72

DB2

October 1998

In this issue

update

DB2 Update

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

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

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

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

Editor
Robert Burgess

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

Subscriptions and back-issues
A year's subscription to *DB2 Update*, comprising twelve monthly issues, costs £245.00 in the UK; $365.00 in the USA and Canada; £251.00 in Europe; £257.00 in Australasia and Japan; and £255.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1994 issue, are available separately to subscribers for £21.00 ($31.00) each including postage.

Printed in England.

Preparing for DB2 Version 6

IBM announced DB2 Version 6 at the International DB2 User Group in May1998. As a part of this announcement, IBM is renaming the product DB2 Universal Database for OS/390, for the sake of consistency across its DB2 product line. IBM is planning a general availability date of June 1999.

This new release includes many new and exciting features including large objects, triggers, user-defined functions, user-defined data types, and more. However, an important item for every DB2 shop to understand is what is *not* in DB2 Version 6. For the first time IBM is removing features from DB2 – and you will need to prepare your DB2 subsystems for Version 6 by removing the soon-to-be-unsupported features from your installation. Let's examine each of the features that will be removed as from DB2 Version 6.

INDEXES

Two types of index are available to DB2 – type 1 and type 2. Type 2 indexes were introduced with DB2 Version 4 and should be the standard index type implemented in your shop. Most organizations already favour the creation of type 2 indexes over type 1 indexes because:

- Type 2 indexes eliminate index locking (the predominant cause of contention in most pre-Version 4 DB2 applications).
- Type 2 indexes do not use index subpages.
- Type 2 indexes are the only type supported for ASCII encoded tables.
- Many newer DB2 features cannot be used unless Type 2 indexes are used. These features include: row-level locking, data sharing, full partition independence, uncommitted reads, UNIQUE WHERE NOT NULL, and CPU and sysplex parallelism.

From DB2 Version 6, type 1 indexes will no longer be supported by DB2. All your indexes must be type 2 before migrating to DB2 Version 6. It is wise to begin this migration as soon as possible because

of the benefits outlined above. If you are on DB2 Version 3 or an earlier release you cannot implement type 2 indexes because they are not supported. In this case you should move to DB2 Version 4 or a later release as soon as possible to begin migrating your indexes to type 2 in preparation for DB2 Version 6.

To find all type 1 indexes, issue the following SQL statement:

```
SELECT CREATOR, NAME
FROM SYSIBM.SYSINDEXES
WHERE INDEXTYPE = ' ';
```

SHARED READ-ONLY DATA

Shared Read-Only Data (SROD) was provided as a new feature of DB2 in Version 2.3. SROD provided a way for the same DB2 database to be read by multiple DB2 subsystems without implementing distributed data or sysplex data sharing. However, the shared object must be started ACCESS(RO) and all data access is read-only. When the data needs to be updated only one of the subsystems, the one marked as the owner, can update the data.

SROD is complex to implement, limited in functionality, and not frequently implemented. From DB2 Version 6, SROD support is removed. To support SROD-like functionality you will need to convert to data distribution or data sharing. To find all SROD databases, issue the following SQL statement:

```
SELECT NAME, BPOOL, ROSHARE
FROM SYSIBM.SYSDATABASE
WHERE ROSHARE IN ('O', 'R');
```

RECOVER INDEX

Through DB2 Version 5, the RECOVER INDEX utility is used to re-create indexes from current data. RECOVER INDEX scans the table on which the index is based and regenerates the index based on the actual data. Indexes are always recovered from actual table data, not from image copy and log data.

With DB2 Version 6, the function of the RECOVER INDEX utility changes. Instead of rebuilding indexes from the current data, RECOVER INDEX will actually recover the index by reading an image copy of the index dataset. So, with DB2 Version 6 you can use

the COPY utility to make back-ups of DB2 indexes and the RECOVER utility to restore them.

To provide equivalent functionality for re-creating an index from the current data, IBM provides a new utility called REBUILD INDEX. The REBUILD INDEX utility works in exactly the same way as the RECOVER INDEX used to function.

Organizations should begin changing all of their current RECOVER INDEX jobs to use REBUILD INDEX syntax instead. The REBUILD INDEX syntax is available in DB2 Version 5 and Version 4 (with PTF PQ09842). After you migrate to DB2 Version 6, the RECOVER INDEX utility will cease to function if the proper index back-up copies are not available to use during recovery.

HOST VARIABLES WITHOUT COLONS

DB2 programmers know that host variables used in SQL statements in a program should be preceded by a colon. So, if a host variable is named 'HV', it should be coded in the SQL statement as ':HV'. However, many programmers do not know that, through Version 5, DB2 programs tolerate host variables that are not preceded by a colon. This feature ends as of DB2 Version 6.

The reasoning from IBM for eliminating this feature is that it is getting too difficult for DB2 to differentiate host variables from SQL. This is because of the rising complexity of SQL and the new features being added to DB2. Therefore, for DB2 Version 6 and onward, all host variables must be prefixed with a colon.

This change should not impact many programs because most organizations have DB2 standards that dictate all host variables will begin with a colon. However, because DB2 has tolerated host variables without a colon through DB2 Version 5, you should inspect all DB2 SQL statements in application programs to ensure compliance prior to migrating to DB2 Version 6.

DATASET PASSWORDS

A little-used feature of DB2 is the ability to provide security via dataset passwords. Using the DSETPASS key word of the CREATE

TABLESPACE and CREATE INDEX statement, it is possible to password-protect DB2 datasets.

This feature disappears with DB2 Version 6. If you need to protect your DB2 datasets outside DB2 security, you can use RACF, CA-ACF2, CA-Top Secret, or whatever security package you have installed at your site to accomplish this.

To find datasets that are password-protected using DSETPASS, issue the following SQL statement:

```
SELECT 'INDEX ', CREATOR, NAME
FROM SYSIBM.SYSINDEXES
WHERE DSETPASS <> '        '
UNION ALL
SELECT 'TSPACE', DBNAME, NAME
FROM SYSIBM.SYSTABLESPACE
WHERE DSETPASS <> '        '
```

STORED PROCEDURE REGISTRATION

After coding a stored procedure, you must register information about that stored procedure in the DB2 system catalog. This process is in sharp contrast to the manner in which other database objects are recorded in the system catalog. Typically, when an object is created, DB2 automatically stores the meta-data description of that object in the appropriate DB2 Catalog tables. For example, to create a new table the CREATE TABLE statement is issued and DB2 automatically records the information in multiple system catalog tables (SYSIBM.SYSTABLES, SYSIBM.SYSCOLUMNS, and possibly SYSIBM.SYSFIELDS).

Because stored procedures are not created within DB2, nor are they created using DDL, the database administrator must use SQL INSERT statements to populate the SYSIBM.SYSPROCEDURES system catalog table with the meta-data for the stored procedure.

The following SQL provides an example of an INSERT to register a stored procedure:

```
INSERT INTO SYSIBM.SYSPROCEDURES
  (PROCEDURE, AUTHID, LUNAME, LOADMOD, LINKAGE,
   COLLID, LANGUAGE, ASUTIME, STAYRESIDENT,
   IBMREQD, RUNOPTS, PARMLIST, RESULT_SETS,
   WLM_ENV, PGM_TYPE, EXTERNAL_SECURITY,
```

```
  COMMIT_ON_RETURN)
VALUES
 ('PROCNAME', ' ', ' ', 'LOADNAME', ' ',
  'COLL0001', 'COBOL', 0, 'Y',
  'N', ' ', 'NAME CHAR(20) INOUT', 1,
  ' ', 'M', 'N', 'N');
```

This SQL statement registers a stored procedure written in COBOL and named PROCNAME with a load module named LOADNAME. It uses a package with a collection-id of COLL0001. Any location can execute this procedure. The program stays resident and uses the DB2 SPAS (not Workload Manager), and no limit is set on the amount of time it can execute before being cancelled. Furthermore, the stored procedure uses one input/output parameter, and the parameter cannot be null.

This method of registering stored procedures changes in DB2 Version 6. Instead of the INSERT statement, CREATE and ALTER statements are provided for registering stored procedures to the DB2 system catalog. Additionally, a new catalog table named SYSIBM.SYSROUTINES replaces SYSIBM.SYSPROCEDURES. This new table will store information on triggers, user-defined functions, and stored procedures. The meta-data for all these 'routines' will be provided to the system catalog by means of DDL statements.

A number of organizations have implemented processes for creating and updating stored procedures that include registration. These processes will need to be modified for DB2 Version 6. Additionally, if your organization uses a third party tool to register or change stored procedure information, be sure that it will be changed to support the new DB2 Version 6 DDL syntax.

DB2 PRIVATE PROTOCOL DISTRIBUTED DATA

Distributed data support was added to DB2 as of Version 2.2. At that time, IBM had not formulated its DRDA framework. In DB2 Version 2.2, Distributed Unit of Work (DUW) capability was provided solely through a private protocol which did not support any industry standards. As of DB2 Version 3, both the private protocol DUW and full DRDA DUW is supported. Private protocol is also referred to as system-directed access and DRDA protocol is also referred to as application-directed access.

Application-directed data access is the more powerful of the two options. With application-directed access, explicit connections are required. Furthermore, application-directed distributed access conforms to the DRDA standard.

However, DB2 also provides system-directed access to distributed DB2 data. The system-directed access is less flexible than application-directed access because:

- It does not use the open, DRDA protocol, but instead uses a DB2-only private protocol.
- It is viable for DB2-to-DB2 distribution only.
- Connections cannot be explicitly requested, but are implicitly performed when distributed requests are initiated.

Although system-directed access does not conform to DRDA, it does provide the same levels of distributed support as application-directed access – remote request, RUW, and DUW. System-directed access is requested using three-part table names.

As of DB2 Version 6, three part names can be used with DRDA. This provides static SQL support for distributed requests using three-part names. DB2 private protocol distribution is still available with DB2 Version 6, but IBM has indicated that it will be removed in a future release – as such, you should consider migrating away from DB2 private protocol distribution.

SYNOPSIS

Version 6 is the first release of DB2 to take features out of the product. Organizations must understand what is being removed, know how to provide similar functionality with other DB2 features, and develop a plan to migrate away from the non-supported features. It is not too early to begin planning and migration now. The sooner you remove the old technology, the sooner you can move to the latest and greatest version of DB2 when it becomes available.

Craig S Mullins
VP Operations
PLATINUM Technology (USA)

Peculiarities of the CREATE statement clauses

This article will discuss the peculiarities of the CREATE statement clauses in DB2 for MVS Versions 4.1 and 5.1 that should be taken into consideration by the DBA. In addition, environmental considerations (such as DASD device geometry) as they pertain to the CREATE clauses will be presented.

DB2/MVS Version 4.1 and 5.1 are very similar in functionality and features. Version 5.1 has several SQL changes but only a few of these are directly related to indexes and tablespaces. In DB2/MVS Version 5.1, specific changes of interest were introduced to the following:

- The LARGE clause for the CREATE TABLESPACE statement.
- The PIECESIZE clause for the ALTER and CREATE INDEX statements.
- The CCSID clause for the CREATE DATABASE and CREATE TABLESPACE statements.

The most important clauses, and the peculiarities to be observed while creating DB2 tablespaces and/or indexspaces, are discussed below.

THE USING CLAUSE

The USING clause in the CREATE tablespace/indexspace statements indicates whether the dataset(s) that will support the tablespace/ indexspace are to be defined by the DBA (user) or by DB2 itself. This is accomplished by specifying either a USING clause with the VCAT parameter (for user defined datasets) or the STOGROUP parameter (for DB2 managed datasets). If DB2 is to define the dataset(s), the USING STOGROUP clause could be followed with space allocation parameters (PRIQTY, SECQTY) and an ERASE rule. If the USING clause is not specified at all, DB2 will define the dataset(s) using the default STOGROUP for the database, with default values for PRIQTY, SECQTY, and ERASE.

Most MVS installations use IBM's DFSMS/SMS, a product with multiple functional components such asDFSMS/DFP, DFSMS/DSS,

and DFSMS/HSM. This subsystem and its components can manage all or part of the DB2 datasets in your installation, depending on how it is implemented.

All VSAM Linear Datasets (LDSs) are processed by the VSAM part of DFSMS/DFP. When the USING clause is followed by the VCAT parameter, the DBA (user) is responsible for requesting the allocation of the required VSAM LDS using IDCAMS statements. When the USING clause is followed by the STOGROUP parameter, the DB2 DBMS will allocate the required VSAM LDS – applying the parameters specified in the USING clause, such as the PRIQTY and SECQTY.

There are a couple of potential advantages of using user-defined datasets, such as having the certainty of knowing where the datasets are being placed (which can be crucial for performance expectations), as well as having a tighter control on DASD utilization. These potential advantages can be reproduced with a careful implementation of DB2-managed datasets in connection with DFSMS.

DB2-managed datasets have two flavours – non-SMS-managed and SMS-managed. Non-SMS-managed implies that specific DASD volumes are assigned to DB2 storage groups (STOGROUPs). These specific DASD volumes have to be monitored by the DBA to ensure that sufficient space is available for application growth and normal operations, such as execution of the REORG utility, which deletes and recreates the underlying VSAM LDS.

SMS-managed implies that there are certain 'agreements' between DB2 and SMS on how to allocate and expand DB2 VSAM LDSs. These 'agreements' are supported by the DFSMS ACS routines, usually under the jurisdiction of your storage administration friends.

DB2 datasets managed by SMS is the environment that should be used in medium to large DB2 shops – although it may feel a bit uncomfortable not knowing where your DB2 datasets are placed out there in the DASD farm. It is also difficult to predict, monitor, and control dataset performance in an ever-changing environment. But having SMS-managed DB2 datasets certainly takes away a lot of the mundane work associated with dataset placement and space administration. It is also important to keep in mind that IBM and third-party companies are

continuously improving the functionality and capabilities of DFSMS and related system software.

The peculiarity of this item is that, for user-defined DB2 tablespaces and indexspaces, no PRIQTY or SECQTY values can be specified in the DDL. Thus, in addition to creating the DDL, the DBA must generate the IDCAMS statements necessary to create the VSAM LDSs that correspond to the CREATE DDL statements. For additional information on user-defined VSAM LDS, refer to the IBM publication *DB2 Administration Guide, Volume 1*, as well as the Define Cluster command in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

TYPES OF TABLESPACES

DB2/MVS has three types of tablespace – simple, segmented, and partitioned. The presence of the SEGSIZE or NUMPARTS clauses, or neither of them, determines the tablespace type. When neither the SEGSIZE nor NUMPARTS clause is specified, the tablespace to be created is a simple one. When the SEGSIZE is specified, the tablespace to be created is a segmented one. When NUMPARTS is specified, the tablespace to be created is partitioned.

In DB2 Version 4.1, the maximum size for *all* these tablespaces is 64GB. The type of tablespace to be used will dictate the maximum VALID primary quantity (PRIQTY) value that can be specified in the CREATE statement. In DB2/MVS Version 4.1, simple and segmented (non-partitioned) tablespaces can use a maximum PRIQTY value of 2GB; partitioned tablespaces can use a maximum PRIQTY value ranging from one to four gigabytes, depending on the number of partitions specified (from one to 64 partitions).

In DB2 Version 5.1, with the introduction of the CREATE LARGE TABLESPACE, the maximum number of partitions for a partitioned tablespace (only) has changed from 64 to 256, and the maximum size of each partition now can be 4GB, regardless of the number of partitions. These changes allow DB2 large partitioned tablespaces to be 1TB in size.

The peculiarity for this item is that, regardless of the type of tablespace

you are creating, you can specify a PRIQTY of 4GB (this can happen in both DB2 Version 4.1 and Version 5.1). Assuming that you have DASD devices with enough space, DFSMS will allocate a 4GB VSAM LDS, but DB2 will only use what it is supposed to. For example, for a non-partitioned tablespace, if you specify a PRIQTY of 3GB, DB2 will only use the first two gigabytes of the VSAM LDS – the rest will *always* remain unused. This waste will be compounded every time DB2 requests a new primary extent on another DASD volume with the 3GB PRIQTY. As you can see, this peculiarity can be very costly and can easily go undetected. I have been told that this problem will be corrected in DB2/MVS Version 6.

The following SQL query against the DB2 catalog will show whether you have any tablespaces that were created with improper PRIQTY values on a DB2 Version 4.1 subsystem:

```
SELECT A.DBNAME, A.NAME, A.PARTITIONS, A.SEGSIZE,  B.PARTITION, B.PQTY,
B.SQTY
 FROM SYSIBM.SYSTABLESPACE A,  SYSIBM.SYSTABLEPART  B
   WHERE ( ( A.DBNAME = B.DBNAME ) AND ( A.NAME = B.TSNAME ) ) AND
   ( ( ( A.PARTITIONS BETWEEN 17 AND 32 ) AND ( PQTY*4 > 2Ø97152) ) OR
     ( ( A.PARTITIONS BETWEEN 33 AND 64) AND ( PQTY*4 > 1Ø48576) ) OR
     ( ( A.PARTITIONS = Ø ) AND ( PQTY*4 > 2Ø97152 ) )
                    )
      ORDER BY A.DBNAME, A.NAME, B.PARTITION
            FOR FETCH ONLY;
```

The following SQL query against the DB2 catalog will show whether you have any indexspaces that were created with improper PRIQTY values on a DB2 Version 4.1 subsystem:

```
SELECT A.IXCREATOR, A.IXNAME, A.PARTITION, A.PQTY
     FROM SYSIBM.SYSINDEXPART  A
 WHERE
   (  ( A.PQTY*4  > 2Ø97152 ) AND
   ( 17 <= ( SELECT MAX(PARTITION)  FROM SYSIBM.SYSINDEXPART B
 WHERE ( A.IXCREATOR = B.IXCREATOR ) AND ( A.IXNAME = B.IXNAME ) )
            AND
   32 >= ( SELECT MAX(PARTITION)   FROM SYSIBM.SYSINDEXPART B
 WHERE ( A.IXCREATOR = B.IXCREATOR ) AND ( A.IXNAME = B.IXNAME )  )
        )
     )
 UNION
SELECT A.IXCREATOR, A.IXNAME, A.PARTITION, A.PQTY
     FROM SYSIBM.SYSINDEXPART   A
```

```
 WHERE
    (  ( A.PQTY*4  > 1Ø48576 ) AND
    ( 33 <= ( SELECT MAX(PARTITION)  FROM SYSIBM.SYSINDEXPART B
 WHERE ( A.IXCREATOR = B.IXCREATOR ) AND ( A.IXNAME = B.IXNAME  ) )
         AND
 64 >= ( SELECT MAX(PARTITION)  FROM SYSIBM.SYSINDEXPART B
 WHERE ( A.IXCREATOR = B.IXCREATOR ) AND ( A.IXNAME = B.IXNAME ) )
         )
     )
UNION
SELECT A.IXCREATOR, A.IXNAME, A.PARTITION, A.PQTY
     FROM SYSIBM.SYSINDEXPART   A
 WHERE
    (  ( A.PQTY*4  > 2Ø97152 ) AND
    ( Ø = ( SELECT MAX(PARTITION)  FROM SYSIBM.SYSINDEXPART B
 WHERE ( A.IXCREATOR = B.IXCREATOR ) AND ( A.IXNAME = B.IXNAME ) )
         )
    )
ORDER BY 1, 2, 3
FOR FETCH ONLY;
```

The following SQL query will show whether there are any tablespaces that were created with an incorrect PRIQTY value on a DB2 Version 5.1 subsystem:

```
SELECT A.DBNAME, A.NAME, A.PARTITIONS, A.SEGSIZE, B.PARTITION, B.PQTY,
B.SQTY
    FROM SYSIBM.SYSTABLESPACE A,
              SYSIBM.SYSTABLEPART    B
 WHERE ( ( A.DBNAME = B.DBNAME ) AND ( A.NAME = B.TSNAME ) ) AND
   ( A.TYPE NOT = 'L' ) AND
   ( ( ( A.PARTITIONS BETWEEN 17 AND 32 ) AND ( PQTY*4 > 2Ø97152) ) OR
   ( ( A.PARTITIONS BETWEEN 33 AND 64) AND ( PQTY*4 > 1Ø48576) ) OR
   ( ( A.PARTITIONS = Ø ) AND ( PQTY*4 > 2Ø97152 ) )
                     )
      ORDER BY A.DBNAME, A.NAME, B.PARTITION
            FOR FETCH ONLY;
```

In DB2/MVS Version 5.1, you can specify the maximum amount of DASD space that will be used by DB2 for non-partitioned indexes using the PIECESIZE clause. The previously listed query for Version 4.1 indexes will run on a DB2 Version 5.1 subsystem, but it will not take the PIECESIZE clause into consideration.

I believe there is no hard-and-fast rule on how you should choose between single, segmented, or partitioned tablespaces. The following

items are provided for your consideration when choosing which type of tablespace to use:

- Very small tables (fewer than 32 pages) with low update activity can be stored in segmented tablespaces with other small tables. Storing multiple tables on the same tablespace will reduce DB2 overhead at run-time (fewer datasets are opened). Possible drawbacks are that a load replace on a table will clear all other tables in the same tablespace, and that any operations/utilities that affect a single tablespace will affect all tables within it.
- Segmented tablespaces should be used for medium-sized tables of less than 2GB. The reasoning for this is that a single dataset can hold all of the data, reducing DB2 overhead. Segmented tablespaces have advantages over simple tablespaces, such as:
  - A mass delete of all rows of a table operates faster.
  - A COPY utility will not copy empty pages.
  - At insertion time, some read operations are avoided because the space map contains the required information.

  A drawback to using segmented tablespaces is the additional overhead of maintaining the tablespace space map. Note that performance requirements may push a medium size table to be implemented using a partitioned tablespace schema, where I/O parallelism could be engaged to improve DASD I/O throughput.
- Partitioned tablespaces should be used for large tables. Large is a term relative your specific installation – I would suggest that large is any multi-million row table or any tablespace with over 10GB. Partitioned tablespaces should also be used for any tables with high-performance requirements where I/O parallelism (and potentially, CPU parallelism) could be utilized.

  There are several advantages to using partitioned tablespaces, as well as some disadvantages. These are well documented in several publications, including the *DB2 Administration Guide*. Recently, there have been several articles and technical papers promoting the widespread use of partitioned tablespaces, regardless of tablespace size, in order to allow the DB2 Optimizer

to choose multiple I/O operations against the tablespaces/index-spaces.

Suffice to say, for large volumes of data or high-performance requirements, partitioned tablespaces should be the first choice to consider for implementation.

THE SEGSIZE CLAUSE

The presence of the SEGSIZE clause of the CREATE TABLESPACE statement indicates to DB2 that the tablespace is to be segmented. It also specifies how many pages are to be assigned to each segment. A segment is a logical grouping of physical pages of the tablespace that will store rows for a single table. Multiple tables can be stored in the same segmented tablespace. SEGSIZE values range from four to 64 in multiples of four.

In choosing a segment size for a tablespace, take into consideration the number of pages that will be used by each one of the tables to be stored in the tablespace, as well as the geometry of the DASD device where you are placing the tablespace. In addition, the SEGSIZE value influences the way DB2 reads pages while doing sequential prefetch (as documented in the *DB2 Administration Guide, Tablespace scans of Segmented Tablespaces)*.

Assuming the tablespace is managed by DB2, DB2 will allocate the tablespace using the PRIQTY and SECQTY values specified in the DDL. At the time the tablespace is created, DB2 initializes the first two pages of the tablespace (the header page, and the space map page) – no other pages in the tablespace are processed. At the time a new table is created in the tablespace, DB2 will obtain the amount of space necessary to store a segment, this space being equal to the tablespace page size multiplied by the SEGSIZE clause (let’s call this value the segment space value).

One of the peculiarities for this item is that the segment space value can interfere with the way DB2 allocates space. For example, if you specify a SEGSIZE of sixty-four 4KB pages, but only specify a PRIQTY of 48KB, when DB2 needs to allocate a segment, the segment space required will be 256KB. Because 256 is not evenly

divisible by 48, there will be some space allocated but never used. The net result will be some wasted DASD between segments. This problem can be alleviated by having a PRIQTY value specified that is larger than the segment space value, or by ensuring that the PRIQTY and/or SECQTY values will allocate space in tracks or cylinder boundaries.

I recommend that PRIQTY and SECQTY values for segmented tablespaces take three factors into consideration:

- The SEGSIZE value itself (because of the prefetch implications mentioned in the previous paragraph).
- The DASD device geometry.
- The tablespace page size (4KB or 32KB).

The relationship between DASD device geometry and the tablespace page size in reference to the SEGSIZE value has a relatively minor impact on the actual database design. It could have a significant impact on DASD utilization and/or database performance. The tables in Figure 1 and Figure 2 show the lowest common denominator in kilobytes between the segment space value (page size multiplied by SEGSIZE value) and the track and cylinder capacity for 3380 and 3390 devices, for each possible SEGSIZE value. These values represent the amount of space required to ensure that a segment is allocated in a track or cylinder boundary, thus reducing the chance of wasting DASD space between segments. Note that these values do not take into consideration the header page nor the space map page at the beginning of the tablespace, which amount to 8KB or 128KB depending on the page size.

The tables can be used as follows. Let us say, for example, you need to create a tablespace to store 800,000KB of data. You want to use a segmented tablespace, using buffer pool BP1 (4KB in size), and you know that you will be using 3390 devices. You choose the SEGSIZE that you think will be best for your application requirements, let's say 32, because you know you will be doing lots of tablespace scans (sequential prefetch reads). From the 3390 DASD table (Figure 2), the QTY value to ensure cylinder boundary utilization for a SEGSIZE of 32 is 5,760KB. Divide 800,000 by 5,760 and the result is 138.88. This

3380 DASD device geometry (40KB/track, 600KB/cylinder)

	4KB pages			32KB pages		
SEGSIZE value	Page size *SEGSIZE	QTY for tracks	QTY for cylinders	Page size *SEGSIZE	QTY for tracks	QTY for cylinders
4	16	80	1200	128	640	9600
8	32	160	2400	256	1280	19200
12	48	240	1200	384	1920	9600
16	64	320	4800	512	2560	38400
20	80	80	1200	640	3200	9600
24	96	480	2400	768	3840	19200
28	112	560	8400	896	4480	67200
32	128	640	9600	1024	5120	76800
36	144	720	3600	1152	5760	28800
40	160	160	2400	1280	6400	19200
44	176	880	13200	1408	7040	105600
48	192	960	4800	1536	7680	38400
52	208	1040	15560	1664	8320	124480
56	224	1120	16800	1792	8960	134400
60	240	240	1200	1920	9600	9600
64	256	1280	19200	2048	10240	153600

Figure 1: Amount of space required on 3380 device

3390 DASD device geometry (48KB/track, 720KB/cylinder)

SEGSIZE value	4KB pages Page size *SEGSIZE	QTY for tracks	QTY for cylinders	32KB pages Page size *SEGSIZE	QTY for tracks	QTY for cylinders
4	16	48	720	128	384	5760
8	32	96	1440	256	768	11520
12	48	48	720	384	384	5760
16	64	192	2880	512	1536	23040
20	80	240	720	640	1920	28800
24	96	288	1440	768	2304	34560
28	112	336	5040	896	2628	40320
32	128	384	5760	1024	3072	46080
36	144	432	2160	1152	3456	51840
40	160	480	1440	1280	3840	57600
44	176	528	7920	1408	4224	63360
48	192	576	2880	1536	4608	69120
52	208	624	9360	1664	4992	74880
56	224	672	10080	1792	5376	80640
60	240	720	720	1920	5760	86400
64	256	768	11520	2048	6144	92160

Figure 2: Amount of space required on 3390 device

is not an even number and therefore you should use 140 – multiply back by 5,760 and your new QTY value is 806,400.

THE PRIQTY CLAUSE

The PRIQTY clause will determine the minimum primary space to be allocated for a DB2-managed object, be it a tablespace or an indexspace. DB2 knows the page-size corresponding to the buffer pool being used in the CREATE statement, whether it is 4KB or 32KB, and computes the smallest multiple of the page-size not less than the PRIQTY value specified as the amount of space to be requested for the VSAM LDS allocation. For example, if the PRIQTY is 50, the buffer pool specified is BP0, and page-size is 4KB; 50 divided by four equals 12.5, so DB2 uses 13 * 4 = 52 as the value to pass to DFSMS/DFP for the allocation of the VSAM LDS.

The resulting space allocated by DFSMS could be greater than the space requested by DB2, because DFSMS/DFP allocates VSAM LDS in track or cylinder boundaries depending on the primary *and* secondary amounts requested for the allocation. Following the above example, assuming the VSAM LDS is to be created on a DASD device with 3380 geometry where each track can store ten 4KB pages, 52KB will require two tracks for storage. Two tracks represent 80KB, but DB2 is only using 52KB. The net result is some DASD wastage.

The PRIQTY value has minimum restrictions. The minimum requirements are dependent on the page-size of the tablespace, determined by the type of buffer pool specified in the CREATE statement. Specifically, for tablespaces and indexspaces that will use a 4KB size buffer pool (BP0 through BP49), the minimum amount of space that can be requested is 12KB. If you specify less than the minimum, DB2 will automatically adjust the value at allocation time.

Tablespaces are the only DB2 objects than can use the 32KB buffer pools (BP32K, BP32K1-BP32K9). The minimum amount of space that can be requested for a tablespace using 32KB size buffer pool is 96KB. Again, if you specify less than the minimum, DB2 will automatically adjust the value at allocation time.

The PRIQTY value also has maximum restrictions, as previously

discussed. For additional information refer to the *SQL Reference* manual for the specific DB2 version in use.

The PRIQTY value also has a close relationship with the SEGSIZE clause, as previously discussed. I suggest that you spend a couple of hours at your installation, experimenting with different PRIQTY values and SEGSIZE values to get a clear idea of the relationship of these two clauses.

An external factor that you could specify that has an influence on the PRIQTY values is the DFSMS/DFP limitation that VSAM datasets can't exceed 123 extents across multiple volumes. For DB2, the number of extents is reduced to 118, because either the primary space allocation, or each secondary space allocation, of a VSAM dataset could be fulfilled by DFSMS/DFP using up to five DASD extents. DB2 needs the five extents cushion to avoid the potential problem of having an excessive number of extents on a dataset. For additional information, refer to the Define Cluster command in the *DFSMS/MVS: Access Method Services for VSAM catalog.*

Additionally, the PRIQTY value itself will have an impact on how much space DFSMS/DFP will allocate for the VSAM LDS on DB2's behalf. Bear in mind how DFSMS allocates the VSAM dataset. Firstly, it allocates the primary space on a candidate volume (in one to five extends, depending on how fragmented the DASD device was at the time of the allocation request). When the space allocated is exhausted, if the VSAM dataset has not reached the 118 extends limit, DB2 will request from DFSMS that the space allocated to the VSAM LDS be expanded by an additional SECONDARY QUANTITY (SECQTY) amount. If there is no space available on the current volume, DFSMS will find another candidate volume in either the STOGROUP's DASD POOL or in the SMS pool, depending on the installation parameters. Once a candidate volume is found, DFSMS will allocate the new extend on the new volume using the primary space allocation quantity. This peculiarity of VSAM allocation logic can generate a serious amount of DASD waste. For example, a segmented tablespace will require 1.25GB of DASD and the PRIQTY is defined at 1.25GB and SECQTY is defined at 0.125GB. After the primary space allocated is used, DB2 will request for an extent of

0.125GB. Let us assume the DASD volume is full. DFSMS will be forced to look for available space on another volume and it will allocate another 1.25GB. The total space allocated is 2.5GB – but you know that for segmented tablespaces DB2 will only use the first two gigabytes of the dataset, and it will never use that extra 0.5GB allocated on the second space. Thus, more DASD is wasted. You can prevent this by having some standards at your installation that minimize the risk of over-allocating DASD for DB2 datasets. The trade-off is that your DB2 datasets will have more extents.

As you can see, there are several peculiarities pertaining to the PRIQTY clause. The PRIQTY clause has 'relationships' with other clauses of the CREATE statements that sometimes are not obvious to the DBA. External factors such as DASD device geometry and DFSMS/DFP behaviour will have a significant impact on how well DB2 uses the space allocated.

An important aspect of choosing and specifying the PRIQTY values is its operational implications. For example, if a DB2 tablespace or indexspace is created with an insufficient PRIQTY value, and the underlying VSAM dataset(s) reach the maximum number of VSAM extends, 'Resource unavailable' messages will start appearing and someone will end up calling you in the middle of the night.

Similarly, if the DB2 object was created with a large PRIQTY value, it is possible that, during the next execution of the REORG utility, the utility won't be able to get the space it requires to re-allocate the VSAM LDS, and the utility job will terminate abnormally. Database growth happens randomly, and the only way to avoid 'Resource unavailable' problems is by conducting periodic reviews of your different tablespaces and indexspaces to ensure they are not expanding into a critical number of extents, or that they are not about to exhaust your DASD pool.

So, how do you determine the appropriate PRIQTY for your DB2 tablespaces and indexspaces? I believe a basic requirement is a clear knowledge and understanding of how the storage environment at your shop is set up to handle DB2 databases, and to review that environment on a regular basis. What types of DASD devices are available? What is the average amount of contiguous space regularly available on your

DASD devices? What happens to your DASD pools during month-end processing or other special days? How fast are your applications growing, etc?

I have several recommendations for PRIQTY values:

- You should never take the defaults. Always ensure that the PRIQTY values are allocated in track or cylinder boundaries, since that is how DFSMS/DFP allocates datasets. Make sure that the PRIQTY has a good 'relationship' with the SEGSIZE clause (if any) as well as with the DASD device geometry.
- The maximum PRIQTY value should match or be less than the average amount of free space that you will find on your DASD pool at *any* time. For example, if you are using SMS to handle the allocation and placement of your DB2 datasets, and the average amount of free space in these volumes is around 1GB, make sure that you don't set your PRIQTY value to a value larger than 1GB. If you do, it is very likely that reorganization utility jobs would be unable to re-allocate the VSAM datasets it had just deleted – because there was no single DASD volume in the SMS pool that could satisfy the PRIQTY amount requested. This recommendation also allows the data to spread across multiple devices, which potentially could help to minimize DASD contention.
- For large tablespaces (ie more than 10GB), consider requesting a pool of dedicated DASD volumes from your storage administration friends. These large tablespaces may require this special attention in order to facilitate operations (such as to avoid no space available error conditions), as well as to specifically place the datasets on certain DASD devices/controllers for performance reasons.
- Ensure that the SECQTY value will not affect the way DFSMS will allocate the VSAM LDS. Specifically, DFSMS/DFP will determine the size of the Control Area (CA) to be allocated based on the *smaller* of the two allocation quantities (primary or secondary) specified in the define command. Thus, if your SECQTY value is less than a cylinder, even if your PRIQTY is over a cylinder, the VSAM LDS will be allocated in tracks instead

of cylinders, thus potentially creating a storage and/or performance problem. For more details, refer to the Define Cluster IDCAMS command.

THE SECQTY CLAUSE

The SECQTY value specifies the amount of space DB2 will request when there is no more space to be used. The SECQTY value can't be larger than 128MB. I have been told that this limitation will be removed in DB2 Version 6.

I like to specify the SECQTY value depending on the PRIQTY specified. If the PRIQTY size is larger than 512MB, I like to specify the maximum SECQTY value (128MB). For PRIQTY values of less than 512MB, I suggest that the SECQTY value is equal to 20 percent of the PRIQTY value, or that the maximum allowed SECQTY value is specified, such as in the case of tables with heavy insertion activity. Always ensure that the SECQTY value uses track or cylinder boundaries, so it will not impact the way the VSAM LDS is allocated, as mentioned previously.

CONCLUSION

DASD I/O is still the slowest part of a DBMS engine. Efficient allocation and utilization of DASD resources can have significant performance implications for a DB2 database. The less I/O performed against a tablespace/indexspace, the better an application will perform.

DASD utilization and management is changing rapidly and significantly with the introduction of RAID technologies. For example, IBM's RVA DASD device uses hardware compression to reduce the actual amount of DASD needed to store data. Typical reduction rates for DB2 datasets are in the order of one to four, depending on the type of data as well as the tablespace characteristics, such as FREEPAGE and FREEPCT. Compression is performed at the DASD device itself and neither DB2 nor the MVS operating system is aware of this compression happening. In addition, the concept of DASD device geometry loses a lot of relevance on these new devices, because, although they do emulate it, it does not affect the way they store and retrieve the data. Similarly, in relation to performance, dataset extends

lose their importance, because the RVA devices store data all over multiple small disks, so extends are only pertinent as far as the 118 limitation.

As far as DB2 is concerned, the implications of these new technologies are many. First of all, many of the ideas and concepts expressed in this article will become irrelevant. In time, the newer DASD subsystems, with automatic hardware compression features, will take care of the space allocated versus space used issue – since it will compress allocated but not used space to a very small amount of actual DASD space. However, for those of us still using *real* 3380 and 3390 devices, most of the items listed above are still relevant.

I believe that understanding your storage administration environment as well as the peculiarities of the DB2/MVS DBMS CREATE clauses described above, will produce a better DB2 database – one that can use DASD resources as efficiently as possible, and that could potentially be a better performer.

Antonio Salcedo
Lead System Programmer/DBA (USA)

Simulating a production environment – part 2

This month we continue the article on simulating a production environment in DB2.

```
*-------------------------------------
*    GENERAL STRUCTURE OF PROGRAM                                -
*-------------------------------------
*
 0000-MAIN.

     ACCEPT W-PARM FROM SYSIN.
     DISPLAY 'W-PARM-DBNAME'  W-PARM-DBNAME.
     DISPLAY 'W-PARM-TBOWNER' W-PARM-TBOWNER.
     PERFORM A100-BEGIN
        THRU A100-BEGIN-EXIT.
     PERFORM B100-INITIALIZE
        THRU B100-INITIALIZE-EXIT.
     MOVE 0 TO PROCESS-END.
```



```
     PERFORM C100-READ-SYSTBLS
        THRU C100-READ-SYSTBLS-EXIT
             UNTIL PROCESS-END = 1.
*
     EXEC SQL
        COMMIT
     END-EXEC.
*
     MOVE 0 TO PROCESS-END.
     PERFORM C200-READ-SYSTBLSP
        THRU C200-READ-SYSTBLSP-EXIT
             UNTIL PROCESS-END = 1.
*
     EXEC SQL
        COMMIT
     END-EXEC.
*
     MOVE 0 TO PROCESS-END.
     PERFORM C300-READ-SYSINDXS
        THRU C300-READ-SYSINDXS-EXIT
             UNTIL PROCESS-END = 1.
*
     EXEC SQL
        COMMIT
     END-EXEC.
*
     MOVE 0 TO PROCESS-END.
     PERFORM C400-READ-SYSCOLMS
        THRU C400-READ-SYSCOLMS-EXIT
             UNTIL PROCESS-END = 1.
*
     EXEC SQL
        COMMIT
     END-EXEC.
*
     MOVE 0 TO PROCESS-END.
     PERFORM C500-READ-SYSCOLDT
        THRU C500-READ-SYSCOLDT-EXIT
             UNTIL PROCESS-END = 1.
*
     PERFORM Z100-END
        THRU Z100-END-EXIT.

*

*-------------------------------------
*    OPEN FILES
*-------------------------------------
*
 A100-BEGIN.
*
```

```
*    READY TRACE.
     OPEN INPUT SYSTBLS
                SYSTBLSP
                SYSINDXS
                SYSCOLMS
                SYSCOLDT.
     OPEN OUTPUT REPOUT.
*
 A100-BEGIN-EXIT.
     EXIT.
*
*-------------------------------------
*    INITIALIZE VARIABLE FIELDS
*-------------------------------------
*
 B100-INITIALIZE.
*
     MOVE SPACES TO W-SYSTBLS
                    W-SYSTBLSP
                    W-SYSINDXS
                    W-SYSCOLMS
                    W-SYSCOLDT.
*
     MOVE ZEROS TO W-TBL-CARD-NUM
                    W-TBL-NPAGES-NUM
                    W-TBL-PCTRCOMP-NUM
                    W-TBLSP-NACTIVE-NUM
                    W-CLUSTERRATIO-NUM
                    W-FIRSTKEY-CARD-NUM
                    W-FULLKEY-CARD-NUM
                    W-NLEAF-NUM
                    W-NLEVELS-NUM
                    W-COLCARD-NUM
                    W-FREQUENC-NUM.
*
 B100-INITIALIZE-EXIT.
     EXIT.
*
*-------------------------------------
*    READ INPUT FILES RESULTANT OF THE RUN OF PGM PCATV3EX
*-------------------------------------
  C100-READ-SYSTBLS.
     READ SYSTBLS  AT END
                   MOVE 1 TO PROCESS-END
                   GO TO C100-READ-SYSTBLS-EXIT.
     IF FILE-STATUS = 00
        NEXT SENTENCE
     ELSE
        PERFORM Z100-END
           THRU Z100-END-EXIT.
*
```

```
     MOVE R-SYSTBLS      TO W-SYSTBLS.
     MOVE W-TBL-CARD         TO W-TBL-CARD-NUM.
     MOVE W-TBL-NPAGES       TO W-TBL-NPAGES-NUM.
     MOVE W-TBL-PCTRCOMP     TO W-TBL-PCTRCOMP-NUM.
*
     PERFORM D100-UPDATE-SYSTBLS
        THRU D100-UPDATE-SYSTBLS-EXIT.
*
 C100-READ-SYSTBLS-EXIT.
     EXIT.
 C200-READ-SYSTBLSP.
     READ SYSTBLSP   AT END
                  MOVE 1 TO PROCESS-END
                  GO TO C200-READ-SYSTBLSP-EXIT.
     IF FILE-STATUS = 00
        NEXT SENTENCE
     ELSE
        PERFORM Z100-END
           THRU Z100-END-EXIT.

     MOVE R-SYSTBLSP     TO W-SYSTBLSP.
     MOVE W-TBLSP-NACTIVE TO W-TBLSP-NACTIVE-NUM.
*
     PERFORM D200-UPDATE-SYSTBLSP
        THRU D200-UPDATE-SYSTBLSP-EXIT.
*
 C200-READ-SYSTBLSP-EXIT.
     EXIT.
 C300-READ-SYSINDXS.
*
     READ SYSINDXS   AT END
                  MOVE 1 TO PROCESS-END
                  GO TO C300-READ-SYSINDXS-EXIT.
     IF FILE-STATUS = 00
        NEXT SENTENCE
     ELSE
        PERFORM Z100-END
           THRU Z100-END-EXIT.
*
     MOVE R-SYSINDXS     TO W-SYSINDXS.
     MOVE W-CLUSTERRATIO   TO W-CLUSTERRATIO-NUM.
     MOVE W-FIRSTKEY-CARD  TO W-FIRSTKEY-CARD-NUM.
     MOVE W-FULLKEY-CARD   TO W-FULLKEY-CARD-NUM.
     MOVE W-NLEAF          TO W-NLEAF-NUM.
     MOVE W-NLEVELS        TO W-NLEVELS-NUM.
*
     PERFORM D300-UPDATE-SYSINDXS
        THRU D300-UPDATE-SYSINDXS-EXIT.
*
 C300-READ-SYSINDXS-EXIT.
     EXIT.
```

```
*
 C400-READ-SYSCOLMS.
*
     READ SYSCOLMS   AT END
                  MOVE 1 TO PROCESS-END
                  GO TO C400-READ-SYSCOLMS-EXIT.
     IF FILE-STATUS = 00
        NEXT SENTENCE
     ELSE
        PERFORM Z100-END
           THRU Z100-END-EXIT.

     MOVE R-SYSCOLMS     TO W-SYSCOLMS.
     MOVE W-COLCARD      TO W-COLCARD-NUM.
*
     PERFORM D400-UPDATE-SYSCOLMS
        THRU D400-UPDATE-SYSCOLMS-EXIT.
*
 C400-READ-SYSCOLMS-EXIT.
     EXIT.
*
*
 C500-READ-SYSCOLDT.
*
     READ SYSCOLDT   AT END
              MOVE 1 TO PROCESS-END
              GO TO C500-READ-SYSCOLDT-EXIT.
*
     IF FILE-STATUS = 00
        MOVE R-SYSCOLDT TO W-SYSCOLDT
        MOVE W-FREQUENC TO W-FREQUENC-NUM
     ELSE
        PERFORM Z100-END
           THRU Z100-END-EXIT.
*
     IF W-COLDT-TBNAME NOT = W-PREV-TBNAME
     OR W-COLDT-NAME NOT = W-PREV-COLNAME
        IF W-PREV-TBNAME = SPACES
            EXEC SQL
               OPEN FETCH_SYSCOLDT
            END-EXEC
        ELSE
            EXEC SQL
               CLOSE FETCH_SYSCOLDT
            END-EXEC
            EXEC SQL
               OPEN FETCH_SYSCOLDT
            END-EXEC.
*
     MOVE W-COLDT-TBNAME TO W-PREV-TBNAME
     MOVE W-COLDT-NAME TO W-PREV-COLNAME
```

```
      *
           PERFORM C510-FETCH-SYSCOLDT
              THRU C510-FETCH-SYSCOLDT-EXIT.
      *
      *
       C500-READ-SYSCOLDT-EXIT.
           EXIT.
      *
      *

 C510-FETCH-SYSCOLDT.
      *
           MOVE W-COLDT-TBNAME TO NAME-TABLE.
      *
           EXEC SQL
               FETCH  FETCH_SYSCOLDT
                  INTO  :W-FETCH-COLDT-TBNAME,
                        :W-FETCH-COLDT-NAME,
                        :W-FETCH-COLVALUE,
                        :W-FETCH-FREQUENC
           END-EXEC.
      *
           IF SQLCODE = 0
              PERFORM D500-UPDATE-SYSCOLDT
                 THRU D500-UPDATE-SYSCOLDT-EXIT
           ELSE
              IF SQLCODE = +100
                 PERFORM X100-DBERROR-DB2
                    THRU X100-DBERROR-DB2-EXIT.
      *
       C510-FETCH-SYSCOLDT-EXIT.
           EXIT.
      *-------------------------------------
      *    UPDATES SYSIBM.SYSTABLES                                   -
      *-------------------------------------
      *
       D100-UPDATE-SYSTBLS.
      *
           MOVE W-TBL-NAME TO NAME-TABLE
      *
           EXEC SQL
               UPDATE  SYSIBM.SYSTABLES
               SET
                 CARD = :W-TBL-CARD-NUM-R,
                 NPAGES = :W-TBL-NPAGES-NUM-R,
                 PCTROWCOMP = :W-TBL-PCTRCOMP-NUM-R
               WHERE CREATOR = :W-PARM-TBOWNER
                 AND NAME = :W-TBL-NAME
           END-EXEC.
      *
           IF SQLCODE = 0
```

```
           ADD 1 TO COUNT-TBL-UPD
        ELSE
           IF SQLCODE = +100
              PERFORM X100-DBERROR-DB2
                 THRU X100-DBERROR-DB2-EXIT.
*
 D100-UPDATE-SYSTBLS-EXIT.
        EXIT.
*

*-------------------------------------
*    UPDATES SYSIBM.SYSTABLESPACE                                  -
*-------------------------------------
*
 D200-UPDATE-SYSTBLSP.
*
        MOVE W-TBLSP-NAME TO NAME-TABLE
*
        EXEC SQL
            UPDATE  SYSIBM.SYSTABLESPACE
            SET
              NACTIVE = :W-TBLSP-NACTIVE-NUM-R
            WHERE DBNAME = :W-PARM-DBNAME
              AND NAME = :W-TBLSP-NAME
        END-EXEC.
*
        IF SQLCODE = 0
           ADD 1 TO COUNT-TBL-UPD
        ELSE
           IF SQLCODE = +100
              PERFORM X100-DBERROR-DB2
                 THRU X100-DBERROR-DB2-EXIT.
*
 D200-UPDATE-SYSTBLSP-EXIT.
        EXIT.
*
*-------------------------------------
*    UPDATES SYSIBM.SYSINDEXES
*-------------------------------------
*
 D300-UPDATE-SYSINDXS.
*
        MOVE W-INDXS-NAME TO NAME-TABLE
*
        EXEC SQL
            UPDATE  SYSIBM.SYSINDEXES
            SET
              FIRSTKEYCARD = :W-FIRSTKEY-NUM-R,
              FULLKEYCARD  = :W-FULLKEY-NUM-R,
              NLEAF = :W-NLEAF-NUM-R,
              NLEVELS = :W-NLEVELS-NUM-R,
```

```
           CLUSTERRATIO = :W-CLUSTERRAT-NUM-R
         WHERE TBCREATOR = :W-PARM-TBOWNER
           AND TBNAME = :W-INDXS-TBNAME
           AND NAME = :W-INDXS-NAME
     END-EXEC.
*
     IF SQLCODE = 0
        ADD 1 TO COUNT-TBL-UPD
     ELSE
        IF SQLCODE = +100
           PERFORM X100-DBERROR-DB2
              THRU X100-DBERROR-DB2-EXIT.
*
 D300-UPDATE-SYSINDXS-EXIT.
     EXIT.
*

*-------------------------------------
*    UPDATES SYSIBM.SYSCOLUMNS
*-------------------------------------
*
 D400-UPDATE-SYSCOLMS.
*
     MOVE W-COLMS-NAME TO NAME-TABLE
*
     EXEC SQL
         UPDATE  SYSIBM.SYSCOLUMNS
         SET
           COLCARD  = :W-COLCARD-NUM-R,
           LOW2KEY  = :W-LOW2KEY,
           HIGH2KEY = :W-HIGH2KEY
         WHERE TBCREATOR = :W-PARM-TBOWNER
           AND TBNAME = :W-COLMS-TBNAME
           AND NAME = :W-COLMS-NAME
        END-EXEC.
*
     IF SQLCODE = 0
        ADD 1 TO COUNT-TBL-UPD
     ELSE
        IF SQLCODE = +100
           PERFORM X100-DBERROR-DB2
              THRU X100-DBERROR-DB2-EXIT.
*
 D400-UPDATE-SYSCOLMS-EXIT.
     EXIT.
*
*-------------------------------------
*    UPDATES SYSIBM.SYSCOLDIST
*-------------------------------------
*
 D500-UPDATE-SYSCOLDT.
```

```
*
     MOVE W-COLDT-NAME TO NAME-TABLE
*
     EXEC SQL
         UPDATE  SYSIBM.SYSCOLDIST
         SET
           COLVALUE  = :W-COLVALUE,
           FREQUENCY = :W-FREQUENC-NUM-R
         WHERE CURRENT OF FETCH_SYSCOLDT
     END-EXEC.
*
     IF SQLCODE = 0
        ADD 1 TO COUNT-TBL-UPD
     ELSE
        IF SQLCODE = +100
           PERFORM X100-DBERROR-DB2
              THRU X100-DBERROR-DB2-EXIT.
*
 D500-UPDATE-SYSCOLDT-EXIT.
     EXIT.
*
*

*-------------------------------------
*    CALL DB2 ERROR ROUTINE                                          -
*-------------------------------------
*
 X100-DBERROR-DB2.
*
     MOVE SQLCODE TO ERROR-CODE.
     DISPLAY MSG2.

     PERFORM X200-CALL-MODULE
        THRU X200-CALL-MODULE-EXIT

     PERFORM Z100-END
        THRU Z100-END-EXIT.
*
 X100-DBERROR-DB2-EXIT.
     EXIT.
*
 X200-CALL-MODULE.
*
     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 > 8.
*
```

```
           GO X200-CALL-MODULE-EXIT.
      *
      ****************************************************************
      * PRINT MESSAGE TEXT                                           *
      ****************************************************************
       ERROR-PRINT.
      *
           WRITE REPREC FROM ERROR-TEXT (ERROR-INDEX)
              AFTER ADVANCING 1 LINE.
      *
       X200-CALL-MODULE-EXIT.
           EXIT.
      *-------------------------------
      *    PROGRAM END                                              -
      *-------------------------------
      *
       Z100-END.
      *
      *    RESET TRACE.

           CLOSE SYSTBLS
                 SYSTBLSP
                 SYSINDXS
                 SYSCOLMS
                 SYSCOLDT
                 REPOUT.

           DISPLAY '************************************'
           DISPLAY 'UPDATED RECORDS' COUNT-TBL-UPD
           DISPLAY '************************************'.

           STOP RUN.
      *
       Z100-END-EXIT.
           EXIT.
```

JCL TO RUN PROGRAMS

```
//RUNSYS   JOB (ACCT#),'CATLG PROGS',MSGLEVEL=(1,1),CLASS=x,
//         MSGCLASS=x,NOTIFY=&SYSUID,TIME=1440
//*
//JOBLIB   DD  DSN=DSN310.SDSNEXIT,DISP=SHR
//         DD  DSN=DSN310.SDSNLOAD,DISP=SHR
//         DD  DSN=SYS1.COB2LIB,DISP=SHR
//*****************************************************
//*   RUN CATALOG PROGRAMS
//*****************************************************
//STEP1   EXEC PGM=IKJEFT01,DYNAMNBR=20,REGION=3M
//SYSTSPRT DD  SYSOUT=*
```

```
//SYSPRINT DD  SYSOUT=*
//SYSABEND DD  SYSOUT=*
//SYSABOUT DD  SYSOUT=*
//SYSDBOUT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSTBLS  DD  DSN=TEST.SYSTBLS,DISP=SHR
//SYSTBLSP DD  DSN=TEST.SYSTBLSP,DISP=SHR
//SYSINDXS DD  DSN=TEST.SYSINDXS,DISP=SHR
//SYSCOLMS DD  DSN=TEST.SYSCOLMS,DISP=SHR
//SYSCOLDT DD  DSN=TEST.SYSCOLDT,DISP=SHR
//REPOUT   DD  DSN=TEST.REPOUT,DISP=SHR
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(xxxx)
 RUN  PROGRAM(PCATV3EX/PCATV3UP) PLAN9SYSDPGB) -
      LIB('TEST.DB2.LOAD')
 END
//SYSIN    DD *
xxxxxxxxyyyyyyyy
//*
```

Note: the following fields are variable according to your installation:

- ‘xxxxxxxx’ is the database name.
- ‘yyyyyyyy’ is the creator of the tables and indexes.

Before you run the program(s), you must bind a plan with two PACKAGES or two DBRMS – one for each program.

DB2CATSP

DB2CATSP displays values from the SYSTABLESPACE output file produced by the catalog extraction program PCATV3UP.

```
/* REXX */
DD1="I"TIME("S")
DD2="O"TIME("S")
DSN1="'xxxxxxxx.SYSTBLSP'"
DSN2="'xxxxxxxx.SYSTBLOP'"

"ALLOC F("DD1") SHR REUSE DA("DSN1")"
"EXECIO * DISKR "DD1" (FINIS STEM TBL.)"
"FREE F("DD1")"
/*****  READS THRU INPUT FILE UNTIL SPECIFIED RECORD IS REACHED *****/

NM=Ø
SAY 'DO YOU WANT THE WHOLE DATABASE?'
PULL VAR1
```

```
IF VAR1='YES' THEN
   NOP
ELSE
  DO
   SAY 'WHAT TABLESPACE DO YOU WANT TO CHANGE?'
   PULL VAR2
  END
DO A=1 TO TBL.Ø
   PARSE VALUE TBL.A WITH 1 NAME 7 . ,
                          9 NACTIVE 13 .
   DO
     IF VAR1='YES' THEN
        NOP
     ELSE
        IF VAR2 ¬= NAME then
           ITERATE
        ELSE
           NOP
   END

/*******  PROCESSES INPUT VALUES ******/

   DO
     NACTIVE=C2X(NACTIVE)
     DO
       IF NACTIVE='FFFFFFFF'  THEN
           NOP
       ELSE
           NACTIVE=X2D(NACTIVE)
     END

/*******  DISPLAYS SPECIFIC FILE RECORD *********/

     LIST.SCR=LEFT(NAME,8)'  'RIGHT(NACTIVE,8,Ø)
     SAY 'TBLSP     NACTIVE  '
     SAY LIST.SCR
     PULL . NACTIVE .

/*******  PREPARES CHANGED FIELDS TO REWRITE FILE RECORD ******/

     IF NACTIVE¬='' THEN
        DO
          NAME=LEFT(NAME,8)
          IF NACTIVE/='FFFFFFFF'  &,
             DATATYPE(NACTIVE)=NUM  THEN
             NACTIVE=D2C(NACTIVE,4)
          ELSE
             IF NACTIVE='FFFFFFFF' THEN
                NACTIVE=COPIES(X2C('FF'),4)
             ELSE
                DO
```

```
                  A=A-1
                  SAY '***ERROR: NACTIVE IS INVALID'
                  ITERATE
                END

          NM=NM+1
          PARSE VALUE NAME||LEFT(NACTIVE,9,Ø)  WITH TBLO
          TBLO.NM=TBLO
        END
     ELSE
        NOP

/*******  PROCESSES NEXT RECORD ON INPUT FILE ******/

     IF VAR1='YES' THEN
        NOP
     ELSE
        DO
         SAY 'WHAT TABLESPACE DO YOU WANT TO CHANGE? '
         PULL VAR2
         A=1
         ITERATE
        END
   END
END

"ALLOC F("DD2") SHR REUSE DA("DSN2")"
"EXECIO "NM" DISKW "DD2" (FINIS STEM TBLO.)"
"FREE F("DD2")"
RETURN

/* REXX */
```

DB2CATTB

DB2CATTB displays values from the SYSTABLES output file result produced by the extraction program PCATV3EX.

```
/* REXX */

DD1="I"TIME("S")
DD2="O"TIME("S")
DSN1="'XXXXXXXX.SYSTBLS'"
DSN2="'XXXXXXXX.SYSTBLSO'"
"ALLOC F("DD1") SHR REUSE DA("DSN1")"
"EXECIO * DISKR "DD1" (FINIS STEM TBL.)"
"FREE F("DD1")"

/*****  READS THRU INPUT FILE UNTIL SPECIFIED RECORD IS REACHED *****/
```

```
NM=Ø
SAY 'DO YOU WANT THE WHOLE DATABASE?'
PULL VAR1
IF VAR1='YES' THEN
   NOP
ELSE
  DO
   SAY 'WHAT TABLE DO YOU WANT TO CHANGE?'
   PULL VAR2
  END
DO A=1 TO TBL.Ø
   PARSE VALUE TBL.A WITH 1 NAME 7 . ,
                          19 CARD 23 . ,
                          28 NPAGES 32 . ,
                          37 COMP 39 .
   DO
     IF VAR1='YES' THEN
        NOP
     ELSE
        IF VAR2 ¬= NAME THEN
           ITERATE
        ELSE
           NOP
   END

/*******  PROCESSES INPUT VALUES ******/
   DO
     CARD=C2X(CARD)
     NPAGES=C2X(NPAGES)
     COMP=C2X(COMP)
     DO
       IF  CARD='FFFFFFFF'  THEN
           NOP
       ELSE
           CARD=X2D(CARD)
       IF  NPAGES='FFFFFFFF'  THEN
           NOP
       ELSE
           NPAGES=X2D(NPAGES)
       IF  COMP='FFFF'  THEN
           NOP
       ELSE
           COMP=X2D(COMP)
     END

/*******  DISPLAYS SPECIFIC FILE RECORD *********/

     LIST.SCR=LEFT(NAME,8)' 'RIGHT(CARD,8,Ø)'  ',
              RIGHT(NPAGES,8,Ø)'     'RIGHT(COMP,4,Ø)
     SAY 'TABLE    CARD        NPAGES    PCTCOMP'
```

```
     SAY LIST.SCR
     PULL . CARD NPAGES COMP .


/*******  PREPARES CHANGED FIELDS TO REWRITE FILE RECORD ******/

     IF CARD¬='' THEN
        DO
          NAME=LEFT(NAME,18)
          IF  CARD/='FFFFFFFF' &,
              DATATYPE(CARD)=NUM  THEN
              CARD=D2C(CARD,4)
          ELSE
             IF  CARD='FFFFFFFF' THEN
                 CARD=COPIES(X2C('FF'),4)
             ELSE
                 DO
                   A=A-1
                   SAY '***ERROR: CARD IS INVALID'
                   ITERATE
                 END

          IF  NPAGES/='FFFFFFFF' &,
              DATATYPE(NPAGES)=NUM  THEN
              NPAGES=D2C(NPAGES,4)
          ELSE
             IF  NPAGES='FFFFFFFF' THEN
                 NPAGES=COPIES(X2C('FF'),4)
             ELSE
                 DO
                   A=A-1
                   SAY '***ERROR: NPAGES IS INVALID'
                   ITERATE
                 END

          IF  COMP/='FFFF' &,
              DATATYPE(COMP)=NUM  THEN
              COMP=D2C(COMP,2)
          ELSE
             IF  COMP='FFFFFFFF' THEN
                 COMP=COPIES(X2C('FF'),2)
             ELSE
                 DO
                   A=A-1
                   SAY '***ERROR: COMP IS INVALID'
                   ITERATE
                 END

/*************  PROCESSES NEXT RECORD ON INPUT FILE ***********/
    NM=NM+1
          PARSE VALUE NAME||LEFT(CARD,9,Ø)||LEFT(NPAGES,9,Ø),
```

```
                        ||LEFT(COMP,4,Ø) WITH TBLO
          TBLO.NM=TBLO
        END
     ELSE
        NOP

     IF VAR1='YES' THEN
        NOP
     ELSE
        DO
         SAY 'WHAT TABLE DO YOU WANT TO CHANGE?'
         PULL VAR2
         A=1
         ITERATE
        END
   END
END

"ALLOC F("DD2") SHR REUSE DA("DSN2")"
"EXECIO "NM" DISKW "DD2" (FINIS STEM TBLO.)"
"FREE F("DD2")"
RETURN

/* REXX */
```

DB2CATIX

DB2CATIX displays values from the SYSINDEXES output file produced by the extraction program PCATV3EX.

```
/* REXX */

DD1="I"TIME("S")
DD2="O"TIME("S")
DSN1="'XXXXXXXX.SYSINDXS'"
DSN2="'XXXXXXXX.SYSINDXO'"

"ALLOC F("DD1") SHR REUSE DA("DSN1")"
"EXECIO * DISKR "DD1" (FINIS STEM TBL.)"
"FREE F("DD1")"

/*****  READS THRU INPUT FILE UNTIL SPECIFIED RECORD IS REACHED *****/

NM=Ø
SAY 'DO YOU WANT THE WHOLE DATABASE?'
PULL VAR1
IF VAR1='YES' THEN
   NOP
ELSE
  DO
```

```
   SAY 'WHAT TABLE OR INDEX DO YOU WANT TO CHANGE?'
   PULL VAR2
  END

DO A=1 TO TBL.Ø
   PARSE VALUE TBL.A WITH 1 TBNAME 7 . ,
                          19 NAME 27 . ,
                          37 CLUSTRAT 39 . ,
                          41 FIRSTKEY 45 . ,
                          5Ø FULLKEY  54 . ,
                          59 NLEAF    63 . ,
                          68 NLEVELS  7Ø .
   DO
     IF VAR1='YES' THEN
        NOP
     ELSE
        IF VAR2 ¬= TBNAME &,
           VAR2 ¬= NAME   THEN
           ITERATE
        ELSE
           NOP
   END
/*******  PROCESSES INPUT VALUES ******/

   DO
     CLUSTRAT=C2X(CLUSTRAT)
     FIRSTKEY=C2X(FIRSTKEY)
     FULLKEY=C2X(FULLKEY)
     NLEAF=C2X(NLEAF)
     NLEVELS=C2X(NLEVELS)
     DO
       IF CLUSTRAT='FFFF'  THEN
           NOP
       ELSE
           CLUSTRAT=X2D(CLUSTRAT)
       IF FIRSTKEY='FFFFFFFF'   THEN
           NOP
       ELSE
           FIRSTKEY=X2D(FIRSTKEY)
       IF FULLKEY='FFFFFFFF'    THEN
           NOP
       ELSE
         FULLKEY=X2D(FULLKEY)
       IF NLEAF='FFFFFFFF'  THEN
           NOP
       ELSE
           NLEAF=X2D(NLEAF)
       IF NLEVELS='FFFF'  THEN
           NOP
       ELSE
         NLEVELS=X2D(NLEVELS)
```

```
     END

/*******  DISPLAYS SPECIFIC FILE RECORD *********/
     LIST.SCR=LEFT(TBNAME,8)'  'LEFT(NAME,8)'    'RIGHT(CLUSTRAT,4,Ø),
              '     'RIGHT(FIRSTKEY,8,Ø)'   'RIGHT(FULLKEY,8,Ø),
              '   'RIGHT(NLEAF,8,Ø)'    'RIGHT(NLEVELS,4,Ø)
     SAY 'TABLE     INDEX    CLUSTRATIO   FIRSTKEY   FULLKEY       NLEAF
    NLEVELS'
     SAY LIST.SCR
     PULL . . CLUSTRAT FIRSTKEY FULLKEY NLEAF NLEVELS .

/*******  PREPARES CHANGED FIELDS TO REWRITE FILE RECORD ******/

     IF CLUSTRAT¬='' THEN
        DO
          TBNAME=LEFT(TBNAME,18)
          NAME=LEFT(NAME,18)
          IF CLUSTRAT/='FFFF'  &,
             DATATYPE(CLUSTRAT)=NUM  THEN
             CLUSTRAT=D2C(CLUSTRAT,2)
          ELSE
             IF CLUSTRAT='FFFFFFFF' THEN
                CLUSTRAT=COPIES(X2C('FF'),2)
             ELSE
                DO
                  A=A-1
                  SAY '***ERROR: CLUSTRAT IS INVALID'
                  ITERATE
                END

/*******  PREPARES FIRSTKEY VALUES THAT WERE CHANGED  ******/

    IF FIRSTKEY/='FFFF'  &,
             DATATYPE(FIRSTKEY)=NUM  THEN
             FIRSTKEY=D2C(FIRSTKEY,4)
          ELSE
             IF FIRSTKEY='FFFFFFFF' THEN
                FIRSTKEY=COPIES(X2C('FF'),4)
             ELSE
                DO
                  A=A-1
                  SAY '***ERROR: FIRSTKEY IS INVALID'
                  ITERATE
                END

/*******  PREPARES FULLKEY VALUES THAT WERE CHANGED  ******/

          IF FULLKEY/='FFFF'  &,
             DATATYPE(FULLKEY)=NUM  THEN
             FULLKEY=D2C(FULLKEY,4)
          ELSE
```

```
            IF FULLKEY='FFFFFFFF' THEN
               FULLKEY=COPIES(X2C('FF'),4)
            ELSE
               DO
                 A=A-1
                 SAY '***ERROR: FULLKEY IS INVALID'
                 ITERATE
               END

/*******  PREPARES NLEAF VALUES THAT WERE CHANGED  ******/
          IF NLEAF/='FFFF'  &,
             DATATYPE(NLEAF)=NUM  THEN
             NLEAF=D2C(NLEAF,4)
          ELSE
             IF NLEAF='FFFFFFFF' THEN
                NLEAF=COPIES(X2C('FF'),4)
             ELSE
                DO
                  A=A-1
                  SAY '***ERROR: NLEAF IS INVALID'
                  ITERATE
                END
/*******  PREPARES NLEVELS VALUES THAT WERE CHANGED  ******/

          IF NLEVELS/='FFFF'  &,
             DATATYPE(NLEVELS)=NUM  THEN
             NLEVELS=D2C(NLEVELS,2)
          ELSE
             IF NLEVELS='FFFFFFFF' THEN
                NLEVELS=COPIES(X2C('FF'),2)
             ELSE
                DO
                  A=A-1
                  SAY '***ERROR: NLEVELS IS INVALID'
                  ITERATE
                END

/*******  PROCESSES NEXT RECORD ON INPUT FILE  ******/

    NM=NM+1
          PARSE VALUE TBNAME||NAME||LEFT(CLUSTRAT,4,Ø)||,
              LEFT(FIRSTKEY,9,Ø)||LEFT(FULLKEY,9,Ø)||,
              LEFT(NLEAF,9,Ø)||LEFT(NLEVELS,4,Ø)  WITH TBLO
          TBLO.NM=TBLO
       END
     ELSE
       NOP
     IF VAR1='YES' THEN
       NOP
     ELSE
       DO
```

```
          A=A+1
          PARSE VALUE TBL.A WITH 1  TBNAME 7 .
          IF VAR2 = TBNAME THEN
            DO
              A=A-1
              ITERATE
            END
          ELSE
            DO
              SAY 'WHAT TABLE OR INDEX DO YOU WANT TO CHANGE?'
              PULL VAR2
              A=1
              ITERATE
            END
        END
   END
END
"ALLOC F("DD2") SHR REUSE DA("DSN2")"
"EXECIO "NM" DISKW "DD2" (FINIS STEM TBLO.)"
"FREE F("DD2")"
RETURN

/* REXX */
```

DB2CATCD

DB2CATCD displays values from the SYSCOLDIST output file produced by the catalog extraction program PCATV3EX.

```
/* REXX */

DD1="I"TIME("S")
DD2="O"TIME("S")
DSN1="'XXXXXXXX.SYSCOLDT'"
DSN2="'XXXXXXXX.SYSCOLOT'"
"ALLOC F("DD1") SHR REUSE DA("DSN1")"
"EXECIO * DISKR "DD1" (FINIS STEM TBL.)"
"FREE F("DD1")"

/*****  READS THRU INPUT FILE UNTIL SPECIFIED RECORD IS REACHED *****/

NM=Ø
SAY 'DO YOU WANT THE WHOLE DATABASE?'
PULL VAR1

IF VAR1='YES' THEN
   NOP
ELSE
  DO
```

```
   SAY 'WHAT TABLE DO YOU WANT TO CHANGE?'
   PULL VAR2
  END

DO A=1 TO TBL.Ø
   PARSE VALUE TBL.A WITH 1  TBNAME 7 .,
                          19 COLNAME 37 . ,
                          37 COLVALUE 77 . ,
                          291 FREQUENC 293 .
   DO
     IF VAR1='YES' THEN
        NOP
     ELSE
        IF VAR2 = TBNAME THEN
           NOP
        ELSE
           ITERATE
   END

/*******  PROCESSES FREQUENCY VALUES ******/
   DO
     FREQUENC=C2X(FREQUENC)
       IF FREQUENC='FFFF'   THEN
         DO
           A=A+1
           PARSE VALUE TBL.A WITH 1  TBNAME 7 .
           IF VAR2 = TBNAME THEN
             DO
               A=A-1
               ITERATE
             END
           ELSE
             DO
               SAY 'WHAT TABLE DO YOU WANT TO CHANGE?'
               PULL VAR2
               A=1
               ITERATE
             END
         END
       ELSE
           FREQUENC=X2D(FREQUENC)
   END
```

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

Iolanda Lopes
Database Administator
Companhia Seguros Mundial Confiança (Portugal)

Character versus numeric data types

Most DBAs have faced the situation where one of their applications requires a four-byte code that is used to identify products, accounts, or some other business object, and all of the codes are numeric and will stay that way. But, for reporting purposes, users or developers want the codes to print out with leading zeros. So, the users request that the column be defined as CHAR(4) to ensure that leading zeros are always shown. What are the drawbacks, if any, to doing this?

Well, there are drawbacks! Without proper edit checks, inserts and updates could put invalid alphabetic characters into the product code. This can be a very valid concern if *ad hoc* data modifications are permitted. This is rare in production databases, but data problems can still occur if the proper edit checks are not coded into every program that can modify the data. Let's assume that proper edit checks are coded and will never be by-passed. This removes the data integrity question.

There is another problem that is related to filter factors. Consider the possible number of values that a CHAR(4) column and a SMALLINT column can assume. Even if edit checks are coded for each, DB2 is not aware of these and assumes that all combinations of characters are permitted. DB2 uses base 37 maths when it determines access paths for character columns – under the assumption that 26 alphabetic letters, 10 numeric digits, and a space will be used. This adds up to 37 possible characters. For a four-byte character column there are 37^4 (1,874,161) possible values.

A SMALLINT column can range from -32,768 to 32,767, producing 65,536 possible small integer values. The drawback here is that negative or five-digit product codes could be entered. However, if we adhere to our proper edit check assumption, the data integrity problems will be avoided here as well.

DB2 will use the HIGH2KEY and LOW2KEY values to calculate filter factors. For character columns, the range between HIGH2KEY and LOW2KEY is larger than numeric columns because there are more total values. The filter factor will be larger for the numeric data

type than for the character data type – which may influence DB2 to choose a different access path. For this reason, favour the SMALLINT over the CHAR(4) definition.

It might be possible to solve the leading zeros problem using other methods. When using QMF, you can ensure that leading zeros are shown by using the 'J' edit code. Report programs can be coded to display leading zeros easily enough by moving the host variables to appropriate display fields. *Ad hoc* access through other reporting tools typically provides a parameter that can enable leading zeros to be displayed.

In general, it is wise to choose a data type which is closest to the domain for the column. If the column is to store numeric data, favour choosing a numeric data type (SMALLINT, INTEGER, DECIMAL, or floating point). In addition, always be sure to code appropriate edit checks to ensure data integrity.

Craig S Mullins
VP Operations
PLATINUM Technology (USA)

January 1995 – October 1998 index

Items below are references to articles that have appeared in *DB2 Update* since January 1995. References show the issue number followed by the page number(s). Back-issues of *DB2 Update* are available back to Issue 15 (January 1994). See page 2 for details.

Sterling Software has announced Version 1.5 of Vision:Webaccess, with new access to SQL Server and DB2 data, via a standard Web browser. Users can point-and-click through queries, view and refresh previously created reports and charts, and create *ad hoc* reports and charts.

The new release connects to DB2 via IBM's client middleware. Persistent database connection eliminates the need to reconnect to databases constantly, and there's no client installation, configuration, or maintenance.

For further information contact:
Sterling Software, 1800 Alexander Bell Drive, Reston, VA 22091, USA.
Tel: (703) 264 8000.
Sterling Software, 1 Longwalk Road, Stockley Park, Uxbridge, Middlesex, UB11 1DB.
Tel: (0181) 867 8000.
URL: http://www.storage.sterling.com.

* * *

Infotel has announced MASTER-UTIL for DB2, for automated and integrated management of reorganizations and back-up copies. With its MaserReorg and MasterCopy components, MASTER-UTIL automatically ensures that DB2 application performance is maintained and that the DB2 environment is secure, but with full operational awareness and control.

MasterReorg prioritizes and schedules DB2 reorganizations, based on the degree of disorganization of the DB2 objects and batch window constraints. It maintains and maximizes the performance of DB2 application programs, optimizes the use of the batch window, and ensures completion of the reorganizations prior to the start of the on-line session.

MasterCopy automatically manages the back-up copies of DB2 tablespaces according to the restore criteria defined by the site and taking into account batch window constraints. It assures availability at any given time of all objects necessary to perform a recovery.

For further information contact:
Infotel, 15438 North Florida Avenue, Suite 204, Tampa, FL 33613, USA.
Tel: (813) 264 2090.
Infotel Software, Craven House, 14-18 York Road, Wetherby, West Yorkshire, LS22 6SL, UK.
Tel: (01937) 584 200.
URL: http://www.infotelcorp.com.

* * *

IBM has announced DB2 Spatial Extender, an optional feature of DB2 DataJoiner Version 2.1.1. By using existing geographic information systems applications or a single SQL statement directly, it can query spatial information and/or join this data with normal business data from various sources. The product includes support for Oracle 8 and Sybase SQL Anywhere via DataJoiner Version 2.1.1, which provides transparent Data Definition Language support and supports complex configurations and large database needs.

For further information contact your local IBM representative.