the API begins with the DriverManager class. It provides a method called getConnection that is used to establish a persistent connection to any of a number of available data sources. The general format of the method invocation is:

```
DriverManager.getConnection(name-of-data-source);
```

But there's a trick – the data source name must be specified in a special URL format:

```
jdbc:<subprotocol>:<driver-specific-stuff>
```

The subprotocol will generally be the name of a database product or vendor, while the third node will often be a server or location name, and may include a port number as well. For example, to connect to DB2 for OS/390, use a URL that looks like this:

```
jdbc:db2os390:location_name
```

where 'location_name' is defined in the DB2 catalog table SYSIBM.LOCATIONS. To connect to DB2 UDB on Unix, use a URL that looks like this:

```
jdbc:db2://server_name:port_nbr/db_name
```

To illustrate, let's assume that there is a DB2 subsystem known by the location name ORLANDO. Assume further that there is a table of automobile data (called INV_TAB) being used by a car dealership to keep track of its inventory. The goal is to first establish a connection to the subsystem, then a non-prepared Statement object. That Statement object is used to execute the SQL query represented by the String object referred to as q. While the next() method continues to return true, we continue retrieving the next row from the result set and send it to standard output (a rather trivial use of the data!). Then we close both the statement and the database connection. A JDBC example with dynamic SQL code to accomplish these tasks follows:

```
import java.sql.*;
public class JDBCExample1 {
  public static void main(String args[ ]) throws Exception {
     Class.forName("ibm.sql.DB2Driver");
     String sourceURL = "jdbc:db2os390:ORLANDO";
     Connection dbConn = DriverManager.getConnection(sourceURL);
     Statement stmt = dbConn.createStatement( );
     String q = "SELECT VIN, YEAR, MODEL FROM INV_TAB";
     q = q + " WHERE MSRP > 18000.00";
     ResultSet rs = stmt.executeQuery(q);
     while (rs.next( )) {
          String v = rs.getString(1);
          int y = rs.getInt(2);
          String m = rs.getString(3);
          System.out.println("Row data = " + v + " " + y + " " + m);
     }
     stmt.close( );
     dbConn.close( );
  }  // end of method
}  // end of class
```

Note that, since we know the column names, we had the option of retrieving column values using a slightly different technique:

```
String v = rs.getString("VIN");
int y = rs.getInt("YEAR");
String m = rs.getString("MAKE");
```

Column names used in getXXX() methods are case-insensitive. While the use of column names may be more self-documenting, the use of ordinal column numbers will probably provide slightly better performance. Furthermore, there are some cases where the column number is required.

The next example accomplishes the same task as the previous example, but now the statement is prepared (ie compiled) prior to execution:

```
import java.sql.*;
public class JDBCExample2 {
  public static void main(String args[ ]) throws Exception {
     Class.forName("ibm.sql.DB2Driver");
     String sourceURL = "jdbc:db2os390:ORLANDO";
     Connection dbConn = DriverManager.getConnection(sourceURL);
     String q = "SELECT VIN, YEAR, MODEL FROM INV_TAB";
     q = q + " WHERE MSRP > ?";
     PreparedStatement stmt = dbConn.prepareStatement(q);
     stmt.setDouble(1,18000.00);
     ResultSet rs = stmt.executeQuery();
     while (rs.next( )) {
          String v = rs.getString(1);
          int y = rs.getInt(2);
          String m = rs.getString(3);
          System.out.println("Row data = " + v + " " + y + " " + m);
     }
     stmt.close( );
     dbConn.close( );
  }
}
```

Only five lines of code have changed from the previous example. The question mark used in the WHERE clause is a marker parameter. A value must be provided for all marker parameters prior to running the query. The PreparedStatement interface provides one setXXX() method for each valid X/Open-defined data type; this is how IN parameters are set. Naturally, PreparedStatement objects can also be used for SQL with no parameters.

Although the example above doesn't show it, PreparedStatement

objects are most useful when executed repeatedly because the cost of statement compilation is incurred only once.

Now let's assume that we have a stored procedure called SHOW_CARS that does the same type of SELECT statement as seen in the previous two examples and returns a single result set. The code to run such a stored procedure might look like the following:

```
import java.sql.*;
public class JDBCExample3 {
  public static void main(String args[ ]) throws Exception {
     Class.forName("ibm.sql.DB2Driver");
     String sourceURL = "jdbc:db2os390:ORLANDO";
     Connection dbConn = DriverManager.getConnection(sourceURL);
     CallableStatement stmt = dbConn.prepareCall("{call SHOW_CARS}");
     ResultSet rs = stmt.executeQuery();
     while (rs.next( )) {
          String v = rs.getString(1);
          int y = rs.getInt(2);
          String m = rs.getString(3);
          System.out.println("Row data = " + v + " " + y + " " + m);
     }
     stmt.close( );
     dbConn.close( );        // could have been re-used
  } // end of method
} // end of class
```

When using stored procedures, the programmer has the option of the SQL-92 CALL syntax enclosed within curly brackets (the escape syntax) as shown above. The escape sequence should be favoured over the DBMS-specific call syntax because it leads to more portable code.

Also, although the example doesn't show it, JDBC CallableStatement objects support the use of IN parameters through methods inherited from PreparedStatement. Furthermore, OUT parameters can be registered using an overloaded method called registerOutParameter() that is specific to this interface.

In the event that the stored procedure returns two (or more) separate result sets, the code would have to look something like this:

```
CallableStatement stmt;
stmt = dbConn.prepareCall("{call SHOW_CARS}");
stmt.execute();  // notice the different method call!
ResultSet rs = stmt.getResultSet();
// process first result set (as shown previously)
stmt.getMoreResults();
```

```
rs = stmt.getResultSet();
// process second result set
```

This process can be repeated any number of times.

JDBC ENHANCEMENTS

The Java Platform 2, commonly known as JDK 1.2, was officially released in December 1998 to coincide with the Java Business Expo in New York. Some enhancements include the introduction of the Java Foundation Classes (JFC), CORBA support, and a new collection framework. A complete list of new features can be found at http://java.sun.com/products/jdk/1.2/docs/relnotes/features.html.

For the purposes of this article, the enhancements to the JDBC spec – officially called JDBC 2.0 – are the most interesting. The ResultSet interface was expanded to include new methods that support more robust ways to move through the rows in an answer set (see the following code) above and beyond the next() method that was shown in the previous code examples. Naturally, these new features can be exploited only if the driver being used supports the JDBC 2.0 spec.

```
import java.sql.*;
public class JDBCExample4 {
  public static void main(String args[]) throws Exception {
     Class.forName("ibm.sql.DB2Driver");
     String sourceURL = "jdbc:db2os390:ORLANDO";
     Connection dbConn = DriverManager.getConnection(sourceURL);
     Statement stmt = dbConn.createStatement();
     String q = "SELECT VIN, YEAR, MODEL FROM INV_TAB";
     ResultSet rs = stmt.executeQuery(q);
     rs.next( );   // sitting on first row
     rs.absolute(10);   // sitting on tenth row
     rs.previous( );    // sitting on ninth row
     rs.relative(-2);   // sitting on seventh row
     rs.relative(3);    // sitting on tenth row again
     rs.first( );  // sitting on the first row again
     stmt.close( );
     dbConn.close( );
  } // end of method
} // end of class
```

Also new for JDBC 2.0, the batch update facility allows a Statement object to submit a set of heterogeneous update commands together as a single unit to the underlying DBMS. Depending on the network infrastructure and JDBC driver, this approach can yield impressive

performance gains compared with the traditional approach of individual statement execution. As a rule, autocommit mode should be disabled to prevent JDBC from committing the transaction when executeBatch() is called.

The executeBatch() method submits a batch of commands to the underlying DBMS for execution. Commands are executed in the order in which they were added to the batch. ExecuteBatch() returns an array of update counts for the commands that were executed. A 'for' loop is a handy way to spin through the update counts, as follows:

```
import java.sql.*;
public class JDBCExample5 {
  public static void main(String args[]) throws Exception {
     Class.forName("ibm.sql.DB2Driver");
     String sourceURL = "jdbc:db2os390:ORLANDO";
     Connection dbConn = DriverManager.getConnection(sourceURL);
     dbConn.setAutoCommit(false);
     Statement stmt = dbConn.createStatement();
     stmt.addBatch("DELETE FROM INV_TAB WHERE YEAR < 1990");
     stmt.addBatch("UPDATE INV_TAB SET MSRP = MSRP * 1.05");
     int[] counts = stmt.executeBatch();
     for (int x = 0; x < counts.length; x++) {
          System.out.print("Nbr rows affected by statement # " + x);
          System.out.println(" is " + counts[x]);
     }
     dbConn.commit();
     stmt.close( );
     dbConn.close( );
  } // end of method
} // end of class
```

Only DDL and DML commands that return a simple update count may be executed as part of a batch. Also, although the example doesn't show it, PreparedStatements and CallableStatements can be used in batch updates.

FURTHER READING

There's a lot more to JDBC than I have covered in this article. To expand upon your new-found knowledge, consider purchasing one of the many books on the market that cover the topic of JDBC either directly or indirectly. Two of the best are *JDBC Database Access with Java* by G Hamilton *et al* (ISBN 0-201-30995-5) and *Teach Yourself Database Programming With JDBC in 21 Days* by Ashton Hobbs (ISBN 1-57521-123-8).

Also worthwhile is a visit to IBM's *DB2 for OS/390 Database Connectivity* Web site at http://www.software.ibm.com/data/db2/os390/jdbc.html.

The good folks at the IBM International Technical Support Centers have written a relevant redbook entitled *Integrating Java with Existing Data and Applications on OS/390* (SG24-5142-00). This is also available at http://publib.boulder.ibm.com/pubs/pdfs/redbooks/sg245142.pdf. Finally, Sun maintains a Web site devoted to JDBC at http://java.sun.com/products/jdbc/index.html. It's information straight from the horse's mouth.

Editor's note: in a future article, the author will examine the emerging standard of embedded SQL in Java, SQLJ.

If you would like to discuss this article further, the author can be contacted at jbradford@gr.com.

John T Bradford
Greenbrier and Russel (USA)

Timestamp checking program

THE PROBLEM

Every DBA and programmer has experienced the dreaded –818 or –805 error, indicating that the version of the load module you are running doesn't match the version of the package or plan in the DB2 catalog. If I had a dollar for every one of these that I have found and fixed, I think I would be a rich man today. With the advent of automated software configuration managers (such as Endevor or ChangeMan) the number of occurrences has been minimized, but the recent addition of a non-Endevor managed Y2K testing environment to our site brought up the problem again. I decided to develop something that would enable us to do a timestamp 'health check' on a given group of load modules so that we could quickly determine what needed to be fixed and what could be done to fix it.

OUR SOLUTION

Timestamps or consistency tokens can be found easily for a given package or DBRM from the catalog, but I know of no quick and accurate way to find the timestamp within the object code that makes up the load module. The COBOL II compiler stores the timestamp in a section of the load module called the Constant Global Table (CGT) as two four-byte literals which, for some reason best known to the compiler, have their order reversed from that in which they are stored in the catalog (ie LOAD timestamp = catalog timestamp bytes 5-8 plus catalog timestamp bytes 1-4). I decided to retrieve the timestamp (or list of timestamps in the case of package versions) from the catalog and the DBRM and then search through the load module to see whether I could find a string that matched. If I found a match I considered that the load module was OK; if I didn't, I assumed that there would be a timestamp error. This approach is not foolproof, but it will certainly work in 99% of the cases, which is a lot better than what we had.

This situation is further complicated by the use of static linking. If all modules are dynamically linked then we can assume that there is a one-to-one correspondence between a load module and a package, but if static linking is in use then a single load module can contain CSECTs that correspond to many packages (and conversely a single package can correspond to a CSECT in multiple load modules). To get around this, I used the AMBLIST utility to dump out information about the CSECTs contained in the module and built a table in my program that cross-referenced module and CSECT information. Depending on how many modules you are checking, the AMBLIST step can be quite time-consuming, so I set this as an option to be used only if necessary.

ASSUMPTIONS AND DEPENDENCIES

The following code allows for static or dynamic linking. It assumes that packages are being used (ie static plans are not supported, although it should not be too hard to add this) and package versioning is catered for. The program currently only checks COBOL II programs, but it should be simple to extend it to other language types once you know the format of the timestamp string within the load module. The timestamp check program uses DB2REXX, a REXX-DB2 interface

program from *General DB2/REXX interface*, *DB2 Update*, Issue 53, March 1997, but any program that allows communication between REXX and DB2 could be substituted.

There are a number of site-specific areas in the code. Firstly, our site standards allow us to assume that each application has its own package collection, and that this will match the first two characters of the program name. I use this information to restrict the number of programs checked so that the elapsed time will be reasonable (particularly when using AMBLIST processing), but you can break it up in whatever way is appropriate to your installation. Secondly, the load and DBRM library names are site-specific and you will need to tailor these for your installation. Lastly, it is possible that the format of the AMBLIST output will vary from site to site, particularly if you are using a different linkage editor (we use the MVS binder), so if you use the static link option it may be necessary to tweak the code that builds the cross reference list.

PROGRAM OUTPUT

The timestamp check program checks all combinations of the catalog, program, and DBRM timestamps to see whether they match. The amount of output received is controlled by the first parameter to the program, the error level. If this is 'E', only error level messages are displayed. This means that error messages are displayed only if the catalog and the program do not match (ie the program will receive a –805 when it runs). This is the normal mode that the program should be run in. If the error level is 'W' then all mismatches will be displayed regardless of whether they will cause the program to fail or not (for instance catalog does not match DBRM or DBRM does not match catalog).

Some sample output is shown below:

```
Error level = E, only load module timestamp errors will be printed
Dynamic linking specified, module xref table will not be built

Reading package details from catalog
Running timestamp comparison
TN1Ø73Ø
   No package timestamp matched in load module TN1Ø73Ø
   DBRM timestamp does not match any package
   Load module and DBRM timestamps match
```

```
TN2Ø25Ø
   No package timestamp matched in load module TN2Ø25Ø
   DBRM timestamp does not match any package
   Load module and DBRM timestamps do not match
```

Normally, you will be concerned only about whether the program matches the catalog, and, if not, whether the program matches the DBRM. If the program and the DBRM do match, simply binding the package again will fix the problem. If they do not, you will most likely have to recompile the program. Note that because error level is set to E in this example, DBRM/catalog and load module/DBRM matching messages are printed only when the load module and the catalog do not match.

THE CODE

Three objects are supplied:

- GENAMB – this REXX EXEC generates the AMBLIST commands necessary to produce the module cross reference listing. This will be required only if you are using static linking. This is invoked using GENAMB xx, where 'xx' is the prefix for the load modules you wish to generate commands for. This prefix may be any length.
- TSCHECK2 – this REXX EXEC is the timestamp checking program and is invoked with the following parameters:

  ```
  TSCHECK2 ssid collid errlvl linkopt
  ```

  where:

  – 'ssid' is the DB2 subsystem-id to check against.

  – 'collid' is the name of the collection to check.

  – 'errlvl' is the error level, as described above (E or W).

  – 'linkopt' is a flag to denote whether static linking is used (Y for static linking, N for dynamic).
- JCL to run the programs. Some tweaking will be required if you are not going to use the static linking option (ie remove the first two steps and modify the IF processing).

GENAMB

```
/*  REXX */
arg appl
say ""

/*  The following dataset names are site specific.  Be sure to */
/*  alter them to match your site's naming standards           */
if appl = "EL" then env = 'CPSR'
else env = 'COMM'
loadlib.Ø = 4
loadlib.1 = "TLM."env".ACPT.CICSLOAD"
loadlib.2 = "TLM."env".PROD.CICSLOAD"
loadlib.3 = "TLM."env".ACPT.LOADLIB"
loadlib.4 = "TLM."env".PROD.LOADLIB"
dsname = ""
do i = 1 to loadlib.Ø
  dsname = dsname||"'"||loadlib.i||"' "
end
pattern = appl"*"
"ALLOC F(LOADSCAN) DA("dsname") SHR REUSE"
ADDRESS ISPEXEC "LMINIT DATAID(LOADDSID)  DDNAME(LOADSCAN) ENQ(SHR)"
ADDRESS ISPEXEC "LMOPEN DATAID("loaddsid") OPTION(INPUT)"
ADDRESS ISPEXEC "LMMLIST DATAID("loaddsid") OPTION(LIST)",
     "MEMBER(LOADMEM) STATS(YES) PATTERN("pattern")"
listcc = rc
if rc <> Ø then do
  if rc = 4 then say "No members match pattern" pattern
  else say "Error executing LMMLIST (1). Return code =" listcc
  zispfrc = 8
  ADDRESS ISPEXEC "VPUT (ZISPFRC)"
  exit 8
end
say "AMBLIST commands generated for the following members"
say ""
do while (listcc = Ø)
  say "  " loadmem loadlib.zllib
  queue "  LISTLOAD OUTPUT=XREF,DDN=LOADLIB"zllib",MEMBER="loadmem
  ADDRESS ISPEXEC "LMMLIST DATAID("loaddsid") MEMBER(LOADMEM)",
      "STATS(YES)"
  listcc = rc
end
if listcc > 8 then do
  say "Error executing LMMLIST (2). Return code =" listcc
  zispfrc = 8
  ADDRESS ISPEXEC "VPUT (ZISPFRC)"
  exit 8
end
queue ""
say " "
say queued() "AMBLIST commands written to output"
```

```
"EXECIO * DISKW AMBCMD (FINIS"
if rc <> 0 then delstack
exit:
ADDRESS ISPEXEC "LMCLOSE DATAID("loaddsid")"
ADDRESS ISPEXEC "LMFREE DATAID("loaddsid")"
"FREE F(LOADSCAN)"
out:
exit
```

TSCHECK2

```
/*  REXX */
arg ssid appl errlvl static_flag
/*                                                                  */
/*  Explanations of input parameters:                               */
/*     ssid - DB2 subsystem ID to check timestamps in               */
/*     appl - collection ID to check                                */
/*     errlvl - 'E' print messages only on timestamp error          */
/*            - 'W' print all messages and warnings                 */
/*       Defaults to E                                              */
/*     static_flag - 'N' assume all load modules are dynamically    */
/*                    linked (ie no AMBLIST processing)             */
/*                 - 'Y' assume load modules are statically         */
/*                    linked and build module cross reference table*/
/*                                                                  */
maxrows = 5000
prev_pkg = ""
pkg_timestamp. = ""
pkg_count = 1
zispfrc = 0
if errlvl = "" then errlvl = "E"
if static_flag = "" then static_flag = "N"
select
  when errlvl = "E" then,
say "Error level = E, only load module timestamp errors will be printed"
  when errlvl = "W" then,
    say "Error level = W, all errors and warnings will be printed"
  otherwise do
    say "Invalid value for error level supplied - " errlvl
    say "Error level E assumed"
    errlvl = "E"
  end
end
select
  when static_flag = "Y" then,
    say "Static linking specified, module xref table will be built"
  when static_flag = "N" then,
    say "Dynamic linking specified, module xref table will not be built"
  otherwise do
```

```
    say "Invalid value for static flag supplied" static_flag
    say "Dynamic linking assumed, module xref table will not be built"
    static_flag = "N"
  end
end

say ""
if static_flag = "Y" then do
  /*  Build module cross reference table    */
  say "Building module cross reference table"
  call BuildXREF
end
/*  Allocate and open load and DBRM libraries  */
call AllocLibs
/*  Read package details from DB2 catalog                          */
say "Reading package details from catalog"
query = "SELECT NAME, CONTOKEN",
        "FROM SYSIBM.SYSPACKAGE",
        "WHERE COLLID = '"appl"'",
        "ORDER BY 1"
ADDRESS ISPEXEC "SELECT PGM(DB2REXX)"
if rc <> 0 | (sqlcode <> 0  & sqlcode <> 100)| cafcode <> 0 then do
  say "Error occurred selecting from SYSTABLES"
  say query
  say "  RC = " rc
  say "  SQLCODE = " sqlcode
  say "  CAFCODE = " cafcode
  say "  CAFREASON = " cafreason
  zispfrc = 8
  signal exit
end
say "Running timestamp comparison"
do i = 1 to rows
  /*  Build a list of timestamps for versions of each package     */
  /*  When package name changes we can start to run the checks    */
  if name.i <> prev_package & i <> 1 then do
    call RunChecks
    pkg_count = 1
  end
  /*  Build up table of timestamps for each package version */
  pkg_timestamp.pkg_count = contoken.i
  pkg_count = pkg_count + 1
  prev_package = name.i
end
call RunChecks
exit:
ADDRESS ISPEXEC "LMCLOSE DATAID("loaddsid")"
ADDRESS ISPEXEC "LMFREE DATAID("loaddsid")"
ADDRESS ISPEXEC "LMCLOSE DATAID("dbrmdsid")"
ADDRESS ISPEXEC "LMFREE DATAID("dbrmdsid")"
"FREE F(LOADSCAN)"
```

```
"FREE F(DBRMSCAN)"
ADDRESS ISPEXEC "VPUT (ZISPFRC)"
exit

CheckLoad:
  ADDRESS ISPEXEC "LMMFIND DATAID("loaddsid") MEMBER("module")",
       "STATS(YES)"
  if rc > 8 then do
    say "Error executing LMMFIND for load member" module
    zispfrc = 8
    signal exit
  end
  if rc = 8 then return
  if rc = Ø then found_load = "YES"
  get_rc = Ø
  /*  Read each record from the given load module and check whether  */
  /*  we find a match with either the package (ie catalog) timestamp */
  /*  or the DBRM timestamp.  Remember that the timestamp is         */
  /*  stored with the first and second full words reversed in a      */
  /*  load module                                                    */
  do until (get_rc > Ø)
    ADDRESS ISPEXEC "LMGET DATAID("loaddsid") MODE(INVAR)",
        "DATALOC(LOADLINE) DATALEN(LENVAR) MAXLEN(3276Ø)"
    get_rc = rc
    if get_rc > 8 then do
      say "Error during LMGET for load module" prev_package
      say "Return code = " rc
      zispfrc = 8
      signal exit
    end
    dbrm_load_timestamp = substr(dbrm_timestamp,5,4)||,
                          substr(dbrm_timestamp,1,4)
    if pos(dbrm_load_timestamp,loadline) > Ø then,
      match_load_dbrm = "YES"
    do j = 1 to pkg_count - 1
      load_timestamp = substr(pkg_timestamp.j,5,4)||,
                       substr(pkg_timestamp.j,1,4)
      if pos(load_timestamp,loadline) > Ø then do
        match_load = "YES"
        leave
      end
    end
    if match_load_dbrm = "YES" & match_load = "YES" then return
  end
return
```

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

Matthew Keene (Australia)

Using a relational database for data warehouses

Editor's note: although this article is independent of RDBMS, it is appropriate for DB2, as well as for Oracle7, Sybase, Ingres, etc.

WHAT IS A DATA WAREHOUSE?

A data warehouse is a concept not a product. It is the compiling, assembling, and consolidating of application data common to user communities at a single logical point. Typical uses include *ad hoc* queries, 'what if' queries, data matching, trend analysis, and other sophisticated information functions.

Warehouse data is generally extracted from OLTP databases that are optimized for transaction processing. Data warehouses are optimized for information processing.

A data warehouse can be described as a read-only database that provides users with access to consolidated, historic, or static data extracted from operational databases, usually augmented with external data.

Read-only and static data require breaking application mindsets. Building a data warehouse means abandoning many traditional IT concepts including methodology, normalization, and back-up and recovery.

METHODOLOGY

Standard IT methodology is built for application systems – a data warehouse is not an application system. ISO9001 methodologies usually require:

- A description of business processes and flows.
- A description of application system interfaces.
- A conceptual data model and entity descriptions.
- Unit, module, and integration testing.

A data warehouse has none of these:

- There are no business processes because the data warehouse is providing information to improve decision-making.
- There are no application system interfaces because the data warehouse only extracts the data it needs.
- There is no conceptual model because the data warehouse schema is an amalgamation schema combining data from multiple operational and external databases.
- There is no unit, module, or integration testing because there are no units.

Most methodologies do allow some activity overlap but are essentially sequential. Almost all data warehouse activities can be done in parallel.

NORMALIZATION

Normalization is another ISO9001 standard. The question is not should normalization be used but to which form (level). The fundamental normalization rationale is the elimination of storage anomalies. Figure 1 shows a table of the differences between OLTP and data warehouse applications.

Note the response time for the data warehouse (*). Having a response time of hours is untimely. Traversing hundreds of gigabytes or tens of terabytes takes time. It is important to reduce that time to the minimum to make the data warehouse as useful as possible. Normalization produces many tables.

Queries that need data from many tables must do an SQL join. Joins take time and that time increases geometrically as more tables are involved. The obvious answer is a denormalized schema – its name is the star schema and this is discussed later.

BACK-UP AND RECOVERY

Back-up takes a considerable time. A recent test showed that a system with fifteen high-speed tape drives could achieve a back-up rate of about 500GB per hour. It would take this expensive system 20 hours

Function	*OLTP*	*Data warehouse*
Data content	Current values	Archival and aggregations
Data organization	Application	Amalgamation
Data nature	Dynamic	Static; read only
Data structure	Complex	Simple
Accessibility	High	Low to moderate
Usage	Highly structured	Highly unstructured
Response time	<2 seconds	*

Figure 1: OLTP and data warehouse comparison

to back-up a 10TB data warehouse. Less efficient systems can easily require more than 24 hours, making the data warehouse inaccessible.

A data warehouse is a database, and databases must be backed-up so that they can be recovered in case of data loss. That is true for standard databases but is untrue for read-only data warehouses.

Because there are no updates, there is no need to back-up the entire database every time. The only back-up required is for new data that has been appended. The data warehouse append window should allow for the creation of an incremental tape. The incremental tapes can be used for recovery. If the incremental tapes become unusable then the final fall-back position is to use the operational system's back-up tapes to recreate the data warehouse.

DATA WAREHOUSE SCHEMA

The data warehouse schema should not be an OLTP schema, as shown by Figure 1. OLTP schemas are typically normalized to provide efficient data access for a large number of transactions using few rows from a few tables.

Data warehouse queries usually involve accessing many rows. Data warehouse queries can easily access gigabytes or terabytes of data. If those many rows are in many normalized tables then the performance penalty is severe.

The data warehouse schema must be denormalized to provide minimum tables. The minimum number is one, but one row is impractical because of excessive DASD requirements.

The recommended schema is a star or variation of this. A star schema is divided into two table types – fact and dimension. The fact table is at the star centre whereas the dimension tables are at its points. The fact table contains the columns to be measured such as sales and units. The dimension tables hold the measuring columns such as date and location.

Fact tables are usually displayed horizontally:

```
date_dimkey   location_dimkey    $$$  units
```

dimkey is the 'foreign key' to the dimension table, such as date. The date_dimkey does not contain the actual date but is an odometer. The first date in the data warehouse is numbered 1. Dimension tables are usually displayed vertically because of their hierarchical nature.

date_dimkey	57
year	1999
quarter	first
month_number	2
month_name	February
week_number	9
day	26
dayofweek	Friday

Figure 2: Example date dimension table

A date dimension table is shown in Figure 2, giving example values for 26 February 1999. Certain columns in Figure 2, such as month_name and dayofweek, are descriptive text. Dimension tables allow rapid drill down or up because they are arranged in hierarchical order. It is unnormalized because February would be redundantly stored 28 times. If month_number is used as a foreign key to month_name then an additional join is required. The goal of a star schema is to reduce joins.

Recent benchmarking has shown that a query that took about 60 minutes using OLTP schema was reduced to about 1 minute using star – an improvement of 60 times!

Sometimes, DASD considerations require that descriptive text is normalized. This variation of star is a snowflake. Studies also show that most queries require SUM. Summing a billion rows is not fast. The better way is to build aggregation tables:

- Fact by date
- Fact by location
- Fact by date by location.

The last aggregation table can be very large because it requires a 'Cartesian product' of date and location. If there are 1,000 locations, then the table contains 365/6 thousand rows. A schema having some normalization and aggregations is called a melted snowflake.

It should be evident that a melted snowflake schema will differ substantially from a typical OLTP schema, regardless of its normalization level. Physical implementation of a star or variation requires careful planning, including using little known features.

Editor's note: in a future article, the author will provide a logical melted snowflake schema with its physical schema and will provide specific recommendations for DB2.

Eric Garrigue Vesely
Principal
Analyst Workbench Consulting (Malaysia)

BMC has announced XBM Enterprise Snapshot, aimed at increased automation, availability, and reliability of CICS and DB2, IMS, and VSAM databases. The software enables BMC tools and utilities to take advantage of the various snapshot point-in-time copies of data now available. This allows those utilities to perform processing in parallel with regular production workloads, resulting in greater applications availability.

Utilities enabled by XBM include COPY PLUS for DB2, REORG PLUS for DB2, UNLOAD PLUS for DB2, and CHECK PLUS for DB2.

For further information contact:
BMC Software, 2101 CityWest Boulevard, Houston, TX 77042-2827, USA.
Tel: (713) 918 8800.
BMC Software, Compass House, 207-215 London Road, Camberley, Surrey, GU15 3EY, UK.
Tel: (01276) 24622.
URL: http://www.bmc.com.

* * *

Cisco Systems has announced Cisco Transaction Connection (CTRC), providing TCP/IP end users and servers with access to DB2 databases using SNA. The software and router system based on Cisco IOS software is designed to replace Unix and NT gateways for database access.

Jointly developed with StarQuest Software, CTRC uses the Distributed Relational Database Architecture (DRDA) protocol to access remote databases using a standard messaging format over TCP/IP or SNA. It converts TCP/IP data requests into SNA messages and forwards them to the host.

For further information contact:
Cisco, 5305 Gulf Drive. Suite 1, New Port Richey, FL 34652, USA.
Tel: (813) 817 0131.
URL: http://www.cisco.com.

* * *

DB2 users can benefit from Software AG's announcment of support for Java applications on OS/390 mainframes. Although Bolero creates Java Byte Code that can theoretically run on all platforms that have a Java Virtual Machine, Java implementations are different on mainframes, NT, and Unix. The OS/390 version supports mainframes specifically, and allows Bolero applications to store persistent objects in DB2 databases and be used in the CICS Open Transaction Environment.

For further information contact:
Software AG (UK), Charter Court, 74/78 Victoria Street, St Albans, Herts, AL1 3XH, UK.
Tel: (01727) 844 455.
Software AG of North America, 11190 Sunrise Valley Drive, Reston, VA 22091, USA.
Tel: (703) 860 5050.
URL: http://www.software-ag.com.

* * *