Craig S. Mullins

Return to Home Page

April 2001

DB2 update

DB2 Table Space Options

By Craig S. Mullins

Although DB2 data is accessed at the table level, those skilled in DB2 database design and administration know that the actual data is actually stored in a structure known as a table space. Each table space correlates to one or more individual physical VSAM data sets that are used to house the actual DB2 data. When designing DB2 databases, DBAs can choose from three types of table spaces, each one useful in different circumstances. The three types of table spaces are:

- Simple table spaces
- Segmented table spaces
- Partitioned table spaces

In general, the predominant table space type to use for most applications is the segmented table space. Segmented table spaces provide a good combination of features that mix ease of use and setup with performance and functionality. Many organizations adhere to standards stating that new DB2 table spaces should be segmented table spaces unless a compelling reason exits to choose one of the other table space types. You should consider using the other types of DB2 table spaces in the following cases:

- Use partitioned table spaces when you wish to encourage parallelism. Although DB2 can and will use parallel access techniques for non-partitioned table spaces, partitioning data helps DB2 exploit parallelism.
- Consider using partitioned table spaces when the amount of data to be stored is very large (more than 1 million pages). You will have more control over the placement of data in separate underlying data sets using partitioned table spaces. This is often a concern with larger DB2 tables.
- Use partitioned table spaces to reduce utility processing time and decrease contention. It is possible to execute DB2 utilities against single partitions without impacting concurrent access to data in other partitions. Furthermore, the utilities

will run faster against a single partition than against the entire table space and you will have more control over driving your utility workload. For example, you may not have sufficient time in the batch window to run a REORG of a four million page segmented table space, but you might have the time to run a REORG of one partition of that table space nightly. With four partitions of one million pages (or perhaps more partitions containing even fewer pages) you may be able to REORG one partition a night.

- Implement partitioned table spaces to improve data availability. For example, if the data is partitioned by region, the partitions for the Eastern, Southern, and Northern regions can be made available while the Western region partition is being reorganized.
- Use partitioned table spaces to improve recoverability. Once again, consider the ramifications if the data is partitioned by region. If an error impacts data for the Eastern region only, then only the Eastern partition needs to be recovered. The Southern, Northern, and Western regions can remain online, because they are not

impacted by the problem in the Eastern region's data.

- Consider partitioned table spaces to isolate specific data areas in dedicated data sets. If there are specific data "hot spots" that have higher data modification and/or access activity, you may be able to improve application performance by isolating the "hot spot" into a single partition that can be tuned for the specific type of application access.
- Use a simple table space only when you need to mix data from different tables on one page. Simple table spaces will mix data from each table assigned to the table space on each table space page. A segmented table space will not because each segment in the segmented table space is assigned to a single table. If you have two tables that are very frequently joined you might consider loading them into a single simple table space, ensuring that each row loaded from the first table is immediately followed by all of the rows from the second table that will be joined to the first table. This can minimize I/O for retrieval. However, DB2 will not maintain this ordering when the data is

changed, so this approach is generally useful only for static data.

Partitioning Considerations

DB2 can handle up to 254 partitions per table space. The actual limit on number of partitions depends on the DSSIZE of the table space. Large table spaces are those which specify the LARGE parameter or have a DSSIZE greater than 4GB. The LARGE parameter was introduced with V5; DSSIZE with V6. A large table space can have from 1 to 254 partitions. Non-large table spaces are limited to no more than 64 partitions, as are any table spaces created in a version prior to DB2 V5.

For non-LARGE partitioned table spaces, the number of partitions impacts the maximum size of the data set partition as follows:

Number of Partitions	Maximum Data Set Size
1 to 16	4 GB
17 to 32	2 GB
33 to 64	1 GB

Keep these limitations in mind as you design your partitioned table spaces.

As a general rule of thumb try to define table space partitions such that no one partition is more than 20 percent larger than the next largest partition. This provides even growth, which eases DASD monitoring and provides approximately even data access requirements and utility processing times across partitions. This is not a hard-and-fast rule though, especially when dealing with “hot spots.” The “hot spot” partition may be much smaller than the other partitions going against the idea of maintaining evenly distributed partitions. This is okay.

Deciding to use a partitioned table space is not as simple as merely determining the size of the table. In the early days of DB2, size was the primary consideration for choosing a partitioned table space. However, as DB2 has matured and the applications written using DB2 have become modernized, additional considerations will impact your partitioning decisions. Application-level details, such as data contention, performance requirements, degree of parallelism, and the volume of updates to columns in the partitioning index must factor into the decision to use partitioned table spaces.

Sometimes designers try to avoid partitioned table spaces by dividing a table into multiple tables, each with its own table space. This is not wise. Never attempt to avoid a partitioned table space by implementing several smaller table spaces, each containing a subset of the total amount of data. When proceeding in this manner, the designer usually places separate tables into each of the smaller table spaces. This almost always is a bad design decision because it introduces an uncontrolled and unneeded denormalization. Furthermore, when data that logically belongs in one table is separated into multiple tables, SQL operations to access the data as a logical whole are made needlessly complex. One example of this complexity is the difficulty in enforcing unique keys across multiple tables. Although partitioned table spaces can introduce additional complexities into your environment, these complexities never outweigh those introduced by mimicking partitioning with several smaller, identical table spaces. To clarify why this idea is bad, consider these two different ways of implementing a three "partition" solution:

- The first, recommended way is to create the table in a single partitioned table space with three partitions as follows:

```
CREATE DB DB_SAMP;

CREATE TABLESPACE TS_SAMP IN DB_SAMP
      ERASE NO NUMPARTS 3
      (PART 1
       USING STOGROUP SG_SAMP1
       PRIQTY 2000   SECQTY 50
       COMPRESS NO,

       PART 2
```

```
        USING STOGROUP SG_SAMP2
        PRIQTY 4000   SECQTY 150
        COMPRESS YES,

        PART 3
        USING STOGROUP SG_SAMP3
        PRIQTY 1000
        SECQTY 50
        COMPRESS YES)

      LOCKSIZE PAGE   BUFFERPOOL BP1   CLOSE
NO;

CREATE TABLE TB_SAMP . . . IN
DB_SAMP.TS_SAMP;
```

- The second, ill-advised way is to create three table spaces each with its own table as follows:

```
CREATE DB DB_SAMP;

CREATE TABLESPACE TS_SAMP1 IN DB_SAMP
      USING STOGROUP SG_SAMP1
      PRIQTY 2000   SECQTY 50
      ERASE NO COMPRESS NO
      LOCKSIZE PAGE   BUFFERPOOL BP1   CLOSE
NO;

CREATE TABLESPACE TS_SAMP2 IN DB_SAMP
      USING STOGROUP SG_SAMP2
```

```
     PRIQTY 4000   SECQTY 150
     ERASE NO COMPRESS YES
     LOCKSIZE PAGE   BUFFERPOOL BP1   CLOSE
NO;

CREATE TABLESPACE TS_SAMP3 IN DB_SAMP
     USING STOGROUP SG_SAMP3
     PRIQTY 1000
     SECQTY 50
     ERASE NO COMPRESS YES
     LOCKSIZE PAGE   BUFFERPOOL BP1   CLOSE
NO;


CREATE TABLE TB_SAMP1 . . . IN
DB_SAMP.TS_SAMP1;

CREATE TABLE TB_SAMP2 . . . IN
DB_SAMP.TS_SAMP2;

CREATE TABLE TB_SAMP3 . . . IN
DB_SAMP.TS_SAMP3;
```

Now consider how difficult it would be to retrieve data in the second implementation if you did not know which “partition” the data resides in, or if the data could reside in multiple partitions.

Using the first example a simple SELECT will work.

```
SELECT *
FROM TB_SAMP
```

```
WHERE COL1 = :HOST-VARIABLE;
```

In the second example, a UNION is required.

```
SELECT *
FROM TB_SAMP1
WHERE COL1 = :HOST-VARIABLE
UNION ALL
SELECT *
FROM TB_SAMP2
WHERE COL1 = :HOST-VARIABLE
UNION ALL
SELECT *
FROM TB_SAMP3
WHERE COL1 = :HOST-VARIABLE;
```

If other tables need to be joined the “solution” becomes even more complex. Likewise if data must be updated, inserted, or deleted and you do not know which “partition” contains the impacted data. The bottom line: avoid bypassing DB2 partitioning using your own pseudo-partitions.

Partitioning Pros and Cons

Before deciding to partition a table space, weigh the pros and cons. Consult the following list of advantages and disadvantages before implementation:

Advantages of a partitioned table space:

- Each partition can be placed on a different DASD volume to increase access efficiency.

- Partitioned table spaces are the only type of table space that can hold more than 64GB of data (the maximum size of simple and segmented table spaces). A partitioned table space with extended addressability (EA-enabled) can hold up to 16 terabytes of data. Without being EA-enabled a partitioned table space can store up to about 1 TB of data.

- Start and stop commands can be issued at the partition level. By stopping only specific partitions, the remaining partitions are available to be accessed thereby promoting higher availability.

- Free space (PCTFREE and FREEPAGE) can be specified at the partition level enabling the DBA to isolate data “hot spots” to a specific partition and tune accordingly.

- Partitioning can optimize Query I/O, CPU, and Sysplex parallelism by removing disk contention as an issue because partitions can be spread out across multiple devices.

- Table space scans on partitioned table spaces can skip partitions that are excluded based on the query predicates. Skipping entire partitions can improve overall query performance for table space scans because less data needs to be accessed.

- The clustering index used for partitioning can be set up to decrease data contention. For example, if the table space will be partitioned by DEPT, each department (or range of compatible departments) could be placed in separate partitions. Each department is in a discrete physical data set, thereby reducing inter-departmental contention due to multiple departments coexisting on the same data page. Note that contention remains for data in non-partitioned indexes (although this contention has been significantly reduced by in recent versions of DB2).

- DB2 creates a separate compression dictionary for each table space partition. Multiple dictionaries tend to cause better overall compression ratios. In addition, it is more likely that the partition-level compression dictionaries can be rebuilt more frequently than non-partitioned dictionaries. Frequent rebuilding of the compression dictionary can lead to a better overall compression ratio.

- The REORG, COPY, and RECOVER utilities can execute on table spaces at the partition level. If these utilities are set to execute on partitions instead of on the entire table space, valuable time can be saved by processing only the partitions that need to be reorganized, copied, or recovered. Partition independence and resource serialization further increase the availability of partitions during utility processing.

Disadvantages of a partitioned table space:

- Only one table can be defined in a partitioned table space. This is not necessarily a disadvantage because most DBAs follow a one-table-per-table-space rule.

- The columns of the partitioning index cannot be updated. To change a value in one of these columns, you must delete the row and then reinsert it with the new values.

- The range of key values for which data will be inserted into the table must be known and stable before you create the partitioning index. To define a partition, a range of values must be hard-coded into the partitioning index definition. These ranges will be used to distribute the data throughout the partitions. If you provide a stop-gap partition to catch all the values lower (or higher) than the defined range, monitor that partition to ensure that it does not grow dramatically or cause performance problems if it is smaller or larger than most other partitions.

- After you define the method of partitioning, the only way to change it is to ALTER the partitioning index to change the LIMITKEY values and reorganize any impacted partitions. Prior to V6 you had to drop and redefine both the partitioning index and table space to change LIMITKEY specifications.

In general, partitioned table spaces are becoming more useful. You might even want to consider using partitioning for most table spaces (instead of segmented), especially if parallelism is an issue. At least, consider partitioning table spaces that are accessed in a read only manner by long-running batch programs. Of course, very small table spaces are rarely viable candidates for partitioning, even with DB2’s advanced I/O, CPU, and Sysplex parallelism features. This is true because the smaller the amount of data to access, the more difficult it is to break it into pieces large enough such that concurrent, parallel processing will be helpful.

When using partitioned table spaces, try to place each partition of the same partitioned table space on separate DASD volumes. Failure to do so can negatively affect the performance of query parallelism performed against those partitions. Disk drive head contention will occur because concurrent access is being performed on separate partitions that co-exist on the same device. Of course, with some of the newer storage devices, such as the ESS Shark hardware from IBM, data set placement is a non-issue because of the way in which data is physically stored on the device.

Summary

DB2 provides three different types of table spaces, each of which has its own distinct set of advantages and disadvantages for use depending upon the situation. As a DBA you should understand the implementation details of each type of table space and be prepared to choose the right type of table space for each situation.

From DB2 Update (Xephon) April 2001.