118

DB2 update

August 2002

In this issue

DB2 Update

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

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

Subscriptions and back-issues
A year's subscription to *DB2 Update*, comprising twelve monthly issues, costs £255.00 in the UK; $380.00 in the USA and Canada; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1999 issue, are available separately to subscribers for £22.50 ($33.75) each including postage.

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

Editor
Trevor Eddolls

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

Temporary tables – declared and created

DB2 for OS/390 has provided the ability to create temporary tables since Version 5. But the initial functionality was practical only in certain circumstances because of some inherent limitations. This first type of temporary table is known as a created temporary table. IBM's support for temporary tables has expanded as of Version 7 – now DB2 offers two different types of temporary tables: created and declared.

WHY USE TEMPORARY TABLES?

But before we investigate these two types of temporary tables, let's first address why anyone would want or need to use a temporary table in the first place.

One potential usage of temporary tables is to store intermediate SQL results. Consider, for example, the results of one query having to be used in a subsequent query. Instead of rerunning the first query (or combining it with the subsequent query), the results of the first query can be stored in a temporary table. Then the temporary table can be joined into the second query without incurring the overhead of rerunning the first query. This is especially useful if the first query is particularly complex or inefficient.

Or what about result sets that need to be returned more than once by the same program? Consider this scenario: a complex multi-table join is coded, the running of which consumes a lot of resources. Furthermore, that join statement needs to be run three times during the course of the program. Instead of running the join three times you can run it once and populate a temporary table with the results. The next two times you can simply read the temporary table, which might be more efficient than re-executing the complex, resource-consuming, multi-table join.

Temporary tables are also useful for enabling non-relational data to be processed using SQL. For example, you can create a global temporary table that is populated with IMS data (or any other non-relational data source) by a program. Then, during the course of that program, the temporary table containing the IMS data can be accessed by SQL statements and even joined to other DB2 tables. The same could be done for data from a flat file, VSAM, IDMS, or any other non-relational data.

Another reason for IBM's inclusion of temporary table support in DB2 is to make conversion from other relational products easier. Microsoft SQL Server and Oracle both have supported temporary tables for quite some time now. Without such support in DB2, it was very difficult for developers to convert or port their Oracle or SQL Server applications to DB2. IBM alleviated this problem by enabling temporary table support in DB2.

Now let's examine the two types of temporary tables supported by DB2.

CREATED TEMPORARY TABLES

A created temporary table exists only as long as the process that uses it. Temporary tables are created using the CREATE GLOBAL TEMPORARY TABLE statement. When created, the schema for the table is stored in the DB2 system catalog (SYSIBM.SYSTABLES) just like any other table, but the TYPE column is set to 'G' to indicate a global temporary table. Created temporary tables are sometimes referred to as global temporary tables, but this is confusing since declared temporary tables are also referred to as global declared tables.

It is important to remember that a created global temporary table must be created using a DDL CREATE statement before it can be used in any program.

A created temporary table is instantiated when it is referenced in an OPEN, SELECT INTO, INSERT, or DELETE statement, not when it is created. Each application process that uses the temporary table creates a new instance of the table for its use. When using a created temporary table, keep the following in mind:

- Because they are not persistent, some typical database operations including locking, logging, and recovery do not apply to created temporary tables.
- Indexes cannot be created on created temporary tables so all access is by a complete table scan.
- Constraints cannot be created on created temporary tables.

- A null is the only default value permitted for columns of a created temporary table.
- Created temporary tables cannot be referenced by DB2 utilities.
- Created temporary tables cannot be specified as the object of an UPDATE statement.
- When deleting from a created temporary table, all rows must be deleted.
- Although views can be created on created temporary tables, the WITH CHECK OPTION cannot be specified.

Work file datasets are used to manage the data of created temporary tables. The work database (DSNDB07) is used as storage for processing SQL statements that require working storage – not just for created temporary tables. So if you are using created temporary tables be sure to examine the *DB2 Installation Guide* for tactics to estimate the disk storage required for temporary work files.

When a temporary work file result table is populated using an INSERT statement, it uses work file space. No other process can use the same work file space as that temporary work file table until the table goes away. The space is reclaimed when the application process commits or rolls back, or when it is deallocated, depending on which RELEASE option was used when the plan or package was bound. It is a good idea to keep the work files in a separate buffer pool to make it easier to monitor. IFCID 0311 in performance trace class 8 can be used to distinguish these tables from other uses of the work file.

DECLARED TEMPORARY TABLES

With Version 7 of DB2, IBM introduced declared temporary tables. Actually, to be more accurate, declared temporary tables were made available in the intermediate DB2 Version 6 refresh.

This new type of temporary table is different from a created temporary table and overcomes many of their limitations. The first significant difference between declared and created temporary tables is that declared temporary tables are specified using a DECLARE statement in an application program, and not using a DDL CREATE statement.

Because they are not persistent they do not have descriptions in the DB2 Catalog.

Additionally, declared temporary tables offer significant features and functionality not provided by created temporary tables. Consider:

- Declared temporary tables can have indexes and CHECK constraints defined on them.
- You can issue UPDATE statements and positioned DELETE statements against a declared temporary table.
- You can implicitly define the columns of a declared temporary table and use the result table from a SELECT.

So, declared temporary tables offer much more flexibility and functionality than created temporary tables. To 'create' an instance of a declared temporary table you must issue the DECLARE GLOBAL TEMPORARY TABLE statement inside an application program. That instance of the declared temporary table is known only to the process that issues the DECLARE statement. Multiple concurrent programs can be executing using the same declared temporary table name because each program will have its own copy of the temporary table.

But more work is required to use a declared temporary table than there is to use a created temporary table. Before you can declare temporary tables you must create a temporary database and table spaces for them to use. This is accomplished by specifying the AS TEMP clause on a CREATE DATABASE statement. Then, you must create segmented table spaces in the temporary database. Only one temporary database for declared temporary tables is permitted per DB2 subsystem.

When a DECLARE GLOBAL TEMPORARY TABLE statement is issued, DB2 will create an empty instance of the temporary table in the temporary table space. INSERT statements are used to populate the temporary table. Once inserted, the data can be accessed, modified, or deleted. When the program completes, DB2 will drop the instance of the temporary table. Also, be aware that users of temporary tables must have been granted USE authority on the temporary tablespace.

The following example shows a DECLARE statement that can be issued from an application program (assuming the temporary database and table spaces have been defined already):

```
DECLARE GLOBAL TEMPORARY TABLE TEMP_EMP
  (EMPNO      CHAR(6)     NOT NULL,
   FIRSTNME   VARCHAR(12) NOT NULL,
   MIDINIT    CHAR(1)     NOT NULL,
   LASTNAME   VARCHAR(15) NOT NULL,
   WORKDEPT   CHAR(3),
   PHONENO    CHAR(4)
  );
```

This creates a declared temporary table named TEMP_EMP. Additionally, you can use the LIKE clause to DECLARE a temporary table that uses the same schema definition as another currently existing table. You can use the INCLUDING IDENTITY COLUMN ATTRIBUTES clause to copy the IDENTITY columns as well. For example:

```
DECLARE GLOBAL TEMPORARY TABLE TEMP_PROJ
 LIKE DSN871Ø.PROJ
 INCLUDING IDENTITY
 ON COMMIT PRESERVE ROWS;
```

This example shows how to use the INCLUDING IDENTITY clause. However, the sample table DSN8710.PROJ does not use an IDENTITY column, so this statement would not work – it is shown as an example only.

Notice also the ON COMMIT PRESERVE ROWS clause in the previous example. The ON COMMIT clause specifies what action DB2 is to take with the data in the declared temporary table when the program issues a COMMIT statement. There are two options – PRESERVE or DELETE rows. Specifying PRESERVE ROWS indicates that the rows of the table are to be kept. Beware, though, that the PRESERVE ROWS option impacts thread reuse. You will not be able to reuse threads for any application process that contains, at its most recent COMMIT, an active declared temporary table defined using the PRESERVE ROWS option of the ON COMMIT clause. The other option, which is the default, is ON COMMIT DELETE ROWS. In that case all of the rows of the table are deleted as long as there are no cursors defined using WITH HOLD.

SCROLLABLE CURSORS

Scrollable cursors, another new feature of DB2 Version 7, require declared temporary tables. A scrollable cursor provides the ability to scroll forward and backward through the data once the cursor is open. This can be achieved using nothing but SQL – no host language code (eg COBOL, C) is required to facilitate a scrollable cursor in DB2 V7. A scrollable cursor makes navigating through SQL result sets much easier.

The data from a scrollable cursor is maintained in a declared temporary table. DB2 uses this mechanism to facilitate scrolling through data in multiple ways – forward, backward, or to a specific position.

So, keep in mind that, even if you do not choose to use temporary tables in your application programs, you may need to implement them to support scrollable cursors.

DECLARED TEMPORARY TABLE STORAGE

Before using declared temporary tables, the temporary database and temporary tablespaces must be defined to store the temporary data. For example:

```
CREATE DATABASE TEMPDB AS TEMP;
CREATE TABLESPACE TEMPTS
 IN TEMPDB
 SEGSIZE 4
 BUFFERPOOL BP7;
```

The tablespace is created as a temporary tablespace by virtue of its being in the temporary database.

The page size of the temporary tablespace must be large enough to hold the longest row in the declared temporary table. The size of a row in the declared temporary table might be considerably larger than the size of the row in the table for which the scrollable cursor is used. As with a regular table, the size of the row depends on the number of columns that are stored in the declared temporary table and the size of each column.

An in-depth discussion of calculating the storage requirements for declared temporary table tablespaces is provided in the *DB2 Installation Guide*. Be sure to refer to that manual before implementing declared temporary tables or any features that rely on declared temporary tables (eg scrollable cursors).

Keep in mind, too, that when there is more than one temporary tablespace defined to the DB2 subsystem, DB2 will select which temporary tablespaces it will use for scrollable cursor processing.

DECLARE OR CREATE?

With all of the limitations of created temporary tables why would anyone still want to use them instead of declared temporary tables?

Well, there are a few potential problems with declared temporary tables, too. First of all, the SYSPACKDEP catalog table will not show dependencies for declared temporary tables, but it will for created temporary tables. Secondly, some DBAs are wary of allowing database structures to be created by application programmers inside an application program. With limited DDL and database design knowledge it may not be wise to trust programmers to get the table structure correct. Furthermore, the additional management of the temporary database and table spaces can become an administrative burden.

So, created temporary tables are still useful – in the right situations. They should be considered primarily when no updating of temporary data is needed and access to the temporary data is purely sequential.

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

What is the Soundex function?

This article looks at 'Soundex' – what it is, its history and how it works, how DB2 incorporates it, how to use it with the DIFFERENCE function, and, finally, deployment considerations for using it.

WHAT IS SOUNDEX?

There has often been a need to search a list of words for a word whose spelling you are not quite sure of. This is especially true with a list of names. How do you spell SMITH? Could it be SMYTHE? Or perhaps you made a typo when entering the name and typed SIMTH instead, This is where Soundex comes in. Basically, Soundex is a way of comparing one character string with another, based on phonetics rather than on the actual characters that make up the character string.

THE HISTORY OF SOUNDEX AND HOW IT WORKS

The Soundex method was developed in the early part of the 20th Century in the USA. It converts a character string (of any length) into a code made up of four characters. The first character of the code is the first character of the word you are converting. The remaining three characters of the code are taken from the table below by replacing each letter of the word with the corresponding number (or 'no value') shown:

Letter in word	*Number in code*
B,F,P,V	1
C,G,J,K,Q,S,X,Z	2
D,T	3
L	4
M,N	5
R	6
H,Y,W	No value
A,E,I,O,U	No value

Note:

- If the word begins with a 'no value' character, then it is retained (see example *d* below).
- Consecutive letters which generate the same code number are represented by a single code number (see example *d* below).

(For a complete set of rules to use when using the Soundex method to find names, look at: http://www.nara.gov/genealogy/coding.html.)

So let's see how we would convert the word SMITH. Using the table above, work through each letter of Smith and see what it converts to:

Letter	*Converts to*
S	S
M	5
I	-
T	3
H	-

So SMITH converts to S53, but because the code has to be four characters long, we pad it with a zero at the end to give a code of S530.

An important point to bear in mind here is how to deal with the ‘no value’ letters. Should you go through the word disregarding them before doing any conversion, or should you just represent them as dashes (as in the above example)? A common example to illustrate this dilemma is to find the Soundex value of the word Ashcroft. Using the method shown above, Ashcroft maps to:

Word:	A	S	H	C	R	O	F	T
Code:	A	2	-	2	6	-	1	3

So the Soundex code for Ashcroft isA226 (remembering that the code is only four characters long). Now look what would have happened if we had eliminated the ‘no value’ letters before doing the conversion:

ASHCROFT would become ASCRFT, which would generate a code of:

Word:	A	S	C	R	F	T
Code:	A	2	2	6	1	3

The ‘S’ and ‘C’ have the same code number, so are represented by a single code number, so the code would be A261. Which method does DB2 use? See example *e* in the next section to see how DB2 handles this – it uses the first method! (ie Ashcroft is A226).

A word of caution when implementing Soundex comparisons – the comparison is only as good as what is input. Again, let’s use the word Smith as an example. Consider the following Soundex values:

```
Word            Soundex value
Smith           S53Ø
Smirh           S56Ø
Smiyh           S5ØØ
```

Just by typing ‘r’ and ‘y’ instead of ‘t’ (‘r’ and ‘y’ are the letters either side of ‘t’ on the keyboard) in the word Smith give very different Soundex code values.

DB2 AND SOUNDEX

UDB DB2 has, since V5.2, had a Soundex built-in function, which accepts as input a character string and returns a four-character Soundex representation of the input string.

Let’s look at some examples using DB2 (UDB 7.2 FP4 on Windows 2000):

```
    Query                                Soundex value
(a) >db2 values(soundex('Fred'))            F630
(b) >db2 values(soundex('Smith'))           S530
(c) >db2 values(soundex('Simth'))           S530
(d) >db2 values(soundex('Apple'))           A140
(e) >db2 values(soundex('Ashcroft'))        A226
```

Note:

- You can see that Smith and Simth have the same soundex value, which is good and bad. If you are looking for Smith then it will be found even if you enter Simth – unfortunately, Simth will also match with ‘sameday’ and ‘snotty’. So if you are looking for a Mr Smith, then you will also match on Mr Sameday etc.
- You can see in example *d* that, for the word Apple, the soundex value begins with the letter A. Also, the two Ps are treated as a single P, with the soundex value of 1.
- Example *e* shows how DB2 handles the conversion of the word Ashcroft, which shows that DB2 does *not* eliminate the ‘no value’ letters before doing the conversion.

THE DIFFERENCE FUNCTION

There is another built-in function which can be used in conjunction with the Soundex function and that is the DIFFERENCE function. This function accepts as input two character strings, and returns an integer value between 0 and 4, where 4 indicates the highest level of match. I think the values should be interpreted as relative to each other and not as absolute. What I mean by this is that 3 represents a better match than 2 but not as good a match as 4. I think this is the only way to really interpret this. An example is shown below:

```
    Query                                          Difference value
(e) >db2 values(difference('Smith','simth'))             4
(f) >db2 values(difference('Smith','simthe'))            4
(g) >db2 values(difference('Smith','sameday'))           4
```

```
(h)  >db2 values(difference('Smith','snotty'))            4
(i)  >db2 values(difference('Smith','smirh'))             3
```

Note:

- Both simth and smithe give a maximum difference value (4) when compared with Smith (as do the words sameday and snotty).
- If you are using the DIFFERENCE function, you cannot compare a character string with a soundex value – well, you can, but it will be a case of garbage in garbage out, as shown in:

  ```
  >db2 values(difference('Smith',soundex('smith')))
  ```

  returns a difference value of 2, but this value is as a result of doing a difference on 'Smith' and 'S530' (the soundex value of Smith).

You can combine the Soundex and DIFFERENCE functions to try to overcome some of the problems mentioned above. Create a small table (tab21) with the following values:

```
>db2 select * from tab21

ID          NAME
---------- ----------
1           Fred
2           Sue
3           Smith
4           Smithe
```

Now let's go back to our Smith/Smirh problem. Soundex returns different values for these words (S530/S560), so using a query such as:

```
>db2 "select * from tab21 where (soundex(name) = soundex('smirh'))
```

will not return any rows. If we now use the DIFFERENCE function as follows:

```
>db2 values(difference('Smith','smirh'))
```

it returns a value of 3.

So when searching for the word Smith, we could combine the two previous queries as follows:

```
>db2 "select * from tab21 where ((soundex(name) = soundex('smirh')) or
difference(name,'smirh') > 2)"
```

```
ID          NAME
---------- ----------
3           Smith
4           Smithe
```

This query will at least return the value for Smith. This works well for a four row table, but you would have to thoroughly test any query that runs against a multi-million row table!

WHY CAN'T I USE THE LIKE FUNCTION?

You can of course use the LIKE function to find words, but it will not find SMITH when you type in SIMTH%. The LIKE function has its uses, but in this particular situation of trying to find names, the Soundex function is, I feel, the better option.

DEPLOYMENT CONSIDERATIONS

The Soundex algorithm was developed for American sounding words. I guess it works for most English words, but have not tried it in, say, French or German (or even Japanese). If you have DB2 in these countries, you should perform similar tests to those shown above to determine the boundaries that you can search within.

If you are going to use Soundex comparisons, then it may be worth creating a new column with the Soundex value in it. This means that you can create an index on this column, and then in your application you can convert the name to its Soundex value before searching the table.

CONCLUSION

Soundex can be a very powerful weapon in your arsenal for searching for words where you are not too sure of the spelling. However, it requires thorough testing, tables need to be designed with this method in mind, and any SQL needs to be explained to eliminate undesirable access paths. If you have applications which do a lot of name searching, give it a try!

C Leonard
Freelance Consultant (UK)

New utility control statements in DB2 UDB for OS/390 Version 7

INTRODUCTION

In Version 7.0 of DB2 Universal Database for OS/390 or z/OS, IBM has moved all the database utilities like COPY, RECOVER, QUIESCE, etc into a separate suite. Some new features have been introduced that obviously are to help in the rapid development of DB2 utility jobs. A new utility called EXEC SQL could be used to create mapping tables for online REORGs. In this article, we look at three new utility control statements and discuss their features, usage, and drawbacks. These statements are coded in the SYSIN dataset of a job step executing DSNUPROC or DSNUTILB.

The LISTDEF utility control statements are useful in extracting catalog information and generating lists on which other utility processes can be run. The TEMPLATE statements provide a means of defining a model for naming and allocating datasets to be used by the standard utilities. The OPTIONS statement provides a means of previewing generated JCL, and specifying the actions to take in case of failure in a particular step. These control statements will be very effective in shops where rigid and well defined naming standards are in use. We will briefly examine each of these.

LISTDEF

The LISTDEF utility control statement is a pattern matching statement. It helps to generate a list of objects on which to run the utilities. The LISTDEF control statement consists of the keyword LISTDEF and:

- The mandatory name of the list (18 alphanumeric characters).
- A mandatory INCLUDE clause followed by one or more INCLUDE or EXCLUDE clauses. These clauses are processed in top down order. Hence, an EXCLUDE can omit an INCLUDEd clause, and *vice versa* – thus the order is important. Also, if we used a

PARTLEVEL clause in an INCLUDE, it can be EXCLUDEd only by another PARTLEVEL clause.

- The type of objects the list contains – it can be either TABLESPACES or INDEXSPACES only, irrespective of the object type used in the initial catalog look-up (see next item). If you specify an initial object type to be a table, it will use the associated tablespace in the list and so on.
- The object to be used for the initial catalog look-up – this can be either a DATABASE, a TABLESPACE, another LIST created by the LISTDEF command, a TABLE, an INDEXSPACE, or an INDEX.

A good feature of the LISTDEF control statement is the ability to specify wildcards or pattern matching using '*' and '%' for any string of characters and '_' and '?' for a single character. Other features of the syntax allow you to select indexspaces that have been created with or altered to have the copy option set to YES or NO. A very interesting feature is the ability to select all objects based on RI, which selects parent and child objects in both directions for a given table. Finally, there are options to select only the LOBs, or the BASE objects or ALL. The PARTLEVEL option allows operations on selected partitions only.

Some list definitions that are prohibited are the reference to catalog and directory objects and all inclusive lists like DATABASE *, DATABASE %, TABLESPACE *.*, TABLE %, etc. Some utilities like COPY and RECOVER can process a list without the object type, while others like the QUIESCE need to have the object type specified.

The LISTDEF control statements must be defined in the SYSIN dataset prior to the utility control statements that reference it. They may also be defined in datasets and allocated to a DDname, which may be referred to by the OPTIONS control statement using the LISTDEFDD clause. The default LISTDEFDD is SYSLISTD. If both SYSIN and LISTDEFDD refer to the same list, SYSIN takes precedence.

To execute the LISTDEF control statement, Select authorization is required on SYSTABLES, SYSINDEXES, and SYSTABLESPACES as well as the necessary authorization to execute the utility in which the list will be used. The LISTDEF utility control statement executes in the UTILINIT phase.

Some examples are provided below for the LISTDEF control statement.

All tablespaces in database EMPLOYEE:

```
LISTDEF  X  INCLUDE  TABLESPACES DATABASE EMPLOYEE
```

All tablespaces in database ACCOUNT:

```
LISTDEF ACCT INCLUDE TABLESPACES DATABASE ACCOUNT
```

All indexspaces that have been created with, or have been altered to, COPY YES in database SALARY:

```
LISTDEF  SALARY  INCLUDE INDEXSPACES  COPY YES DATABASE  SALARY
```

The table SALARY_TABLE is in a list called SALTBL; the type of object may be omitted here as the table object selected implies a tablespace:

```
LISTDEF SALTBL   INCLUDE  TABLE  USERID.SALARY_TABLE
```

The indexes like SALTBLX% are in a list called SALNDX; the type of object may be omitted here as the index object implies an indexspace:

```
LISTDEF SALNDX   INCLUDE  INDEX   USERID.SALTBLX%
```

Create a list, named EMPLIST, of all tablespaces and indexspaces in the EMPLOYEE database and exclude the tablespace CONTRACT and its indexes:

```
LISTDEF  EMPLIST    INCLUDE TABLESPACE  EMPLOYEE.*
                    INCLUDE INDEXSPACE  EMPLOYEE.*X%
                    EXCLUDE TABLESPACE  EMPLOYEE.CONTRACT
                    EXCLUDE INDEXSPACE  EMPLOYEE.CONTRACTX%
```

Having defined the above list, we can use it in the manner shown below:

```
QUIESCE   TABLESPACE  LIST  EMPLIST
```

(Remember that some utilities need to have the object type defined.)

Another way the above list can be used is:

```
COPY LIST EMPLIST
```

In the above example, where the EMPLIST list is used to copy objects of a database based on certain Include and Exclude conditions, it is necessary to define the datasets to copy the list of objects generated by the LISTDEF. The TEMPLATE utility control statement is a handy and powerful method of accomplishing this task with minimum effort and very few lines of coding.

TEMPLATE

The TEMPLATE utility control statement is very useful for allocating datasets, without using JCL DD cards. In its simplest form, it defines dataset naming standards or conventions. Optionally it can also define allocation parameters like size, device type, and other attributes. As it uses the MVS DYNALLOC macro, it is constrained by its limitations. This statement requires no privileges to execute it. However, the privileges will be checked when a utility refers a template. The TEMPLATE control statement executes in the UTILINIT phase.

The TEMPLATE utility control statement is defined using the keyword TEMPLATE, followed by an 8-character alphanumeric name, and other keywords or options that control the dataset definition. Almost all attributes of the DD statement may be defined using these options, which may be classified as common options, disk options, and tape options.

The common options specify UNIT, DSN, MODELDCB, BUFNO, DATACLAS, MGMTCLAS, STORCLAS, RETPD/EXPDL, VOLUMES, VOLCNT, GDGLIMIT, and DISP parameters.

The disk options specify SPACE, space allocation units (CYL, TRK, MB), and three new options – PCTPRIME, MAXPRIME, and NBRSECND, which are explained later.

The tape options specify UNCNT (number of devices to be allocated), STACK (defaults to NO – do not stack output datasets contiguously on same tape volume), JES3DD (DDname for JES3 processing), and TRTCH (track recording technique – COMP for compacted format, NOCOMP – for standard format, and a default of NONE).

The options similar to the JCL DD statement options have the same functionality.

In addition to these options, a well-defined set of symbolic variables are available, which cater to almost every conceivable naming convention. These symbolic variables are used in a manner similar to those used in JCL PROCs.

For example, using the above options we can define a TEMPLATE MYTMP as:

```
    TEMPLATE MYTMP DSN 'mytemp.gdg.version(+1)'
```

to create the next version of the GDG named above. The quotes are required for special characters like the (+1).

Another example, shown here:

```
    TEMPLATE  TEMPL2  DSN &DB..&TS..D2&JDATE..T&TIME..P&PART
```

is used to define a dataset with the database name as the first-level node followed by the tablespace name, julian date (YYYYDDD) and time (HHMMSS), and partition number. A symbolic for the VCAT name would have been handy here.

All the symbolics are self-explanatory. The &PART will substitute a five-digit part number. For non-partitioned objects, it will return five zeros (00000). This may not be acceptable to some database administrators because they would prefer to use 00001 for non-partitioned objects. There are more symbolic variables and they are classified into job, utility, object, and date and time variables. The utility symbolic variables are very interesting as they are highly suitable for image copy jobs.

For example:

- &ictype. or &ic. would substitute I, F, or C for incremental, full, or changelimit image copy.
- &locrem. or &lr. would substitute L for COPYDDN *ddname1* and R for RECOVERYDDN *ddname1*.
- &pribac. or &pb. would substitute P for COPYDDN *ddname2* and B for RECOVERYDDN *ddname2*.

For a discussion of all the symbolic variables and the options and their permissible values, refer to the *Utility Guide and Reference for DB2 for OS/390 Version 7*.

The three new options that relate to the space allocation, namely PCTPRIME, MAXPRIME, and NBRSECND, are briefly discussed here. The SPACE is defined as (primary,secondary) in units of CYL, TRK, or MB. There are three other options that control how the space is allocated, namely PCTPRIME, MAXPRIME, and NBRSECND. These take an integer value as the argument. The PCTPRIME specifies the percentage of the required space to be obtained as the primary quantity. The default is 100% and this is preferred to avoid secondary extents. The MAXPRIME specifies the maximum allowable primary space allocation in the units specified for the SPACE option and it overrides the primary value specified. It also defines the basis for the PCTPRIME calculation. The NBRSECND indicates how the remaining space will be divided after the allocation of the primary based on the above two options and defaults to 10.

If SPACE is specified, both primary and secondary must be specified along with the allocation unit. If SPACE is not specified, DB2 estimates the space required based on the utility type and does the allocation request. In case the estimated space cannot be allocated on one device unit as a primary allocation, the PCTPRIME option may be used to reduce the primary allocation.

For example, if DB2 estimates the total space for a dataset used by a utility to be 3000 cylinders, and suppose that we can allocate only a maximum primary of 2000 cylinders, we can code the following:

```
TEMPLATE XYZ DSN(&DB..&TS..D&DATE) PCTPRIME 75 NBRSECND 5
```

This will mean that 2250 cylinders will be allocated as the primary and the remaining 750 cylinders will be allocated through 5 extents of 150 cylinders each.

If we used the SPACE option and coded the above example as:

```
TEMPLATE XYZ DSN(&DB..&TS..D&DATE) SPACE(3000,300) CYL MAXPRIME 2000
  NBRSECND 5
```

it would mean that 2000 would be allocated for the primary and the remaining would be allocated as 5 secondary extents of 200 cylinders each.

If we also specified a PCTPRIME of 75 in the above example as:

```
TEMPLATE XYZ DSN(&DB..&TS..D&DATE) SPACE(3000,300) CYL  PCTPRIME 75
                               MAXPRIME 2000      NBRSECND 5
```

in this case, the primary allocation would be 1500 cylinders and there would be 5 secondary extents of 300 cylinders each.

If the SPACE keyword is omitted, DB2 will estimate the size of the dataset and allocate them. (All SPACE allocations are made with the RLSE option.) However, the estimated sizes are on the higher side and may be too high for the DYNALLOC macro to handle. In that case, error message DSNU1034I is issued. This may be handled through a specific DD card or using the options explained above.

The TEMPLATE utility control statement may be specified in the SYSIN dataset, preceding the utility that would reference it. Using the TEMPLATE and LISTDEF statements, we can easily develop the JCL for a COPY utility as shown below. All the statements would be in the SYSIN DD dataset of a standard DSNUPROC or DSNUTILB job step.

```
TEMPLATE templ    UNIT 3490
        DSN(vcatname.&DB..&TS..&LOCREM.&JDATE..T&TIME..P&PART)
                    RETPD  60   STACK YES
LISTDEF     empl  INCLUDE  DATABASE EMPLOYEE
COPY  LIST  empl  COPYDDN(templ) RECOVERYDDN(templ) FULL YES
```

The above will generate local-site primary copy datasets named like:

```
vcatname.dbname.tsname.Lyyyyddd.Thhmmss.P00000
```

and remote-site primary copy datasets named like:

```
vcatname.dbname.tsname.Ryyyyddd.Thhmmss.P00000
```

The datasets are written to tape and would be retained for 60 days and the output datasets are written contiguously on the same tape without unmounting or rewinding.

If we modified it as:

```
TEMPLATE templ    UNIT 3490
         DSN(vcatname.&DB..&TS..&LR.&PB.&MO.&DA..T&TIME..P&PART)
                        RETPD  60   STACK YES
LISTDEF  empl     INCLUDE  DATABASE EMPLOYEE
COPY LIST empl COPYDDN(templ,templ) RECOVERYDDN(templ,templ) FULL YES
```

it will generate local-site primary and back-up copy datasets with names like:

```
vcatname.dbname.tsname.LPmmdd.Thhmmss.PØØØØØ
vcatname.dbname.tsname.LBmmdd.Thhmmss.PØØØØØ
```

and remote-site primary and back-up copy datasets with names like:

```
vcatname.dbname.tsname.RPmmdd.Thhmmss.PØØØØØ
vcatname.dbname.tsname.RBmmdd.Thhmmss.PØØØØØ
```

Optionally, the TEMPLATE statements may be specified in one or more datasets and referred to by a TEMPLATEDD(DDname) statement in the OPTIONS utility control statement. The default DDname is SYSTEMPL.

The LISTDEF and TEMPLATE utility control statements provide very flexible methods of extracting database object lists and defining datasets for execution of utilities. The OPTIONS utility control statement provides mechanisms for previewing the generated JCL using those two control statements. It also specifies what action to take in the event of errors or warnings encountered during the utility processing. Finally, it also provides a means of selecting different lists and templates, which must have been defined earlier.

OPTIONS

The OPTIONS utility control statement is defined using the keyword OPTIONS and followed by other parameters like PREVIEW, LISTDEFDD, TEMPLATEDD, EVENT, and OFF.

PREVIEW is used to indicate that the utility must not be executed, but the LISTDEF and TEMPLATE are to be expanded. The function is similar to the PREVIEW JCL PARM shown below:

```
//STEP1    EXEC  DSNUPROC,UID='DBNAME.ICOPY',UTPROC='PREVIEW',
//                            SYSTEM='TEST'
```

The OPTIONS parameter may be turned on and off in the SYSIN DD dataset. However, the JCL option would be in force for the entire step, even if the OPTIONS were specified in the SYSIN dataset.

LISTDEFDD and TEMPLATEDD are followed by a *DDname* and they define the DDname that points to a pre-defined dataset containing LISTDEF or TEMPLATE statements respectively.

EVENT specifies what action to take in the case of ITEMERROR and WARNING during list processing. EVENT ITEMERROR may be set to SKIP or HALT. When set to HALT, it stops processing on encountering a return code of 8 on any list object that is being processed. When set to SKIP, that item is skipped and the remainder of the list objects is processed. However, if there are more lists specified, following the list which had an ITEMERROR, they would not be processed.

The WARNING parameter may be used to reset the return codes on warning conditions and may be set to RC0, RC4, or RC8. RC0 lowers the warning to a zero return code and should be avoided; RC4 is the default and it does not change the return code; RC8 resets the return code to 8.

OFF would restore all the defaults of the OPTIONS statement and is the same as specifying:

```
OPTIONS LISTDEFDD SYSLISTD  TEMPLATEDD SYSTEMPL
             EVENT (ITEMERROR,  HALT, WARNING, RC4)
```

The authorization requirements for the OPTIONS control statement would be the same as those of the TEMPLATE control statement.

The example below illustrates the use of OPTIONS to preview the generated JCL:

```
OPTIONS PREVIEW
TEMPLATE templ    UNIT 349Ø
            DSN(vcatname.&DB..&TS..&LOCREM.&JDATE..T&TIME..P&PART)
                         RETPD  6Ø   STACK YES
LISTDEF   empl     INCLUDE  DATABASE EMPLOYEE
COPY LIST empl COPYDDN(templ) RECOVERYDDN(templ) FULL YES
```

A list of objects selected by one or more LISTDEF statements may be previewed as shown below:

```
OPTIONS PREVIEW
LISTDEF LIST1 INCLUDE  TABLESPACES  EMPLOYEE.*
              EXCLUDE  TABLESPACE   EMPLOYEE.SALARY
LISTDEF LIST2 INCLUDE  TABLESPACES  EMPLOYEE.SALARY
```

The OPTIONS may be used without a LISTDEF and a TEMPLATE as shown below. The first modify will generate a return code of 4 because there will be no image copies existing. The EVENT parameter is used to set that return code to 0. The OPTIONS is reset before processing the second MODIFY statement:

```
OPTIONS EVENT(WARNING,RCØ)
MODIFY RECOVERY TABLESPACE  EMPLOYEE.SALARY  AGE(*)
OPTIONS OFF
MODIFY  RECOVERY TABLESPACE EMPLOYEE.VACATION  AGE(45)
```

The example below PREVIEWs the COPY utility and overrides it with the ITEMERROR and SKIP options. It shows the use of the TEMPLATE and LISTDEF utility control statements along with the OPTIONS control statement.

```
OPTIONS PREVIEW
LISTDEF LIST1  INCLUDE  TABLESPACES  TABLESPACE ACCOUNT.*
               EXCLUDE  TABLESPACES  TABLESPACE ACCOUNT.OVERTIME
               INCLUDE  COPY YES  INDEXSPACES ACCOUNT.*
LISTDEF LIST2  INCLUDE  TABLESPACES  TABLESPACE EMPLOYEE.SALARY
               INCLUDE  TABLESPACES  TABLESPACE EMPLOYEE.VACATION
               INCLUDE  TABLESPACES  TABLESPACE EMPLOYEE.TRAINING
OPTIONS EVENT(ITEMERROR,SKIP)
TEMPLATE templ    UNIT 349Ø
         DSN(vcatname.&DB..&TS..&LOCREM.&JDATE..T&TIME..P&PART)
                      RETPD  6Ø   STACK YES
COPY LIST LIST1 SHRLEVEL CHANGE COPYDDN(templ) RECOVERYDDN(templ)
                FULL YES
COPY LIST LIST2 SHRLEVEL REFERENCE COPYDDN(templ) RECOVERYDDN(templ)
                FULL YES
```

CONCLUSION

The LISTDEF statement can be used to define a list of objects on which the utility can operate. The INCLUDE and EXCLUDE clauses give sufficient flexibility in defining any combination of objects. It would be possible to generate list definitions based on tablespace sets for recovery purposes.

The TEMPLATE statement would create datasets following a standard naming convention using the symbolic variables provided and the various dataset attributes. The TEMPLATE control statement can be used to automatically estimate and allocate output datasets required by the utility. For example, in a REORG job, the DDnames for the parameters like UNLDDN, WORKDDN, COPYDDN, RECOVERYDDN, etc, may be TEMPLATEd as a matter of convenience. Even though DB2 uses higher space estimates for templating, the use of the RLSE parameter ensures that the space is allocated optimally.

The OPTIONS statement can be used on any utility. The LISTDEF option is supported by almost all the online utilities except CHECK DATA (CHECK INDEX is supported), CHECK LOB, LOAD, REPAIR, and STOSPACE. The new UNLOAD utility also supports LISTDEF. The TEMPLATE option may be used where necessary.

The preview option of the OPTIONS statement would enable the review of the generated statements before they are actually executed. The options to control execution flow in case of failure or warnings would save time.

It would have been convenient to define a list of objects for a utility based on certain catalog statistics and then run the utility on that list. For example, identify a list of tablespaces that need to be REORGed based on the catalog values and then execute REORG on those.

These utility control statements should greatly enhance a DBA's life and productivity. And we may not have to develop any REXX programs to generate JCL for DB2 online utility job streams.

REFERENCE

DB2 Universal Database for OS/390 and z/OS, Utility Guide and Reference, Version 7.

Jaiwant K Jonathan
DB2 DBA
QSS Inc (USA)

Have you come across any undocumented features in DB2 Version 7.2? Why not share your discovery with others? Send your findings to us at any of the addresses shown on page 2.

DB2 for OS/390 V5 and federated systems

This article deals with the problem of executing a distributed query directed to multiple data sources from a DB2 for OS/390 V5 environment. Such an extension of DB2 for OS/390 V5 functionality is accomplished through the federated database system set up on DB2 UDB V7.2 for WIN/NT. A simple distributed request submitted by two different applications is chosen to illustrate the procedure steps needed to realize the idea.

A DB2 federated system is a special type of database management system that provides heterogeneous distributed query capability. With DB2 UDB V7.2 and the separate product DB2 Relational Connect, it is possible with a single SQL query to access data located on different platforms, both IBM and non-IBM. Distributed query can reference data from multiple sources such as the DB2 database family, OLE DB sources, Oracle and Sybase and/or Microsoft SQL Server databases. This type of query is called a distributed request and is restricted to read-only mode. End users and client applications perceive the referenced data as residing in a single collective database. Relational Connect is not required for accessing the DB2 family data sources.

The components of a federated system are a DB2 server, called a federated server, and multiple data sources. A federated server includes a federated database, while a data source consists of an instance of a relational DBMS and the database(s) within that instance. The system catalog of the federated database, called the global catalog, beside data about its own objects, stores information about data sources and specific tables, views, and functions in them. A client application sends a query to the federated database, which then directs it to the appropriate data source, obtains the required data, and returns the results to the application. A distributed request can be in any of the following forms – subquery, set operators (union, intersect, except), or join. Through DB2 SQL, table columns of any data types that DB2 supports, except LOB data types, can be referenced. The update of data is also possible by using the pass-through facility and a data source’s own SQL dialect.

In our examples we use a distributed join of the well-known sample employee and department tables DSN8510.EMP (stored in DB2 for

OS/390 V5.1) and DSN8610.DEPT (in DB2 for OS/390 V6.1). Connection is established from the application requester DBT (DB2 for OS/390 V5.1) to the federated server SAMPLE (DB2 UDB V7.2 for WIN/NT), where all information relevant to the referenced tables and access methods were previously stored in the global catalog of the federated database. The federated server acts then as an application requester, while application servers are DB2 for OS/390 V5.1 (DBT at our installation) and DB2 for OS/390 V6.1 (DB6 installed on a different machine).

Two application programs are chosen to illustrate how to execute a query that references data residing at different locations. In the first example, sample application program DSNTEP2 is used to access remote data, while in the second the same query is executed through QMF.

EXAMPLE 1

```
//SYSUSER1  JOB MSGCLASS=X,NOTIFY=&SYSUID
//JOBLIB   DD  DSN=DSN51Ø.SDSNLOAD,DISP=SHR
//STEPØØØ1 EXEC PGM=IKJEFTØ1,DYNAMNBR=2Ø
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DBT)
 RUN  PROGRAM(DSNTEP2) PLAN(DSNTEP51) -
      LIB('DBT51Ø.RUNLIB.LOAD') PARMS('/ALIGN(MID)')
 END
//*
//SYSIN    DD *
   EXEC SQL CONNECT TO SAMPLE;
   SELECT a.WORKDEPT, b.DEPTNAME, a.EMPNO, a.FIRSTNME, a.LASTNAME
   FROM EMP5 a, DEPT6 b
   WHERE a.WORKDEPT = b.DEPTNO
   ORDER BY 1, 3;
/*
//
```

To execute this query, a series of prerequisites has to be met. Customization of the Distributed Data Facility of DB2 for OS/390, TCP/IP, DB2 UDB, and federated system is needed, as well as binding some plans and packages. A summary of the necessary steps is shown below.

Step 1: DDF customization

During the DB2 for OS/390 installation or migration, process customization of the Distributed Data Facility is done. Some parameters are specific to TCP/IP and some to APPC, but in order to use DRDA TCP/IP support, APPC support has to be customized and active as well. Most DDF parameters are stored in DB2 subsystem parameter module DSNZPARM through macro DSN6FAC.

The TCP/IP port and resynchronization port numbers and APPC LU name are stored in a BSDS communication record using the change log inventory utility. This is done for both DB2 subsystems, but only the corresponding part of DSNTIJUZ JCL for one of them (DBT in our examples) is presented below:

```
//DSNTLOG EXEC PGM=DSNJUØØ3,COND=(4,LT)
//STEPLIB  DD  DISP=SHR,DSN=DSN51Ø.SDSNLOAD
//SYSUT1   DD  DISP=OLD,DSN=DBTC51Ø.BSDSØ1
//SYSUT2   DD  DISP=OLD,DSN=DBTC51Ø.BSDSØ2
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSIN    DD  *
 DDF    LOCATION=LOC2,LUNAME=LU2,
        NOPASSWD,RESPORT=5Ø2ØØ,PORT=446Ø
```

Step 2: identifying DB2 subsystem to VTAM

With PORT and RESPORT defined in BSDS, DDF accepts TCP/IP connections from clients and allows connections to other DRDA servers. To identify DB2 to VTAM, beside the LUNAME definition in BSDS, the APPL has to be defined in VTAM.

Step 3: defining DB2 to TCP/IP

Communication between DB2 for OS/390 and the TCP/IP address space is enabled through the OpenEdition MVS kernel address space. DDF is the DB2 address space that actually communicates with the OpenEdition MVS kernel address space. Modifications to TCP/IP datasets relevant to DB2 and the given task are described below.

TCPIP.PROFILE.TCPIP

```
PORT
  …
  4460 TCP OMVS               ; drda for DBT
  4463 TCP OMVS               ; drda for DB6
 50200 TCP OMVS               ; resync port for DBT
 50203 TCP OMVS               ; resync port for DB6
 51164 TCP OMVS               ; db2cDB2 for SAMPLE
 51165 TCP OMVS               ; db2iDB2  for SAMPLE
```

TCPIP.HOSTS.LOCAL

This dataset contains the IP addresses and the related host names. Directly below and in the subsequent text, change the pattern for IP addresses and host names for the local OS/390 host (MVSHOST5), the other OS/390 host (MVSHOST6), and the federated server to the appropriate ones for your environment.

```
HOST : xxx.xxx.xxx.xxx : MVSHOST5 ::::
HOST : fff.fff.fff.fff : FEDSERV  ::::
```

TCPIP.ETC.SERVICES

This dataset describes correlation between TCP/IP service names and port numbers.

```
…
drdadbt          4460/tcp       drda for DBT
db2cDB2         51164/tcp       connection port for SAMPLE
db2iDB2         51165/tcp       interrupt port for SAMPLE
```

Step 4: populate Communication Database (CDB)

Communication database tables have to be populated only at the DB2 for OS/390 application requester (DBT in our case). The partial content of these tables, with data necessary for our task, follows.

SYSIBM.LOCATIONS

```
LOCATION         LINKNAME IBMREQD PORT                             TPN
---------------- -------- ------ -------------------------------- ----
LOC1             LU1      N       446
LOC3             LU3      N       4463
SAMPLE           UDB72    N       51164
```

SYSIBM.IPNAMES

```
LINKNAME SECURITY_OUT USERNAMES IBMREQD IPADDR
-------- ------------ -------- ------ -------------------------------
LU1      A                      N       xxx.xxx.xxx.xxx
```

```
LU3      A                      N       yyy.yyy.yyy.yyy
UDB72    P           O          N       fff.fff.fff.fff
```

SYSIBM.USERNAMES

```
TYPE AUTHID   LINKNAME NEWAUTHID PASSWORD IBMREQD
---- -------- -------- -------- -------- -------
O             UDB72    db2admin  passwØ1  N
```

Step 5: customization on WIN/NT platform

The TCP/IP configuration files for DB2 UDB server are stored in ETC subdirectories:

```
C:\WINNT\system32\drivers\etc\Hosts
…
xxx.xxx.xxx.xxx                   MVSHOST5
yyy.yyy.yyy.yyy                   MVSHOST6
```

The services file contains all ports assigned to applications for the TCP/IP connection process:

```
C:\WINNT\system32\drivers\etc\Services
…
drda            446/tcp
drdadbt        446Ø/tcp
drdadb6        4463/tcp
jdbcappl       6789/tcp
rportdsn       5Ø2Ø/tcp  #two-phase commit
db2cDB2DASØØ  5ØØØØ/tcp  #connection port for the DB2 instance DB2DASØØ
db2iDB2DASØØ  5ØØØ1/tcp  #interrupt port for the DB2 instance DB2DASØØ
rportdbt      5Ø2ØØ/tcp  #two-phase commit
rportdb6      5Ø2Ø3/tcp  #two-phase commit
db2cDB2       51164/tcp  #connection port for the DB2 instance DB2
db2iDB2       51165/tcp  #interrupt port for the DB2 instance DB2
db2cDB2CTLSV  51166/tcp  #connection port for the DB2 instance DB2CTLSV
db2iDB2CTLSV  51167/tcp  #interrupt port for the DB2 instance DB2CTLSV
```

Step 6: customization of DB2 Connect

DB2 Connect installed with DB2 UDB is used for connections to DRDA servers. To connect to the remote location, UDB searches information stored in its directories. The following commands were used to catalog nodes, database aliases, and DCS database aliases at our DB2 UDB V7.2 for WIN/NT requester and to bind utilities at both DRDA servers (DB2 for OS/390 V5 and V6).

```
CATALOG TCPIP NODE N5DBT REMOTE MVSHOST5 SERVER 446Ø OSTYPE MVS;
CATALOG DCS DATABASE D5DBT AS LOC2;
CATALOG DATABASE D5DBT AS DB5DBT AT NODE  N5DBT AUTHENTICATION DCS;

CATALOG TCPIP NODE N6DB6 REMOTE MVSHOST6 SERVER 4463 OSTYPE MVS;
CATALOG DCS DATABASE D6DB6 AS LOC3;
CATALOG DATABASE D6DB6 AS DB6DB6 AT NODE N6DB6 AUTHENTICATION DCS;

CONNECT TO DB5DBT USER SYSUSER USING PASSØ5;
BIND "D:\Program files\sqllib\bnd\@ddcsmvs.lst" blocking all sqlerror
continue messages ddcsmsg5.txt grant public;
CONNECT RESET;

CONNECT TO DB6DB6 USER SYSUSER USING PASSØ6;
BIND "D:\Program files\sqllib\bnd\@ddcsmvs.lst" blocking all sqlerror
continue messages ddcsmsg6.txt grant public;
CONNECT RESET;
```

Step 7: customization of a federated system

In our example, the distributed query is directed to two data sources, both belonging to the DB2 family. Federated support for DB2 family data sources is enabled as the default during installation of DB2 UDB V7.2 on Windows NT. Relational Connect is not necessary for requests directed to DB2 databases.

The objects required to establish and use a federated system are wrappers, servers, and nicknames. Additional objects include user mappings (relating to authentication), data type mappings (associations between data source data type and DB2 data type), function mappings (local function to DB2 function mappings), and index specifications (used to improve performance). In our case, the required federated objects plus user mappings are created.

The overview of the tasks done are:

1 Connect to a federated database.

2 Create a wrapper for each database type that will be included in the federated system. Communication between federated server and data source as well as data retrieval are enabled through mechanisms called wrappers. The wrapper identifies the module (dll or library) used to access a particular type of data source.

3 Create a server to define a data source to the federated system. Server data includes the server name, server type, server version, wrapper name, authorization information, and server options.

Create a user mapping if the user ID or password at the federated server is different from the user ID or password at the data source.

5 Create a nickname for each table or view at the data source. Nicknames identify data source tables and views. Applications can reference them in queries only by those assigned identifiers and not by actual names (except in pass-through mode where its data source name has to be specified). To end users, nicknames appear similar to aliases.

6 Query the tables using nicknames.

The actual commands used are:

```
CONNECT TO SAMPLE USER db2admin USING passwØ1;
CREATE WRAPPER DRDA LIBRARY 'drda.dll';

CREATE SERVER DB5DBT TYPE DB2/39Ø VERSION 5.1 WRAPPER DRDA AUTHID
SYSUSER PASSWORD PASSØ5 OPTIONS (NODE 'N5DBT', DBNAME 'LOC2');
CREATE USER MAPPING FOR db2admin SERVER DB5DBT OPTIONS(REMOTE_AUTHID
'SYSUSER', REMOTE_PASSWORD 'PASSØ5');
CREATE NICKNAME db2admin.EMP5 FOR DB5DBT.DSN851Ø.EMP;

CREATE SERVER DB6DB6 TYPE DB2/39Ø VERSION 6.1 WRAPPER DRDA AUTHID
SYSUSER PASSWORD PASSØ6 OPTIONS (NODE 'N6DB6', DBNAME 'LOC3');
CREATE USER MAPPING FOR db2admin SERVER DB6DB6 OPTIONS(REMOTE_AUTHID
'SYSUSER', REMOTE_PASSWORD 'PASSØ6');
CREATE NICKNAME db2admin.DEPT6 FOR DB6DB6.DSN861Ø.DEPT;
```

Step 8: binding packages and plans

In our first example, a sample PL/I program DSNTEP2 is used, which enables the execution of SQL statements using Distributed Unit of Work (DUW). For a DB2 for OS/390 application requester (DBT) to be able to access data from the remote location, the proper package must also be created at the federated server (SAMPLE) and included in the local plan for the application. At run time, program DSNTEP2 will use the correct package, depending on the location where it executes. The JCL is shown below:

```
//SYSUSERB JOB MSGCLASS=X,REGION=4M,TIME=144Ø,NOTIFY=&SYSUID
//STEPØØØ1 EXEC PGM=IKJEFTØ1,DYNAMNBR=2Ø
//DBRMLIB  DD  DSN=DBT51Ø.DBRMLIB.DATA,DISP=SHR
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
```

```
 DSN SYSTEM(DBT)
 BIND PACKAGE (SAMPLE.DSNTEP2) MEMBER(DSNTEP2) ACT(REP) ISO(CS) -
   SQLERROR(NOPACKAGE) VALIDATE(BIND)
 BIND PLAN(DSNTEP51)  PKLIST(*.DSNTEP2.*) ACT(REP) ISO(CS)
 END
//
```

EXAMPLE 2

Since using QMF is very convenient for *ad hoc* queries with the possibility to get customized reports, we performed the necessary tasks to enable execution of the given distributed request using QMF V3.3.0. Connection to a remote location is possible with the QMF command CONNECT TO, and after establishing the connection, all subsequent QMF commands are directed to that location.

PROC1:

```
CONNECT TO SAMPLE
RUN QUERY1
```

QUERY1:

```
SELECT a.WORKDEPT, b.DEPTNAME, a.EMPNO, a.FIRSTNME, a.LASTNAME
FROM EMP5 a, DEPT6 b
WHERE a.WORKDEPT = b.DEPTNO
ORDER BY 1, 3
```

The other and probably more convenient option is to start the QMF session directly at the remote location. In that case the following start-up parameters should be set: DSQSSUBS, which specifies the ID of the DB2 subsystem on which QMF is installed, and DSQSDBNM, which specifies the location to connect to when starting a QMF session. In our case, if a QMF session is started with DSQSSUBS=DBT, DSQSDBNM=SAMPLE, distributed request can be executed directly from the federated server.

The following job will bind the QMF/MVS V3.3.0 installation programs from a DB2 for OS/390 Application Requester (DBT) into DRDA Application Server (SAMPLE):

```
//SYSUSERB JOB MSGCLASS=X,REGION=4M,TIME=144Ø,NOTIFY=&SYSUID
//* ===============================================================
//* 1. Change the QMFTPRE, DB2EXIT, and DB2LOAD values below to   ---
//*    the high-level qualifiers used at your DB2/MVS             ---
//*    installation.                                              ---
//* 2. Change DBT to your DB2/MVS subsystem ID.                       ---
//* 3. Change DB2ADMIN to your DB2 DRDA Application Server            ---
//*    authorization ID. This value might be determined by an         ---
//*    outbound ID translation recorded in the DB2/MVS                ---
//*    Communications Database.                                       ---
//* 4. Change SAMPLE to the location name of the DB2 DRDA             ---
//*    Application Server as defined in the DB2/MVS                   ---
//*    Communications Database.                                       ---
//* 5. (Optional) You may change the plan name DSQSI33Ø in            ---
//*    PLAN(DSQSI33Ø) below (in the BIND PLAN command) as             ---
//*    beneficial to your installation and maintenance tasks.         ---
//*    If you do so then assure that the plan name is also            ---
//*    changed in and coordinated with the following jobs:            ---
//*       DSQ1EDJ2, DSQ1BDJ3, DSQ1EDJ4, DSQ1EDX1 and DSQ1EDX2.        ---
//*                                                                   ---
//* * NOTE BEFORE THIS JOB CAN BE EXECUTED:                           ---
//*   1) DRDA COMMUNICATIONS BETWEEN THE DB2/MVS APPLICATION          ---
//*      REQUESTER AND THE DB2 APPLICATION SERVER MUST BE DEFINED ---
//*      AND OPERATIONAL.                                             ---
//*   2) THE DB2/MVS SUBSYSTEM MUST BE STARTED WITH DDF ACTIVE.       ---
//*   3) THE DB2 DRDA APPLICATION SERVER MUST BE STARTED.             ---
//* ==================================================================
//DSQBIND  PROC RGN='2Ø48K',
//             QMFTPRE='QMF33Ø',
//             DB2EXIT='SYS1.DBT51Ø.SDSNEXIT',
//             DB2LOAD='DSN51Ø.SDSNLOAD'
//BIND1   EXEC PGM=IKJEFTØ1,REGION=&RGN
//STEPLIB  DD  DSN=&DB2EXIT.,DISP=SHR
//         DD  DSN=&DB2LOAD.,DISP=SHR
//SYSTSPRT DD  SYSOUT=*,DCB=BLKSIZE=121
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//DBRMLIB  DD  DSN=&QMFTPRE..DSQDBRM,DISP=SHR
//         PEND
//* ==================================================================
//DSQBIND1 PROC RGN='2Ø48K',
//             QMFTPRE='QMF33Ø',
//             DB2EXIT='SYS1.DBT51Ø.SDSNEXIT',
//             DB2LOAD='DSN51Ø.SDSNLOAD'
//BIND2    EXEC PGM=IKJEFTØ1,REGION=&RGN
//STEPLIB  DD  DSN=&DB2EXIT.,DISP=SHR
//         DD  DSN=&DB2LOAD.,DISP=SHR
//SYSTSPRT DD  SYSOUT=*,DCB=BLKSIZE=121
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//DBRMLIB  DD  DSN=&QMFTPRE..DSQDBRM,DISP=SHR
//GRANT   EXEC PGM=IKJEFTØ1,REGION=&RGN
//STEPLIB  DD  DSN=&QMFTPRE..DSQLOAD,DISP=SHR
//         DD  DSN=&DB2EXIT.,DISP=SHR
//         DD  DSN=&DB2LOAD.,DISP=SHR
//SYSTSPRT DD  SYSOUT=*,DCB=BLKSIZE=121
```

```
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SQL2  EXEC PGM=IKJEFTØ1,REGION=&RGN
//STEPLIB  DD  DSN=&QMFTPRE..DSQLOAD,DISP=SHR
//         DD  DSN=&DB2EXIT.,DISP=SHR
//         DD  DSN=&DB2LOAD.,DISP=SHR
//SYSTSPRT DD  SYSOUT=*,DCB=BLKSIZE=121
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//BIND3  EXEC PGM=IKJEFTØ1,REGION=&RGN,
//        COND=((1,GE,SQL2),(3,LT,SQL2))
//STEPLIB  DD  DSN=&DB2EXIT.,DISP=SHR
//         DD  DSN=&DB2LOAD.,DISP=SHR
//SYSTSPRT DD  SYSOUT=*,DCB=BLKSIZE=121
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//DBRMLIB  DD  DSN=&QMFTPRE..DSQDBRM,DISP=SHR
//GRANT3  EXEC PGM=IKJEFTØ1,REGION=&RGN,
//        COND=((8,LE,BIND3),(1,GE,SQL2),(3,LT,SQL2))
//STEPLIB  DD  DSN=&QMFTPRE..DSQLOAD,DISP=SHR
//         DD  DSN=&DB2EXIT.,DISP=SHR
//         DD  DSN=&DB2LOAD.,DISP=SHR
//SYSTSPRT DD  SYSOUT=*,DCB=BLKSIZE=121
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//         PEND
//* =================================================================
//DSQEXSQL PROC RGN='2Ø48K',
//             QMFTPRE='QMF33Ø',
//             DB2EXIT='SYS1.DBT51Ø.SDSNEXIT',
//             DB2LOAD='DSN51Ø.SDSNLOAD'
//TSODSN  EXEC PGM=IKJEFTØ1,REGION=&RGN
//STEPLIB  DD  DSN=&QMFTPRE..DSQLOAD,DISP=SHR
//         DD  DSN=&DB2EXIT.,DISP=SHR
//         DD  DSN=&DB2LOAD.,DISP=SHR
//SYSTSPRT DD  SYSOUT=*,DCB=BLKSIZE=132
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//         PEND
//BINDP1  EXEC PROC=DSQBIND
//BIND1.SYSTSIN DD *
DSN SYSTEM(DBT)
BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQABSQL)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQABINS)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
BIND PLAN(DSQSI33Ø) PKLIST (*.Q.*) ISOLATION(CS) VALIDATE(BIND) -
      CURRENTSERVER(SAMPLE) ACTION(REPLACE) RETAIN CURRENTDATA(NO)
//* =================================================================
//DSQCTBL EXEC DSQEXSQL
//TSODSN.SYSTSIN DD *
DSN SYSTEM(DBT)
RUN PROGRAM(DSQØBSQL) PLAN(DSQSI33Ø)
//* -----------------------------------------------------------------
//TSODSN.SYSIN DD *
 CREATE REGULAR TABLESPACE DSQTSCT1 MANAGED BY SYSTEM USING
('DSQTSCT1');
 CREATE REGULAR TABLESPACE DSQTSCT2 MANAGED BY SYSTEM USING
('DSQTSCT2');
 CREATE REGULAR TABLESPACE DSQTSCT3 MANAGED BY SYSTEM USING
('DSQTSCT3');
 CREATE REGULAR TABLESPACE DSQTSGOV MANAGED BY SYSTEM USING
('DSQTSGOV');
 CREATE REGULAR TABLESPACE DSQTSLOG MANAGED BY SYSTEM USING
('DSQTSLOG');
 CREATE REGULAR TABLESPACE DSQTSPRO MANAGED BY SYSTEM USING
('DSQTSPRO');
 CREATE REGULAR TABLESPACE DSQTSRDO MANAGED BY SYSTEM USING
('DSQTSRDO');
 CREATE REGULAR TABLESPACE DSQTSSYN MANAGED BY SYSTEM USING
('DSQTSSYN');
 CREATE REGULAR TABLESPACE DSQTSDEF MANAGED BY SYSTEM USING
('DSQTSDEF');
CREATE TABLE Q.DSQ_RESERVED
  (QMFREQ1 CHAR(16),
   QMFREQ2 CHAR(1) NOT NULL)
IN DSQTSRDO;
COMMENT ON TABLE Q.DSQ_RESERVED IS 'QMF CONTROL TABLE';
COMMENT ON COLUMN Q.DSQ_RESERVED.QMFREQ1 IS 'RESERVED FOR QMF';
COMMENT ON COLUMN Q.DSQ_RESERVED.QMFREQ2 IS 'RESERVED FOR QMF';
GRANT SELECT ON Q.DSQ_RESERVED TO PUBLIC;
CREATE TABLE Q.RESOURCE_TABLE (
  RESOURCE_GROUP  CHAR(16) NOT NULL,
  RESOURCE_OPTION CHAR(16) NOT NULL,
  INTVAL          INTEGER,
  FLOATVAL        FLOAT,
  CHARVAL         VARCHAR(8Ø) )
IN DSQTSGOV;
COMMENT ON TABLE Q.RESOURCE_TABLE IS
 'RESOURCE TABLE FOR QMF GOVERNOR';
COMMENT ON COLUMN Q.RESOURCE_TABLE.RESOURCE_GROUP IS
 'RESOURCE GROUP NAME USED BY QMF GOVERNOR';
COMMENT ON COLUMN Q.RESOURCE_TABLE.RESOURCE_OPTION IS
 'RESOURCE OPTION NAME';
COMMENT ON COLUMN Q.RESOURCE_TABLE.INTVAL IS
 'INTEGER OPTION VALUE';
COMMENT ON COLUMN Q.RESOURCE_TABLE.FLOATVAL IS
 'FLOATING POINT OPTION VALUE';
COMMENT ON COLUMN Q.RESOURCE_TABLE.CHARVAL IS
 'CHARACTER OPTION VALUE';
```

```
CREATE INDEX Q.RESOURCE_INDEX ON Q.RESOURCE_TABLE
 (RESOURCE_GROUP,RESOURCE_OPTION);
CREATE VIEW Q.RESOURCE_VIEW
  (RESOURCE_GROUP,
   RESOURCE_OPTION,
   INTVAL,
   FLOATVAL,
   CHARVAL)
 AS SELECT RESOURCE_GROUP, RESOURCE_OPTION, INTVAL,
           FLOATVAL, CHARVAL FROM Q.RESOURCE_TABLE;
GRANT SELECT ON Q.RESOURCE_VIEW TO PUBLIC;
DELETE FROM Q.RESOURCE_TABLE
 WHERE RESOURCE_GROUP='SYSTEM' AND RESOURCE_OPTION='SCOPE'
  AND INTVAL IS NULL AND FLOATVAL IS NULL;
INSERT INTO Q.RESOURCE_TABLE
       ( RESOURCE_GROUP, RESOURCE_OPTION, FLOATVAL, INTVAL, CHARVAL )
 VALUES( 'SYSTEM', 'SCOPE', NULL,
          NULL, 'INDICATE WHETHER GOVERNOR IS ACTIVE' );
DELETE FROM Q.RESOURCE_TABLE
 WHERE RESOURCE_GROUP='SYSTEM' AND RESOURCE_OPTION='ROWPROMPT'
  AND INTVAL=25000 AND FLOATVAL IS NULL;
INSERT INTO Q.RESOURCE_TABLE
       ( RESOURCE_GROUP, RESOURCE_OPTION, FLOATVAL, INTVAL, CHARVAL )
 VALUES('SYSTEM', 'ROWPROMPT', NULL,
         25000, 'PROMPT USER AFTER FETCHING 25000 ROWS');
DELETE FROM Q.RESOURCE_TABLE
 WHERE RESOURCE_GROUP='SYSTEM' AND RESOURCE_OPTION='ROWLIMIT'
  AND INTVAL=100000 AND FLOATVAL IS NULL;
INSERT INTO Q.RESOURCE_TABLE
       ( RESOURCE_GROUP, RESOURCE_OPTION, FLOATVAL, INTVAL, CHARVAL )
 VALUES('SYSTEM', 'ROWLIMIT', NULL,
          100000, 'CANCEL AFTER FETCHING 100000 ROWS');
DELETE FROM Q.RESOURCE_TABLE
 WHERE RESOURCE_GROUP='SYSTEM' AND RESOURCE_OPTION='TIMEPROMPT'
   AND FLOATVAL IS NULL
   AND INTVAL = 360;
INSERT INTO Q.RESOURCE_TABLE
       ( RESOURCE_GROUP, RESOURCE_OPTION, FLOATVAL, INTVAL, CHARVAL )
 VALUES('SYSTEM', 'TIMEPROMPT', NULL,
        360, 'PROMPT AFTER 6 MINUTES OF CPU TIME'
       );
DELETE FROM Q.RESOURCE_TABLE
 WHERE RESOURCE_GROUP='SYSTEM' AND RESOURCE_OPTION='TIMECHECK'
  AND INTVAL=900 AND FLOATVAL IS NULL;
INSERT INTO Q.RESOURCE_TABLE
       ( RESOURCE_GROUP, RESOURCE_OPTION, FLOATVAL, INTVAL, CHARVAL )
 VALUES('SYSTEM', 'TIMECHECK', NULL,
             900, 'CHECK TIME AT 15 MINUTE INTERVALS');
DELETE FROM Q.RESOURCE_TABLE
 WHERE RESOURCE_GROUP='SYSTEM' AND RESOURCE_OPTION='TIMELIMIT'
   AND FLOATVAL IS NULL
   AND INTVAL = 144Ø;
INSERT INTO Q.RESOURCE_TABLE
       ( RESOURCE_GROUP, RESOURCE_OPTION, FLOATVAL, INTVAL, CHARVAL )
 VALUES('SYSTEM', 'TIMELIMIT', NULL,
         144Ø, 'CANCEL AFTER 24 MINUTES OF CPU TIME');
CREATE TABLE Q.PROFILES
  ( CREATOR        CHAR(8) NOT NULL,
   "CASE"          CHAR(18),
    DECOPT         CHAR(18),
    CONFIRM        CHAR(18),
    WIDTH          CHAR(18),
    LENGTH         CHAR(18),
    LANGUAGE       CHAR(18),
    SPACE          CHAR(5Ø),
    TRACE          CHAR(18),
    PRINTER        CHAR(8),
    TRANSLATION    CHAR(18) NOT NULL,
    PFKEYS         VARCHAR(31),
    SYNONYMS       VARCHAR(31),
    RESOURCE_GROUP CHAR(16),
    MODEL          CHAR(8),
    ENVIRONMENT    CHAR(8)
  )
IN DSQTSPRO;
COMMENT ON TABLE Q.PROFILES IS
 'QMF USER PROFILE TABLE';
COMMENT ON COLUMN Q.PROFILES.CREATOR IS
 'LOGON USER ID OR SYSTEM';
COMMENT ON COLUMN Q.PROFILES."CASE" IS
 'TERMINAL INPUT CASE OPTION';
COMMENT ON COLUMN Q.PROFILES.DECOPT IS
 'NUMERIC DECIMAL OUTPUT FORMAT';
COMMENT ON COLUMN Q.PROFILES.CONFIRM IS
 'DATA CHANGE CONFIRMATION OPTION';
COMMENT ON COLUMN Q.PROFILES.WIDTH IS
 'PRINT COMMAND DEFAULT WIDTH';
COMMENT ON COLUMN Q.PROFILES.LENGTH IS
 'PRINT COMMAND DEFAULT LENGTH';
COMMENT ON COLUMN Q.PROFILES.LANGUAGE IS 'QUERY LANGUAGE OPTION';
COMMENT ON COLUMN Q.PROFILES.SPACE IS
 'DBSPACE NAME FOR SAVE DATA';
COMMENT ON COLUMN Q.PROFILES.TRACE IS
 'QMF TRACE INDICATORS';
COMMENT ON COLUMN Q.PROFILES.PRINTER IS
 'PRINT COMMAND DEFAULT GDDM NICKNAME';
COMMENT ON COLUMN Q.PROFILES.TRANSLATION IS
 'NATIONAL LANGUAGE FOR ROW';
COMMENT ON COLUMN Q.PROFILES.PFKEYS IS
 'PF KEY DEFINITION TABLE NAME';
COMMENT ON COLUMN Q.PROFILES.SYNONYMS IS
 'SYNONYM DEFINITION TABLE NAME';
```

```
COMMENT ON COLUMN Q.PROFILES.RESOURCE_GROUP IS
 'RESOURCE GROUP NAME USED BY QMF GOVERNOR';
COMMENT ON COLUMN Q.PROFILES.MODEL IS
 'TYPE OF QUERY';
COMMENT ON COLUMN Q.PROFILES.ENVIRONMENT IS
 'QMF ENVIRONMENT';
CREATE UNIQUE INDEX Q.PROFILEX ON Q.PROFILES (CREATOR ASC , TRANSLATION
 ASC, ENVIRONMENT ASC);
INSERT INTO Q.PROFILES
 ( CREATOR, "CASE", DECOPT, CONFIRM, WIDTH, LENGTH, LANGUAGE, SPACE,
   TRACE, PRINTER, TRANSLATION, PFKEYS, SYNONYMS, RESOURCE_GROUP, MODEL,
   ENVIRONMENT ) VALUES
 ( 'SYSTEM'                    -- CREATOR
 , 'UPPER'                     -- CASE
 , 'PERIOD'                    -- DECOPT
 , 'YES'                       -- CONFIRM
 , '132'                       -- WIDTH
 , '6Ø'                        -- LENGTH
 , 'SQL'                       -- LANGUAGE
 , 'DSQTSDEF'                  -- SPACE
 , 'NONE'                      -- TRACE
 , ' '                         -- PRINTER
 , 'ENGLISH'                   -- TRANSLATION
 ,  NULL                       -- PFKEYS
 , 'Q.COMMAND_SYN_TSO'         -- SYNONYMS
 , 'SYSTEM'                    -- RESOURCE_GROUP
 ,  NULL                       -- MODEL
 , 'TSO'                       -- ENVIRONMENT
 );
INSERT INTO Q.PROFILES
 ( CREATOR, "CASE", DECOPT, CONFIRM, WIDTH, LENGTH, LANGUAGE, SPACE,
   TRACE, PRINTER, TRANSLATION, PFKEYS, SYNONYMS, RESOURCE_GROUP, MODEL,
   ENVIRONMENT ) VALUES
 ( 'SYSTEM'                    -- CREATOR
 , 'UPPER'                     -- CASE
 , 'PERIOD'                    -- DECOPT
 , 'YES'                       -- CONFIRM
 , '132'                       -- WIDTH
 , '6Ø'                        -- LENGTH
 , 'SQL'                       -- LANGUAGE
 , 'DSQTSDEF'                  -- SPACE
 , 'NONE'                      -- TRACE
 , ' '                         -- PRINTER
 , 'ENGLISH'                   -- TRANSLATION
 ,  NULL                       -- PFKEYS
 ,  NULL                       -- SYNONYMS
 , 'SYSTEM'                    -- RESOURCE_GROUP
 ,  NULL                       -- MODEL
 , 'CICS'                      -- ENVIRONMENT
 );
INSERT INTO Q.PROFILES
```

```
  ( CREATOR, "CASE", DECOPT, CONFIRM, WIDTH, LENGTH, LANGUAGE, SPACE,
    TRACE, PRINTER, TRANSLATION, PFKEYS, SYNONYMS, RESOURCE_GROUP, MODEL,
    ENVIRONMENT ) VALUES
  ( 'SYSTEM'                    -- CREATOR
  , 'UPPER'                     -- CASE
  , 'PERIOD'                    -- DECOPT
  , 'YES'                       -- CONFIRM
  , '132'                       -- WIDTH
  , '6Ø'                        -- LENGTH
  , 'SQL'                       -- LANGUAGE
  , 'DSQTSDEF'                  -- SPACE
  , 'NONE'                      -- TRACE
  , ' '                         -- PRINTER
  , 'ENGLISH'                   -- TRANSLATION
  ,  NULL                       -- PFKEYS
  , 'Q.COMMAND_SYN_CMS'         -- SYNONYMS
  , 'SYSTEM'                    -- RESOURCE_GROUP
  ,  NULL                       -- MODEL
  , 'CMS');                     -- ENVIRONMENT
 CREATE TABLE Q.OBJECT_DIRECTORY
   (OWNER CHAR(8) NOT NULL ,
    NAME VARCHAR(18) NOT NULL ,
    TYPE CHAR(8) NOT NULL ,
    SUBTYPE CHAR(8) ,
    OBJECTLEVEL INTEGER NOT NULL ,
    RESTRICTED CHAR(1) NOT NULL ,
    MODEL CHAR(8) ,
    CREATED TIMESTAMP ,
    MODIFIED TIMESTAMP ,
    LAST_USED TIMESTAMP)
 IN DSQTSCT1;
 COMMENT ON TABLE Q.OBJECT_DIRECTORY IS
  'QMF SAVED ITEM DIRECTORY TABLE';
 COMMENT ON COLUMN Q.OBJECT_DIRECTORY.OWNER IS
  'AUTHORIZATION ID OF ITEM  OWNER';
 COMMENT ON COLUMN Q.OBJECT_DIRECTORY.NAME IS
  'NAME OF QMF ITEM';
 COMMENT ON COLUMN Q.OBJECT_DIRECTORY.TYPE IS
  'TYPE OF ITEM';
 COMMENT ON COLUMN Q.OBJECT_DIRECTORY.SUBTYPE IS
  'SUBTYPE OF ITEM';
 COMMENT ON COLUMN Q.OBJECT_DIRECTORY.OBJECTLEVEL IS
  'VERSION OF INTERNAL REPRESENTATION OF ITEM';
 COMMENT ON COLUMN Q.OBJECT_DIRECTORY.RESTRICTED IS
  'OBJECT ACCESS RESTRICTED INDICATION';
 COMMENT ON COLUMN Q.OBJECT_DIRECTORY.MODEL IS
  'TYPE OF QUERY';
 COMMENT ON COLUMN Q.OBJECT_DIRECTORY.CREATED IS
  'OBJECT CREATION DATE';
 COMMENT ON COLUMN Q.OBJECT_DIRECTORY.MODIFIED IS
  'OBJECT MODIFICATION DATE';
```

```
COMMENT ON COLUMN Q.OBJECT_DIRECTORY.LAST_USED IS
 'OBJECT LAST USAGE DATE';
CREATE UNIQUE INDEX Q.OBJECT_DIRECTORYX ON Q.OBJECT_DIRECTORY
  (OWNER ASC,
   NAME ASC);
CREATE TABLE Q.OBJECT_REMARKS
  (OWNER CHAR(8) NOT NULL ,
   NAME VARCHAR(18) NOT NULL ,
   TYPE CHAR(8) NOT NULL ,
   REMARKS VARCHAR(254))
IN DSQTSCT2;
COMMENT ON TABLE Q.OBJECT_REMARKS IS
 'QMF SAVED ITEM COMMENTS TABLE';
COMMENT ON COLUMN Q.OBJECT_REMARKS.OWNER IS
 'AUTHORIZATION ID OF QMF ITEM OWNER';
COMMENT ON COLUMN Q.OBJECT_REMARKS.NAME IS
 'NAME OF QMF ITEM';
COMMENT ON COLUMN Q.OBJECT_REMARKS.TYPE IS
 'TYPE OF QMF ITEM';
COMMENT ON COLUMN Q.OBJECT_REMARKS.REMARKS IS
 'COMMENTS ABOUT QMF ITEM';
CREATE UNIQUE INDEX Q.OBJECT_REMARKSX ON Q.OBJECT_REMARKS
  (OWNER ASC,
   NAME ASC);
CREATE TABLE Q.OBJECT_DATA
  (OWNER CHAR(8) NOT NULL ,
   NAME VARCHAR(18) NOT NULL ,
   TYPE CHAR(8) NOT NULL ,
   SEQ SMALLINT NOT NULL ,
   APPLDATA VARCHAR(3600) FOR BIT DATA)
IN DSQTSCT3;
COMMENT ON TABLE Q.OBJECT_DATA IS
 'QMF SAVED ITEM DATA TABLE';
COMMENT ON COLUMN Q.OBJECT_DATA.OWNER IS
 'AUTHORIZATION ID OF QMF ITEM OWNER';
COMMENT ON COLUMN Q.OBJECT_DATA.NAME IS
 'NAME OF QMF ITEM';
COMMENT ON COLUMN Q.OBJECT_DATA.TYPE IS
 'TYPE OF QMF ITEM';
COMMENT ON COLUMN Q.OBJECT_DATA.SEQ IS
 'ROW SEQUENCE NUMBER';
COMMENT ON COLUMN Q.OBJECT_DATA.APPLDATA IS
 'QMF ITEM DATA';
CREATE UNIQUE INDEX Q.OBJECT_OBJDATAX ON Q.OBJECT_DATA
  (OWNER ASC,
   NAME  ASC,
   SEQ   ASC);
CREATE TABLE Q.OBJECT_DATA2
  (OWNER CHAR(8) NOT NULL ,
   NAME VARCHAR(18) NOT NULL ,
   TYPE CHAR(8) NOT NULL ,
   SEQ SMALLINT NOT NULL ,
   APPLDATA VARCHAR(3600) FOR BIT DATA)
IN DSQTSCT3;
COMMENT ON TABLE Q.OBJECT_DATA2 IS
 'QMF TEMPORARY SAVED ITEM DATA TABLE';
COMMENT ON COLUMN Q.OBJECT_DATA2.OWNER IS
 'AUTHORIZATION ID OF QMF ITEM OWNER';
COMMENT ON COLUMN Q.OBJECT_DATA2.NAME IS
 'NAME OF QMF ITEM';
COMMENT ON COLUMN Q.OBJECT_DATA2.TYPE IS
 'TYPE OF QMF ITEM';
COMMENT ON COLUMN Q.OBJECT_DATA2.SEQ IS
 'ROW SEQUENCE NUMBER';
COMMENT ON COLUMN Q.OBJECT_DATA2.APPLDATA IS
 'QMF ITEM DATA';
CREATE UNIQUE INDEX Q.OBJECT_OBJDATA2X ON Q.OBJECT_DATA2
  (OWNER ASC,
   NAME  ASC,
   SEQ   ASC);
GRANT INSERT ON Q.OBJECT_DATA2 TO PUBLIC;
CREATE TABLE Q.ERROR_LOG
  (DATESTAMP         CHAR(8)        NOT NULL,
  "TIMESTAMP"        CHAR(5)        NOT NULL,
   USERID            CHAR(8)        NOT NULL,
   MSG_NO            CHAR(8)        NOT NULL,
   MSGTEXT           VARCHAR(254)   NOT NULL)
IN DSQTSLOG;
COMMENT ON TABLE Q.ERROR_LOG IS
 'QMF ERROR RECORDING LOG';
COMMENT ON COLUMN Q.ERROR_LOG.DATESTAMP IS
 'DATE OF ERROR RECORDING';
COMMENT ON COLUMN Q.ERROR_LOG."TIMESTAMP" IS
 'TIME OF ERROR RECORDING';
COMMENT ON COLUMN Q.ERROR_LOG.USERID IS
 'LOGON ID OF USER THAT ERROR RECORDING IS FOR';
COMMENT ON COLUMN Q.ERROR_LOG.MSG_NO IS
 'QMF MESSAGE NUMBER';
COMMENT ON COLUMN Q.ERROR_LOG.MSGTEXT IS
 'ERROR MESSAGE';
CREATE TABLE Q.COMMAND_SYN_TSO
 ( VERB               CHAR(18)     NOT NULL,
   OBJECT             VARCHAR(31),
   SYNONYM_DEFINITION VARCHAR(254) NOT NULL,
   REMARKS            VARCHAR(254)
 ) IN DSQTSSYN;
COMMENT ON TABLE  Q.COMMAND_SYN_TSO IS
 'QMF TSO COMMAND SYNONYM TABLE';
COMMENT ON COLUMN Q.COMMAND_SYN_TSO.VERB IS
 'NAME OF THE VERB';
```

```
COMMENT ON COLUMN Q.COMMAND_SYN_TSO.OBJECT IS
 'NAME OF THE OBJECT';
COMMENT ON COLUMN Q.COMMAND_SYN_TSO.SYNONYM_DEFINITION IS
 'DEFINITION OF SYNONYM';
COMMENT ON COLUMN Q.COMMAND_SYN_TSO.REMARKS IS
 'COMMENTS ABOUT SYNONYM';
CREATE UNIQUE INDEX Q.COMMAND_SYN_TSOX ON Q.COMMAND_SYN_TSO
 ( VERB   ASC,
   OBJECT ASC );
GRANT SELECT ON Q.COMMAND_SYN_TSO TO PUBLIC;
INSERT INTO Q.COMMAND_SYN_TSO VALUES (
  'DPRE',
   NULL,
  'RUN Q.DSQAER1P',
  'QMF DISPLAY PRINTED REPORT APPLICATION' );
INSERT INTO Q.COMMAND_SYN_TSO VALUES (
  'ISPF',
   NULL,
  'TSO DSQAEZ1P P(''&ALL'')',
  'QMF ISPF BRIDGE APPLICATION' );
INSERT INTO Q.COMMAND_SYN_TSO VALUES (
  'BATCH',
   NULL,
  'TSO DSQABB11 PNAME(DXYEABMP)',
  'QMF BATCH APPLICATION' );
INSERT INTO Q.COMMAND_SYN_TSO VALUES (
  'LAYOUT',
   NULL,
  'TSO DSQAELØA',
  'QMF LAYOUT APPLICATION' );
CREATE TABLE Q.COMMAND_SYN_CMS
 ( VERB               CHAR(18)     NOT NULL,
   OBJECT             VARCHAR(31),
   SYNONYM_DEFINITION VARCHAR(254) NOT NULL,
   REMARKS            VARCHAR(254)
 ) IN DSQTSSYN;
COMMENT ON TABLE  Q.COMMAND_SYN_CMS IS
 'QMF CMS COMMAND SYNONYM TABLE';
COMMENT ON COLUMN Q.COMMAND_SYN_CMS.VERB IS
 'NAME OF THE VERB';
COMMENT ON COLUMN Q.COMMAND_SYN_CMS.OBJECT IS
 'NAME OF THE OBJECT';
COMMENT ON COLUMN Q.COMMAND_SYN_CMS.SYNONYM_DEFINITION IS
 'DEFINITION OF SYNONYM';
COMMENT ON COLUMN Q.COMMAND_SYN_CMS.REMARKS IS
 'COMMENTS ABOUT SYNONYM';
CREATE UNIQUE INDEX Q.COMMAND_SYN_CMSX ON Q.COMMAND_SYN_CMS
 ( VERB   ASC,
   OBJECT ASC );
GRANT SELECT ON Q.COMMAND_SYN_CMS TO PUBLIC;
```

```
INSERT INTO Q.COMMAND_SYN_CMS VALUES (
  'DPRE',
   NULL,
  'RUN Q.DSQAER2P',
  'QMF DISPLAY PRINTED REPORT APPLICATION' );
INSERT INTO Q.COMMAND_SYN_CMS VALUES (
  'ISPF',
   NULL,
  'CMS DSQAEZ2P',
  'QMF ISPF BRIDGE APPLICATION' );
INSERT INTO Q.COMMAND_SYN_CMS VALUES (
  'BATCH',
   NULL,
  'CMS DSQABB21 DXYEABVP',
  'QMF BATCH APPLICATION' );
INSERT INTO Q.COMMAND_SYN_CMS VALUES (
  'LAYOUT',
   NULL,
  'CMS DSQAELØA',
  'QMF LAYOUT APPLICATION' );
CREATE VIEW Q.DSQEC_QMFOBJS
 ( OWNER, TNAME, TYPE, SUBTYPE, MODEL
 , RESTRICTED, REMARKS, CREATED, MODIFIED, LAST_USED
 , LABEL, LOCATION, OWNER_AT_LOCATION, NAME_AT_LOCATION )
 AS SELECT A.OWNER, A.NAME, A.TYPE, SUBTYPE, MODEL
 , RESTRICTED, REMARKS, CHAR(CREATED), CHAR(MODIFIED), CHAR(LAST_USED)
 , ' ', ' ' , ' ' , ' '
  FROM Q.OBJECT_DIRECTORY A, Q.OBJECT_REMARKS B
 WHERE A.OWNER=B.OWNER AND A.NAME=B.NAME
   AND (A.OWNER=USER OR RESTRICTED='N');
COMMENT ON TABLE Q.DSQEC_QMFOBJS IS
 'DEFAULT VIEW FOR LIST QMF OBJECTS QMF COMMAND';
GRANT SELECT ON Q.DSQEC_QMFOBJS TO PUBLIC;
CREATE VIEW Q.DSQEC_TABS_LDB2
 ( OWNER, TNAME, TYPE, SUBTYPE, MODEL,
   RESTRICTED, REMARKS, CREATED, MODIFIED, LAST_USED,
   LABEL, LOCATION, OWNER_AT_LOCATION, NAME_AT_LOCATION ) AS
 SELECT DISTINCT
     CREATOR, NAME, 'TABLE', TYPE, ' '
   , ' ', REMARKS, ' ', ' ', ' '
   , ' ', ' ', ' ', ' '
   FROM SYSIBM.SYSTABLES A, SYSIBM.SYSTABAUTH B
  WHERE CREATOR = TCREATOR AND NAME=TTNAME
    AND GRANTEE IN (USER,'PUBLIC');
COMMENT ON TABLE Q.DSQEC_TABS_LDB2 IS
 'DEFAULT VIEW FOR LIST TABLES QMF COMMAND';
GRANT SELECT ON Q.DSQEC_TABS_LDB2 TO PUBLIC;
CREATE VIEW Q.DSQEC_COLS_LDB2
 ( OWNER, TNAME, CNAME, REMARKS, LABEL ) AS
 SELECT DISTINCT
```

```
     TBCREATOR, TBNAME, NAME, REMARKS, ' '
   FROM SYSIBM.SYSCOLUMNS, SYSIBM.SYSTABAUTH
  WHERE TCREATOR = TBCREATOR AND TTNAME = TBNAME
    AND GRANTEE IN (USER,'PUBLIC');
COMMENT ON TABLE Q.DSQEC_COLS_LDB2 IS
 'DEFAULT VIEW FOR QMF LIST TABLE COLUMN INFORMATION';
GRANT SELECT ON Q.DSQEC_COLS_LDB2 TO PUBLIC;
 CREATE VIEW Q.DSQEC_ALIASES
  ( OWNER, TNAME, TYPE, SUBTYPE, MODEL, RESTRICTED,
    REMARKS, CREATED, MODIFIED, LAST_USED, LABEL, LOCATION,
    OWNER_AT_LOCATION,
    NAME_AT_LOCATION )
  AS SELECT TABSCHEMA, TABNAME, 'TABLE', TYPE, ' ', ' ',
            REMARKS , ' ', ' ', ' ', ' ', ' ',
            TABLE_SCHEMA(TABNAME,TABSCHEMA),
            TABLE_NAME(TABNAME,TABSCHEMA)
       FROM SYSCAT.TABLES
      WHERE TYPE='A'
        AND USER IN ( TABSCHEMA,
                      TABLE_SCHEMA(TABNAME,TABSCHEMA) );
 COMMENT ON TABLE Q.DSQEC_ALIASES
   IS 'DEFAULT VIEW FOR QMF LIST TABLE ALIASES - DB2 CS V2';
 GRANT SELECT ON Q.DSQEC_ALIASES TO PUBLIC;
COMMIT;
//* ===========================================================
//BINDP2   EXEC PROC=DSQBIND1
//BIND2.SYSTSIN DD *
 DSN SYSTEM(DBT)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQADYSQ)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQAESQL)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQAFSQL)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQAICVS)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQARCTL)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQASUSR)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQABOR)
VALIDATE(BIND)  -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQASV)
```

```
VALIDATE(BIND)   -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQALD)
VALIDATE(BIND)   -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQAPR)
VALIDATE(BIND)   -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQAUPRF)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQASDTA)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQADTVQ)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQACSQL)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQASV2)
VALIDATE(BIND)  -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQAIPEL)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQAHSQL)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQATSQL)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQASSQL)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
//GRANT.SYSTSIN DD *
DSN SYSTEM(DBT)
RUN PROGRAM(DSQØBSQL) PLAN(DSQSI33Ø)
//GRANT.SYSIN DD *
GRANT EXECUTE ON PACKAGE Q.DSQADYSQ TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQAESQL TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQAFSQL TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQAICVS TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQARCTL TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQASUSR TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQABOR  TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQASV   TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQALD   TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQAPR   TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQAUPRF TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQASDTA TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQADTVQ TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQACSQL TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQASV2  TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQAIPEL TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQAHSQL TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQATSQL TO PUBLIC;
GRANT EXECUTE ON PACKAGE Q.DSQASSQL TO PUBLIC;
COMMIT;
//SQL2.SYSTSIN DD *
DSN SYSTEM(DBT)
RUN PROGRAM(DSQØBSQL) PLAN(DSQSI33Ø)
//SQL2.SYSIN DD *
-SET +TRC;
-WITH(SQLØ1) -EXIT 1;
-WITH(SQLØ2) -EXIT 2;
-WITH(SQLØ)  -EXIT 3;
-EXIT Ø;
//BIND3.SYSTSIN DD *
 DSN SYSTEM(DBT)
 BIND PACKAGE(SAMPLE.Q) QUALIFIER(DB2ADMIN) MEMBER(DSQASSQ2)
VALIDATE(BIND) -
      ISOLATION(CS) RELEASE(COMMIT) ACTION(REPLACE) CURRENTDATA(NO)
//GRANT3.SYSTSIN DD *
DSN SYSTEM(DBT)
RUN PROGRAM(DSQØBSQL) PLAN(DSQSI33Ø)
//GRANT3.SYSIN DD *
 GRANT EXECUTE ON PACKAGE Q.DSQASSQ2 TO PUBLIC;
//
```

Nikola Lazovic
Gordana Kozovic
DB2 System Administrators
Postal Savings Bank (Yugoslavia)

DB2 level display via TSO – revisited

When running the *DB2 level display via TSO* program from *DB2 Update* Issue 115, May 2002, an error occurred – the program abended with S0C4. I found that a change from AMODE 24 to AMODE 31 solved the problem.

Jürgen Schaffroth
Juergen.Schaffroth@kreditwerk.de

DB2 news

Quest Software has announced the release of Quest Central for DB2 Version 2.0, promising a unified suite for the management of DB2 databases on both mainframe and distributed platforms.

Version 2.0 extends support to DB2 running on z/OS and OS/390 operating systems with the same GUI used to manage DB2 on Windows, Unix, and Linux.

In addition to providing full support for the mainframe platform, the new version also offers a number of enhancements that extend the performance and tuning capabilities for DB2 on Windows and Unix. DBAs can now capture the complete database workloads to perform SQL analysis on statements, transactions, users' and applications.

Additionally, monitoring capabilities have been extended, allowing users to record and playback key performance metrics and SQL statements during a designated period of time so users can diagnose intermittent performance problems.

For further information contact:
Quest Software, 8001 Irvine Center Drive, Irvine, CA 92618, USA.
Tel: (949) 754 8000.
URL: http://www.quest.com/quest_central/db2.

* * *

IBM has announced 20 new database tools, including two SMART (Self Managing and Resource Tuning) tools to automate database performance tuning and database recovery time.

DB2 Recovery Expert for multi-platforms, provides automated recovery features with diagnostic and self-managing capabilities to minimize database outages, while DB2 Performance Expert consolidates, reports, analyses, and recommends SMART changes on DB2 performance related information.
New DB2 multi-platform tools including High Performance Unload, which unloads and extracts data from DB2 for movement across enterprise systems or for reorganization in place.

DB2 Table Editor accesses, updates, and deletes data across multiple DB2 database platforms, while DB2 Web Query Tool allows the connection of all users directly to multiple enterprise databases, securely and simultaneously, regardless of database size, hardware, operating system, or location. Both these tools now include support for the Informix Dynamic Server.

No fewer than 15 of the new tools are for the zSeries, among which are DB2 Administration Tool and DB2 High Performance Unload, a high-speed DB2 utility for unloading DB2 tables from either a tablespace or an image copy.

Also, DB2 SQL Performance Analyzer provides an analysis of SQL queries without executing them and helps tune queries to achieve maximum performance. It also makes it easier to reduce the escalating costs of database queries by estimating their cost prior to execution.

Meanwhile, DB2 Change Accumulation Tool creates image copies or change files without, it's claimed, impacting on-line operations.

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