110

DB2 update

December 2001

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 1997 issue, are available separately to subscribers for £22.50 ($33.75) each including postage.

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

Editor
Trevor Eddolls

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

Don’t lose your DB2 messages

I have recently seen a couple of MVS/JES3 sites where the DB2 master messages were deleted immediately DB2 terminated. Then the output could only be seen by looking in the SYSLOG, which is really not very convenient.

The output from all started tasks was treated the same way.

A simple solution would be to use the following:

```
//D2E1MSTR PROC LIB='D2EØ.DSNEXIT'
//*
//***************************************************************/
//*        THIS STEP PREVENTS OUTPUT BEING IMMEDIATELY DELETED */
//***************************************************************/
//DUMMY   EXEC PGM=IEBGENER
//SYSPRINT  DD DUMMY
//SYSUT1    DD DSN=DUMMY.TEXT,DISP=SHR
//SYSUT2    DD SYSOUT=T       <— where T is a hold output class
//SYSIN     DD DUMMY
//*
//***************************************************************/
//*        INVOKE DB2                                           */
//***************************************************************/
//IEFPROC EXEC PGM=DSNYASCP,
//             DYNAMNBR=119,
//             REGION=ØM,
//             PARM='ZPARM(DSNZD2E1)'
//STEPLIB   DD DISP=SHR,DSN=&LIB
//          DD DISP=SHR,DSN=D2EØ.DSNLOAD
//BSDS1     DD DISP=SHR,DSN=D2E1.BSDSØ1
//BSDS2     DD DISP=SHR,DSN=D2E1.BSDSØ2
//SYSABEND  DD SYSOUT=D,CHARS=DUMP,FCB=STD3
```

The file DUMMY.TEXT holds one line of text and, as long as that output is on the queue, you can still see all the other messages.

Put the DUMMY step first, then the MVS enqueue for file DUMMY.TEXT is released as soon as that step ends. Otherwise JES3 initiates an enqueue at the start of the started task and maintains it until DB2 is stopped.

This technique could be used with any other started tasks too.

Ron Brown (Germany) © Xephon 2001

Introducing DB2 Everyplace Version 7.2

My, how the world of e-business is changing! It seems like only months ago (but it is really years ago) that business demanded powerful laptop computing to enable employees to do more, to sell more, to create more business. Ah, yes, the promise of e-business and a mobile workforce.

There was a problem with this promise, however. Sure, we have seen the proliferation of ‘dot-coms’ in the millions, and every company these days has some sort of e-business logo, catch-phrase, or marketing collateral. The problem with traditional e-business is that workers were still tied to the bricks-and-mortar of their enterprise, or to hotel rooms – the need to ‘plug-in’ stifled many of the promises that e-business was to deliver. Even powerful laptops need an Ethernet, dial-up, or Token Ring connection, or some sort of Plain Old Telephone Service (POTS) hook-up to keep in touch with the enterprise.

As the spread of pervasive devices such as cellular phones, Personal Digital Assistants (PDAs), and other sophisticated embedded devices continues, e-business has morphed into what we now call ‘pervasive e-business’. The world’s top cellular phone manufacturers (Nokia, Ericsson, Samsung, and Kyocera) all offer some form of integrated PDA with their ‘coolest’ cellular phones. These technologies bring e-business to mobile employees, enabling a flexible and productive work style that is independent of location.

So now that a truly mobile technology infrastructure exists for pervasive e-business, companies are beginning to demand the power of enterprise-class computing (in the form of ‘killer apps’) on these mobile devices. For example, an insurance sales force requires quotations for proposals or scheduling information for customer meetings on a particular day. Vending-machine service people require inventory information for each machine on their scheduled routes, and they must update this data as they restock items. Home health care workers and visiting nurses must be able to download lists of scheduled patients, as well as each patient’s health statistics, and be able to update this information as examinations are conducted throughout the day. A device is illustrated in Figure 1.

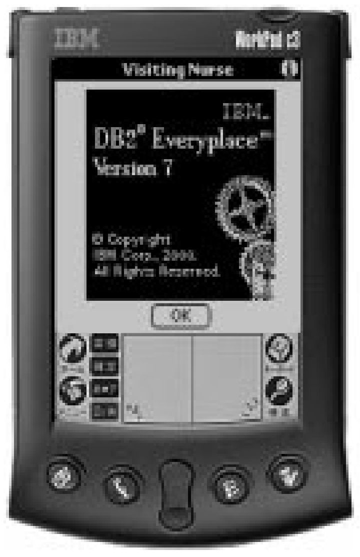


Figure 1: DB2 Everyplace on a PDA

Although all of these mobile workers have diverse needs for data access, all require a reliable software solution that allows them to access their organization's data locally on a mobile device, to modify this data, and then to synchronize these changes with a database on a remote server – all in a timely fashion! The key here is that information collected in the field has to be transported back to the enterprise in a timely, secure, reliable, and efficient manner. This information is fed into transaction-based systems, or data warehousing, Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), or Supply Chain Management (SCM) systems.

There are many reasons for the sudden surge in demand for mobile applications. Mobile computing technology has finally reached the level of performance and capacity needed for enterprise applications. What's more, storage has evolved from a constrained 2MB maximum to a standard 8-32MB and more (IBM MicroDrive enables 1GB of storage on a handheld device!). All of these advances in technology are available in devices whose cost is declining. IDC forecasts that the average price for a smart handheld device will drop from $600 in 2001 to $400 in 2004. Growth in wireless access is creating greater flexibility in how enterprise data is accessed from a mobile device. The IBM Pervasive Computing Division estimates that by 2003 there will be over 1 billion wireless subscribers worldwide, 80% of new

PDAs will be Web-connected, and 63% of Web transactions will be generated from mobile devices. Think about that for a moment! *Sixty-three percent* of Web transactions will *not* originate from the workstations or laptops that we work with everyday. These trends point to a changing e-business landscape, and DB2 Everyplace is leading that change.

The IBM DB2 Everyplace solution provides one- and two-way synchronization of files and data between an enterprise data source and mobile and embedded devices, all in a secure, scalable, open, and robust environment.

THE DB2 EVERYPLACE ENVIRONMENT

A typical DB2 Everyplace environment is shown in Figure 2.

The left hand side of Figure 2 shows the DB2 Everyplace development environment. Mobile applications are migrated from the enterprise –

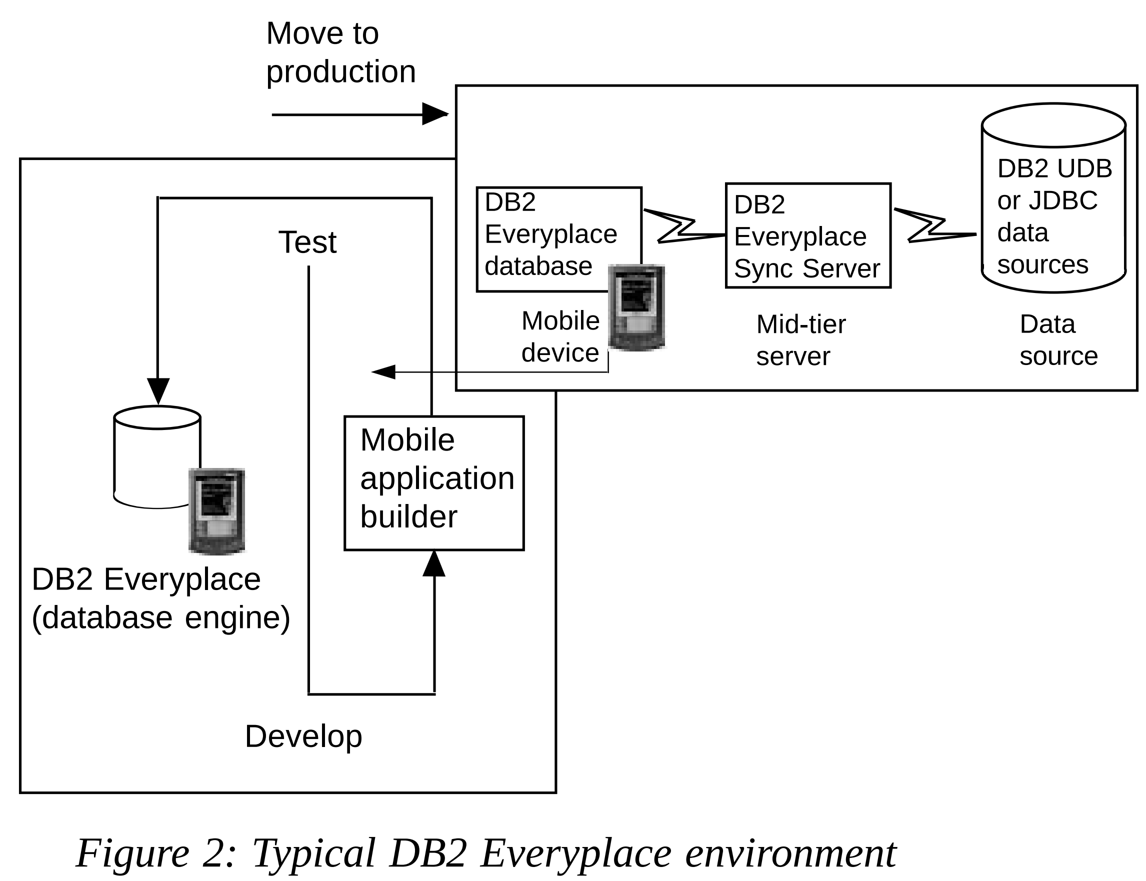

Move to production
DB2 Everyplace database
Mobile device
DB2 Everyplace Sync Server
Mid-tier server
DB2 UDB or JDBC data sources
Data source
Test
Mobile application builder
DB2 Everyplace (database engine)
Develop

Figure 2: Typical DB2 Everyplace environment

or built from the ground up – to run on the pervasive device. DB2 Everyplace comes with the DB2 Everyplace Mobile Application Builder, which can be used to rapidly prototype or build mobile applications. You can also use any off-the-shelf development tool.

Once an application is built and tested, it is deployed to the workforce. The application accesses DB2 Everyplace. As employees use the application, information is synchronized through the DB2 Everyplace SyncServer and ultimately used to update the most valuable asset of the company – its enterprise data store.

THE DB2 EVERYPLACE ENGINE

DB2 Everyplace is an easy-to-use and interoperable relational database engine for pervasive devices such as PDAs, cellular phones, and various embedded devices. DB2 Everyplace is considered easy to use because it relies on standardized interfaces that are accepted in the database community – interfaces such as Java DataBase Connectivity (JDBC), Open DataBase Connectivity (ODBC), or the Call Level Interface (CLI). The DB2 Everyplace solution is considered to be interoperable because it conforms to the Synchronization Mark-up Language (SyncML) specification, an emerging standard that defines the way synchronization information is exchanged. The IBM solution recognizes that eXtensible Markup Language (XML) will provide the basis for data interchange between heterogeneous hardware and software environments, and is the only major pervasive database solution to provide a synchronization architecture that conforms to the SyncML standard.

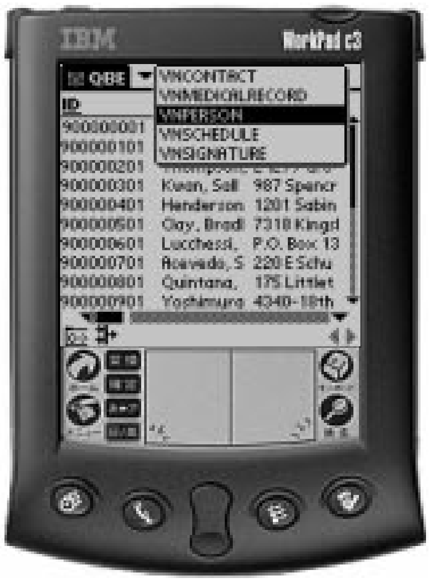
DB2 Everyplace is available on all the major pervasive computing platforms, including: PalmOS, EPOC V5 and V6, Neutrino, Linux and Embedded Linux, Windows CE, and Windows 32-bit operating systems (Windows 95, 98, ME, NT, and 2000).

The amount of space that an application takes up on a device is often referred to as its ‘footprint’. With a ‘fingerprint’ of 150KB, DB2 Everyplace is the smallest relational database engine available on the market today.

DB2 Everyplace is a powerhouse with high performance technologies like main memory database techniques, scrollable cursors, and bi-directional indexes available to all applications. Expect to find Data Definition Language (DDL); a broad spectrum of Data Manipulation Language (DML); constraints and referential integrity support that allows your applications to handle business rules at the database level, and removes the need to add this logic at the application layer; relational operations such as joins; transaction support; the ability to run stored procedures; a wide variety of data types; and much more. DB2 Everyplace offers features associated with relational database management systems.

When running in a Windows 32-bit, Neutrino, or Linux environment, DB2 Everyplace applications can run from read-only media such as DVDs, CD-ROMs, memory sticks, flash cards, or local directory structures. This feature is important, for example, to travelling sales persons who may not have a reliable Internet connection to their enterprise data. For example, a US mid-west crop insurance agent might receive monthly updates via a CD-ROM that is mailed to a remote office. The sales agent simply inserts the CD-ROM into the drive and the application runs from that drive to access all required insurance quotations – see Figure 3.

DB2 Everyplace also comes with tools that let you access your data



```
900000001
900000101
900000201
900000301   Kwan, Sall   987 Spencr
900000401   Henderson    1201 Sabin
900000501   Clay, Bradl  7318 Kingd
900000601   Lucchesi,    P.O. Box 13
900000701   Acevedo, S   220 E Schu
900000801   Quintana,    175 Littlet
900000901   Yoshimura    4340-18th
```

Figure 3: Running applications

without the need to write a single line of code or to construct a single SQL statement. The Query-by-Example (QBE) application empowers you to work with your data ‘out-of-the-box’, with a GUI to horizontally and vertically scroll, update, insert, and delete data stored in your DB2 Everyplace database.

IBM recognizes that today’s businesses don’t just transact across heterogeneous arrays of software and hardware, but in different countries and across different regions of the world. DB2 Everyplace supports Single-Byte Character Set (SBCS)-based, Double-Byte Character Set (DBCS)-based, and UNICODE environments.

THE BRAINS OF SYNCHRONIZATION

The DB2 Everyplace solution comes with an optional HTTP-based bi-directional synchronization infrastructure that allows mobile users to submit changes they have made to local copies of source data. Clients also receive any changes that have been made to source data residing on the enterprise server since the last time they synchronized. Collectively, the management tools and the components of synchronization are known as the Synchronization Server, or SyncServer for short. The SyncServer provides efficient two-way synchronization between mobile devices and DB2 or JDBC-compliant database management systems such as Informix, Oracle, Sybase, and SQL Server (using a third-party driver). The SyncServer runs on Windows NT/2000, Linux, AIX, and the Solaris operating environment.

Connections to JDBC-compliant databases are handled through a JDBC adapter. The JDBC adapter was introduced in DB2 Everyplace Version 7.2 and works with existing DB2 Everyplace clients to deliver relational data to mobile devices. If synchronization will only be performed with a DB2 family database, you can use the DataPropagator (DProp) adapter. DProp adapters use DB2 replication software that is free of charge with distributed versions of DB2. The JDBC adapter does not use DataPropagator for replication; instead it relies on custom code created to work with triggers in the source database. When using the JDBC adapter, users are not required to understand the complexities of a federated environment – the SyncServer handles this transparently.

The SyncServer can be set up in a two-tier or a three-tier environment.

In most implementations, administrators will want to offload the synchronization work from production systems and choose to use a three-tier environment as shown in Figure 4.

Synchronization information is composed in SyncML (a standard XML-based mark-up language for synchronization) and transported in Wireless Binary eXtensible Markup Language (WBXML) for more efficient use of a connection's bandwidth. The transmitted data can optionally be encrypted (at 56- or 128-bit levels) to prevent unauthorized parties from 'sniffing the connection' and logging the data in transit.

The SyncServer requires a Java-based application servlet and DB2 underneath the covers to work. A free version of DB2 Workgroup Edition, for use by the SyncServer, is shipped with DB2 Everyplace. The SyncServer works with any Java-based application server. The application servlet piece of WebSphere Application Server (WAS) is shipped as part of the SyncServer. The SyncServer installation program has recently been enhanced to hide the complexities of installing the SyncServer's required components; it is now literally an 'out-of-the-box' implementation. To support a large number of concurrent synchronizations, administrators should consider scaling the SyncServer with a full feature version of WAS in a Symmetric Multiprocessing (SMP) environment.

The entire DB2 Everyplace environment is managed through an extension of the Control Center called the Mobile Devices Administration Center (MDAC). The MDAC is a tool that provides

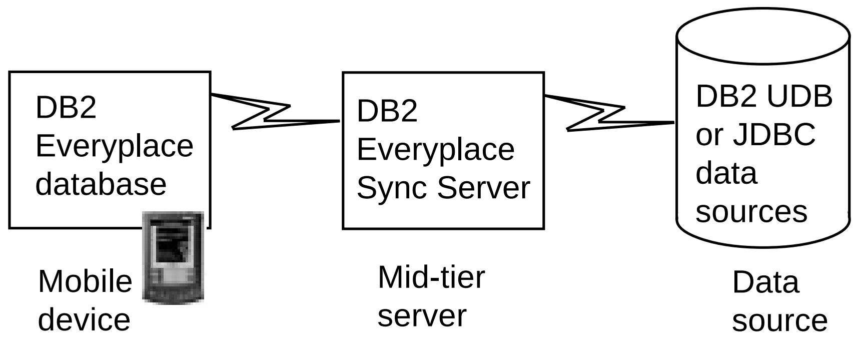


Figure 4: Three-tier environment

synchronization services to groups of users who have similar mobile data synchronization needs. The MDAC allows administrators to perform all sorts of functions. For example, data sources can be horizontally or vertically partitioned to control the data that is synchronized; new indexes can be defined for mobile devices; check clause support can be implemented to enforce business rules; and conflict notifications can be defined. The key feature here is that the MDAC and the SyncServer offer a central point of management for the DB2 Everyplace environment. End users have no idea that DB2 Everyplace is on their devices!

APPLICATION DEVELOPMENT IN DB2 EVERYPLACE-LAND

Standard interfaces such as JDBC and ODBC/CLI can be used by applications to access data that resides in a DB2 Everyplace database. Applications can be built by using C, C++, or Java 'off-the-shelf' development products, or by using the native Software Development Kit (SDK) on your device. For example, Metrowerks Code Warrior, Microsoft Windows CE Toolkits, and IBM VisualAge Micro Edition are all off-the-shelf products that can be used to develop mobile applications for DB2.

DB2 Everyplace comes with a complete and integrated development environment called the DB2 Everyplace Mobile Application Builder (MAB), which supports all of the functions and features that you find in the DB2 Everyplace engine. The MAB is a fourth-generation CASE tool that allows you to generate DB2 Everyplace applications without writing a single line of code! What's more, the code generated by the MAB is native compiled C code. Because this code is compiled, there is no run-time interpreter and therefore no runtime performance penalties – this is illustrated in Figure 5.

Just how easy is the MAB to use? Its environment allows you to visually create and edit forms by selecting controls from a palette. A project tree enables easy management of all objects in the application, and a single click generates and compiles the application. Complete sample applications developed in MAB are included with DB2 Everyplace. The MAB is a free download available from the DB2 Everyplace Web site at www.ibm.com/software/data/db2/everyplace.

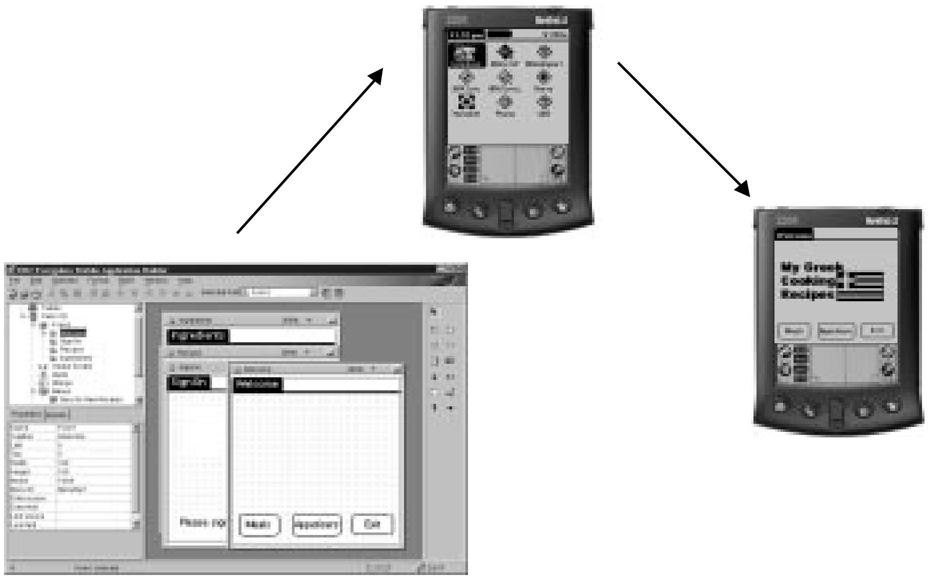


Figure 5: Mobile Application Builder

CONCLUSION

The explosive growth in the use of mobile devices is rapidly driving the market for more robust applications that emulate desktop functionality. Mobile access to enterprise data is required to meet this demand, and DB2 Everyplace delivers a comprehensive solution. The DB2 Everyplace engine provides query and update capabilities for relational data that has been downloaded from an enterprise database, thus extending the reach of mobile workers to enterprise resources. The SyncServer enables timely access to data through an intermittently connected client/server application model, and the Mobile Application Builder provides the tools to quickly generate DB2 Everyplace applications for mobile users. All of these components together make up the DB2 Everyplace offering, which you can use to implement powerful applications to access enterprise data from anywhere and at any time.

Paul C Zikopoulo
Database Specialist
IBM Global Sales Support team (Canada)

Roman B Melnyk
DB2 Information Development team
IBM (Canada)

DB2 stored procedures and dynamic cursors – part 2

This month we conclude the code for a working example of a DYNAMIC cursor.

```
           Ø5  WS-ORGNTR-CLNT-ID-S              PIC 9(Ø9).
           Ø5  FILLER                           PIC X(24)
               VALUE ") AND ORGN_DT  BETWEEN '".
           Ø5  WS-BEGIN-ORGN-DT-S               PIC X(1Ø).
           Ø5  FILLER                           PIC X(Ø7)
               VALUE "' AND '".
           Ø5  WS-END-ORGN-DT-S                 PIC X(1Ø).
           Ø5  FILLER                           PIC X(25)
               VALUE "' AND A.STAT_CD BETWEEN '".
           Ø5  WS-BEGIN-STAT-CD-S               PIC X(Ø2).
           Ø5  FILLER                           PIC X(Ø7)
               VALUE "' AND '".
           Ø5  WS-END-STAT-CD-S                 PIC X(Ø2).
           Ø5  FILLER                           PIC X(Ø1)
               VALUE "'".
           Ø5  WS-ORDER-BY-SECTION-S.
               1Ø  FILLER                       PIC X(1Ø)
                   VALUE ' ORDER BY '.
               1Ø  WS-ORDER-BY-COLUMN-S         PIC X(18).
               1Ø  FILLER                       PIC X(Ø1)
                   VALUE ' '.
               1Ø  WS-ORDER-ASC-DESC-S-1        PIC X(Ø4).
               1Ø  FILLER                       PIC X(18)
                   VALUE ' ,DTL_STAT_CD_DESC'.
               1Ø  FILLER                       PIC X(Ø1)
                   VALUE ' '.
               1Ø  WS-ORDER-ASC-DESC-S-2        PIC X(Ø4).
      *
       Ø1  WS-ORDER-BY.
           Ø5  FILLER                       PIC X(27)
               VALUE ' ORDER BY DTL_STAT_CD_DESC '.
           Ø5  WS-ORDER-ASC-DESC            PIC X(Ø4).
           Ø5  FILLER                       PIC X(25)  VALUE SPACES.
      *****************************************************************
      *             DB2ERRØ1 ERROR HANDLER COPY BOOK
      *****************************************************************
       Ø1  WS-DB2ERRØ1-LINKAGE.
           COPY DB2CPER1.

       Ø1 WS-DB2ERRØ1                       PIC X(Ø8)  VALUE 'DB2ERRØ1 '.
      *****************************************************************
```

```
*             S Q L C A        AREA
*****************************************************************
     EXEC SQL INCLUDE SQLCA END-EXEC.
*****************************************************************
*             DCLGEN FOR THE ABC REFERRAL TABLE
*****************************************************************
      EXEC SQL INCLUDE ABCCD1Ø2 END-EXEC.
*
      EXEC SQL INCLUDE ABCCD1Ø8 END-EXEC.
*
 Ø1 WS-RECEIVED-SENT-LOC     USAGE SQL TYPE IS
                             RESULT-SET-LOCATOR VARYING.
*
     EXEC SQL
       DECLARE RECEIVED_CSR CURSOR WITH RETURN FOR
           DYNAMIC-RECEIVED
     END-EXEC.
*
     EXEC SQL
       DECLARE SENT_CSR CURSOR WITH RETURN FOR
           DYNAMIC-SENT
     END-EXEC.
*****************************************************************
*             WORKING STORAGE ENDS HERE
*****************************************************************
 Ø1  FILLER                         PIC X(34)   VALUE
     'SPDYNACR WORKING STORAGE ENDS HERE'.
*****************************************************************
*             L I N K A G E    S E C T I O N
*****************************************************************
 LINKAGE SECTION.
*** DLL PASSED DATA
 Ø1  SP-CALL-TYPE                PIC  X(Ø1).
 Ø1  SP-CLNT-ID                  PIC S9(Ø9)  COMP.
 Ø1  SP-BEGIN-ORGN-DT            PIC  X(1Ø).
 Ø1  SP-END-ORGN-DT              PIC  X(1Ø).
 Ø1  SP-STAT-CD                  PIC  X(Ø2).
 Ø1  SP-SORT-BY-COLUMN           PIC  X(18).
 Ø1  SP-SORT-ASC-DESC            PIC  X(Ø4).
 Ø1  SP-ERROR-NUM                PIC  X(Ø2).
 Ø1  SP-ERROR-DESC               PIC  X(136).
*****************************************************************
*          P R O C E D U R E    D I V I S I O N
*****************************************************************
 PROCEDURE DIVISION USING SP-CALL-TYPE
                          SP-CLNT-ID
                          SP-BEGIN-ORGN-DT
                          SP-END-ORGN-DT
                          SP-STAT-CD
                          SP-SORT-BY-COLUMN
                          SP-SORT-ASC-DESC
                          SP-ERROR-NUM
                          SP-ERROR-DESC.
*****************************************************************
* MAINLINE
*  DETERMINE IF THE LIST TO BE RETURNED IS FOR THE RCV OR SENT LST
*****************************************************************
 ØØØØ-MAINLINE.
     MOVE 'ØØ' TO SP-ERROR-NUM
     INITIALIZE   SP-ERROR-DESC
*
     EVALUATE SP-CALL-TYPE
         WHEN 'R'
             PERFORM 1ØØØ-RECEIVED-LIST  THRU 1ØØØ-EXIT
         WHEN 'S'
             PERFORM 2ØØØ-SENT-LIST      THRU 2ØØØ-EXIT
         WHEN OTHER
             MOVE 'Ø8' TO SP-ERROR-NUM
             MOVE 'CALL TYPE NOT R OR S - SPDYNACR'
                    TO SP-ERROR-DESC
     END-EVALUATE
*
     PERFORM 99ØØ-GOBACK  THRU 99ØØ-EXIT.
 ØØØØ-EXIT.
     EXIT.
*****************************************************************
* PREPARE THE DYNAMIC SQL AND THEN OPEN THE CURSOR FOR THE RCV LST
*****************************************************************
 1ØØØ-RECEIVED-LIST.
     MOVE SP-CLNT-ID        TO WS-RECIP-CLNT-ID-R
     MOVE SP-BEGIN-ORGN-DT  TO WS-BEGIN-ORGN-DT-R
     MOVE SP-END-ORGN-DT    TO WS-END-ORGN-DT-R
     MOVE SP-SORT-BY-COLUMN TO WS-ORDER-BY-COLUMN-R
     MOVE SP-SORT-ASC-DESC  TO WS-ORDER-ASC-DESC-R-1
                               WS-ORDER-ASC-DESC-R-2
*** IF STATUS CODE SORT THEN REALLY SORT BY DETAIL STATUS
***  CODE DESCRIPTION ONLY
     IF SP-SORT-BY-COLUMN = 'STAT_CD'
         MOVE SP-SORT-ASC-DESC TO WS-ORDER-ASC-DESC
         MOVE WS-ORDER-BY      TO WS-ORDER-BY-SECTION-R
     END-IF
*** IF ALL STATUS CODES ARE REQUIRED PLUG THE BEGINNING
***  AND ENDING STATUS WITH LOW AND HIGH VALUES
     MOVE SP-STAT-CD     TO WS-BEGIN-STAT-CD-R
     IF SP-STAT-CD = '  '
         MOVE '99'       TO WS-END-STAT-CD-R
     ELSE
         MOVE SP-STAT-CD TO WS-END-STAT-CD-R
     END-IF
*
     MOVE WS-DYNAMIC-SQL-RECEIVED TO WS-DYNAMIC-SQL-TXT
     EXEC SQL PREPARE
         DYNAMIC-RECEIVED FROM :WS-DYNAMIC-SQL
     END-EXEC
     EVALUATE SQLCODE
         WHEN Ø
             CONTINUE
         WHEN OTHER
             MOVE 'Ø8'                   TO SP-ERROR-NUM
             MOVE '1ØØØ-RECEIVED-LIST' TO DB2Ø2Ø-LOCATION
             MOVE 'ERROR ON THE RECEIVED SQL PREPARE'
                                         TO DB2Ø2Ø-USER-DISPLAY-ARE
             PERFORM 9ØØØ-DB2-FATAL-ERROR  THRU 9ØØØ-EXIT
     END-EVALUATE
*
     PERFORM 7ØØ1-OPEN-RECEIVED-CSR  THRU 7ØØ1-EXIT
     EVALUATE SQLCODE
         WHEN Ø
             CONTINUE
         WHEN 1ØØ
             MOVE 'Ø4'                       TO SP-ERROR-NUM
             MOVE 'NO RECEIVED ROWS FOUND' TO SP-ERROR-DESC
         WHEN OTHER
             MOVE 'Ø8'                   TO SP-ERROR-NUM
             MOVE '1ØØØ-RECEIVED-LIST' TO DB2Ø2Ø-LOCATION
             MOVE 'ERROR ON OPEN - RECEIVED_CSR'
                                         TO DB2Ø2Ø-USER-DISPLAY-ARE
             PERFORM 9ØØØ-DB2-FATAL-ERROR  THRU 9ØØØ-EXIT
     END-EVALUATE.
 1ØØØ-EXIT.
     EXIT.
*****************************************************************
* PREPARE THE DYNAMIC SQL AND THEN OPEN THE CURSOR FOR THE
*  SENT LIST
*****************************************************************
 2ØØØ-SENT-LIST.
     MOVE SP-CLNT-ID        TO WS-REFRR-CLNT-ID-S
                               WS-ORGNTR-CLNT-ID-S
     MOVE SP-BEGIN-ORGN-DT  TO WS-BEGIN-ORGN-DT-S
     MOVE SP-END-ORGN-DT    TO WS-END-ORGN-DT-S
     MOVE SP-SORT-BY-COLUMN TO WS-ORDER-BY-COLUMN-S
     MOVE SP-SORT-ASC-DESC  TO WS-ORDER-ASC-DESC-S-1
                               WS-ORDER-ASC-DESC-S-2
*** IF STATUS CODE SORT THEN REALLY SORT BY DETAIL STATUS
***  CODE DESCRIPTION ONLY
     IF SP-SORT-BY-COLUMN = 'STAT_CD'
         MOVE SP-SORT-ASC-DESC TO WS-ORDER-ASC-DESC
         MOVE WS-ORDER-BY      TO WS-ORDER-BY-SECTION-S
     END-IF
*** IF ALL STATUS CODES ARE REQUIRED PLUG THE BEGINNING
***  AND ENDING STATUS WITH LOW AND HIGH VALUES
     MOVE SP-STAT-CD     TO WS-BEGIN-STAT-CD-S
     IF SP-STAT-CD = '  '
         MOVE '99'       TO WS-END-STAT-CD-S
     ELSE
         MOVE SP-STAT-CD TO WS-END-STAT-CD-S
     END-IF
*
     MOVE WS-DYNAMIC-SQL-SENT TO WS-DYNAMIC-SQL-TXT
     EXEC SQL PREPARE
         DYNAMIC-SENT FROM :WS-DYNAMIC-SQL
     END-EXEC
     EVALUATE SQLCODE
         WHEN Ø
             CONTINUE
         WHEN OTHER
             MOVE 'Ø8'              TO SP-ERROR-NUM
             MOVE '2ØØØ-SENT-LIST' TO DB2Ø2Ø-LOCATION
             MOVE 'ERROR ON THE SENT SQL PREPARE'
                                    TO DB2Ø2Ø-USER-DISPLAY-AREA
             PERFORM 9ØØØ-DB2-FATAL-ERROR  THRU 9ØØØ-EXIT
     END-EVALUATE
*
     PERFORM 7ØØ3-OPEN-SENT-CSR  THRU 7ØØ3-EXIT
     EVALUATE SQLCODE
         WHEN Ø
             CONTINUE
         WHEN 1ØØ
             MOVE 'Ø4'                  TO SP-ERROR-NUM
             MOVE 'NO SENT ROWS FOUND' TO SP-ERROR-DESC
         WHEN OTHER
             MOVE 'Ø8'              TO SP-ERROR-NUM
             MOVE '2ØØØ-SENT-LIST' TO DB2Ø2Ø-LOCATION
             MOVE 'ERROR ON OPEN - SENT_CSR'
                                    TO DB2Ø2Ø-USER-DISPLAY-AREA
             PERFORM 9ØØØ-DB2-FATAL-ERROR  THRU 9ØØØ-EXIT
     END-EVALUATE
 2ØØØ-EXIT.
     EXIT.
*****************************************************************
* OPEN THE RECEIVED LIST CURSOR
*****************************************************************
 7ØØ1-OPEN-RECEIVED-CSR.
     EXEC SQL
         OPEN RECEIVED_CSR
     END-EXEC.
 7ØØ1-EXIT.
     EXIT.
*****************************************************************
* OPEN THE SENT LIST CURSOR
*****************************************************************
 7ØØ3-OPEN-SENT-CSR.
```

```
           EXEC SQL
               OPEN SENT_CSR
           END-EXEC.
       7ØØ3-EXIT.
           EXIT.
      *****************************************************************
      * SET RETURN CODE AND ERROR MESSAGE FOR FATAL DB2 ERRORS.
      *  CALL WS-DB2ERRØ1 USING WS-DB2ERRØ1-LINKAGE
      *****************************************************************
       9ØØØ-DB2-FATAL-ERROR.
           MOVE SQLCODE        TO WS-SQLCODE
           MOVE 'SQLCODE = '   TO SP-ERROR-DESC
           MOVE WS-SQLCODE     TO SP-ERROR-DESC (11:5)
           MOVE 'REASON = '    TO SP-ERROR-DESC (18:9)
           MOVE SQLCA (19:7Ø) TO SP-ERROR-DESC (27:7Ø)
           MOVE 'SPDYNACR' TO DB2Ø2Ø-PROGRAM
           MOVE SQLCA      TO DB2Ø2Ø-SQLCA
           CALL WS-DB2ERRØ1 USING WS-DB2ERRØ1-LINKAGE
           PERFORM 99ØØ-GOBACK  THRU 99ØØ-EXIT.
       9ØØØ-EXIT.
           EXIT.
      *****************************************************************
      * GOBACK TO CALLING MODULE.
      *****************************************************************
       99ØØ-GOBACK.
           GOBACK.
       99ØØ-EXIT.
           EXIT.
```

Tim Albrecht
DB2 Database Administrator (USA)

Using check constraints to simulate domains

DB2 has provided table check constraints for a number of releases now (since Version 4), but many organizations have yet to capitalize on their functionality. Check constraints enable enhanced data integrity without requiring procedural logic (such as in stored procedures and triggers). Let's examine the basics of table check constraints.

A constraint is basically a restriction placed on the data values that can be stored in a column or columns of a table. Of course, most RDBMS products provide several different types of constraint, such as referential constraints (to define primary and foreign keys) and unique constraints (to prohibit duplicates).

Check constraints place specific data value restrictions on the contents of a column through the specification of a Boolean expression. The expression is explicitly defined in the table DDL and is formulated in much the same way that SQL WHERE clauses are formulated. Any attempt to modify the column data (ie during INSERT and UPDATE processing) will cause the expression to be evaluated. If the modification conforms to the Boolean expression, the modification is permitted to continue. If not, the statement will fail with a constraint violation.

This functionality is great for simulating the relational concept of a domain. A domain is basically the set of valid values that a column or data type can take on. Check constraints only simulate domains, though, because there are other features provided by domains that are not provided by check constraints. One such feature is that columns pooled from separate domains must not be compared or operated on by expressions that require the same type of data for both operands. For domains to be supported more fully the DBMS must support both check constraints and strong type-checking for user-defined and built-in data types. This prohibits users allowing ridiculous operations, such as comparing IQ to shoe size or adding Australian dollars to euros.

FORMING CHECK CONSTRAINTS

Check constraints are written using recognizable SQL syntax. This makes them easy to implement for anyone who has even a passing familiarity with SQL. The check constraint consists of two components – a constraint name and a check condition.

The constraint name is an SQL identifier and is used to reference or identify the constraint. The same constraint name cannot be specified more than once for the same table. If a constraint name is not explicitly coded, DB2 will create a unique name automatically for the constraint.

The check condition defines the actual constraint logic. The check condition can be defined using any of the basic predicates (>, <, =, <>,

<=, >=), as well as BETWEEN, IN, LIKE, and NULL. Furthermore, AND and OR can be used to string conditions together.

There are, however, restrictions on how check constraints are formulated. Some of these restrictions include:

- Other tables may not be accessed in the check condition – it can refer only to columns of the table on which it is defined.
- It cannot be defined on a LOB or ROWID column.
- It can be up to 3800 bytes long, not including redundant blanks.
- Only a limited subset of SQL operations are permitted. Check constraints cannot contain any subselects, built-in or user-defined functions, CAST functions (other than those created when the UDT was created), host variables, parameter markers, special registers, columns with a field procedure, CASE expressions, quantified predicates, EXISTS predicates, or the NOT logical operator. Furthermore, if a check-condition refers to a long string column, the reference must occur within a LIKE predicate.
- The first operand of the check constraint must be the name of a column contained in the table for which the constraint is defined.
- The other operand must be either another column name in the same table or a constant value.
- If the second operand is a constant, it must be compatible with the data type of the first operand. If the second operand is a column, it must be the same data type as the first column specified.

CHECK CONSTRAINT EXAMPLES

Check constraints enable the DBA or database designer to specify more robust data integrity rules directly into the database. Consider the following example:

```
CREATE TABLE EMP
      (EMPNO          INTEGER
      CONSTRAINT CHECK_EMPNO
      CHECK (EMPNO BETWEEN 100 and 25000),
      EMP_ADDRESS     VARCHAR(70),
      EMP_TYPE        CHAR(8)
CHECK (EMP_TYPE IN ('TEMP',   'FULLTIME', 'CONTRACT')),
EMP_DEPT         CHAR(3)       NOT NULL WITH DEFAULT,
SALARY           DECIMAL(7,2)  NOT NULL
CONSTRAINT CHECK_SALARY
CHECK (SALARY < 50000.00),
COMMISSION       DECIMAL(7,2),
BONUS            DECIMAL(7,2)
);
```

The CREATE statement for the EMP table contains three different check constraints:

1 The name of the first check constraint for the EMP table is CHECK_EMPNO. It is defined on the EMPNO column. The constraint ensures that the EMPNO column can contain values that range from 100 to 25000 (instead of the domain of all valid integers).

2 The second check constraint for this table is on the EMP_TYPE column. This is an example of an unnamed constraint. Though this is possible, it is not recommended. It is best to always provide an explicit constraint name in order to ease identification and administration. This specific constraint restricts the values that can be placed into EMP_TYPE as: ‘TEMP’, ‘FULLTIME’, and ‘CONTRACT’; no other values would be accepted.

3 The last check constraint on this table is named CHECK_SALARY. It effectively ensures that no employee can be entered with a salary of more than $50,000. (Now who would want to work there?)

COLUMN VERSUS TABLE-LEVEL CONSTRAINT

The first check constraint examples we have reviewed show a column-level check constraint. However, check constraints may also be coded at the table-level. A column-level check constraint is defined in the DDL immediately after the column. Appropriately enough, a table-level check constraint is defined after all the columns of the table have already been defined.

It is quite common for business rules to require access to multiple columns within a single table. When this situation occurs, it is wise to code the business rule into a check constraint at the table level, instead of at the column level. Of course, any column-level check constraint can also be defined at the table level as well. In terms of functionality, there is no difference between an integrity constraint defined at the table level and the same constraint defined at the column level. Let's augment our sample table DDL to add two table-level check constraints:

```
CREATE TABLE EMP
      (EMPNO           INTEGER
      CONSTRAINT CHECK_EMPNO
      CHECK (EMPNO BETWEEN 100 and 25000),
      EMP_ADDRESS      VARCHAR(70),
      EMP_TYPE         CHAR(8)
      CHECK (EMP_TYPE IN ('TEMP',   'FULLTIME', 'CONTRACT')),
      EMP_DEPT         CHAR(3)       NOT NULL WITH DEFAULT,
      SALARY           DECIMAL(7,2)  NOT NULL
      CONSTRAINT CHECK_SALARY
      CHECK (SALARY < 50000.00),
      COMMISSION       DECIMAL(7,2),
      BONUS            DECIMAL(7,2),
      CONSTRAINT COMM_VS_SALARY
      CHECK (SALARY > COMMISSION),
      CONSTRAINT COMM_BONUS
      CHECK (COMMISSION>0 OR BONUS > 0),
      );
```

The CREATE statement for the EMP table has been modified to contain two table-level check constraints having the following ramifications:

1 The name of the first table-level check constraint for the EMP table is COMM_VS_SALARY. This constraint will ensure that no employee can earn more commission than salary.

2 The second table-level check constraint is named COMM_BONUS. This constraint will ensure that every employee either earns a commission or a bonus (or, possibly, both).

CHECK CONSTRAINT BENEFITS

So what are the benefits of check constraints? The primary benefit is the ability to enforce business rules directly in each database without requiring additional application logic. Once defined, the business rule is physically implemented and cannot be bypassed. Check constraints also provide the following benefits:

- Because there is no additional programming required, DBAs can implement check constraints without involving the application programming staff. This effectively minimizes the amount of code that must be written by the programming staff. With the significant application backlog within most organizations, this can be the most crucial reason to utilize check constraints.
- Check constraints provide better data integrity. Because check constraints are always executed whenever the data in the column on which they are defined is to be modified, the business rule is not bypassed during *ad hoc* processing and dynamic SQL. When business rules are enforced using application programming logic instead, the rules can not be checked during *ad hoc* processes.
- Check constraints promote consistency. Because they are implemented once, in the table DDL, each constraint is always enforced. Constraints written in application logic, on the other hand, must be executed within each program that modifies any data to which the constraint applies. This can cause code duplication and inconsistent maintenance resulting in inaccurate business rule support.
- Typically check constraints coded in DDL will outperform the corresponding application code.

The overall impact of check constraints will be to increase application development productivity while at the same time promoting higher data integrity.

CHECK CONSTRAINTS, NULLS, AND DEFAULTS

An additional consideration for check constraints is the relational NULL. Any nullable column also defined with a check constraint can be set to null. When the column is set to null, the check constraint evaluates to unknown. Because null indicates the lack of a value, the presence of a null will not violate the check constraint.

Additionally, DB2 provides the ability to specify defaults for table columns – both system-defined defaults (pre-defined and automatically set by the DBMS) and user-defined defaults. When a row is inserted or loaded into the table and no value is specified for the column, the column will be set to the value that has been identified in the column default specification. For example, we could define a default for the EMP_TYPE column of our sample EMP table as follows:

```
EMP_TYPE      CHAR(8)      DEFAULT 'FULLTIME'
CHECK (EMP_TYPE IN ('TEMP',   'FULLTIME', 'CONTRACT')),
```

If a row is inserted without specifying an EMP_TYPE, the column will default to the value ‘FULLTIME’.

A problem can arise when using defaults with check constraints. Most DBMS products do not perform semantic checking on constraints and defaults. The DBMS, therefore, will allow the DBA to define defaults that contradict check constraints. Furthermore, it is possible to define check constraints that contradict one another. Care must be taken to avoid creating this type of problem.

Examples of contradictory constraints are depicted below:

```
CHECK (EMPNO > 10 AND EMPNO <9)
```

In this case, no value is both greater than 10 and less than 9, so nothing could ever be inserted.

```
EMP_TYPE      CHAR(8)       DEFAULT 'NEW'
CHECK (EMP_TYPE IN ('TEMP',   'FULLTIME', 'CONTRACT')),
```

In this case, the default value is not one of the permitted EMP_TYPE values according to the defined constraint. No defaults would ever be inserted.

```
CHECK (EMPNO > 10)
CHECK (EMPNO >= 11)
```

In this case, the constraints are redundant. No logical harm is done, but both constraints will be checked, thereby impacting the performance of applications that modify the table in which the constraints exist.

Other potential semantic problems could occur if the constraint contradicts a referential integrity DELETE or UPDATE rule, if two constraints are defined on the same column with contradictory conditions, or if the constraint requires that the column be NULL, but the column is defined as NOT NULL.

OTHER POTENTIAL HAZARDS

Take care when using the LOAD utility on a table with check constraints defined to it. By specifying the ENFORCE NO parameter you can permit DB2 to load data that does not conform to the check constraints (as well as the referential constraints). Although this eases the load process by enabling DB2 to bypass constraint checking, it will place the table space into a check pending state. You can run CHECK DATA to clear this state (or force the check pending off by using START with the FORCE option or the REPAIR utility). If you do not run CHECK DATA, constraint violations may occur, causing dirty data.

SUMMARY

Check constraints provide a very powerful vehicle for supporting business rules in the database. They can be used to simulate relational domains. Because check constraints are non-bypassable, they provide better data integrity than corresponding logic programmed into the application. It is a wise course of action to use check constraints in your database designs to support data integrity, domains, and business rules in all of your relational database applications.

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

Have you come across any undocumented features in DB2 Version 7? Why not share your discovery with others? Send your findings to the editor, Trevor Eddolls, at any of the addresses shown on page 2 or e-mail trevore@xephon.com.

DB2 stored procedures – a cookery book for COBOL shops

YOUR SITUATION

You are part of an OS/390 or z/OS-shop, which runs DB2 Version 4.1, 5.1, or 6.1 and, hopefully, soon Version 7.1.

You have a lot of COBOL skills to support your legacy systems.

You need to:

1 Fetch data from the OS/390-platform into some GUI/HTML-based applications on Windows NT or a Web platform.

2 Update data on OS/390 from a GUI/HTML-based application.

And you want these actions done by business logic in a secure and manageable fashion.

You are concerned about performance and data integrity, and you need to know where data is used, because *you* know that data changes over time, and this calls for deep knowledge about where the data is used.

You want to be sure that business data is managed in a business-like manner and thus you want data processed via static SQL.

Your ‘new’ employees do not ‘talk’ COBOL, your ‘old’ employees do not ‘talk’ Visual Basic/VBScript etc, and your customers want their legacy data presented in a Windows GUI or on the Web in HTML.

If a piece of business-logic has to be developed, you want only one copy of it, and all applications that need to execute it to be able to do so in an easy, simple, and straightforward manner.

One of the solutions to this situation is to use stored procedures.

WHAT ARE STORED PROCEDURES?

Stored Procedures (SP) is a technology that in this article covers the use of stored procedures, User Defined Functions (UDF), User

Defined Types (UDT), and triggers.

SPs and UDFs are programs that can be written in COBOL, C, Java, SQL Procedure Language (SPL), or REXX.

UDTs use SPs to verify data beyond normal domain-support.

The reason for connecting triggers to the technology is that triggers can call an SP in the trigger body text.

SPs execute in a Stored Procedure Address Space (SPAS), which can be either DB2- or WLM-managed. If you plan on using COBOL on OS/390, it has to be LE COBOL, since Language Environment/390 is a requirement.

The stored procedure technology is not a new technology. It has been part of both Oracle (PL/SQL) and SQL Server (Transact SQL) and a number of other database vendors’ product lines for ages.

SP in DB2 for OS/390 was first introduced in DB2 Version 4.

In DB2 for OS/390 Version 5, it became useful even if registration in the catalog table SYSPROCEDURES had to be done via DML (INSERT, UPDATE, and DELETE).

In DB2 UDB for OS/390 Version 6, the technology had matured significantly and new commands to update the system catalog (SYSIBM.SYSROUTINES and SYSIBM.SYSPARMS) were introduced.

The unique thing about DB2 SPs is that they can be built on static SQL, which means that the execution overhead is minimized in comparison with both Oracle and SQL Server.

In Oracle and SQL Server the ‘BIND’ process is done on a dynamic basis at least once per server start, which means that access path selection *can* change between server starts, depending on data statistics.

In DB2 the access path is determined at BIND time and will only change if objects are dropped.

DB2 keeps track of where objects are used in the SYSPACKDEP and SYSPLANDEP tables in the system catalog.

If you have to change table/column layouts you can find the SPs that use the tables/views by selecting from these tables.

Among the strengths of the technology is the environment independence. You can activate SPs from VB, ODBC, CICS, batch, etc, which means that you can activate precisely the same program logic from a variety of different platforms.

Since this article is about COBOL, I will concentrate on showing you how things can be done in COBOL.

STRATEGY

In our shop we have a database strategy in which DB2 for OS/390 is the platform where we want to keep all our business data; DB2 for OS/390 is the platform where we are focused on keeping developers educated to code and manage business applications.

We have chosen the Microsoft platform for a client/server environment, and WebSphere as the strategic platform for developing Web-based applications.

If we want to access data from a client/server platform, we have chosen to use SPs, since it gives us good integrity control and a nice tracking ability through the system catalog table (SYSIBM.SYSROUTINEAUTH).

Since all the SQL in our SPs is static SQL, we can track how data is accessed through the system catalog table SYSIBM.SYSPACKDEP.

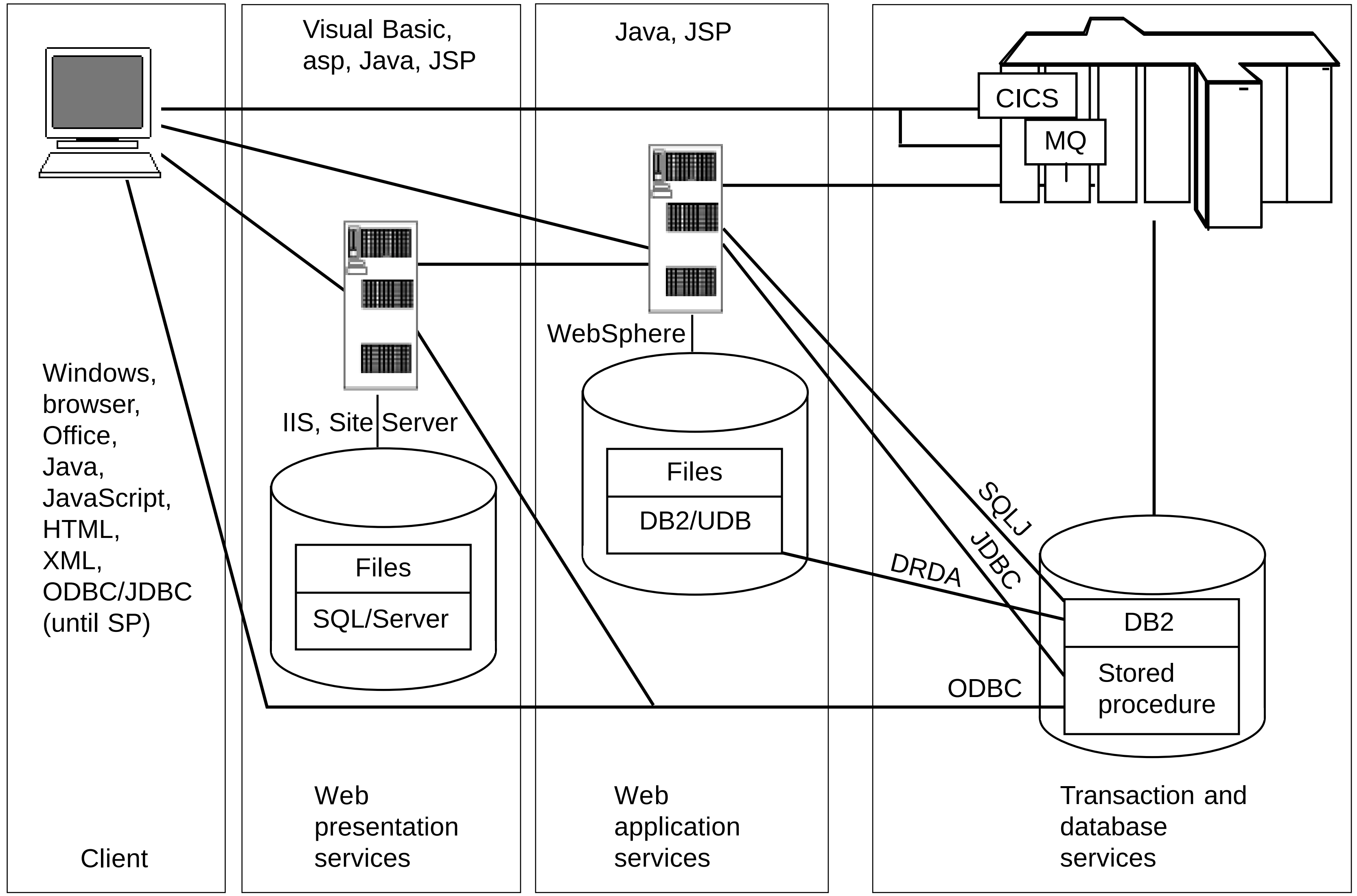
We want all update logic and most of the data retrieval logic to be static, since this gives us the most reliable and secure run-time environment.

Since our 200+ programmers are already very clever at building COBOL applications, we thought that using this area of expertise might be beneficial to our company.

We used to build applications in COBOL, which executed in MVS as batch programs or as CICS programs.

As you can see from Figure 1, the complexity in the application

Figure 1: System and data architecture


Visual Basic, asp, Java, JSP
Java, JSP
CICS
MQ
Windows, browser, Office, Java, JavaScript, HTML, XML, ODBC/JDBC (until SP)
IIS, Site Server
WebSphere
Files
SQL/Server
Files
DB2/UDB
SQLJ
JDBC
DRDA
DB2
Stored procedure
ODBC
Client
Web presentation services
Web application services
Transaction and database services

framework has increased, and the importance of knowing where data is accessed has got a whole new meaning. The benefits of having a real-time and always updated data dictionary (DB2's system catalog) has become even more obvious to us.

WHY SP?

SPs can be written by COBOL programmers, who are extremely knowledgeable in legacy systems, and since SPs can be activated in both Microsoft Office via VB and in HTML via VBScript, it seems to be a good technology to use when you are in the transition between the 'old' clean OS/390-world and the 'new' client/server and Web world because you want to develop the business logic only once.

SPs can return result sets, where normal MVS programs return a record at a time. Therefore you have to build logic to deal with the result sets.

The traditional benefit of the technology is the ability to reduce network traffic, where, instead of building a lot of requests and sending these to DB2 and receiving answers, you can do more logic per Unit Of Work (UOW) with less traffic on the network.

Having business logic close to data and having a tight coupling between the logic and data helps you keep integrity issues to a minimum.

To show how a given COBOL program can be transformed into an SP you can see the transformation steps in the sample application below.

Traditional COBOL programming logic:

….

```
EXEC SQL DECLARE CURSOR CU1 FOR
SELECT NAME, ADDRESS.
FROM MYTABLE
      WHERE NAME = :V1NAME
END-EXEC
EXEC SQL OPEN CU1
PERFORM UNTIL EOF
    EXEC SQL
```

```
            FETCH  CU1 INTO
                          :NAME,
                          :ADDRESS
    END-EXEC
    ….
END-PERFORM
EXEC SQL CLOSE CU1
…...
```

The stored procedure part of the above (JBTEST) in COBOL:

```
….
WORKING-STORAGE SECTION.
….
EXEC SQL DECLARE  CU1 CURSOR WITH  RETURN
FOR SELECT NAME, ADDRESS.
FROM MYTABLE
      WHERE NAME = :V1NAME
END-EXEC
….
LINKAGE SECTION.
Ø1 V1NAME PIC X(2Ø).
Ø1 V2RC PIC X(5).
…..
PROCEDURE DIVISION USING V1NAME, V2RC
……
EXEC  SQL
       OPEN CU1
END-EXEC
MOVE  ,OK'  TO V2RC
……
```

The WITH RETURN tells the stored procedure to build a result set that can be fetched in the calling program.

The COBOL program calling the stored procedure JBTEST:

```
….
EXEC SQL
          CALL JBTEST (:V1NAME, :V2RC)
END-EXEC.
IF SQLCODE = +466 THEN PERFORM
    EXEC SQL ASSOCIATE LOCATOR (:LOC)
              WITH PROCEDURE JBTEST
    END-EXEC
    EXEC SQL  ALLOCATE CU1 CURSOR
              FOR RESULT  SET :LOC
    END-EXEC
    IF V2RC = ,OK' THEN PERFORM UNTIL ROW-NOT-FOUND
```

```
         EXEC SQL FETCH CU1 INTO
                     :NAME
                     :ADDRESS
          END-EXEC
          …..
    END-PERFORM
    EXEC SQL
               CLOSE CU1
    END-EXEC...
END-PERFORM
```

The ASSOCIATE LOCATOR and ALLOCATE is conceptually the same as DECLARE CURSOR and OPEN.

After these two statements, the logic is the same as with traditional programming.

The reason for the test on SQLCODE +466 is that this return code tells the calling program that the SP has built one or more result set(s).

The same SP can be called from VB via ADO:

```
….
Set cmdJBTEST = New ADODB.Command
….
$ set connection to DB2
CmdJBTEST.CommandType = adCmdStoredProc
CmdJBTEST.CommandText = "JBTEST"
CmdJBTEST.Parameters.Append
CmdJBTEST.CreateParameter('V1NAME', adChar, adParamInput -
                                                   20, 'Svenn-Aage')
CmdJBTEST.Parameters.Append
CmdJBTEST.CreateParameter('V2RC', adChar, adParamOutput -
                                                   5, '   ')
Set rsReturnData = cmdJBTEST.Execute
$ loop to fetch resultset plus cleanup.
```

Remember when you write SPs to use data formats known to all the environments that can activate the SP. A good rule-of-thumb is to use display formats.

In VBScript you can use the same kind of ADO functionality as above.

BUILDING BLOCKS

To use a technology as stored procedures you have to build a technical infrastructure that makes it possible for developers to concentrate on building business logic.

We have chosen a set of tools for development, monitoring, and production use listed below.

Other vendors have solutions that cover the same kind of functionality and you can ask them what solutions they can give you.

The building blocks for your COBOL-SP environment must at least include the following points:

- Program translation procedures (JCL).
- SYSROUTINES/SYSPARMS management (SYS-PROCEDURES in V5).
- Run-time environment (WLM/DB2-SPAS and client/server requirements).
- Debugging tools.
- Monitoring tools.
- Abend/dump tools.
- Programming practices/standards.
- Security guidelines.
- Production turnover tools.

These are discussed below.

PROGRAM TRANSLATION SP

First of all you have to be running Language Environment and you have to use COBOL for OS/390 and VM Version 2.2 or higher.

When setting up the COBOL translation JCL procedures for SPs you have to include the following in the PARM list in the DB2 precompiler step to activate the RRSAF (Recoverable Resource manager Services Attachment Facility) module, DSNHLIR:

```
//DB2PREC  EXEC PGM=DSNHPC,
//  PARM=(,….ATTACH(RRSAF)')
```

In the linkage editor step you have to include the RRS stub in the SYSLIN DD input:

```
//LKED     EXEC  PGM=IEWL ….,
//SYSLIN   DD  *
  ….
         INCLUDE SYSLIB(DSNRLI)
        ….
```

Now you are ready to define your programs to DB2.

In DB2 Version 5.1 this is done in the system catalog table SYSIBM. SYSPROCEDURES via INSERT/UPDATE/DELETE statements.

In DB2 Version 6.1 the two system catalog tables SYSIBM.SYSROUTINES and SYSIBM.SYSPARMS are populated via CREATE/ALTER PROCEDURE, as shown below.

SYSROUTINES AND SYSPARMS

In DB2 Version 6.1 the maintenance of Stored Procedure information in the system catalog is done via the commands CREATE PROCEDURE, ALTER PROCEDURE, and DROP PROCEDURE.

The activation is done via a START/STOP PROCEDURE.

To display information you can use the DISPLAY PROCEDURE command.

We have chosen standard settings for CREATE PROCEDURE, and for the SP JBTEST it will look like this:

```
CREATE PROCEDURE SYSPROC.JBTEST
      (
        IN       V1NAME       CHAR(2Ø),
        OUT      V2RC         CHAR(5)
      )
RESULT SET 1
EXTERNAL NAME JBTEST
LANGUAGE COBOL
NOT DETERMINISTIC
FENCED
MODIFIES SQL
COLLID STPROC                - You can choose your own
WLM ENVIRONMENT (yourwlm, *)
ASUTIME 3ØØØØØ               - To stop looping SPs.
```

```
STAY RESIDENT YES            - Important for performance
PROGRAM TYPE MAIN
SECURITY DB2
COMMIT ON RETURN NO          - Important for consistency
```

ALTER PROCEDURE can change almost anything but the parameter list, where you have to DROP and CREATE again.

Sample ALTER:

```
ALTER PROCEDURE SYSPROC.JBTEST
      RESULT SET 1
      EXTERNAL NAME JBTEST
      LANGUAGE COBOL
      NOT DETERMINISTIC
      MODIFIES SQL DATA
      NO DBINFO
      COLLID STPROC
      WLM ENVIRONMENT (yourwlm, *)
      ASUTIME LIMIT 300000
      STAY RESIDENT YES
      PROGRAM TYPE MAIN
      SECURITY DB2
      COMMIT ON RETURN NO
```

To address SPs in a data sharing environment, IBM has added the SCOPE() parameter to the START/STOP/DISPLAY syntax. This feature was implemented via APAR PQ29031 and PQ35094.

SCOPE(GROUP) used in the DISPLAY command shows where the procedure is used in all data sharing members for the given data sharing group. With STOP/START, it is used to ensure that ALL data sharing members are addressed.

When you make changes to your SP settings remember to STOP your procedure before changing it and start it again afterwards:

- Step1:

  ```
  -STOP PROCEDURE (SYSPROC.JBTEST) SCOPE(GROUP)
  ```

- Step2:

  ```
  ALTER …
  ```

- Step3:

  ```
  -START PROCEDURE (SYSPROC.JBTEST) SCOPE(GROUP)
  ```

To see where JBTEST is active, issue the DISPLAY command:

```
     -DISPLAY PROCEDURE (SYSPROC.JBTEST) SCOPE(GROUP)
```

Result:

```
DSNX94ØI .DB2A DSNX9DIS DISPLAY PROCEDURE REPORT FOLLOWS -
——— SCHEMA=SYSPROC
DSNX9DIS PROCEDURE JBTEST HAS NOT BEEN ACCESSED OR IS NOT DEFINED
DSNX9DIS DISPLAY PROCEDURE REPORT COMPLETE
DSN9Ø35I .DB2A BEGIN OF DISPLAY FOR MEMBER: DB2A
----------------------------------------------------------------
——— SCHEMA=SYSPROC
PROCEDURE           STATUS ACTIVE QUEUED MAXQUE TIMEOUT WLM_ENV
JBTEST              STARTED  Ø      Ø       Ø      Ø    yourwlm
DSNX9DIS DISPLAY PROCEDURE REPORT COMPLETE
————END OF DISPLAY FOR MEMBER: DB2B    ————
DSN9Ø22I .DB2A DSNX9COM ,-DISPLAY PROC' NORMAL COMPLETION
```

As you can see from the above display output, the result comes from two different data sharing members.

For more information about the commands you must consult the DB2 manuals.

RUN-TIME

You can choose to make your Stored Procedure Address Spaces (SPAS) DB2 or WLM-managed (Work Load Manager).

There are several differences in these set-ups:

- The DB2-managed SPAS is set up via the DSNZPARM field STORPROC and the actual procedure running the program is DSNX9STP.
- The WLM-managed SPAS is set up via WLM policies (see your MVS system programmer about this) and is referenced via the above CREATE/ALTER PROCEDURE, where you can have a multitude of different SPASs.

It is recommended that you use WLM-managed address spaces.

WLM is the environment that we see most of the new development stuff from IBM addressing. So it seems that DB2-managed SPAS is only for test use.

DB2 SPAS cannot scale, whereas WLM-managed SPAS can scale with many active address spaces (started tasks), and you can easily create WLM address spaces for different purposes and with different access characteristics.

Some of the ISV-products (eg Compuware XPEDITER stored procedure feature) require use of WLM-managed SPAS, and I have not come across any that require DB2-managed SPAS.

If you plan on using UDF (User Defined Functions) you have to use WLM.

If SPASs are working as started tasks and if your WLM policy is set up to have a limited number of SPs active at any given time, WLM will start new ones when you fill them with running SPs.

When the activity-level drops below given points, WLM will stop the WLM-managed SPASs when they have been idle for a specified period. This gives you a powerful and dynamic environment.

When you want to run your DB2 SPs from Windows, you should install the DB2 Client Application Enabler on your workstation and either install DB2 Connect Personal Edition on your workstation or install DB2 Connect Enterprise Edition on a server in your environment.

You have to catalog your hostdb connnection via CATALOG statements (consult your DB2 UDB for NT documentation for the syntax).

DEBUGGING TOOL

Compuware’s XPEDITER/TSO has a feature to debug COBOL programs and this feature requires that SPs run in a WLM-managed Stored Procedure Address Space.

It works with the actual load module active in the WLM-managed DB2 SPAS and the debugging is done via interception of the program code from your active TSO session.

Seen from a programmer’s point of view, the debugging is done via ISPF, just as from a normal COBOL program.

MONITOR TOOL

We use Landmark’s TMON/DB2 to monitor DB2.

TMON/DB2 has a powerful SQL Capture feature, which can present the content of a given host variables and can trap given SQL codes. This is very useful when you run your SPs from different environments and want to know what the content is of the input and output host variables and SQL codes.

But be careful. If you do not build a reasonable filter for the trace of these SQL codes and host varable presentations, TMON/DB2 will generate a lot of output and consume a considerable amount of ressources too.

ABEND/DUMP

Compuware's Abendaid and Viewing Facility are used for formatting and viewing purposes.

During compilation of COBOL programs, Xpediter forms symbolic output files (the mapping between load modules and program source code). AbendAid and Viewing facility uses the symbolic output files to map the load module and source code. The programmer will look at the COBOL source when he sees the dump, instead of using PSWs to find the errors when the program dumps.

SP dumps are located in the output of the WLM address space that executed the SP.

The jobname for a stored procedure is found via the OS/390 command:

```
     /D WLM,APPLENV=yourwlm
```

where *yourwlm* is the content of the WLM_ENVIRONMENT column in SYSIBM.SYSROUTINES.

We strongly recommend that you use the name of the internal WLM as your actual started task name, since this gives you the option of having to look only in the SYSIBM.SYSROUTINES column WLM_ENVIRONMENT.

If you plan to run DB2-managed SPAS, look in the DSNZPARM field STORPROC.

Programming:

- Do not program an abend in an SP.
- Always use external formats in in/out parameters because you will never know when you have a new ‘customer’ who does not understand DECIMAL or other fields.
- Send a return code to the caller if something is wrong, and let the caller decide on abnormal termination issues.
- If you plan on using standardized alert routines via focal point products you should use an alert mechanism that is not rolled back. We plan on using MQ for this purpose.
- Always test for SQL codes in both the SP and the calling program.

SECURITY

The best solution to handle stored procedure security is to use RACF, since by doing so you separate the security parts from the DB2 parts.

When you DROP an SP you drop the SYSROUTINEAUTH entries as well, which is the normal cascading practice of DB2.

If you use normal DB2 security you have to build the following:

```
GRANT CREATEIN, ALTERIN, DROPIN ON schema xx to yy
```

where *xx* is the schema name (ie SYSPROC) and *yy* is the GRANTEE (ie PUBLIC).

```
GRANT EXECUTE ON PROCEDURE xx to yy
```

where *xx* is the name of the stored procedure (ie MYPROC) and *yy* is the GRANTEE (ie PUBLIC).

Always use a generic term as GRANTEE, since the cascading issues might be a nightmare.

We recommend that you use schema names other than SYSPROC for your home-grown stored procedures, since this will give you better management options.

PRODUCTION TURNOVER

When an SP is moved from development to production you should use the same procedure that you normally use and build the handling of SYSROUTINES /SYSPARMS into this routine. A schematic solution might look like this for an existing procedure *xx*:

- Copy old production load module, DBRM, and Bind statement to a fallback library.
- Copy the old create procedure statement to the fallback library.
- STOP PROCEDURE *xx* (the procedure).
- Copy new load module, DBRM, and Bind statement to production.
- Execute the Bind-statement.
- Run the DROP/CREATE PROCEDURE statement.
- Run any required GRANT statements.
- START PROCEDURE *xx*.

REFERENCES

IBM has a Web page for stored procedures at http://www-4.ibm.com/software/data/db2/os390/spb/exciting/.

A redbook giving a lot of information about the set-up is SG24-5759-00, *DB2 UDB for OS/390 Version 6 Management Tool Package*.

CONCLUSION

The SP technology is a fine technology to build a bridge between platforms on the programming and reusability side. It is close to the data that it actually handles, and there is a good cross-reference between business-logic and data dictionary (if you call the system catalog that).

If you plan on using stored procedures developed in Java, DB2 UDB on Windows NT has a stored procedure builder, which makes it easy to build SPs via drag-and-drop technology.

I wish you good luck with your implementation of the technology.

Svenn-Aage Sønderskov (sas@jyskebank.dk)
Jyske Bank (Denmark)

A formatting routine for DSNTIAUL unload files

If you migrate from one DB2 system to another, ie from DB2 for OS/2 to DB2 for NT, you have to compare the data on the two systems. This is often done by comparing the unload files, which is easy to do if the columns are in character or variable character format. Unfortunately, in most cases you have numeric data in integer, small integer, or floating point format. To translate these formats to a displayable form is a cumbersome job. So I wrote a little procedure to interpret the load statement, which is written together with the unloaded data to the SYSPUNCH file. The load statement includes the column names, the column format, and the column length, so that all necessary information is available to split the unload data into the right portions – see the load statement produced by unloading SYSIBM.SYSTABLESPACES with DSNTIAUL below:

```
LOAD DATA LOG NO INDDN SYSRECØØ INTO TABLE
    SYSIBM.SYSTABLES
 (
 NAME         POSITION(       1        )  VARCHAR                     ,
 CREATOR      POSITION(      21        )  CHAR(                   8)  ,
 TYPE         POSITION(      29        )  CHAR(                   1)  ,
 DBNAME       POSITION(      3Ø        )  CHAR(                   8)  ,
 TSNAME       POSITION(      38        )  CHAR(                   8)  ,
 DBID         POSITION(      46        )  SMALLINT                    ,
 OBID         POSITION(      48        )  SMALLINT                    ,
 COLCOUNT     POSITION(      5Ø        )  SMALLINT                    ,
 EDPROC       POSITION(      52        )  CHAR(                   8)  ,
 VALPROC      POSITION(      6Ø        )  CHAR(                   8)  ,
 CLUSTERTYPE  POSITION(      68        )  CHAR(                   1)  ,
 CLUSTERRID   POSITION(      69        )  INTEGER                     ,
 CARD         POSITION(      73        )
 NPAGES       POSITION(      77        )  INTEGER                     ,
 PCTPAGES     POSITION(      81        )  SMALLINT                    ,
 IBMREQD      POSITION(      83        )  CHAR(                   1)  ,
 REMARKS      POSITION(      84        )  VARCHAR                     ,
 PARENTS      POSITION(     34Ø        )  SMALLINT                    ,
 CHILDREN     POSITION(     342        )  SMALLINT                    ,
 KEYCOLUMNS   POSITION(     344        )  SMALLINT                    ,
 RECLENGTH    POSITION(     346        )  SMALLINT                    ,
 STATUS       POSITION(     348        )  CHAR(                   1)  ,
 KEYOBID      POSITION(     349        )  SMALLINT                    ,
 LABEL        POSITION(     351        )  VARCHAR                     ,
 CHECKFLAG    POSITION(     383        )  CHAR(                   1)  ,
   CHECKRID     POSITION(     384        )  CHAR(                  4) ,
   AUDITING     POSITION(     388        )  CHAR(                  1) ,
   CREATEDBY    POSITION(     389        )  CHAR(                  8) ,
   LOCATION     POSITION(     397        )  CHAR(                 16) ,
   TBCREATOR    POSITION(     413        )  CHAR(                  8) ,
   TBNAME       POSITION(     421        )  VARCHAR                   ,
   CREATEDTS    POSITION(     441        ) TIMESTAMP EXTERNAL(    26) ,
   ALTEREDTS    POSITION(     467        ) TIMESTAMP EXTERNAL(    26) ,
   DATACAPTURE  POSITION(     493        )  CHAR(                  1) ,
   RBA1         POSITION(     494        )  CHAR(                  6) ,
   RBA2         POSITION(     5ØØ        )  CHAR(                  6) ,
   PCTROWCOMP   POSITION(     5Ø6        )  SMALLINT                  ,
   STATSTIME    POSITION(     5Ø8        )  TIMESTAMP EXTERNAL(   26) ,
   CHECKS       POSITION(     534        )  SMALLINT                  ,
   CARDF        POSITION(     536        )  FLOAT(                 53) ,
   CHECKRID5B   POSITION(     544        )  CHAR(                  5) ,
   ENCODING_SCHEME  POSITION(     549    )  CHAR(                  1)
   )
```

The formatted data is written in a file, one line for each column in the form 'column name = data' (see below), and can be examined by the ISPF editor. The formatted output from LISTUNLD, displaying the unload file from record 1 to record 3 of SYSIBM.SYSTABLESPACES, is shown below:

```
—U8Ø2259.DBT2.UNLOADØØ.DATAF— Record 1

NAME                   = SYSCOPY
CREATOR                = SYSIBM
TYPE                   = T
DBNAME                 = DSNDBØ6
TSNAME                 = SYSCOPY
DBID                   = 6
OBID                   = 46
COLCOUNT               = 19
EDPROC                 =
VALPROC                =
CLUSTERTYPE            =
CLUSTERRID             = Ø
CARD                   = 718Ø91
NPAGES                 = 23937
PCTPAGES               = 98
IBMREQD                = Y
REMARKS                =
PARENTS                = Ø
CHILDREN               = Ø
KEYCOLUMNS             = Ø
RECLENGTH              = 1919
STATUS                 =
KEYOBID               = Ø
LABEL                 =
CHECKFLAG             =
CHECKRID              =
AUDITING              =
CREATEDBY             = SYSIBM
LOCATION              =
TBCREATOR             =
TBNAME                =
CREATEDTS             = 1985-Ø4-Ø1-ØØ.ØØ.ØØ.ØØØØØØ
ALTEREDTS             = 1998-Ø3-14-18.3Ø.Ø5.441262
DATACAPTURE           =
RBA1                  =
RBA2                  =
PCTROWCOMP            = Ø
STATSTIME             = 2ØØ1-Ø8-24-22.Ø4.4Ø.966181
CHECKS                = Ø
CARDF                 = 718Ø91   (AF5ØB, E 4)hex
CHECKRID5B            =
ENCODING_SCHEME       = E

—U8Ø2259.DBT2.UNLOADØØ.DATAF— Record 2

NAME                  = SYSFIELDS
CREATOR               = SYSIBM
TYPE                  = T
DBNAME                = DSNDBØ6
TSNAME                = SYSDBASE
DBID                  = 6
OBID                  = 21
COLCOUNT              = 13
EDPROC                =
VALPROC               =
CLUSTERTYPE           =
CLUSTERRID            = Ø
CARD                  = Ø
NPAGES                = Ø
PCTPAGES              = Ø
IBMREQD               = Y
REMARKS               =
PARENTS               = 1
CHILDREN              = Ø
KEYCOLUMNS            = Ø
RECLENGTH             = 1879
STATUS                =
KEYOBID               = Ø
LABEL                 =
CHECKFLAG             =
CHECKRID              =
AUDITING              =
CREATEDBY             = SYSIBM
```

```
LOCATION              =
TBCREATOR             =
TBNAME                =
CREATEDTS             = 1985-Ø4-Ø1-ØØ.ØØ.ØØ.ØØØØØØ
ALTEREDTS             = 1985-Ø4-Ø1-ØØ.ØØ.ØØ.ØØØØØØ
DATACAPTURE           =
RBA1                  =
RBA2                  =
PCTROWCOMP            = Ø
STATSTIME             = 2ØØ1-Ø8-24-22.Ø4.4Ø.966181
CHECKS                = Ø
CARDF                 = Ø   (Ø E -65)hex
CHECKRID5B            =
ENCODING_SCHEME       = E

—U8Ø2259.DBT2.UNLOADØØ.DATAF— Record 3

NAME                  = SYSTABLESPACE
CREATOR               = SYSIBM
TYPE                  = T
DBNAME                = DSNDBØ6
TSNAME                = SYSDBASE
DBID                  = 6
OBID                  = 17
COLCOUNT              = 34
EDPROC                =
VALPROC               =
CLUSTERTYPE           =
CLUSTERRID            = Ø
CARD                  = 811
NPAGES                = 811
PCTPAGES              = 16
IBMREQD               = Y
REMARKS               =
PARENTS               = Ø
CHILDREN              = 2
KEYCOLUMNS            = 2
RECLENGTH             = 171
STATUS                =
KEYOBID               = Ø
LABEL                 =
CHECKFLAG             =
CHECKRID              =
AUDITING              =
CREATEDBY             = SYSIBM
LOCATION              =
TBCREATOR             =
TBNAME                =
CREATEDTS             = 1985-Ø4-Ø1-ØØ.ØØ.ØØ.ØØØØØØ
ALTEREDTS             = 2ØØØ-Ø5-2Ø-12.1Ø.33.149676
DATACAPTURE           =
RBA1                 =
RBA2                 =
PCTROWCOMP           = Ø
STATSTIME            = 2ØØ1-Ø8-24-22.Ø4.4Ø.966181
CHECKS               = 1
CARDF                = 811   (32B, E 2)hex
CHECKRID5B           =
ENCODING_SCHEME      = E
```

The procedure is given the file name of the unload file. To determine the name of the SYSPUNCH file it is assumed that the SYSPUNCH file has the same name as the UNLOAD file, but has added the qualifier PUNCH as the last but one qualifier, eg:

```
UNLOAD:      SYS1.UNLOAD.DATA.BESTF
SYSPUNCH:    SYS1.UNLOAD.DATA.PUNCH.BESTF
```

The procedure is designed to format a small range of data in the foreground of TSO, but it can be used in batch for larger amounts of data too.

The procedure is called in the following form:

```
LISTUNLD filename startrec recno ( options
```

where:

- filename is the dataset name of the unload file.
- startrec is the first record that should be displayed.
- recno is the number of records to display.

The options must be separated by a '(' from the parameters. To debug, turn on the REXX trace. Use the FILE option to keep the OUTPUT file after exiting the procedure.

LISTUNLD

```
/* REXX */
/*****************************************************************/
/* LISTUNLD is used to show the output (SYSREC) from DB2 DSNTIAUL */
/* program in a readable form. The rule, how to interpret the     */
/* output record, is taken from the corresponding load statement   */
/* produced from DSNTIAUL via SYSPUNCH                            */
/*                                                                */
/* Format of the procedure call :                                 */
/*                                                                */
/* LISTUNLD filename strec recno                                   */
/*                                                                 */
/* filename : name of the unload file                              */
/* strec    : record to start formatting   default( 1 )            */
/* recno    : number of records to format  default( 999999 )       */
/*                                                                 */
/* The filename of the punchfile which include the load statement  */
/* is assumed to have the same name but has included 'PUNCH'       */
/* as the last but one qualifier                                   */
/*******************************************************************/
trace o
 arg unlout recnr recanz '(' options
/*———Start default values ————————————————*/
debug = 'N'
file  = 'N'
recmax = 999999
recstr = 1
/*———End   default values ————————————————*/
 do while options ¬= ''
   parse upper var options opt options
   select
     when left(opt,4) = 'FILE'
       then file = 'Y'
     when left(opt,5) = 'DEBUG'
       then debug = 'Y'
     otherwise do
       say opt '—Option in error'
       say ' '
       unlout = '?'
       end
   end
 end
 if unlout = ''  then unlout = '?'
 if unlout = '?'
    then do
         say '*******************************************************'
         say '* Procedure LISTUNLD                                  *'
         say '*                                                     *'
         say '* Syntax : LISTUNLD filename strec recno ( otions     *'
         say '*                                                     *'
         say '* Parameters:                                         *'
         say '*                                                     *'
         say '* filename : Name of the unload dataset               *'
         say '*            if filename = ? this help is displayed   *'
         say '* strec    : Start display with that record           *'
         say '* recno    : number of records to display             *'
         say '*                                                     *'
         say '* Options  (options must be seperated by blanks)      *'
         say '*                                                     *'
         say '* DEBUG    : Trace the procedure                      *'
         say "* FILE     : Keep the temporary file when exiting the *"
```

```
        say '*           procedure                                 *'
        say '*                                                     *'
        say '*           Option FILE is the default when running   *'
        say '*           in batchmode                              *'
        say '*******************************************************'
        exit 99
        end
if debug  = 'Y' then trace r
if recnr  = ''  then recnr = recstr
if recanz = ''  then recanz = recmax
unlout = strip(unlout,'B',"'")
unlout = strip(unlout,'B','"')
st_llq = lastpos('.',unlout) + 1
llq = substr(unlout,st_llq)
interpret "parse var unlout q1 '."llq"' q2"
pchout = q1'.PUNCH.'llq
if SYSDSN("'"pchout"'") ¬= 'OK'
   then do
        say 'The file of the load statement could not be found'
        say "Its name should be "pchout
        exit 99
        end
qual = 'ID'right(random(99999),5,'Ø')
tempname = "'"userid()".LISTUNLD."qual".DATAV'"
"ALLOC F(OUT) DS("tempname") NEW"
call read_sql
"EXECIO" queued() "DISKW OUT ( FINIS"
"FREE F(OUT)"
sys = sysvar(sysenv)                    /* FORE or  BACK  */
if sys = "BACK"
   then do
        /* Procedure is running in batchmode */
        say 'File' tempname 'was allocated and written'
        end
   else do
        /* Procedure is running in foreground */
        'ISPEXEC EDIT DATASET('tempname')'
        if file = 'N'
           then do
                xx = outtrap(xx.)
                "DEL "tempname
                xx = outtrap(off)
                end
        end
exit
```

Editor's note: the code will be concluded in the next issue.

Roman Hawlitschek
System Programmer (Germany)

DB2 news

IBM has announced Version 4.0 of its WebSphere Site Analyzer for Web site visitor statistics and analysis. It has a new Web Tracker for near real-time data capture and reporting, support for Windows NT/2000, Linux, AIX, and Solaris, and Web log analysis.

It's shipped with DB2 and supports Oracle, provides shopping cart analysis for WebSphere Commerce Suite users, and provides proxy traffic analysis for WebSphere Edge Server users, as well as campaign analysis and rule effectiveness.

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

* * *

EMC has extended its ControlCenter Database Tuner performance management software, adding support for CLARiiON storage systems in Oracle database environments and DB2 UDB support for Symmetrix boxes.

The performance management tool helps monitor and configure, control, and fine-tune databases, host operating systems, and Symmetrix or CLARiiON systems.

It collects, correlates, and presents database mapping and performance data, which can be customized for DBAs, system administrators, and storage managers, helping to optimize Symmetrix and CLARiiON performance across database applications.

It creates a dynamic view of enterprise information assets that can be shared locally or over the Internet.

For further information contact:
EMC, 35 Parkwood Drive, Hopkinton, MA 01748, USA.
Tel: (508) 435 1000.
URL: http://www.emc.com/products/storage_management/controlcenter.jsp.

* * *

Veritas and IBM have teamed to ship two new storage virtualization products for DB2 UDB: Veritas Database Edition for DB2 and Veritas Database Edition/HA for DB2, management tools that help optimize the performance of database applications and simplify the administration of the underlying DB2 storage and servers. The high availability version adds Veritas software clustering software.

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

* * *

IBM has upgraded its Content Manager, based on DB2, with improved capabilities including faster search and load capabilities, better report distribution features, and broad access to all forms of content. The federated approach lets users store, access, and analyse critical information in any format, regardless of where it resides.

Specifically, improved access to both structured and unstructured information now includes Microsoft Exchange and FileNet documents.

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