104 DB2 update

June 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 ($33.50) 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/db2update.html; you will need to supply a word from the printed issue.

Editor

Trevor Eddolls

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.

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 contact us at any of the addresses above and we will send you a copy of our *Notes for Contributors*, or you can download a copy from www.xephon.com/contnote.html.

Business Intelligence with DB2 Universal Database Version 7.1

In *DB2 Update* Issue 100, February 2001, I wrote an article entitled *E-business enhancements with DB2 Universal Database Version 7.1*. In that article I wrote about some of the major enhancements that were added to the most recent version of DB2 to enable businesses to conduct electronic commerce more easily, faster, and more reliably. In that article, DB2 was positioned in the *Build* and *Run* quadrants of the Cartesian plane in Figure 1. DB2 had some enhancements, such as identity columns and declared temporary tables, that helped application developers build or migrate database applications. DB2 was also more strongly positioned with Version 7.1 in the *Run* quadrant with enhancements such as the XML Extender and increased log limit size helping companies run more scalable business environments.

Although there was a lot of focus on e-business enhancements in DB2 Version 7.1, there were some powerful enhancements for Business Intelligence (BI) as well. The BI enhancements strongly position DB2 (and its associated products) in the *Leverage* quadrant. The *Leverage* quadrant is the final stage of the e-business cycle. It takes the information and knowledge that is stored and managed by the database

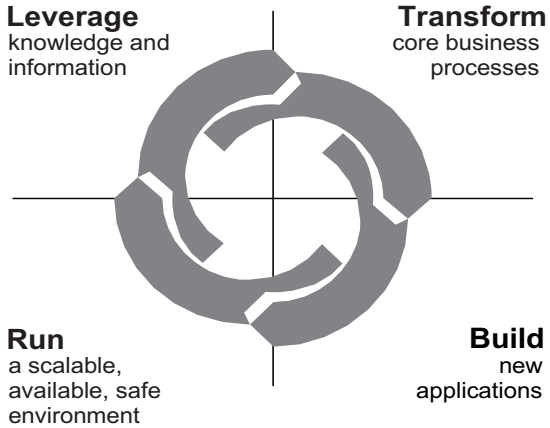

Figure 1: E-business model

(in this case DB2) and leverages that information to make better and more timely business decisions.

The driving force behind BI solutions is to allow information that is managed a chance to help a business 'think outside the box', so to speak. Why should you care about the BI enhancements that were introduced in DB2 Version 7.1? Quite frankly, your competition does. If you want to play in today's game, you have to practise BI!

It's your competition that is using Customer Relationship Management (CRM) software like Siebel to find ways to address customer vulnerability – they are trying to steal *your* customers! The reason the beer was moved to the other side of the store in your local grocery store, where the diapers are, is because a BI application told them to do it. Why is the stinky cheese beside those expensive pre-washed vegetables in the produce aisle? Because someone, somewhere, figured out that you'd buy them together. How are companies able to make these decisions? They analyse data that they collect every day with BI tools. Data is collected through customer loyalty programs like Air Miles and through point-of-purchase analysis. Since much of this data is managed by DB2, it would be a natural extension for DB2 to provide applications and functionality to leverage that information.

This article will discuss the major BI enhancements in DB2 UDB Version 7.1 that let businesses leverage the data that they collect to make better, faster, more informed decisions.

FEDERATED DATABASE SUPPORT

One enhancement to DB2 Version 7.1 is the bundled Federated Database Support component that is available with any DB2 server product. Federated Database Support allows customers to join data (read) from disparate DB2-family (distributed and mainframe) databases in a single transaction.

Customers' environments are rarely homogeneous and the requirement for managing and manipulating 'federations' of data is becoming more and more critical. An additional DB2 product, called DB2 Relational Connect, is required for read-only access to non-DB2 data sources. It's possible using SQL statements to join rows from a table

on a DB2 for S/390 system and an Oracle-based system, using the DB2 SQL API. Results are returned to the client application that issued the SQL statement despite the data residing on two different RDBMSs.

The federated component in DB2 Universal Database Version 7.1 gives users access to many forms of data, including relational data from other vendors (using DB2 Relational Connect) and non-relational data sources from a single DB2 API.

A DB2 federated database system enables users and applications to reference multiple database management systems within a single SQL statement. For example, with DB2's Federated Database Support, you can join data that is located on a DB2 for AIX server table with a table that resides on a DB2 for OS/390 system. Statements of this type are called distributed requests.

To read and write data in other relational databases, there is a product called DB2 DataJoiner. DB2 DataJoiner is built upon existing DB2 technology. It gives the user write and read access to Oracle, Sybase, Informix, SQL Server, and other relational databases, as if they were DB2 data sources. There is a strategic direction to move the read and write DataJoiner functionality for the DB2 family into the core DB2 engine. The first stage of this process was to provide read access between any members of the DB2 family in a single SQL statement as the federated component. The DB2 family is made up of DB2 Universal Database for Windows, OS/2, Linux, Unix, AS/400, VM/VSE, and S/390. The strategic direction for all non-DB2 family databases is to provide read and write access using DB2 Relational Connect. Version 7.2 will add read-only access for SQL Server to DB2 Relational Connect.

In a federated environment, a developer or user would still use the DB2 syntax they understand to obtain or update data found on these federated data sources (including non-DB2 servers). The federated pieces in DB2 have been designed to understand these relational data sources and provide efficient optimization when accessing data across multiple platforms and vendors. This means that the user doesn't need to understand how to write efficient queries that hit an Oracle data source. The federated functionality provided by DB2

allows companies to leverage the skill of employees by focusing efforts on reporting and trend analysis, as opposed to statement tuning.

OLAP STARTER KIT

DB2 has very strong OnLine Analytical Processing (OLAP) offerings that allow users to easily define multidimensional applications, link and populate those applications with warehouse data, and share the applications among a workgroup. When you add one of the DB2 OLAP products to your DB2 installation, you suddenly grow the power of your DB2 server to store and manage both relational and multidimensional databases.

OLAP allows users to ask questions intuitively because the OLAP data is presented in business dimensions. OLAP can perform analyses such as: “Display the profit of my highest and lowest performing products last quarter in my domestic sales regions”. You can use DB2’s OLAP offering to deliver analytic applications that give your business a competitive edge.

Relational OLAP has been extended in the latest version of DB2. New SQL functions for moving aggregates, such as moving average and moving sum, are now available for analytical queries. They comply with the proposed OLAP Addendum to SQL-99. These extended functions are discussed later in this article.

There are two offerings in the DB2 family that provide OLAP services – DB2 OLAP Starter Kit and DB2 OLAP Server. Both packages come with an easy-to-use interface to build and manage OLAP applications. Both packages allow you to create, use, and analyse relational and multidimensional databases. DB2’s OLAP implementation is based on the Hyperion Essbase analytic engine.

With every server copy of DB2 except DB2 Satellite Edition, you get a copy of the DB2 OLAP Starter Kit. The installation of the DB2 OLAP Starter Kit is integrated into any DB2 server install. You can use the DB2 OLAP Starter Kit, a scaled-down version of the full-function DB2 OLAP Server product, to perform multidimensional analysis on your data.

The DB2 OLAP Starter Kit gives you DB2's OLAP power for up to three concurrent users. If you need to extend this power to multiple clients and add some restricted OLAP functionality with this version, you can purchase DB2 OLAP Server.

ADVANCED RELATIONAL SQL

DB2 UDB Version 7.1 includes a number of advanced SQL features that will be useful for analysts who work with enterprise data and rely on a robust and powerful set of functions to aid in the decision-making process. SQL enhancements that are related to BI are referred to as Relational OLAP (Online Analytical Processing) extensions (or ROLAP by those with an affinity for jargon). These new ROLAP extensions are new SQL commands that allow for sophisticated moving averages to be calculated against column functions and ranking functions.

Along with the integrated OLAP offering mentioned in the previous section, DB2 Version 7.1 implements new classes of aggregate functions for use in OLAP applications. These classes are designed to compute functions that are important to OLAP applications, but were difficult if not impossible with traditional SQL.

New SQL functions for moving aggregates, such as moving average and moving sum, are now available for analytical queries and they are in compliance with the proposed OLAP Addendum to SQL-99. These additional computational functions expand your OLAP capabilities with SQL and improve the performance against queries that are typically challenging to write, if possible at all.

Moving aggregate functions are those that operate on a 'window' of the data at a time. Syntactically, moving aggregate functions are regular aggregate functions with additional arguments. Moving aggregates are directed towards business analysts within a firm that need to be empowered with the ability to create sophisticated queries against their data. Since this function can now be done at the SQL level and is therefore driven in the database engine (instead of by third-party tools, which can lead to performance degradation), DB2 can offer better performance for the end user. For example, users may

wish to average a number of consecutive rows of output for the purposes of curve-smoothing. This function would be especially useful for business analysts or economists who work with stock price fluctuations or have a need to seasonally adjust information.

Another ROLAP enhancement in DB2 UDB Version 7.1 is with the grouping of information, built upon the pre-existing ROLLUP function that was introduced in DB2 Version 5.2. Functions that apply to groups of data can now include a form of logic within them to allow for a more sophisticated level of grouping.

THE DB2 SPATIAL EXTENDER

DB2 UDB Version 7.1 introduces the notion of spatially aware SQL queries. Users can now integrate spatial data (locations via coordinates) with normal SQL data in compliance with the OpenGIS Consortium (OGC) and ISO SQLMM standards. The combination of these two technologies gives users access to new types of query that they could not previously run. These queries will result in even more insight into business and social issues. Why is spatial data becoming so important? Think about it: businesses spend thousands, if not millions, of dollars to extract, transform, and load (ETL) information into giant data warehouses. Almost all data has some sort of spatial component to it.

The fact that Nancy Doyle lives at 213 Weatherborn is a spatially enabled piece of data. Now imagine the combination of knowing that Nancy lives at 213 Weatherborn, with the fact that there is public transportation to the country's most upmarket shopping centre. Finally, visualizing this information allows a marketing manager to offer some sort of transit-related promotion to try and capture Nancy's spending money.

Spatial data can be geocoded by DB2 so it can be mapped into graphical representations that offer business analysts a more realistic view into the data. Figure 2 shows an example of address and occurrence data, stored and managed by DB2, being translated to a visual map using a partner tool such as ESRI's ArcInfo. In DB2 UDB Version 7.1, the new Spatial Extender is used to achieve this functionality. The Spatial extender will store and index the spatial data

Figure 2: Example of address and occurrence data

(coordinate information) and allow specific spatial queries against this data.

Figure 2 might represent the result set of an FBI crime data query such as robbery crimes per 100,000 people or car thefts per 100,000 people. Figure 2 could also represent a safe zone for a tarnished water system in the event of an emergency. Whatever the query in Figure 2 represents, the important thing to note here is that your DB2-managed data has been transformed and has become spatially aware. Humans tend to comprehend and digest visuals much more easily than tabulated data. The DB2 Spatial Extender is a new enhancement that helps BI experts better use and leverage their data for better decisions.

ENHANCEMENTS TO THE LOAD UTILITY

The LOAD process is very important to data warehousing environments. Depending on the line of business and the industry, data may need to be refreshed at an hourly rate, or perhaps even faster. For example, a famous cellular provider reloads millions of rows into their data warehouse every few minutes. To meet such needs, some changes to the LOAD and IMPORT utilities have been introduced in

DB2 UDB Version 7.1. These changes can be categorized as:

- IMPORT/LOAD data type extensions – data files can now contain user-defined date and time formats as well as zoned decimal data. In addition, blanks can now be preserved in delimited (DEL) columns that are being placed into CHAR or VARCHAR columns.
- LOAD authority – the LOAD authority, in previous releases only available with DB2 Universal Database for OS/390, is now available for the whole DB2 Universal Database family. Users granted LOAD authority can run the LOAD utility without the need for SYSADM or DBADM authority. This allows users to perform more DB2 functions, and gives database administrators more granular control over the administration of their database.
- Remote load capability – you can specify a remote load through a LOAD API call without changing the LOAD API's prototype. When you specify an input source to be of type SQLU_CLIENT_LOCATION instead, the LOAD utility can invoke the new remote LOAD functionality. Through CLP, a new optional keyword 'CLIENT' can be added to the LOAD command that will invoke a remote load.

DATA WAREHOUSE CENTRE

Data warehousing is the foundation for business intelligence and customer relationship management, so it is important to do it right. Doing it right means accurately translating your business user needs into data models, building an easily accessible data warehouse that continually draws on diverse applications and data sources, and finding a way to maintain that data warehouse in an *ad hoc* environment that is ever-changing.

If you're thinking that data warehousing is a resource-depleting and costly mechanism, you're wrong! DB2 has tools that provide a graphical environment to help you create and manage data warehouses or data marts – all from the Control Center!

There are two offerings in the DB2 family that provide data warehousing

services – DB2 Data Warehouse Center and DB2 Warehouse Manager. Both packages come with an easy-to-use interface, called the Data Warehouse Center, that can be used to build and manage data warehouses.

The power of Visual Warehouse (an old DB2 product used to create data warehouses) and the simplicity of the DB2 Control Center have been merged to provide a single, new user interface for business intelligence customers. You can use the Data Warehouse Center to register and access data sources, define data extraction and transformation steps, populate data warehouses, automate and monitor warehouse management processes, and manage and interchange metadata. The Data Warehouse Center is integrated with a DB2 server installation.

You will often hear the activities involved in data warehousing referred to as ETL (Extract, Transform, and Load). Extract is associated with the extraction of data for input into the database warehouse; for example, moving data from the operational side of your business. Transform refers to massaging (also called cleansing) data so that it is clean and usable in the data warehouse; for example, not having 126 Rory Rd., 126 Rory Road, and 126 Rory in the database. Loading refers to moving that extracted and cleansed data into the data warehouse using DB2 loading utilities.

Using the Data Warehouse Center, you can manage all the activities that encompass the building, loading, and maintenance of a Data Warehouse.

The DB2 Data Warehouse Center is available for Windows NT and Windows 2000 versions of DB2 and is limited to one worker agent that can perform requests on behalf of the administrators.

If you need to run a full-scale data warehouse too, you can purchase the DB2 Warehouse Manager. This product suite comes with a set of agents that can be deployed for data warehouse work, QMF for Windows, Query Patroller (used to manage queries that are submitted against the data warehouse), and the Information Catalog (for meta-data management).

MISCELLANEOUS ENHANCEMENTS

There are various other enhancements to DB2 UDB Version 7.1 that have a 'BI-flavour' to them. For example, you can now convert your ODBC queries into static SQL. This capability provides a performance gain for those canned queries that are repeatedly run against the data warehouse.

The database system monitor now allows you to monitor your DB2 UDB Enterprise – Extended Edition system from a single partition. It collects data and aggregates values across all partitions and returns a single result. This capability provides database administrators with a single point of control for monitoring their entire data warehouse. The database system monitor also collects information on the operation and performance of database activities ranging from reads and writes through to locks and deadlocks.

DB2 Universal Database Version 7.1 also expands support for metadata interchange with support for Object Management Group (OMG) standard Common Warehouse Metadata Interchange for facilitating solution integration among heterogeneous tools. The OMG standard has the support of industry leaders including IBM, Oracle, NCR, and Hyperion.

Finally, there is even an online HTML-based tutorial that you can take to become more familiar with the end-to-end typical business intelligence tasks. The lessons in this tutorial provide step-by-step instructions for data warehousing and OLAP tasks, using the sample databases that come with DB2.

Paul Zikopoulos
Database Specialist
IBM DB2 Sales Support Team (Canada)

Printing out DSNZPARM and DSNHDECP

The March 2001 issue of DB2 Update *contained an article/program for displaying DSNHDECP values. This is a much simpler C++ program for displaying both DSNZPARM and DSNHDECP values – the program just calls an IBM-supplied stored procedure.*

INTRODUCTION

This article looks at a way of printing the DSNZPARM and DSNHDECP values for a DB2 subsystem. This will be done by using a CAF C++ program calling IBM-supplied stored procedure DSNWZP.

WHY?

Why do we want to do this?

The main reason is system documentation – knowing how your system's set up. This information is readily available in online monitors, but being able to print it from a batch job means that you can dump these parameters automatically.

Preventing system parameters being regressed is an important issue. When emergency parameter changes are made – for instance EDM pool changes – it's likely that this will be done by updating the DSNTIJUZ job directly, rather than through the installation CLIST. This can result in the changes not being reflected in the DSNTID*xx* member, and then being lost at the next upgrade. Printing the system parameters to a file and then comparing them after upgrades can help spot such errors.

DSNWZP

DSNWZP is an IBM-supplied stored procedure. It gets bound along with a lot of other stuff in installation job DSNTIJSG. Its reason for existence is the Control Centre. It's the means of passing system information back to the workstation – but we can call it ourselves directly on the mainframe.

The advantage of using DSNWZP is that it's IBM-supplied and therefore IBM-maintained. As new parameters are added, DSNWZP will have them incorporated. Our program is really just a wrapper for DSNWZP. This reduces the maintenance effort on our part to almost zero.

DSNWZP has no inputs, and one output, a VARCHAR 8,600 bytes long in DB2 V5 and 12,000 bytes long in V6. The difference is because, in V5, DSNWZP only outputs DSNZPARM values, whereas under V6, it outputs DSNHDECP as well.

The output format of DSNWZP consists of a series of lines, each separated from the next by a X'25' character. It looks like this:

```
SYSPAUDT/DSN6SYSP/AUDITST/DSNTIPN/ 1/AUDIT TRACE/
1000000000000000000000000000000000
SYSPCDB/DSN6SYSP/CONDBAT/DSNTIPE/ 4/MAX REMOTE CONNECTED/0000000020
SYSPCT/DSN6SYSP/CTHREAD/DSNTIPE/ 2/MAX USERS/00100
.
.
.
```

Each line represents a system parameter and consists of seven fields separated by a slash ('/').

The first field is the internal parameter name as used in the system macro. The second is the name of the macro itself. You will find these used in the DSNTIJUZ job, which assembles and links the system parameters. The third field is the external parameter name as used in the DSNTID*xx* member. The fourth field is the install panel name, and the fifth field the number of the input field on the panel that this parameter relates to. The sixth field is the description as used on the panel, and the last field is the value of the parameter.

Each line is therefore a cross-reference – if you change a parameter in the DSNTIJUZ job directly, you can find the DSNTID*xx* parameter name to change so you can retrofit it, or find the panel in the installation CLIST to regenerate the DSNTID*xx* member and DSNTIJUZ job.

Because it's a stored procedure, DSNWZP needs to be called from static SQL (unless it's called from an ODBC program).

WHY C++?

Why C++? – because it's the easiest way to do it. <evangelization> Believe me, if you know COBOL, Assembler, and C++, C++ is by far the easiest </evangelization>. However, for those who don't have C++ set up on their mainframes, the SQL and techniques mentioned here will work from any other language.

I could have used C instead, but I'm just using a couple of C++ goodies to make things easier:

- Streams – a simple, type-safe output method.
- Object orientation – the ability to define objects. In this exercise the only object I'm going to use is the IBM-supplied I0String object, which will enable easy string manipulation.

THE SQL

The heart of this program is just one SQL call – the call to the stored procedure. Everything else is just set-up, error-checking, and formatting.

A varchar is required:

```
EXEC SQL BEGIN DECLARE SECTION;

struct
  { short int  len;
    char  data 32767 ;
  }
   OUTSTR;
```

I've made it 32KB because the size of the varchar returned is likely to increase in future releases of DB2.

The SQL call looks like this:

```
    EXEC SQL
        CALL DSNWZP(:OUTSTR);
```

I0STRING

I0String is an IBM-provided class that represents strings in C++ (by the way, the second character is a zero, not an oh). The advantage over

using normal C strings is easier allocation and resizing, and the provision of operators to do concatenation and output.

To allocate an I0String, I can just say:

```
I0String str("abcde");
```

or point it at a block of memory:

```
I0String str(charPtr,length)
```

where length is the length I want the string to be. There are a lot of other constructors to create I0Strings from integers, floats, etc.

To print an I0String to standard output:

```
cout << str;
```

Concatenating two strings to make a third string:

```
strc = stra + strb;
```

To use I0String, you need this statement in your program:

```
#include <i0string.h>
```

GENERAL ROUTINES

The program uses a number of routines for setting up CAF and error handling. I've set these up in a separate module because they will come in handy for any future DB2/C++ programs.

To open CAF:

```
int open_for_CAF(const I0String &db2sys,const I0String &plan)
    {
     long retcode;
     unsigned long reascode;
     I0String pl = plan.subString(0,8,' ');
     DSNALI("OPEN        ",(char*)db2sys,(char*)pl,
         &retcode,&reascode);
     if (retcode != 0)
         caf_error("Can't open CAF -",(char*)(db2sys+"/"+pl),
              retcode,reascode);
     return(retcode);
    }
```

The routine takes as input the DB2 subsystem name and plan name. It pads out the plan name before calling DSNALI. If there's an error, it calls caf_error:

```
void caf_error(char *p1,char *p2,long retcode,
                unsigned long reascode)
    {
     cerr << p1;
     if (p2)
         cerr << p2;
     cerr << ",retcode=" << retcode << ",reascode ="
         << IØString(reascode).d2x().rightJustify(8,'Ø') << "\n";
     exit(retcode);
    }
```

This just prints out the subsystem name, plan name, and return, and reason codes. The reason code is converted to hex and padded out with leading zeroes so it's easy to look up in an online message-look-up tool:

```
To close CAF:
 int close_for_CAF()
    {
     long retcode;
     unsigned long reascode;
     DSNALI("CLOSE       ","SYNC",
         &retcode,&reascode);
     if (retcode != Ø)
         caf_error("Error closing CAF",NULL,retcode,reascode);
     return(retcode);
    }
```

For these routines, DSNALI must be defined thus:

```
extern "OS" {int DSNALI(char *func,...);}
```

I've also got a generic DB2-error-handling routine, which takes a C string and the SQLCA as input and prints out a formatted DB2 message:

```
 int process_db2_error(char *text,struct sqlca *sqlca)
    {
     cerr << text << "\n";
     DSNTIAR(sqlca,&error_message,&__dsntiar_len);
     for (int i=Ø;i < __DATA_DIM;i++)
        {
         IØString str(error_message.error_text i ,__DATA_LEN);
         if (str.strip() != "")
             cerr << str << "\n";
        }
     return(Ø);
    }
```

The routine depends on these definitions, which come straight out of the *Application Programming and SQL Guide*:

```
#define __DATA_LEN 120
#define __DATA_DIM 10

struct error_struct
    {
     short int error_len;
     char error_text __DATA_DIM  __DATA_LEN ;
    };

 struct error_struct
    error_message = {__DATA_DIM * __DATA_LEN};

 extern "OS" short int DSNTIAR(struct sqlca *sqlca,
      struct error_struct *error_message,int *data_len);

 int __dsntiar_len = __DATA_LEN;
```

THE MAIN PROGRAM

The program takes the subsystem name as its first parameter. It prints out an error message if there are no parameters:

```
int main(int argv,char** argc)
    {
     if (argv < 2)
        {
         cerr << "subsystem parameter required\n";
         exit(12);
        }
```

The subsystem name (argc[1]) and the program name (argc[0] – this is used as the plan name) are converted to upper case (in case they weren't) and passed to the CAF-opening routine:

```
     IØString subsys(argc[1]);
     subsys = subsys.upperCase();
     IØString plan(argc[Ø]);
     plan = plan.upperCase();
     open_for_CAF(subsys,plan);

The SQL is invoked and checked for error:
     EXEC SQL
         CALL DSNWZP(:OUTSTR);
     if (SQLCODE < Ø)
        {
```

```
         process_db2_error("Error calling DSNWZP",&sqlca);
         exit(12);
        }
```

The returned varchar is formatted and printed:

```
     process_output();

Finally, CAF is closed and we go home:
     close_for_CAF();
     return(Ø);
    }
```

FORMATTING THE OUTPUT

The process_output routine splits the varchar up into lines and passes each one as an I0String to format_line:

```
void process_output()
    {
     char *ptr,*sPtr,*endPtr;
     ptr = sPtr = OUTSTR.data;
     endPtr = OUTSTR.data+OUTSTR.len;
     while (sPtr < endPtr)
        {
         while (ptr < endPtr && *ptr != Øx25)
             ptr++;
         IØString str(sPtr,ptr - sPtr);
         format_line(str);
         sPtr = ++ptr;
        }
    }
```

The format_line calls format_field for each field. It passes to format_field a pointer to the current position in the line, a pointer to the end of the line, the width of the field, and whether the field should be right-justified. If the input line is all blanks, format_line does nothing:

```
void format_line(IØString &line)
    {
     char *ptr,*endPtr;
     if (line.strip() == "")
         return;
     ptr = (char*)line;
     endPtr = ptr + line.size();
     ptr = format_field(ptr,endPtr,8,false);   //Int Parameter Name
     ptr = format_field(ptr,endPtr,8,false);   //Macro Name
     ptr = format_field(ptr,endPtr,18,false);  //Ext Parameter Name
```

```
    ptr = format_field(ptr,endPtr,8,false);   //Install panel
    ptr = format_field(ptr,endPtr,4,true);    //Install Panel Number
    ptr = format_field(ptr,endPtr,Ø,false);   //Description
    cout << "= ";
    ptr = format_field(ptr,endPtr,Ø,false);   //Value
    IØString str(ptr,endPtr - ptr);
    cout << str.strip();                      //rest of Line
    cout << "\n";
   }
```

The format_field prints from the current position up to the next '/' or the end of the line. It gets rid of any leading or trailing blanks and right-justifies the output if required. If the field is shorter than the requested length it pads it out, but it does not truncate longer fields. It returns the new pointer position to the calling routine:

```
char *format_field(char *ptr,char *endPtr,int width,int rjustify)
   {
    char *sPtr = ptr;
    while (ptr < endPtr && *ptr != '/')
        ptr++;
    IØString str(sPtr,ptr - sPtr);
    str = str.strip();
    if (width > Ø)
        if (rjustify)
            str = str.rightJustify(width,' ');
        else
            str = str.leftJustify(width,' ');
    cout << str << " ";
    if (ptr < endPtr)
        ++ptr;
    return(ptr);
   }
```

PREPARATION

The program must be precompiled, compiled, linked, and bound.

The DSNHCPP procedure created during DB2 installation is an example of JCL to do this. Note that the program should be bound as a program object rather than linked as a load module so that it can handle routine names longer than eight characters – it must therefore be bound into a PDSE.

RUNNING THE PROGRAM

JCL to run the program looks like:

```
//STEP1   EXEC PGM=P##WZP
//            PARM='DB2S'
//STEPLIB  DD  DSN=NUDBS.SMITHAC.LOADE,DISP=SHR
//         DD  DSN=SYS2.DB2S.SDSNLOAD,DISP=SHR
//SYSOUT   DD  SYSOUT=V
//SYSPRINT DD  SYSOUT=V
```

THE OUTPUT

The output looks like this:

```
SYSPAUDT DSN6SYSP AUDITST   DSNTIPN  1 AUDIT TRACE =
10000000000000000000000000000000
SYSPCDB  DSN6SYSP CONDBAT   DSNTIPE  4 MAX REMOTE CONNECTED = 0000000020
SYSPCT   DSN6SYSP CTHREAD   DSNTIPE  2 MAX USERS = 00100
SYSPDFRQ DSN6SYSP DLDFREQ   DSNTIPN  16 LEVELID UPDATE FREQUENCY = 00005
SYSPFRQ  DSN6SYSP PCLOSEN   DSNTIPN  14 RO SWITCH CHKPTS = 00005
SYSPIDB  DSN6SYSP IDBACK    DSNTIPE  6 MAX BATCH CONNECT = 00200
SYSPIDF  DSN6SYSP IDFORE    DSNTIPE  5 MAX TSO CONNECT = 00100
SYSPLOGL DSN6SYSP LOGLOAD   DSNTIPN  10 CHECKPOINT FREQ = 0000050000
SYSPMON  DSN6SYSP MON       DSNTIPN  8 MONITOR TRACE = 00000000
SYSPMONS DSN6SYSP MONSIZE   DSNTIPN  9 MONITOR SIZE = 0000008192
SYSPRLFA DSN6SYSP RLFAUTH   DSNTIPP  8 RESOURCE AUTHID = NUDBSSYS
SYSPRLF  DSN6SYSP RLF       DSNTIPO  4 RLF AUTO START = NO
SYSPRLFN DSN6SYSP RLFERR    DSNTIPO  6 RLST ACCESS ERROR = NOLIMIT
SYSPRLFT DSN6SYSP RLFTBL    DSNTIPO  5 RLST NAME SUFFIX = 01
.
.
.
```

Removing leading zeroes on the numeric output fields is left as an exercise for the reader.

CAVEATS

When a new system parameter is added to DSNZPARM through an application of a PTF, it will probably not appear in DSNWZP straight away. It is also unlikely to get saved in the DSNTID*xx* member. Eventually DSNWZP and the installation CLIST get upgraded to deal with the new parameter.

Because DSNWZP is a stored procedure, you cannot run it when in maint mode.

Alan Smith
DB2 Systems programmer
Norwich Union (UK)

Partitioned tablespace management

Partition balancing is one of the activities that need to be done by DBAs on a routine basis. It is neglected mostly because of the difficulty in identifying partitions that are out of balance as well as in re-defining the limit keys. For well-designed databases, the degree of parallelism during a tablespace scan is determined by the total number of pages, the number of partitions, and the partition having the highest number of pages. For example, if a table has 10,000 pages and has 10 partitions, and the partition having the highest number of pages is partition 10 with 2,000 pages, then the highest degree of parallelism that can be achieved is 10,000 divided by 2,000, which is 5. However, if the highest number of pages in any partition is 1,000, then a parallelism degree of 10 can be easily achieved for a tablespace scan.

The number of partitions to have depends largely on the record length and the size of the table. Sometimes, peculiar business cases would dictate the number and size of partitions. Version 5.0 of DB2 allows 255 partitions, numbered from 0 to 254, each of which can be up to 4GB, for a tablespace defined as‘ large’. For a ‘non-large’ tablespace, the partition size can be 1GB, 2GB, or 4GB, depending on the number of partitions being in the ranges 33 to 64, 17 to 32, or 1 to 16 respectively.

The limit keys are critical to partitioning and maintaining a good balance. The limit keys are normally derived from an algorithm that utilizes a reverse timestamp, so as to give a good spread of randomness to the keys. A well-defined algorithm will result in a uniform distribution of the limit keys. However, as time goes on, the partitioning ranges tend to get uneven with the result that some partitions will be crowded while others will be sparse. In this case, the partitions need to be balanced again. Effectively, this means recomputing and redefining the limit keys so as to realize the uniform distribution of rows across each partition.

The query PARTINF3 helps to decide whether a tablespace needs to be re-partitioned. It is best if executed soon after a **runstats** has been

done on all the partitioned tablespaces. It identifies the total rows in the tablespace, the average rows expected in each partition, and also shows the deviation from the average for each partition. Based on this statistic, you may decide to re-partition or leave it alone for some more time.

Query PARTINF3 is shown below:

```
SELECT A.DBNAME, A.TSNAME, A.PARTITION AS P1, A.CARD,
       Y.ACRD AS AVERAGE,
       (A.CARD-Y.ACRD) AS DIFF,
       SUBSTR(A.LIMITKEY,1,10) AS LIMKEY
  FROM SYSIBM.SYSTABLEPART A,
       (SELECT DBNAME, TSNAME, AVG(CARD) AS ACRD
          FROM SYSIBM.SYSTABLEPART
         WHERE TSNAME LIKE 'DB%'
           AND PARTITION > 0
         GROUP BY DBNAME, TSNAME ) AS Y
 WHERE A.TSNAME LIKE 'DB%'
   AND A.PARTITION > 0
   AND Y.DBNAME = A.DBNAME
   AND Y.TSNAME = A.TSNAME
   AND A.DBNAME = 'DBADB001'
   AND A.TSNAME = 'DBATS001'
;
```

The query PARTINF4 calculates the difference between row counts of the successive partitions. It can also be used to give an idea of partition balancing.

Query PARTINF4 is shown below:

```
SELECT A.DBNAME, A.TSNAME, A.PARTITION AS P1, A.CARD AS CARD1,
       SUBSTR(A.LIMITKEY,1,10) AS LIMKEY1,
       Y.ACRD AS AVGCRD,
       (A.CARD-Y.ACRD) AS ROWDEV,
       X.PARTITION AS P2, X.CARD AS CARD2,
       SUBSTR(X.LIMITKEY,1,10) AS LIMKEY2,
       (A.CARD - X.CARD) AS DIFFBETPARTS
 FROM  SYSIBM.SYSTABLEPART A
      ,(SELECT DBNAME, TSNAME, PARTITION, CARD, LIMITKEY
          FROM SYSIBM.SYSTABLEPART
         WHERE TSNAME LIKE 'DB%'
           AND PARTITION > 0) AS X
      ,(SELECT DBNAME, TSNAME, AVG(CARD) AS ACRD
          FROM SYSIBM.SYST ABLEPART
         WHERE TSNAME LIKE 'DB%'
```

```
           AND PARTITION > Ø
         GROUP BY DBNAME, TSNAME ) AS Y
WHERE A.TSNAME LIKE 'DB%'
  AND A.PARTITION > Ø
  AND X.DBNAME = A.DBNAME
  AND X.TSNAME = A.TSNAME
  AND Y.DBNAME = A.DBNAME
  AND Y.TSNAME = A.TSNAME
  AND (X.PARTITION - A.PARTITION) = 1
  AND A.DBNAME = 'DBADBØØ1'
  AND A.TSNAME = 'DBATSØØ1'
;
```

For tablespaces that need partitioning or re-partitioning, the COBOL program PART2 will be a great time-saver. It uses dynamic SQL, to enable it to run on any tablespace, and is written in COBOL. It assumes that there is only one column that constitutes the limit key. (A different version of the program can be coded for concatenated limit keys of two or more columns.) This has to be compiled and bound as a regular COBOL-DB2 batch program, with a suitable plan name. Ensure that the isolation level for the plan is explicitly defined as 'Uncommitted Read'. The default isolation level is 'Repeatable Read' which could be a very nasty surprise when you run this program in production. With the Uncommitted Read isolation, this program can be run during peak times without any impact. The inputs to the program are given as one line in the SYSIN DD card and are described in the execution JCL. One of the inputs is a BUFFER, which may be specified as zero. A non-zero BUFFER will add an extra number of rows as specified by BUFFER to each calculated partitioning range. For example, if we are dividing 1000 rows into 5 partitions, specifying a BUFFER of 10 rows will put 210 rows into each partition instead of 200. The last partition will always end with all 9s.

Sample outputs from the queries and the program are shown, as also is execution JCL for the program.

CONCLUSION

A tablespace will benefit from partitioning if it has grown too big. Be judicious in the use of partitioning. Avoid too much as well as too little. Keeping your partitions in balance will definitely help parallelism and performance.

A crucial parameter to consider for partitioned tablespaces is the LOCKPART parameter. LOCKPART YES is very critical for concurrency as well as parallelism. Unless you have a good reason, this should always be YES. Ensure that the packages are bound with degree ANY for LOCKPART to be really effective.

Remember that dropping a partitioned index will drop the partitioned tablespace. In Version 5.0 of DB2, altering the limit keys would mean dropping the tablespace and recreating it. However, in DB2 Version 7.0, there is the ability to alter the limit keys without having to drop the index and hence the table. However, the revised partitioning will take effect only when the tablespace is REORGed.

SAMPLE OUTPUT FOR PARTINF3

DBNAME	TSNAME	P1	CARD	AVERAGE	DIFF	LIMKEY
DBADB001	DBATS001	1	159064	164179	-5115	7780331
DBADB001	DBATS001	2	159064	164179	-5115	12571170
DBADB001	DBATS001	3	159064	164179	-5115	19912534
DBADB001	DBATS001	4	159064	164179	-5115	36075430
DBADB001	DBATS001	5	159064	164179	-5115	41449281
DBADB001	DBATS001	6	159064	164179	-5115	47699301
DBADB001	DBATS001	7	159064	164179	-5115	63200794
DBADB001	DBATS001	8	159064	164179	-5115	85078772
DBADB001	DBATS001	9	184469	164179	20290	196293021
DBADB001	DBATS001	10	190225	164179	26046	311585808
DBADB001	DBATS001	11	189002	164179	24823	427462016
DBADB001	DBATS001	12	189618	164179	25439	544010900
DBADB001	DBATS001	13	189331	164179	25152	660570808
DBADB001	DBATS001	14	33808	164179	-130371	777383516
DBADB001	DBATS001	15	189269	164179	25090	891514017
DBADB001	DBATS001	16	188631	164179	24452	999999999

where: P1 is the partition number; CARD is the # rows; AVERAGE is total rows / # partitions); DIFF is the difference between CARD and AVERAGE; LIMKEY is the present limit key range.

SAMPLE OUTPUT FOR PARTINF4

DBNAME	TSNAME	P1	CARD1	LIMKEY1	AVGCRD	ROWDEV	P2	CARD2	LIMKEY2	DIFFBETPARTS
DBADB001	DBATS001	1	159064	7780331	164179	-5115	2	159064	12571170	0
DBADB001	DBATS001	2	159064	12571170	164179	-5115	3	159064	19912534	0
DBADB001	DBATS001	3	159064	19912534	164179	-5115	4	159064	36075430	0
DBADB001	DBATS001	4	159064	36075430	164179	-5115	5	159064	41449281	0
DBADB001	DBATS001	5	159064	41449281	164179	-5115	6	159064	47699301	0
DBADB001	DBATS001	6	159064	47699301	164179	-5115	7	159064	63200794	0
DBADB001	DBATS001	7	159064	63200794	164179	-5115	8	159064	85078772	0
DBADB001	DBATS001	8	159064	85078772	164179	-5115	9	184469	196293021	-25405
DBADB001	DBATS001	9	184469	196293021	164179	20290	10	190225	311585808	-5756
DBADB001	DBATS001	10	190225	311585808	164179	26046	11	189002	427462016	1223
DBADB001	DBATS001	11	189002	427462016	164179	24823	12	189618	544010900	-616
DBADB001	DBATS001	12	189618	544010900	164179	25439	13	189331	660570808	287
DBADB001	DBATS001	13	189331	660570808	164179	25152	14	33808	777383516	155523
DBADB001	DBATS001	14	33808	777383516	164179	-130371	15	189269	891514017	-155461
DBADB001	DBATS001	15	189269	891514017	164179	25090	16	188631	999999999	638

where: ROWDEV is the difference between AVERAGE and CARD for partitions under P1, and DIFFBETSPARTS is the difference between CARD of successive parts.

SAMPLE OUTPUT FROM PROGRAM PART2

```
NO OF PARTITIONS:   016
TABLENAME       :   CUST_TABLE
COLUMN NAME     :   CUSTOMER_NO
QUALIFIER       :   PRODDB
BUFFER NUMBER   :   0000

PART #      MAXIMUM VALUE

001      000000000005079058
002      000000000010852150
003      000000000053619052
004      000000000101637718
005      000000000150804623
006      000000000154353346
007      000000000250015914
008      000000000253234629
009      000000000350448650
010      000000000353850738
011      000000000401383781
012      000000000405183658
```

```
Ø13      ØØØØØØØØØ453487ØØ9
Ø14      ØØØØØØØØØ715ØØ879Ø
Ø15      ØØØØØØØØØ771Ø1Ø184
Ø16      Ø99999999999999999
-------------------------------------
```

where MAXIMUM VALUE is the limit key value to use for each partition.

PART2

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. PART.
 AUTHOR. JAIWANT JONATHAN.
 INSTALLATION. JKJ.
 DATE-WRITTEN. OCTOBER, 2ØØØ.
 DATE-COMPILED.
*================================================================*
*REMARKS. THIS IS A PROGRAM TO PARTITION A TABLESPACE EVENLY BASED
*         ON THE TOTAL RECORDS AND THE NUMBER OF RECORDS IN EACH
*         PARTITION
*INPUT: DATASET CONTAINING THE FOLLOWING:
*       PARTN - NO OF PARTITIONS TO DIVIDE INTO (OPTIONAL)
*       COL   - THE COLUMN USED AS THE PARTITIONING KEY
*       TAB   - THE NAME OF THE TABLE
*       CRET  - THE CREATOR/QUALIFIER OF THE TABLE
*
*================================================================*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. IBM-37Ø.
 OBJECT-COMPUTER. IBM-37Ø.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT INPUT-FILE     ASSIGN TO UT-S-SYSIN.
     SELECT REPORT-FILE    ASSIGN TO UT-S-REPORT.
 DATA DIVISION.
 FILE SECTION.
*                  THE INPUT PARAMETERS
 FD  INPUT-FILE
     BLOCK CONTAINS Ø RECORDS
     RECORDING MODE IS F
     LABEL RECORDS ARE OMITTED.
 Ø1  INPUT-FILE-RECORD.
     Ø5  PARTNO                      PIC X(3).
     Ø5  FILLER                      PIC X(1).
     Ø5  COLNAM                      PIC X(18).
     Ø5  FILLER                      PIC X(1).
```

```
     Ø5  TABNAM                       PIC X(18).
     Ø5  FILLER                       PIC X(1).
     Ø5  CRETOR                       PIC X(8).
     Ø5  FILLER                       PIC X(1).
     Ø5  BUFFER                       PIC X(4).
     Ø5  FILLER                       PIC X(25).
*                 PGM PRODUCES A REPORT
 FD  REPORT-FILE
     BLOCK CONTAINS Ø RECORDS
     RECORDING MODE IS F
     LABEL RECORDS ARE OMITTED
     DATA RECORD IS REPORT-FILE-RECORD.
 Ø1  REPORT-FILE-RECORD           PIC X(8Ø).
 WORKING-STORAGE SECTION.
 Ø1  WS-WORK-VAR.
     Ø5  WS-PARTNO                PIC X(3)   VALUE 'ØØØ'.
     Ø5  WS-CRETOR                PIC X(8)   VALUE SPACES.
     Ø5  WS-TABNAM                PIC X(18)  VALUE SPACES.
     Ø5  WS-COLNAM                PIC X(18)  VALUE SPACES.
     Ø5  WS-TOT-ROWS              PIC S9(1Ø) COMP-3.
     Ø5  WS-BREAKLIM              PIC S9(1Ø) COMP-3.
     Ø5  WS-PRTNO                 PIC S9(4)  COMP.
     Ø5  WS-COL-INFO              PIC S9(18) COMP-3.
     Ø5  WS-PARTNO-NUM            PIC 9(4).
     Ø5  WS-BUFFER-NUM            PIC 9(4).
 Ø1  ERROR-MESSAGE.
     Ø5  ERROR-LEN                PIC S9(4)  COMP VALUE +96Ø.
     Ø5  ERROR-TEXT               PIC X(8Ø) OCCURS 8 TIMES
                                         INDEXED BY ERROR-INDEX.
 Ø1 STMTBUF.
     49  STMTLEN       PIC S9(4) COMP VALUE 1ØØ.
     49  STMTCHAR      PIC X(1ØØ).
 Ø1  WS-CT.
     Ø5  WS-CT-SEL                PIC X(23)
                            VALUE ' SELECT  COUNT(*) FROM '.
*    Ø5  WS-CT-CRE                PIC X(18) VALUE SPACES.
*    Ø5  FILLER                   PIC X(1) VALUE SPACES.
     Ø5  WS-CT-TAB                PIC X(27) VALUE SPACES.
     Ø5  FILLER                   PIC X(52) VALUE SPACES.
 Ø1  WS-DT.
     Ø5  WS-DT-SEL                PIC X(1Ø)  VALUE 'SELECT '.
     Ø5  WS-DT-COL                PIC X(18)  VALUE SPACES.
     Ø5  WS-DT-FRM                PIC X(6)  VALUE ' FROM '.
     Ø5  WS-DT-TAB                PIC X(27) VALUE SPACES.
     Ø5  FILLER                   PIC X(39)
                       VALUE ' ORDER BY 1 FOR FETCH ONLY WITH UR'.
 Ø1  WS-OUT.
     Ø5  WS-OUT-PRTNO             PIC 9(4).
     Ø5  FILLER                   PIC X(6)  VALUE SPACES.
     Ø5  WS-OUT-VALUE             PIC X(4Ø).
```

```
     Ø5  FILLER                    PIC X(3Ø) VALUE SPACES.
 Ø1  WS-FLAGS.
     Ø5 WS-FILE-NOT-AT-EOF                      PIC X(3)
                                                VALUE 'NO '.
        88 WS-FILE-AT-EOF                       VALUE 'YES'.
     Ø5 WS-PROCESS-COMPLETE                     PIC X(3)
                                                VALUE 'NO '.
        88 WS-COMPLETE-CN                       VALUE 'YES'.
*******************************************************
** REPORT HEADER STRUCTURE                            *
*******************************************************
 Ø1  HEADER.
         Ø5  FILLER PIC X(35)
             VALUE ' PART NO          VALUE          '.
 Ø1  MSG-SQLERR.
     Ø5  FILLER PIC X(31)
         VALUE ' DSNT493I SQL ERROR, SQLCODE = '.
     Ø5  MSG-MINUS       PIC X(1).
     Ø5  MSG-PRINT-CODE  PIC 9(8).
     Ø5  FILLER PIC X(41) VALUE '            '.
 Ø1  MSG-NOROW.
     Ø5  FILLER PIC X(8Ø)
         VALUE ' NO ROWS FOUND IN TABLE '.
 Ø1  WS-HDR-ROW.
     Ø5  FILLER                    PIC X(1) VALUE SPACES.
     Ø5  WS-HDR-NAME               PIC X(2Ø).
     Ø5  WS-HDR-DTL                PIC X(59).
 77  WS-NOT-FOUND                 PIC S9(8) COMP VALUE +1ØØ .
 77  WS-COUNT                     PIC S9(8) COMP VALUE Ø .
 77  ERROR-TEXT-LEN               PIC S9(8) COMP VALUE +8Ø .
*******************************************************
*   DECLARE CURSOR AND STATEMENT FOR DYNAMIC SQL
*******************************************************
*
     EXEC SQL DECLARE   CT CURSOR FOR CNT END-EXEC.
     EXEC SQL DECLARE  CNT STATEMENT END-EXEC.
     EXEC SQL DECLARE  DT CURSOR FOR STMT END-EXEC.
     EXEC SQL DECLARE  STMT STATEMENT END-EXEC.
     EXEC SQL INCLUDE  SQLCA END-EXEC.
*
 PROCEDURE DIVISION.
*
*******************************************************
* SQL RETURN CODE HANDLING                           *
*******************************************************
     EXEC SQL WHENEVER SQLERROR   GOTO DBERROR END-EXEC.
     EXEC SQL WHENEVER SQLWARNING GOTO DBERROR END-EXEC.
     EXEC SQL WHENEVER NOT FOUND  CONTINUE     END-EXEC.
*
*******************************************************
```

```
* MAIN PROGRAM ROUTINE                                    *
*********************************************************
     OPEN OUTPUT REPORT-FILE.
     OPEN  INPUT INPUT-FILE.
     PERFORM 1000-PROCESS-INPUT UNTIL WS-FILE-AT-EOF.
     CLOSE INPUT-FILE.
     CLOSE REPORT-FILE.
     GOBACK.
 EJECT
 1000-PROCESS-INPUT.
* ---------------------------------------------------------------
* 1. READ EACH RECORD IN INPUT FILE
* 2. CHECK FOR ALL INPUTS. IF PARTNO IS NOT SPECIFIED, THEN
*    USE EXISTING PARTNO FROM THE TABLESPACE.
* ---------------------------------------------------------------
     DISPLAY 'IN 1000-PROCESS-INPUT   PARA...'
     READ INPUT-FILE
       AT END
          SET WS-FILE-AT-EOF TO TRUE
     END-READ
     DISPLAY  'SYSIN RECORD...' PARTNO, TABNAM, COLNAM, CRETOR
     MOVE PARTNO  TO  WS-PARTNO
     MOVE BUFFER  TO  WS-BUFFER-NUM
     IF WS-PARTNO  = '   ' OR  WS-PARTNO = '000'
        PERFORM 2000-GET-PARTNO
           THRU 2000-GET-PARTNO-EXIT
     END-IF
     PERFORM  3000-GET-TOT-ROWS
        THRU  3000-GET-TOT-ROWS-EXIT
     MOVE  WS-PARTNO  TO  WS-PARTNO-NUM
     MOVE  BUFFER  TO  WS-BUFFER-NUM
     COMPUTE WS-BREAKLIM = (( WS-TOT-ROWS + 1 ) / WS-PARTNO-NUM )
                          +  WS-BUFFER-NUM
     MOVE 1 TO WS-COUNT
     MOVE 1 TO WS-PRTNO
     PERFORM  4000-PROCESS-ROWS
        THRU  4000-PROCESS-ROWS-EXIT.
 EJECT
     STOP RUN.
 2000-GET-PARTNO.
* ---------------------------------------------------------------
* THIS PARA GETS THE PARTNO FROM THE SYSTABLESPACE AND SYSTABLE
* USING THE INPUT PARAMETERS TABLENAME AND CREATOR.
* ---------------------------------------------------------------
     DISPLAY 'IN 2000-GET-PARTNO      PARA...'
     MOVE  PARTNO    TO  WS-PARTNO-NUM
     MOVE  TABNAM    TO  WS-TABNAM
     MOVE  COLNAM    TO  WS-COLNAM
     MOVE  CRETOR    TO  WS-CRETOR
     IF  WS-PARTNO-NUM = 0  THEN
```

```
           EXEC SQL
               SELECT PARTITIONS INTO :WS-PARTNO
                 FROM SYSIBM.SYSTABLESPACE A, SYSIBM.SYSTABLES B
                WHERE B.CREATOR = :WS-CRETOR AND B.NAME = :WS-TABNAM
                   AND B.TSNAME = A.NAME AND B.DBNAME = A.DBNAME
           END-EXEC
        END-IF.
 2000-GET-PARTNO-EXIT.
        EXIT.
 3000-GET-TOT-ROWS.
        DISPLAY 'IN 3000-GET-TOT-ROWS PARA ...'
        STRING  CRETOR DELIMITED BY SPACES '.' TABNAM
                DELIMITED BY SPACES
                INTO  WS-CT-TAB
*       MOVE  CRETOR  TO  WS-CT-CRE
*       MOVE  TABNAM  TO  WS-CT-TAB
        MOVE  WS-CT      TO  STMTCHAR
        EXEC SQL
            PREPARE  CNT  FROM  :STMTBUF
        END-EXEC.
        EXEC SQL
            OPEN  CT
        END-EXEC.
        EXEC SQL
            FETCH CT  INTO :WS-TOT-ROWS
        END-EXEC.
        DISPLAY 'AFTER FETCH CT SQLCODE...' SQLCODE
        EXEC SQL
            CLOSE CT
        END-EXEC.
 3000-GET-TOT-ROWS-EXIT.
        EXIT.
 4000-PROCESS-ROWS.
        DISPLAY 'IN  4000-PROCESS-ROWS ...' SQLCODE
        PERFORM  4015-WRITE-HDR
           THRU  4015-WRITE-HDR-EXIT
*
        STRING  CRETOR DELIMITED BY SPACES '.' TABNAM
                DELIMITED BY SPACES INTO  WS-DT-TAB
        MOVE  COLNAM  TO  WS-DT-COL
*       MOVE  TABNAM  TO  WS-DT-TAB
        MOVE  WS-DT      TO  STMTCHAR
        EXEC SQL
            PREPARE  STMT FROM  :STMTBUF
        END-EXEC.
        EXEC SQL
            OPEN  DT
        END-EXEC.
        EXEC SQL
            FETCH DT  INTO :WS-COL-INFO
```

```
    END-EXEC.
    IF SQLCODE = WS-NOT-FOUND
       WRITE REPORT-FILE-RECORD FROM MSG-NOROW
          AFTER ADVANCING 2 LINES
    ELSE
       PERFORM  4010-CHECK-ROWS
          THRU  4010-CHECK-ROWS-EXIT
          UNTIL  SQLCODE EQUAL TO WS-NOT-FOUND  OR
                 WS-COMPLETE-CN
    END-IF
    EXEC SQL
        CLOSE DT
    END-EXEC.
4000-PROCESS-ROWS-EXIT.
    EXIT.
4010-CHECK-ROWS.
    ADD 1  TO  WS-COUNT
    IF  WS-BREAKLIM < WS-COUNT
        PERFORM  4020-WRITE-INFO
           THRU  4020-WRITE-INFO-EXIT
        MOVE 1 TO WS-COUNT
        ADD  1 TO WS-PRTNO
        IF  WS-PRTNO  = WS-PARTNO-NUM
            MOVE  99999999999999999  TO  WS-COL-INFO
            PERFORM  4020-WRITE-INFO
               THRU  4020-WRITE-INFO-EXIT
            SET  WS-COMPLETE-CN  TO  TRUE
            GO TO 4010-CHECK-ROWS-EXIT
        END-IF
    END-IF
    EXEC SQL
        FETCH DT  INTO :WS-COL-INFO
    END-EXEC.
4010-CHECK-ROWS-EXIT.
    EXIT.
4015-WRITE-HDR.
    MOVE  'NO OF PARTITIONS:' TO  WS-HDR-NAME
    MOVE  WS-PARTNO  TO  WS-HDR-DTL
    WRITE  REPORT-FILE-RECORD  FROM  WS-HDR-ROW
    INITIALIZE  WS-HDR-ROW
    MOVE  'TABLENAME       :' TO  WS-HDR-NAME
    MOVE   TABNAM     TO  WS-HDR-DTL
    WRITE  REPORT-FILE-RECORD  FROM  WS-HDR-ROW
    INITIALIZE  WS-HDR-ROW
    MOVE  'COLUMN NAME     :' TO  WS-HDR-NAME
    MOVE   COLNAM     TO  WS-HDR-DTL
    WRITE  REPORT-FILE-RECORD  FROM  WS-HDR-ROW
    INITIALIZE  WS-HDR-ROW
    MOVE  'QUALIFIER       :' TO  WS-HDR-NAME
    MOVE   CRETOR     TO  WS-HDR-DTL
```

```
     WRITE  REPORT-FILE-RECORD  FROM  WS-HDR-ROW
     INITIALIZE  WS-HDR-ROW
     MOVE  'BUFFER NUMBER   :' TO  WS-HDR-NAME
     MOVE   WS-BUFFER-NUM     TO  WS-HDR-DTL
     WRITE  REPORT-FILE-RECORD  FROM  WS-HDR-ROW
     INITIALIZE  WS-HDR-ROW
     WRITE  REPORT-FILE-RECORD  FROM  WS-HDR-ROW.
     INITIALIZE  WS-HDR-ROW
     MOVE  'PART #      MAXIMUM' TO  WS-HDR-NAME
     MOVE  'VALUE'  TO  WS-HDR-DTL
     WRITE  REPORT-FILE-RECORD  FROM  WS-HDR-ROW
            AFTER ADVANCING 2 LINES
     INITIALIZE  WS-HDR-ROW
     WRITE  REPORT-FILE-RECORD  FROM  WS-HDR-ROW.
 4Ø15-WRITE-HDR-EXIT.
     EXIT.
 4Ø2Ø-WRITE-INFO.
     MOVE  WS-COL-INFO  TO  WS-OUT-VALUE
     MOVE  WS-PRTNO     TO  WS-OUT-PRTNO
     WRITE  REPORT-FILE-RECORD  FROM  WS-OUT.
 4Ø2Ø-WRITE-INFO-EXIT.
     EXIT.
 DBERROR.
*                                          **SQL ERROR
          MOVE SQLCODE TO MSG-PRINT-CODE.
          IF SQLCODE < Ø THEN MOVE '-' TO MSG-MINUS.
          WRITE REPORT-FILE-RECORD FROM MSG-SQLERR
             AFTER ADVANCING 2 LINES.
          CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.
          IF RETURN-CODE = ZERO
             PERFORM ERROR-PRINT VARYING ERROR-INDEX
                FROM 1 BY 1 UNTIL ERROR-INDEX GREATER THAN 8
          ELSE
*                                       **ERROR FOUND IN DSNTIAR
*                                       **PRINT ERROR MESSAGE
*            MOVE RETURN-CODE TO RETCODE
*            WRITE MSGREC FROM MSGRETCD
*               AFTER ADVANCING 2 LINES.
          GO TO PROG-END.
*
*****************************************************
*    PRINT MESSAGE TEXT                             *
*****************************************************
 ERROR-PRINT.
          WRITE REPORT-FILE-RECORD FROM ERROR-TEXT (ERROR-INDEX)
             AFTER ADVANCING 1 LINE.
 PROG-END.
*                                           **CLOSE FILES
          CLOSE INPUT-FILE
                REPORT-FILE.
```

```
          GOBACK.
```

EXECUTION JCL

```
//JOBCARD1
//*
//*
//JOBLIB  DD DSN=Yourhlq.DSNLOAD,DISP=SHR
//               DD DSN=Your load lib dataset, DISP=SHR
//*********************************************************************
//*
//PART2    EXEC PGM=IKJEFT1B,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSDBOUT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//REPORT   DD DSN=the report dataset name,DISP=(NEW,KEEP),
//         SPACE=(TRK,(10,20)),DCB=(LRECL=80,RECFM=FBA,BLKSIZE=3120),
//         UNIT=SYSDA
//SYSABOUT DD SYSOUT=*
//*
//SYSTSIN  DD *
  DSN SYSTEM(ssid)
  RUN PROGRAM (PART2)      -
      PLAN    (planname)
  END
//*
//SYSIN    DD *
016 CUSTOMER_NO         CUST_TABLE           PRODDB   0000
/*
//* Please follow record layout as given below for the SYSIN card
//*   05  PARTNO                       PIC X(3).
//*   05  FILLER                       PIC X(1).
//*   05  COLNAM                       PIC X(18).
//*   05  FILLER                       PIC X(1).
//*   05  TABNAM                       PIC X(18).
//*   05  FILLER                       PIC X(1).
//*   05  CRETOR                       PIC X(8).
//*   05  FILLER                       PIC X(1).
//*   05  BUFFER                       PIC X(4).
//*   05  FILLER                       PIC X(25).
//*
//*
```

Jaiwant K Jonathan
DB2 DBA
QSS (USA)

Sample user-defined functions – part 2

This month we publish the remaining user-defined functions.

COUNTER – PL/I source code

```
* PROCESS SYSTEM(MVS);
 COUNTER: PROC(UDF_RESULT, UDF_INDR,
               UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
               UDF_DIAG_MSG, UDF_SCRATCHPAD,
               UDF_CALL_TYPE, UDF_DBINFO)
          OPTIONS(MAIN NOEXECOPS REENTRANT);
 /*********************************************************************/
 /*    UDF   : COUNTER                                                */
 /*    OUTPUT: UDF_RESULT   CHAR      COUNTER OUTPUT PARAMETER        */
 /*********************************************************************/
  DCL UDF_RESULT     BIN FIXED(31);     /* RESULT PARAMETER           */
  DCL UDF_INDR       BIN FIXED(15);     /* INDICATOR FOR RESULT       */
  DCL 1 UDF_SCRATCHPAD,                 /* SCRATCHPAD                 */
       3 UDF_SPAD_LEN    BIN FIXED(31),
       3 UDF_SPAD_TEXT   CHAR(100),
        5 COUNTR          BIN FIXED(31);
  %INCLUDE UDFINFO;                          /* DBINFO                */
  DCL (LENGTH,ADDR)   BUILTIN;
  COUNTR=COUNTR+1;
  UDF_RESULT=COUNTR;
 END COUNTER;
```

COUNTERS – PL/I source code

```
* PROCESS SYSTEM(MVS);
 COUNTER: PROC(UDF_PARM1, UDF_RESULT,
                UDF_IND1,  UDF_INDR,
                UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
                UDF_DIAG_MSG, UDF_SCRATCHPAD,
                UDF_CALL_TYPE, UDF_DBINFO)
           OPTIONS(MAIN NOEXECOPS REENTRANT);
 /*********************************************************************/
 /*    UDF   : COUNTER                                                */
 /*    INPUT : UDF_PARM1    INTEGER   COUNTER STEP VALUE              */
 /*    OUTPUT: UDF_RESULT   INTEGER   COUNTER OUTPUT PARAMETER        */
 /*********************************************************************/
  DCL UDF_PARM1      BIN FIXED(31);     /* INPUT PARAMETER            */
  DCL UDF_RESULT     BIN FIXED(31);     /* RESULT PARAMETER           */
  DCL UDF_IND1       BIN FIXED(15);     /* INDICATOR FOR INPUT PARM   */
  DCL UDF_INDR       BIN FIXED(15);     /* INDICATOR FOR RESULT       */
  DCL 1 UDF_SCRATCHPAD,                 /* SCRATCHPAD                 */
       3 UDF_SPAD_LEN    BIN FIXED(31),
```

```
      3 UDF_SPAD_TEXT   CHAR(100),
       5 COUNTR          BIN FIXED(31);
 %INCLUDE UDFINFO;                          /* DBINFO               */
 DCL (LENGTH,ADDR)   BUILTIN;
 COUNTR=COUNTR+UDF_PARM1;
 UDF_RESULT=COUNTR;
END COUNTER;
```

CUMULI – PL/I source code

```
* PROCESS SYSTEM(MVS);
 CUMULI: PROC(UDF_PARM1, UDF_RESULT,
               UDF_IND1,  UDF_INDR,
               UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
               UDF_DIAG_MSG, UDF_SCRATCHPAD,
               UDF_CALL_TYPE, UDF_DBINFO)
          OPTIONS(MAIN NOEXECOPS REENTRANT);
 /*********************************************************************/
 /*    UDF   : CUMUL   - CUMULATIVE OPERATION                         */
 /*    INPUT : UDF_PARM1    INTEGER OR SMALLINT PARAMETER             */
 /*    OUTPUT: UDF_RESULT   INTEGER OUTPUT PARAMETER                  */
 /*********************************************************************/
  DCL UDF_PARM1      BIN FIXED(31);     /* INPUT PARAMETER           */
  DCL UDF_RESULT     BIN FIXED(31);     /* RESULT PARAMETER          */
  DCL UDF_IND1       BIN FIXED(15);     /* INDICATOR FOR INPUT PARM  */
  DCL UDF_INDR       BIN FIXED(15);     /* INDICATOR FOR RESULT      */
  DCL 1 UDF_SCRATCHPAD,                 /* SCRATCHPAD                */
       3 UDF_SPAD_LEN    BIN FIXED(31),
       3 UDF_SPAD_TEXT   CHAR(100),
        5 CUMULI         BIN FIXED(31);
  %INCLUDE UDFINFO;                          /* DBINFO               */
  DCL (LENGTH,ADDR)   BUILTIN;
  CUMULI=CUMULI+UDF_PARM1;
  UDF_RESULT=CUMULI;
 END CUMULI;
```

CUMULD – PL/I source code

```
* PROCESS SYSTEM(MVS);
 CUMULD: PROC(UDF_PARM1, UDF_RESULT,
               UDF_IND1,  UDF_INDR,
               UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
               UDF_DIAG_MSG, UDF_SCRATCHPAD,
               UDF_CALL_TYPE, UDF_DBINFO)
          OPTIONS(MAIN NOEXECOPS REENTRANT);
 /*********************************************************************/
 /*    UDF   : CUMUL   - CUMULATIVE OPERATION                         */
 /*    INPUT : UDF_PARM1    DECIMAL PARAMETER                         */
 /*    OUTPUT: UDF_RESULT   DECIMAL PARAMETER                         */
 /*********************************************************************/
```

```
  DCL UDF_PARM1      DECIMAL(15,3);     /* INPUT PARAMETER           */
  DCL UDF_RESULT     DECIMAL(15,3);     /* RESULT PARAMETER          */
  DCL UDF_IND1       BIN FIXED(15);     /* INDICATOR FOR INPUT PARM  */
  DCL UDF_INDR       BIN FIXED(15);     /* INDICATOR FOR RESULT      */
  DCL 1 UDF_SCRATCHPAD,                 /* SCRATCHPAD                */
       3 UDF_SPAD_LEN    BIN FIXED(31),
       3 UDF_SPAD_TEXT   CHAR(100),
        5 CUMULI         PIC'999999999999V999';
  %INCLUDE UDFINFO;                          /* DBINFO               */
  DCL (LENGTH,ADDR)   BUILTIN;
  CUMULI=CUMULI+UDF_PARM1;
  UDF_RESULT=CUMULI;
 END CUMULD;
```

CUMULF – PL/I source code

```
* PROCESS SYSTEM(MVS);
 CUMULF: PROC(UDF_PARM1, UDF_RESULT,
               UDF_IND1,  UDF_INDR,
               UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
               UDF_DIAG_MSG, UDF_SCRATCHPAD,
               UDF_CALL_TYPE, UDF_DBINFO)
          OPTIONS(MAIN NOEXECOPS REENTRANT);
 /*********************************************************************/
 /*    UDF   : CUMUL   - CUMULATIVE OPERATION                         */
 /*    INPUT : UDF_PARM1    FLOAT   PARAMETER                         */
 /*    OUTPUT: UDF_RESULT   FLOAT   PARAMETER                         */
 /*********************************************************************/
  DCL UDF_PARM1      BIN FLOAT(53);     /* INPUT PARAMETER           */
  DCL UDF_RESULT     BIN FLOAT(53);     /* RESULT PARAMETER          */
  DCL UDF_IND1       BIN FIXED(15);     /* INDICATOR FOR INPUT PARM  */
  DCL UDF_INDR       BIN FIXED(15);     /* INDICATOR FOR RESULT      */
  DCL 1 UDF_SCRATCHPAD,                 /* SCRATCHPAD                */
       3 UDF_SPAD_LEN    BIN FIXED(31),
       3 UDF_SPAD_TEXT   CHAR(100),
        5 CUMULI         BIN FLOAT(53);
  %INCLUDE UDFINFO;                          /* DBINFO               */
  DCL (LENGTH,ADDR)   BUILTIN;
  CUMULI=CUMULI+UDF_PARM1;
  UDF_RESULT=CUMULI;
 END CUMULF;
```

DATEUDF – PL/I source code

```
* PROCESS SYSTEM(MVS);
 DATEUDF: PROC(UDF_PARM1, UDF_PARM2, UDF_RESULT,
               UDF_IND1,  UDF_INDR,
               UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
               UDF_DIAG_MSG, UDF_SCRATCHPAD,
               UDF_CALL_TYPE, UDF_DBINFO)
```

```
         OPTIONS(MAIN NOEXECOPS REENTRANT);
/*********************************************************************/
/*    UDF   : DATEUDF                                                */
/*    INPUT : UDF_PARM1    CHAR(10)                                  */
/*          : UDF_PARM2    VARCHAR(1)                                */
/*    OUTPUT: UDF_RESULT   VARCHAR(20)                               */
/*********************************************************************/
 DCL UDF_PARM1      CHAR(10);          /* INPUT PARAMETER           */
 DCL UDF_PARM2      CHAR(1) VAR;       /* INPUT PARAMETER           */
 DCL UDF_RESULT     CHAR(20) VAR;      /* RESULT PARAMETER          */
 DCL UDF_IND1       BIN FIXED(15);     /* INDICATOR FOR INPUT PARM  */
 DCL UDF_INDR       BIN FIXED(15);     /* INDICATOR FOR RESULT      */
 DCL 1 UDF_SCRATCHPAD,                 /* SCRATCHPAD                */
      3 UDF_SPAD_LEN    BIN FIXED(31),
      3 UDF_SPAD_TEXT   CHAR(100);
 EXEC SQL INCLUDE UDFINFO;                  /* DBINFO               */
 DCL (ADDR,LENGTH,SUBSTR,NULL)   BUILTIN;
 EXEC SQL INCLUDE SQLCA;
 /* ********************************************************** */
 /* M:     RETURNS FULL ENGLISH NAME OF THE MONTH.             */
 /* DATEUDF(DATE('2000-08-18'),'W')   ->  'AUGUST'             */
 /* ********************************************************** */
 IF UDF_PARM2='M' THEN DO;
    EXEC SQL SELECT
         CASE MONTH(DATE(:UDF_PARM1))
          WHEN (1)  THEN 'J'||LCASE('ANUARY')
          WHEN (2)  THEN 'F'||LCASE('EBRUARY')
          WHEN (3)  THEN 'M'||LCASE('ARCH')
          WHEN (4)  THEN 'A'||LCASE('PRIL')
          WHEN (5)  THEN 'M'||LCASE('AY')
          WHEN (6)  THEN 'J'||LCASE('UNE')
          WHEN (7)  THEN 'J'||LCASE('ULY')
          WHEN (8)  THEN 'A'||LCASE('UGUST')
          WHEN (9)  THEN 'S'||LCASE('EPTEMBER')
          WHEN (10) THEN 'O'||LCASE('CTOBER')
          WHEN (11) THEN 'N'||LCASE('OVEMBER')
          WHEN (12) THEN 'D'||LCASE('ECEMBER')
         END
         INTO :UDF_RESULT
    FROM SYSIBM.SYSDUMMY1 WITH UR;
 END;
 /* ********************************************************** */
 /* W:     RETURNS FULL ENGLISH NAME FOR THE DAY OF THE WEEK.  */
 /* DATEUDF(DATE('2000-08-18'),'W')   ->  'FRIDAY'             */
 /* ********************************************************** */
 IF UDF_PARM2='W' THEN DO;
    EXEC SQL SELECT
          CASE (DAYOFWEEK(:UDF_PARM1))
           WHEN 1 THEN 'S'||LCASE('UNDAY')
           WHEN 2 THEN 'M'||LCASE('ONDAY')
           WHEN 3 THEN 'T'||LCASE('UESDAY')
```

```
          WHEN 4 THEN 'W'||LCASE('EDNESDAY')
          WHEN 5 THEN 'T'||LCASE('HURSDAY')
          WHEN 6 THEN 'F'||LCASE('RIDAY')
          WHEN 7 THEN 'S'||LCASE('ATURDAY')
        END
        INTO :UDF_RESULT
   FROM SYSIBM.SYSDUMMY1 WITH UR;
END;
/* ********************************************************** */
/* N:     RETURNS DATE IN THE DEFAULT FORMAT 'YYYY-MM-DD'.    */
/* DATEUDF(DATE('2000-08-18'),'N')   ->      '2000-08-18'     */
/* ********************************************************** */
IF UDF_PARM2='N' THEN DO;
   EXEC SQL SELECT DATE(:UDF_PARM1)
        INTO :UDF_RESULT
   FROM SYSIBM.SYSDUMMY1 WITH UR;
END;
/* ********************************************************** */
/* U:     RETURNS DATE IN THE FORMAT       'MM-DD-YYYY'.      */
/* DATEUDF(DATE('2000-08-18'),'U')   ->    '08-18-2000'       */
/* ********************************************************** */
IF UDF_PARM2='U' THEN DO;
   EXEC SQL SELECT CHAR(DATE(:UDF_PARM1),USA)
        INTO :UDF_RESULT
   FROM SYSIBM.SYSDUMMY1 WITH UR;
END;
/* ********************************************************** */
/* E:     RETURNS DATE IN THE FORMAT       'DD.MM.YYYY'.      */
/* DATEUDF(DATE('2000-08-18'),'E')   ->    '18.08.2000'       */
/* ********************************************************** */
IF UDF_PARM2='E' THEN DO;
   EXEC SQL SELECT CHAR(DATE(:UDF_PARM1),EUR)
        INTO :UDF_RESULT
   FROM SYSIBM.SYSDUMMY1 WITH UR;
END;
/* ********************************************************** */
/* S:     RETURNS DATE IN THE FORMAT       'DD MON YYYY'.     */
/* DATEUDF(DATE('2000-08-18'),'S')   ->    '18 AUG 2000'      */
/* ********************************************************** */
IF UDF_PARM2='S' THEN DO;
   EXEC SQL SELECT STRIP(CHAR(DAY(:UDF_PARM1)))
         || ' ' ||
         CASE MONTH(DATE(:UDF_PARM1))
          WHEN (1)  THEN 'J'||LCASE('AN')
          WHEN (2)  THEN 'F'||LCASE('EB')
          WHEN (3)  THEN 'M'||LCASE('AR')
          WHEN (4)  THEN 'A'||LCASE('PR')
          WHEN (5)  THEN 'M'||LCASE('AY')
          WHEN (6)  THEN 'J'||LCASE('UN')
          WHEN (7)  THEN 'J'||LCASE('UL')
          WHEN (8)  THEN 'A'||LCASE('UG')
```

```
          WHEN (9)  THEN 'S'||LCASE('EP')
          WHEN (10) THEN 'O'||LCASE('CT')
          WHEN (11) THEN 'N'||LCASE('OV')
          WHEN (12) THEN 'D'||LCASE('EC')
        END || ' ' ||
        STRIP(CHAR(YEAR(:UDF_PARM1)))
        INTO :UDF_RESULT
   FROM SYSIBM.SYSDUMMY1 WITH UR;
END;
/* ********************************************************* */
/* L: RETURNS DATE IN THE FORMAT        'DD MONTHNAME YYYY'.  */
/* DATEUDF(DATE('2000-08-18'),'S')   -> '18 AUGUST 2000'      */
/* ********************************************************* */
IF UDF_PARM2='L' THEN DO;
   EXEC SQL SELECT STRIP(CHAR(DAY(:UDF_PARM1)))
        || ' ' ||
        CASE MONTH(DATE(:UDF_PARM1))
           WHEN (1)  THEN 'J'||LCASE('ANUARY')
           WHEN (2)  THEN 'F'||LCASE('EBRUARY')
           WHEN (3)  THEN 'M'||LCASE('ARCH')
           WHEN (4)  THEN 'A'||LCASE('PRIL')
           WHEN (5)  THEN 'M'||LCASE('AY')
           WHEN (6)  THEN 'J'||LCASE('UNE')
           WHEN (7)  THEN 'J'||LCASE('ULY')
           WHEN (8)  THEN 'A'||LCASE('UGUST')
           WHEN (9)  THEN 'S'||LCASE('EPTEMBER')
           WHEN (10) THEN 'O'||LCASE('CTOBER')
           WHEN (11) THEN 'N'||LCASE('OVEMBER')
           WHEN (12) THEN 'D'||LCASE('ECEMBER')
        END || ' ' ||
        STRIP(CHAR(YEAR(:UDF_PARM1)))
        INTO :UDF_RESULT
   FROM SYSIBM.SYSDUMMY1 WITH UR;
END;
/* ********************************************************* */
/* J: RETURNS DATE IN THE FORMAT        'YYYYDDD'.           */
/* DATEUDF(DATE('2000-08-18'),'S')   -> '2000231'      '      */
/* ********************************************************* */
IF UDF_PARM2='J' THEN DO;
   EXEC SQL
   SELECT STRIP(CHAR(YEAR(:UDF_PARM1))) ||
          CHAR(DAYS(:UDF_PARM1) -
          DAYS(DATE(STRIP(CHAR(YEAR(:UDF_PARM1)))
          ||'-01-01')) + 1)
   INTO :UDF_RESULT
   FROM SYSIBM.SYSDUMMY1 WITH UR;
END;
/* ********************************************************* */
/* D: RETURNS NUMBER OF DAYS, INCLUDING THE CURRENT DAY      */
/* DATEUDF(DATE('2000-08-18'),'S')   -> '231'          '      */
/* ********************************************************* */
```

```
  IF UDF_PARM2='D' THEN DO;
     EXEC SQL
     SELECT CHAR(DAYS(:UDF_PARM1) -
            DAYS(DATE(STRIP(CHAR(YEAR(:UDF_PARM1)))
            ||'-Ø1-Ø1')) + 1)
     INTO :UDF_RESULT
     FROM SYSIBM.SYSDUMMY1 WITH UR;
  END;
 END DATEUDF;
```

DATEUDFD – PL/I source code

```
* PROCESS SYSTEM(MVS);
 DATEUDF: PROC(UDF_PARM1, UDF_RESULT,
               UDF_IND1,  UDF_INDR,
               UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
               UDF_DIAG_MSG, UDF_SCRATCHPAD,
               UDF_CALL_TYPE, UDF_DBINFO)
          OPTIONS(MAIN NOEXECOPS REENTRANT);
 /*********************************************************************/
 /*    UDF   : DATEUDF                                                */
 /*    INPUT : UDF_PARM1    CHAR(1Ø)                                  */
 /*    OUTPUT: UDF_RESULT   VARCHAR(1Ø)                               */
 /*********************************************************************/
  DCL UDF_PARM1      CHAR(1Ø);          /* INPUT PARAMETER           */
  DCL UDF_RESULT     CHAR(1Ø) VAR;      /* RESULT PARAMETER          */
  DCL UDF_IND1       BIN FIXED(15);     /* INDICATOR FOR INPUT PARM  */
  DCL UDF_INDR       BIN FIXED(15);     /* INDICATOR FOR RESULT      */
  DCL 1 UDF_SCRATCHPAD,                 /* SCRATCHPAD                */
       3 UDF_SPAD_LEN    BIN FIXED(31),
       3 UDF_SPAD_TEXT   CHAR(1ØØ);
  EXEC SQL INCLUDE UDFINFO;                  /* DBINFO               */
  DCL (ADDR,LENGTH,SUBSTR,NULL)   BUILTIN;
  EXEC SQL INCLUDE SQLCA;

  /* ********************************************************** */
  /* RETURNS DATE IN THE DEFAULT FORMAT 'YYYY-MM-DD'.           */
  /* DATEUDF(DATE('2ØØØ-Ø8-18'))   ->   '2ØØØ-Ø8-18'            */
  /* ********************************************************** */
  EXEC SQL SELECT DATE(:UDF_PARM1)
       INTO :UDF_RESULT
  FROM SYSIBM.SYSDUMMY1 WITH UR;
 END DATEUDF;
```

REVERSE – PL/I source code

```
* PROCESS SYSTEM(MVS);
 REVERSE: PROC(UDF_PARM1, UDF_RESULT,
               UDF_IND1,  UDF_INDR,
               UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
```

```
             UDF_DIAG_MSG, UDF_SCRATCHPAD,
             UDF_CALL_TYPE, UDF_DBINFO)
         OPTIONS(MAIN NOEXECOPS REENTRANT);
 /*********************************************************************/
 /*    UDF   : REVERSE                                                */
 /*    INPUT : UDF_PARM1    CHAR      INPUT PARAMETER                 */
 /*    OUTPUT: UDF_RESULT   CHAR      REVERSE INPUT PARAMETER         */
 /*********************************************************************/
  DCL UDF_PARM1      CHAR(4Ø46) VAR;    /* INPUT PARAMETER           */
  DCL UDF_RESULT     CHAR(4Ø46) VAR;    /* RESULT PARAMETER          */
  DCL UDF_IND1       BIN FIXED(15);     /* INDICATOR FOR INPUT PARM  */
  DCL UDF_INDR       BIN FIXED(15);     /* INDICATOR FOR RESULT      */
  DCL 1 UDF_SCRATCHPAD,                 /* SCRATCHPAD                */
       3 UDF_SPAD_LEN    BIN FIXED(31),
       3 UDF_SPAD_TEXT   CHAR(1ØØ);
  %INCLUDE UDFINFO;                     /* DBINFO                    */
  DCL (LENGTH,SUBSTR)   BUILTIN;
  DCL (IC,START)        BIN FIXED(31);
  UDF_RESULT='';
  START=LENGTH(UDF_PARM1);
  DO IC=START TO 1 BY -1;
     UDF_RESULT=UDF_RESULT || SUBSTR(UDF_PARM1,IC,1);
  END;
 END REVERSE;
```

TIMEUDF – PL/I source code

```
* PROCESS SYSTEM(MVS);
 TIMEUDF: PROC(UDF_PARM1, UDF_PARM2, UDF_RESULT,
             UDF_IND1,  UDF_INDR,
             UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
             UDF_DIAG_MSG, UDF_SCRATCHPAD,
             UDF_CALL_TYPE, UDF_DBINFO)
         OPTIONS(MAIN NOEXECOPS REENTRANT);
 /*********************************************************************/
 /*    UDF   : TIMEUDF                                                */
 /*    INPUT : UDF_PARM1    CHAR(8)                                   */
 /*          : UDF_PARM2    VARCHAR(1)                                */
 /*    OUTPUT: UDF_RESULT   VARCHAR(1Ø)                               */
 /*********************************************************************/
  DCL UDF_PARM1      CHAR(8);           /* INPUT PARAMETER           */
  DCL UDF_PARM2      CHAR(1) VAR;       /* INPUT PARAMETER           */
  DCL UDF_RESULT     CHAR(1Ø) VAR;      /* RESULT PARAMETER          */
  DCL UDF_IND1       BIN FIXED(15);     /* INDICATOR FOR INPUT PARM  */
  DCL UDF_INDR       BIN FIXED(15);     /* INDICATOR FOR RESULT      */
  DCL 1 UDF_SCRATCHPAD,                 /* SCRATCHPAD                */
       3 UDF_SPAD_LEN    BIN FIXED(31),
       3 UDF_SPAD_TEXT   CHAR(1ØØ);
  EXEC SQL INCLUDE UDFINFO;                /* DBINFO                 */
  DCL (ADDR,LENGTH,SUBSTR,NULL)   BUILTIN;
  EXEC SQL INCLUDE SQLCA;
```

```
/* ********************************************************** */
/* N:     RETURNS TIME IN THE DEFAULT FORMAT 'HH:MM:SS'.      */
/* TIMEUDF(TIME('17:03:00','N')      ->      '17:03:00'       */
/* ********************************************************** */
IF UDF_PARM2='N' THEN DO;
   EXEC SQL SELECT TRANSLATE(CHAR(TIME(:UDF_PARM1)),':','.')
        INTO :UDF_RESULT
   FROM SYSIBM.SYSDUMMY1 WITH UR;
END;
/* ********************************************************** */
/* M: RETURNS NUMBER OF MINUTES SINCE MIDNIGHT IN FORM 'MMMM' */
/* TIMEUDF(TIME('17:03:00','M')      ->      '1023'           */
/* ********************************************************** */
IF UDF_PARM2='M' THEN DO;
   EXEC SQL SELECT STRIP(CHAR(HOUR(TIME(:UDF_PARM1))*60 +
                   MINUTE(TIME(:UDF_PARM1))))
        INTO :UDF_RESULT
   FROM SYSIBM.SYSDUMMY1 WITH UR;
END;
/* ********************************************************** */
/* S: RETURNS NUMBER OF SECONDS SINCE MIDNIGHT IN FORM 'SSSSS' */
/* TIMEUDF(TIME('17:03:00','S')      ->      '61380'          */
/* ********************************************************** */
IF UDF_PARM2='S' THEN DO;
   EXEC SQL SELECT CHAR((HOUR(TIME(:UDF_PARM1))*60 +
                   MINUTE(TIME(:UDF_PARM1)))*60 +
                   SECOND(TIME(:UDF_PARM1)))
        INTO :UDF_RESULT
   FROM SYSIBM.SYSDUMMY1 WITH UR;
END;
/* ********************************************************** */
/* D: RETURNS TIME IN THE FORMAT 'HHMMSS'                     */
/* TIMEUDF(TIME('17:03:00','D')      ->      '170300'         */
/* ********************************************************** */
IF UDF_PARM2='D' THEN DO;
   EXEC SQL SELECT SUBSTR(CHAR(TIME(:UDF_PARM1)),1,2) CONCAT
                   SUBSTR(CHAR(TIME(:UDF_PARM1)),4,2) CONCAT
                   SUBSTR(CHAR(TIME(:UDF_PARM1)),7,2)
        INTO :UDF_RESULT
   FROM SYSIBM.SYSDUMMY1 WITH UR;
END;
/* ********************************************************** */
/* C: RETURNS TIME IN THE FORMAT 'HH:MMam/pm'                 */
/* TIMEUDF(TIME('17:03:00','C')      ->      '17:03pm'        */
/* ********************************************************** */
IF UDF_PARM2='C' THEN DO;
   EXEC SQL SELECT
   CASE
     WHEN HOUR(TIME(:UDF_PARM1)) = 12
      THEN '12:'||SUBSTR(CHAR(TIME(:UDF_PARM1)),4,2) || LCASE('PM')
     WHEN HOUR(TIME(:UDF_PARM1)) > 12
```

```
        THEN STRIP(CHAR(HOUR(TIME(:UDF_PARM1)) - 12)) || ':' ||
             SUBSTR(CHAR(TIME(:UDF_PARM1)),4,2) || LCASE('PM')
        ELSE STRIP(CHAR(HOUR(TIME(:UDF_PARM1)))) || ':' ||
             SUBSTR(CHAR(TIME(:UDF_PARM1)),4,2) || LCASE('AM')
     END
     INTO :UDF_RESULT
     FROM SYSIBM.SYSDUMMY1 WITH UR;
  END;
 END TIMEUDF;
```

TIMEUDFD – PL/I source code

```
* PROCESS SYSTEM(MVS);
 TIMEUDF: PROC(UDF_PARM1, UDF_RESULT,
               UDF_IND1,  UDF_INDR,
               UDF_SQLSTATE, UDF_NAME, UDF_SPEC_NAME,
               UDF_DIAG_MSG, UDF_SCRATCHPAD,
               UDF_CALL_TYPE, UDF_DBINFO)
          OPTIONS(MAIN NOEXECOPS REENTRANT);
 /*********************************************************************/
 /*    UDF   : TIMEUDF                                                */
 /*    INPUT : UDF_PARM1    CHAR(8)                                    */
 /*    OUTPUT: UDF_RESULT   VARCHAR(10)                                */
 /*********************************************************************/
  DCL UDF_PARM1      CHAR(8);           /* INPUT PARAMETER           */
  DCL UDF_RESULT     CHAR(10) VAR;      /* RESULT PARAMETER          */
  DCL UDF_IND1       BIN FIXED(15);     /* INDICATOR FOR INPUT PARM  */
  DCL UDF_INDR       BIN FIXED(15);     /* INDICATOR FOR RESULT      */
  DCL 1 UDF_SCRATCHPAD,                 /* SCRATCHPAD                */
       3 UDF_SPAD_LEN    BIN FIXED(31),
       3 UDF_SPAD_TEXT   CHAR(100);
  EXEC SQL INCLUDE UDFINFO;                 /* DBINFO                */
  DCL (ADDR,LENGTH,SUBSTR,NULL)   BUILTIN;
  EXEC SQL INCLUDE SQLCA;
  /* ********************************************************* */
  /* RETURNS TIME IN THE DEFAULT FORMAT 'HH.MM.SS'.            */
  /* TIMEUDF(TIME('17:03:00')      ->   '17.03.00'             */
  /* ********************************************************* */
  EXEC SQL SELECT TIME(:UDF_PARM1)
       INTO :UDF_RESULT
  FROM SYSIBM.SYSDUMMY1 WITH UR;
 END TIMEUDF;
```

UDBINFO – include udbinfo declaration from SYSLIB

```
DCL UDF_SQLSTATE   CHAR(5);           /* SQLSTATE RETURNED TO DB2  */
DCL UDF_NAME       CHAR(27) VAR;      /* QUALIFIED FUNCTION NAME   */
DCL UDF_SPEC_NAME  CHAR(18) VAR;      /* SPECIFIC FUNCTION NAME    */
DCL UDF_DIAG_MSG   CHAR(70) VAR;      /* DIAGNOSTIC STRING         */
DCL UDF_CALL_TYPE  BIN FIXED(31);         /* CALL TYPE             */
```

```
DCL 1 UDF_DBINFO,                          /* DBINFO               */
     3 UDF_DBINFO_LLEN BIN FIXED(15),      /* LOCATION LENGTH      */
     3 UDF_DBINFO_LOC  CHAR(128),          /* LOCATION NAME        */
     3 UDF_DBINFO_ALEN BIN FIXED(15),      /* AUTH ID LENGTH       */
     3 UDF_DBINFO_AUTH CHAR(128),          /* AUTHORIZATION ID     */
     3 UDF_DBINFO_CCSID CHAR(48),          /* CCSIDS FOR DB2 OS/390 */
      5 UDF_DBINFO_ESBCS BIN FIXED(31),    /* EBCDIC SBCS CCSID    */
      5 UDF_DBINFO_EMIXED BIN FIXED(31),   /* EBCDIC MIXED CCSID   */
      5 UDF_DBINFO_EDBCS BIN FIXED(31),    /* EBCDIC DBCS CCSID    */
      5 UDF_DBINFO_ASBCS BIN FIXED(31),    /* ASCII SBCS CCSID     */
      5 UDF_DBINFO_AMIXED BIN FIXED(31),   /* ASCII MIXED CCSID    */
      5 UDF_DBINFO_ADBCS BIN FIXED(31),    /* ASCII DBCS CCSID     */
      5 UDF_DBINFO_RESERV1 CHAR(20),       /* RESERVED             */
     3 UDF_DBINFO_QLEN BIN FIXED(15),      /* QUALIFIER LENGTH     */
     3 UDF_DBINFO_QUALIF CHAR(128),        /* QUALIFIER NAME       */
     3 UDF_DBINFO_TLEN BIN FIXED(15),      /* TABLE LENGTH         */
     3 UDF_DBINFO_TABLE  CHAR(128),        /* TABLE NAME           */
     3 UDF_DBINFO_CLEN BIN FIXED(15),      /* COLUMN LENGTH        */
     3 UDF_DBINFO_COLUMN CHAR(128),        /* COLUMN NAME          */
     3 UDF_DBINFO_RELVER CHAR(8),          /* DB2 RELEASE LEVEL    */
     3 UDF_DBINFO_PLATFORM BIN FIXED(31), /* DATABASE PLATFORM    */
     3 UDF_DBINFO_NUMTFCOL BIN FIXED(15), /* # OF TF COLS USED    */
     3 UDF_DBINFO_RESERV1 CHAR(24),        /* RESERVED             */
     3 UDF_DBINFO_TFCOLUMN PTR,            /* -> TABLE FUN COL LIST */
     3 UDF_DBINFO_RESERV2 CHAR(24);        /* RESERVED             */
```

SAMPLE JCL – precompile, link, and bind the DATEUDF function.

```
//SYSADMF  JOB (#ACCOUNT#),'UDF',
//       CLASS=A,MSGLEVEL=(1,1),MSGCLASS=X,
//       NOTIFY=SYSADM,TIME=30,COND=(9,LT),
//       USER=SYSADM
//PRECOM EXEC PGM=DSNHPC,PARM='HOST(PLI),S',
//          REGION=4096K
//DBRMLIB DD DSN=DB2.DBRMLIB(DATEUDF),DISP=SHR
//STEPLIB DD DISP=SHR,DSN=SYS1.DSN610.SDSNEXIT
//        DD  DISP=SHR,DSN=DSN610.SDSNLOAD
//SYSLIB  DD DSN=SKUPNI.DB2.DCL,DISP=SHR
//SYSCIN  DD DSN=&DSNHOUT,DISP=(MOD,PASS),UNIT=3390,
//        SPACE=(800,(500,500))
//SYSPRINT DD SYSOUT=X
//SYSTERM DD SYSOUT=X
//SYSUDUMP DD SYSOUT=X
//SYSUT1 DD SPACE=(800,(500,500),,,ROUND),UNIT=3390
//SYSIN DD DSN=SYSADM.PLI.FUNCTION(DATEUDF),DISP=SHR
//PLI    EXEC PGM=IEL0AA,REGION=4096K,
//      PARM='OBJECT,NODECK,INCLUDE'
//STEPLIB DD DISP=SHR,
// DSN=CEE.SCEERUN
// DD DSN=CEE.SCEELKED,DISP=SHR
```

```
// DD DSN=IEL.V1R1M1.SIELCOMP,DISP=SHR
// DD DSN=SKUPNI.DB2.DCL,DISP=SHR
//SYSPRINT DD SYSOUT=X
//SYSIN DD DSN=&DSNHOUT,DISP=(OLD,DELETE)
//SYSLIN DD DSN=SKUPNI.OBJ(DATEUDF),DISP=SHR
//SYSLIB DD DSN=SKUPNI.DB2.DCL,DISP=SHR
//SYSUT1 DD UNIT=SYSALLDA,DCB=BLKSIZE=1024,
//          SPACE=(1024,(300,60),,CONTIG)
//LINKER EXEC PGM=IEWL,REGION=4096K,
//    PARM='LIST,XREF,RENT,AMODE(31),CASE=MIXED,RMODE=ANY'
//SYSLIB DD DSN=SKUPNI.OBJ,DISP=SHR
//         DD DSN=DSN610.SDSNLOAD,DISP=SHR
// DD DSN=DSN610.RUNLIB.LOAD,DISP=SHR
// DD DSN=CEE.SCEERUN,DISP=SHR
// DD DSN=CEE.SCEELKED,DISP=SHR
//SYSLMOD DD DISP=SHR,
// DSN=DSN610.RUNLIB.LOAD(DATEUDF)
//SYSUT1 DD UNIT=SYSALLDA,DCB=BLKSIZE=1024,
//        SPACE=(1024,(200,20))
//SYSPRINT DD SYSOUT=X
//SYSLIN DD *
       INCLUDE SYSLIB(DATEUDF)
       INCLUDE SYSLIB(DSNRLI)
/*
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB DD DISP=SHR,DSN=SYS1.DSN610.SDSNEXIT
//          DD DISP=SHR,DSN=DSN610.SDSNLOAD
//DBRMLIB DD DISP=SHR,
//       DSN=DB2.DBRMLIB(DATEUDF)
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=X
//SYSPRINT DD SYSOUT=X
//SYSTSIN DD *
      DSN SYSTEM(DSNN)
         BIND PACKAGE(DATEUDF)   -
         MEMBER(DATEUDF)   -
         LIBRARY('DB2.DBRMLIB') -
         ACT(REP) ISO(CS) -
         VALIDATE(BIND)
       END
/*
```

The user-defined function works when WLM is started. Below are some WLM commands and their description:

- D WLM – current state-policy of WLM.
- D WLM,APPLENV=* – check application environment.

- D WLM,APPLENV=DSNNWLM1 – status of appl.environment dsnnwlm1.
- V WLM,APPLENV=DSNNWLM1,QUIESCE – request for termination.
- V WLM,APPLENV=DSNNWLM1,RESUME – restart an application dsnnwlm1.
- V WLM,APPLENV=DSNNWLM1,REFRESH – refresh LE for new load module.
- F WLM,MODE=COMPAT – compat mode.
- F WLM,MODE=GOAL – goal mode.

Bernard Zver (Slovenia)

IBM has announced Version 7.1 of its Content Manager OnDemand for Multiplatforms enterprise report manager, positioned as an alternative to microfiche and hardcopy storage and retrieval.

It enables sites to organize and store any printed output, as well as e-mails and image documents. It supports leading ERP and CRM applications and provides a platform for implementing electronic bill presentment and payment systems.

The new version now supports DB2 UDB V7.1, in addition to Oracle8*i* and SQL Server 2000. Platform support is available for AIX Version 4.3, HP-UX 11, Linux, Solaris 8, and Windows 2000 along with Adobe 4.0.

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

* * *

Candle has announced that the company will provide day-one solution support for CICS Transaction Server for z/OS Version 2 (CICS TS V2) and DB2 Universal Database Server for OS/390 and z/OS, Version 7.

The company also reaffirmed its commitment to the IBM's workload software pricing structure for its CICS, DB2, and IMS solutions.

Candle's day-one support for z/OS Version 1.1 includes OMEGAMON II availability and performance monitors for MVS, CICS, DB2, IMS, DBCTL, SMS, and VTAM, OMEGAVIEW for 3270, OMEGAVIEW II for the Enterprise, OMEGACENTER Gateway, AF/OPERATOR, and AF/REMOTE.

Day-one support for CICS TS V2 DB2 UDB Server for OS/390 and z/OS is covered by OMEGAMON II for CICS and DB2 and CCC for CICS and DB2plex, respectively.

For further information contact:
Candle, 201 N Douglas St, El Segundo, CA 90245, USA.
Tel: (310) 535 3600.
URL: http://www.candle.com/news_events/press_releases/mainframe/dayone_zOS_033001.html.

* * *

Landmark Systems is shipping its TMON for Unix System Services monitor, which is designed to make it possible to change parameters and abort processes without leaving the monitoring console.

It identifies problems, bottlenecks, and availability issues in OS/390 USS resources and enables immediate action to be taken to correct them.

From one workstation, users can access DB2, CICS, IMS, MVS, TCP/IP, or VTAM data to monitor performance.

It monitors key activity components such as global I/O buffers, HFS data sets, processes, threads, and TCP/IP stacks. An exception monitor automates the problem detection process by creating alerts whenever pre-defined problems occur.

For further information contact:
Landmark Systems, 12700 Sunrise Valley Drive, Reston, VA 20191-5804, USA.
Tel: (703) 464 1300.
URL: http://www.landmark.com/products/tmonuss.shtml.