Working with variable data

OVERVIEW

One of the key design issues that is addressed in most DB2 applications is how to implement character data that varies in size from row to row. The basic trade-off is ease-of-use and performance versus storage requirements. It is possible to save DASD storage space by using variable columns instead of placing small amounts of data in a large fixed space. Each variable column carries a 2-byte overhead, however, for storing the length of the data. Additionally, variable columns tend to increase CPU usage and can cause the update process to become inefficient. When a variable column is updated with a larger value, the row becomes larger; if insufficient space is available to store the row, it must be moved to another page. This makes the update and any subsequent retrieval slower.

RULES OF THUMB

Follow these rules when defining variable character columns:

- Avoid variable columns if sufficient DASD is available to store the data using fixed columns. The general rule is: DASD is cheap! Trading the cost of DASD for more efficient development and subsequent performance is often an easy decision to make.
- Though this may be obvious, do not define a variable column when the data does not vary from row to row.
- Do not define a variable column if its maximum length is less than 30 bytes. Furthermore, do not define a variable column if its maximum length is within 10 bytes of the average length of the column. Adhering to these two rules will cause you to choose VARCHAR data types only when they can potentially provide enough DASD savings to offset other costs.
- Consider redefining variable columns by placing multiple rows of fixed length columns in another table or by shortening the columns and placing the overflow in another table.

If, after following these guidelines, VARCHAR columns need to be implemented, do so. However, it is wise to continue to re-assess the decision to use variable character data.

MONITOR THE EFFECTIVENESS OF VARIABLE COLUMNS

Using views and SQL it is possible to query the DB2 catalog to determine the effectiveness of using VARCHAR for a column instead of CHAR. Consider, for example, the PROJNAME column of the DSN8410.PROJ table. It is defined as VARCHAR(24).

To gauge whether VARCHAR is appropriate follow these steps:

1 Create a view that returns the length of the NAME column for every row:

```
CREATE VIEW PROJNAME_LENGTH
     (COL_LGTH)
AS   SELECT LENGTH(PROJNAME)
     FROM   DSN8410.PROJ;
```

2 Then, issue the following query using SPUFI to produce a report detailing the LENGTH and number of occurrences for that length:

```
SELECT       COL_LGTH, COUNT(*)
FROM          PROJNAME_LENGTH
GROUP BY  COL_LGTH
ORDER BY  COL_LGTH
```

This query will produce a report listing the lengths (in this case, from 1 to 24, excluding those lengths which do not occur) and the number of times that each length occurs in the table. These results can be analysed to determine the range of lengths stored within the variable column.

If you are not concerned about this level of detail, the following query can be used instead to summarize the space characteristics of the variable column in question:

```
SELECT  24*COUNT(*),
        SUM(2+LENGTH(PROJNAME)),
        24*COUNT(*)-SUM(2+LENGTH(PROJNAME)),
        24,
        AVG(2+LENGTH(PROJNAME)),
        24-AVG(2+LENGTH(PROJNAME))
FROM    DSN8410.PROJ
```

This query will produce a report similar to the one shown below:

```
SPACE      SPACE          TOTAL   AVG.        AVG.          AVG.
USED AS    USED AS        SPACE   SPACE AS    SPACE AS      SPACE
CHAR(24)   VARCHAR(24)    SAVED   CHAR(24)    VARCHAR(24)   SAVED

158058     96515          61543   24          16            8
```

CALCULATIONS

The following list itemizes the definition for each of the individual columns calculated by this query:

Space used as CHAR(24)
24*COUNT(*)

Space used as VARCHAR(24)
SUM(2+LENGTH(PROJNAME))

Total space saved using VARCHAR
24*COUNT(*)-SUM(2+LENGTH(PROJNAME))

Average space used as CHAR(24)
24

Average space used as VARCHAR(24)
AVG(2+LENGTH(PROJNAME))

Average space saved using VARCHAR
24-AVG(2+LENGTH(PROJNAME))

The query can be modified to be used for any VARCHAR-defined column. The constant 24 can be changed to indicate the maximum length of the variable column as defined in the DDL. Using these tools, you can better judge the actual DASD savings accruing as a result of VARCHAR usage.

SYNOPSIS

There are sound reasons for using variable data types within DB2 databases. Yet, it is important to remember that business conditions change and what may have been a sound reason for using VARCHAR data in the past may no longer be sound. Use the queries in this article to determine if it is still appropriate to use VARCHAR data. In general, do not use VARCHAR for small columns or for columns whose length does not vary considerably.

Craig S Mullins
Senior Technical Advisor
Platinum Technology Inc (USA)

Recovering single tables

INTRODUCTION

The DB2XLAT program described in the following article was initially written to list the DB2 object-ids contained within a segmented tablespace without the need to interrogate DB2. The program was later amended to allow the translation of any number of object-ids. This was to assist in recovering a single table defined within a multi-table segmented tablespace, such as QMF's default tablespace for saving a user's data.

Unlike the normal situation whereby the table definitions are constant and the data is transient, with this type of tablespace both the data and table definitions are changing. This raises several issues with regards to recovery as image copies may not reflect the current contents of the tablespace as known to DB2's catalog/directory. We could overcome this problem if we recovered DB2 as well but it is unlikely that this would be an acceptable approach. The most likely method to adopt would be to recreate the objects under a new database/tablespace and to translate the object-ids to the new ones. The problem can be defined as follows: