This article helps the new DB2® database administrator understand the importance of table spaces and buffer pools. The article also explains how properly designing and tuning table spaces and buffer pools can enhance database performance. [Originally written in 2002, this article has been updated for IBM DB2 V9.7 for Linux®, UNIX®, and Windows®.]

For the database administrator (DBA) who is just stepping into the world of DB2 or for the prospective DBA, the design and performance choices for a new database can be very confusing. This article discusses two areas in which the DBA has important choices to make: table spaces and buffer pools. The design and tuning of table spaces and buffer pools can have a profound impact on how the DB2 server performs.

In this article, the examples use DB2 Version 9.7, Enterprise Server Edition. Most of the examples also apply to downlevel versions, unless otherwise indicated.

The article does the following:

- Defines the types of table spaces and explains how DB2 stores data in table spaces
- Covers configuration options and takes you through the process of creating and managing a table space
- Discusses buffer pools, covering what a buffer pool is and how to create and use it
- Suggests what to consider before deciding to move databases
- Describes how buffer pools and table spaces should be organized to maximize performance

**Learning about table spaces**

All data for a database is stored in a number of table spaces. You can think of a table space as a child and a database as its parent, where the table space (child) cannot have more than one database (parent). Because there are different uses for table spaces, they are classified according to their usage and how they will be managed. There are five different table spaces, named according to their usage:

**Catalog table space**

There is only one catalog table space per database, and it is created when the CREATE DATABASE command is issued. Named SYSCATSPACE by DB2, the catalog table space
holds the system catalog tables. This table space is always created when the database is created.

**Regular table space**
A regular table space stores all permanent data, including regular tables and indexes. It can also hold large data such as LOBs (Large Objects) unless they are explicitly stored in a large table space. A table and its indexes can be divided into separate regular table spaces, if the table spaces are *database-managed space* (DMS) for non-partitioned tables or *system-managed space* (SMS) for partitioned tables. DMS and SMS are described in the section **Table space management**. Catalog table space is an example of a regular table space. By default, the catalog table space is the only regular table space created during the database creation.

**Large table space**
A large table space stores all permanent data just as a regular table space does, including LOBs. This table space type must be DMS, which is the default type. A table created in a large table space can be larger than a table in a regular table space. A large table can support more than 255 rows per data page improving space utilization on data pages. DB2 creates one large table space named USERSPACE1 when a database is created.

**System temporary table space**
A system temporary table space stores internal temporary data required during SQL operations such as sorting, reorganizing tables, creating indexes, and joining tables. At least one system temporary must exist per database. The default created with the database is named TEMPSPACE1.

**User temporary table space**
A user temporary table space stores declared global temporary tables. No user temporary table spaces exist when a database is created. At least one user temporary table space should be created to allow definition of declared temporary tables. User temporary table spaces are optional. No user temporary table space is created by default.

**Table space management**
Table spaces can be managed in one of two ways:

**System-managed space (SMS)**
The operating system manages SMS table spaces. Containers are defined as regular operating system files, and they are accessed using operating system calls. This means that all the regular operating system functions handle the following:

- I/O is buffered by the operating system
- Space is allocated according to the operating system conventions
- The table space is automatically extended when needed

However, containers cannot be dropped from SMS table spaces, and adding new ones is restricted to partitioned databases. In version 9.1 and above, the only default SMS tablespace created during database creation is TEMPSPACE1.

**Database-managed space (DMS)**
DB2 manages DMS table spaces. Containers can be defined either as files, which will be fully allocated with the size given when the table space is created, or as devices. DB2 manages
as much of the I/O as the allocation method and the operating system allow. Extending the containers is possible using the ALTER TABLESPACE command. Unused portions of DMS containers can be also released (starting with version 8). Listing 1 shows how to increase container sizes:

**Listing 1. Increase container size**

```
ALTER TABLESPACE TS1 RESIZE (FILE '/conts/cont0' 2000,
DEVICE '/dev/rcont1' 2000, FILE 'cont2' 2000)
```

You can also use options such as EXTEND or REDUCE to increase or decrease the size of a container.

**How to create and view your table spaces**

When you create a database, three table spaces are created (SYSCATSPACE, TEMPSPACE1, and USRSSPACE1). Listing 2 shows you how to create a database called testdb, connect to it, and list the table spaces using the DB2 command window or the UNIX command line.

**Listing 2. Create, connect, and list**

```
CREATE DATABASE testdb
CONNECT TO testdb
LIST TABLESPACES
```

Listing 3 shows the output from the LIST TABLESPACES command.

**Listing 3. Output from LIST TABLESPACES command**

<table>
<thead>
<tr>
<th>Tablespace ID</th>
<th>Name</th>
<th>Type</th>
<th>Contents</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SYSCATSPACE</td>
<td>Database managed space</td>
<td>All permanent data. Regular table space.</td>
<td>0x0000</td>
</tr>
<tr>
<td>1</td>
<td>TEMPSPACE1</td>
<td>System managed space</td>
<td>System Temporary data</td>
<td>0x0000</td>
</tr>
<tr>
<td>2</td>
<td>USRSSPACE1</td>
<td>Database managed space</td>
<td>All permanent data. Large table space.</td>
<td>0x0000</td>
</tr>
</tbody>
</table>

The CREATE DATABASE command automatically creates the three table spaces in Listing 3. The user can override the default table space creation by including table space specifications in the
command, but a catalog table space and at least one regular or large and one system temporary table space must be created at database creation time. More table spaces of all types (except catalog table space) can be created either with the CREATE DATABASE command, or later using the CREATE TABLESPACE command.

Containers

Each table space has one or more containers. Again, you might think of a container as being a child and a table space as its parent. Each container can belong to only a single table space, but a table space can have many containers. Containers can be added to or dropped from a DMS table space, and their sizes can be modified. Containers can only be added to SMS table spaces on partitioned databases in a partition that does not yet have a container allocated for the table space. When new containers are added, an automatic rebalancing distributes the data across all containers. Rebalancing does not prevent concurrent access to the database.

Table space settings

There are several settings that you can specify for table spaces, either when you create them or later with an ALTER TABLESPACE statement. The following list describes the settings.

Page size

Defines the size of pages used for the table space. Supported sizes include 4K, 8K, 16K, and 32K. The page size limits the row length and column count of tables that can be placed in the table space according to the limits shown in Table 1.

Table 1. Implications of page size

<table>
<thead>
<tr>
<th>Page size</th>
<th>Row size limit</th>
<th>Column count limit</th>
<th>Maximum capacity (DMS tablespace)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 KB</td>
<td>4 005</td>
<td>500</td>
<td>64 GB</td>
</tr>
<tr>
<td>8 KB</td>
<td>8 101</td>
<td>1 012</td>
<td>128 GB</td>
</tr>
<tr>
<td>16 KB</td>
<td>16 293</td>
<td>1 012</td>
<td>256 GB</td>
</tr>
<tr>
<td>32 KB</td>
<td>32 677</td>
<td>1 012</td>
<td>512 GB</td>
</tr>
</tbody>
</table>

Table spaces are limited to 16,777,216 pages, so choosing a larger page size will increase the capacity of the table space.

Extent size

Specifies the number of pages that will be written to a container before skipping to the next container. The database manager cycles repeatedly through the containers as data is stored. This parameter has effect only when there are multiple containers for the table space.

Prefetch size

Specifies the number of pages that are read from the table space when data prefetching is being performed. Prefetching reads in data needed by a query before they are referenced by the query so that the query need not wait for I/O to be performed. Prefetching is selected by the database manager when it determines that sequential I/O is appropriate and that prefetching can help to improve performance.
Overhead and transfer rate

Determines the cost of I/O during query optimization. Both values are measured in milliseconds, and they should be the average for all containers. The overhead is the time associated with I/O controller activity, disk seek time, and rotational latency. The transfer rate is the amount of time necessary to read one page into memory. The default values for a database created in DB2 Version 9 are 7.5 milliseconds and 0.06 milliseconds, respectively. The default values for a database that is migrated from a previous version of DB2 to Version 9 or later are 12.67 milliseconds and 0.18 milliseconds, respectively. These values can be calculated based on hardware specifications.

Example of a CREATE TABLESPACE statement

Listing 4 creates a regular table space, including all the settings from this article.

Listing 4. Creating a table space

```
CREATE TABLESPACE USERSPACE3
  PAGESIZE 8K
  MANAGED BY SYSTEM
  USING ('d:\usp3_cont1', 'e:\usp3_cont2', 'f:\usp3_cont3')
  EXTENTSIZ E 64
  PREFETCHSIZE 32
  BUFFERPOOL BP3
  OVERHEAD 7.5
  TRANSFERRATE 0.06
```

How to view your table space attributes and containers

Specifying the SHOW DETAIL option of the LIST TABLESPACES command shows additional information: LIST TABLESPACES SHOW DETAIL.

Listing 5 shows the output for the USERSPACE1 table space. By default, the three table spaces created at database creation time will be listed.

Listing 5. Output from LIST TABLESPACES SHOW DETAIL command

```
Tablespaces for Current Database

<table>
<thead>
<tr>
<th>Tablespace ID</th>
<th>Name</th>
<th>Type</th>
<th>Contents</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>USERSPACE1</td>
<td>Database managed space</td>
<td>All permanent data. Large table space.</td>
<td>0x0000</td>
</tr>
</tbody>
</table>

Detailed explanation:

- Normal

Total pages = 8192
Useable pages = 8160
Used pages = 96
Free pages = 8064
High water mark (pages) = 96
Page size (bytes) = 4096
Extent size (pages) = 32
Prefetch size (pages) = 32
Number of containers = 1
```

To list the containers needed to use the Tablespace ID from the output above, enter LIST TABLESPACE CONTAINERS FOR 2.
Listing 6. Output from LIST TABLESPACES CONTAINERS command

<table>
<thead>
<tr>
<th>Tablespace Containers for Tablespace 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container ID = 0</td>
</tr>
<tr>
<td>Name = C:\DB2\NODE0000\SQL00004\SQLT0002.0</td>
</tr>
<tr>
<td>Type = Path</td>
</tr>
</tbody>
</table>

The command lists all containers for the specified table space. The path in Listing 6 points to where the container physically resides.

Buffer pools

A buffer pool is associated with a single database and can be used by more than one table space. When considering a buffer pool for one or more table spaces, you must ensure that the table space page size and the buffer pool page size are the same for all table spaces that the buffer pool services. A table space can only use one buffer pool.

When the database is created, a default buffer pool named IBMDEFAULTBP is created, which is shared by all table spaces. More buffer pools can be added using the CREATE BUFFERPOOL statement. The buffer pool size defaults to the size specified by the BUFFPAGE database configuration parameter, but you can override it by specifying the SIZE keyword in the CREATE BUFFERPOOL command. Adequate buffer pool size is essential to good database performance, because it will reduce disk I/O, which is the most time consuming operation. Large buffer pools also have an effect on query optimization, because more of the work can be done in memory.

Block-based buffer pools

Version 8 and higher enables you to set aside a portion of the buffer pool (up to 98%) for block-based prefetching. Block-based I/O improves the efficiency of prefetching by reading a block into a contiguous area of memory instead of scatter-loading it into separate pages. The size of the blocks must be uniform per buffer pool and is controlled by the BLOCKSIZE parameter. The value is the size of the block measured in pages from 2 to 256, the default being 32.

Example of CREATE BUFFERPOOL statement

For an example of the CREATE BUFFERPOOL statement, enter: `CREATE BUFFERPOOL BP3 SIZE 2000 PAGESIZE 8K`

This buffer pool is assigned to USERSPACE3 on this article's CREATE TABLESPACE example and is created before creating the table space. Note that the page sizes of 8K for the buffer pool and table space are the same. If you create the table space after creating the buffer pool, you can leave out the BUFFER POOL BP3 syntax in the CREATE TABLESPACE statement. Instead, you can use the ALTER TABLESPACE command to add the buffer pool to the existing table space by entering `ALTER TABLESPACE USERSPACE3 BUFFERPOOL BP3`.

How to view your buffer pool attributes

You can list buffer pool information by querying the SYSCAT.BUFFERPOOLS system view, as shown in Listing 7.
Listing 7. Querying SYSCAT.BUFFERPOOLS

```
SELECT * FROM SYSCAT.BUFFERPOOLS
```

<table>
<thead>
<tr>
<th>BPNAME</th>
<th>BUFFERPOOLID</th>
<th>DBPGNAME</th>
<th>NPAGES</th>
<th>PAGESIZE</th>
<th>ESTORE</th>
<th>NUMBLOCKPAGES</th>
<th>BLOCKSIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBMDEFAULTBP</td>
<td>1</td>
<td>-</td>
<td>1000</td>
<td>4096</td>
<td>N</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

1 record(s) selected.

To find out which buffer pool is assigned to table spaces, run the query shown in Listing 8.

Listing 8. Querying SYSCAT.TABLESPACES

```
SELECT TBSPACE, BUFFERPOOLID FROM SYSCAT.TABLESPACES
```

<table>
<thead>
<tr>
<th>TBSPACE</th>
<th>BUFFERPOOLID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCATSPACE</td>
<td>1</td>
</tr>
<tr>
<td>TEMPSPACE1</td>
<td>1</td>
</tr>
<tr>
<td>USERSPACE1</td>
<td>1</td>
</tr>
</tbody>
</table>

3 record(s) selected.

The BUFFERPOOLID is shown in the query in Listing 8, enabling you to see which buffer pool is associated with each table space.

Visual diagram of how a database holds table spaces

Now that you know what a table space and buffer pool are and how to create them, Figure 1 shows an example of how they are visually organized within a database.

Figure 1. Table spaces and buffer pools

The example database has five table spaces: one catalog, two regular, one large, and one system temporary table space. No user temporary table space was created. There are eight containers. In this example, buffer pools might be assigned as follows:

- BP1 (4K) to SYSCATSPACE and USERSPACE2
- BP2 (8K) to USERSPACE1
- BP3 (32K) to LARGESPACE and SYSTEMP1
Examining performance implications

In general, when designing table space and container placement on physical devices, the goal is to maximize I/O parallelism and buffer utilization. To achieve that goal, you need a thorough understanding of the database design and applications. Only then can you determine such issues as whether segregating two tables to different devices will lead to parallel I/O, or whether a table should be created in a separate table space so it can be fully buffered.

Start designing the physical layout of a new database by designing the table space organization, as shown in the following steps.

1. Determine the constraints given by the table designs. These might result in having to use more than one regular table space.
2. Consider whether having the tables in table spaces with different settings is likely to significantly increase performance.
3. Design a tentative table space.
4. Consider buffer pool utilization, which might lead you to make some changes to the previous table space design.
5. Allocate containers to the table spaces.

This process is iterative, and the design should be verified with stress-testing and benchmarking. Clearly, arriving at the best design can be quite an intensive effort, so the time it takes can only be justified if the database performance must be the best possible. As a rule:

- Start out with the simplest feasible design.
- Add complexity only when there is a sufficient performance justification for it based on testing.

Often a slight degradation in performance is well worth the reduced complexity of administering and maintaining a simpler database design. DB2 has sophisticated resource-management logic, which standardly produces very good performance without elaborate design.

Table space organization

Each table, depending on how it is accessed most frequently, has a most efficient set of table space settings: PAGESIZE, EXTENTSIZE, and PREFETCHSIZE.

The catalog table space and system temporary table spaces should usually be allocated as SMS. There is no reason to have more than one temporary table space of the same page size, and usually one with the largest page size is sufficient.

The salient question is whether to split up the user data into multiple table spaces or not. One consideration is the utilization of pages. Rows cannot be split between pages, so tables with long rows require the appropriate page size. However, there cannot be more than 255 rows on a page, so tables with short rows do not utilize the whole page.

For example, a table with a row length of 12 bytes placed in a table space with 32K page size utilizes only about 10% of each page, which is calculated as (255 rows * 12 bytes) + 91 bytes of
overhead) / 32k page size = ~10%. This is only a consideration if the table is large, which means the wasted space is significant. It also makes I/O and buffering less efficient, because the actual useful content of each page is small.

If a table can either be placed into a smaller page size table space or fully utilize a larger page size, then the most frequent method of access determines which one is better. If typically more rows are accessed sequentially (maybe the table is clustered), then the larger page size is more efficient. If rows are accessed randomly, then the smaller page size enables DB2 to make better use of the buffer, because more pages fit into the same storage area.

After you group the tables by page size, access frequency and type determine whether further grouping the data into separate table spaces is warranted. EXTENTSIZEx is the number of pages of data that will be written to a container before writing to the next container (if multiple containers exist in the table space).

PREFETCHSIZE specifies the number of pages to be read from the table space when data prefetching is being performed. Prefetching is used when the Database Manager determines that sequential I/O is appropriate and that prefetching can help to improve performance (typically large table scans). It is a good practice to explicitly set the PREFETCHSIZE value as a multiple of the EXTENTSIZEx value for your table space and the number of table space containers. For example, if the EXTENTSIZEx is 32 and there are four containers, then good PREFETCHSIZE values would be 128, 256, and so on. If one or more heavily used tables require a different set of these parameters than the values that are best for the rest of the table space performance, put the heavily used tables into a separate table space to improve overall performance.

If prefetching is an important factor in a table space, consider setting aside part of the buffer for block-based I/O. The block size should be equal to the PREFETCHSIZE.

**Buffer pool utilization**

The most important reason to use more than one user table space is to manage buffer utilization. A table space can be associated with only one buffer pool, but one buffer pool can be used for more than one table space.

The goal of buffer pool tuning is to help DB2 make the best possible use of the memory available for buffers. The overall buffer size has a significant effect on DB2 performance, because a large number of pages can significantly reduce I/O, which is the most time-consuming operation. However, if the total buffer size is too large, and there is not enough storage to allocate them, a minimum system buffer pool for each page size is allocated, and performance is sharply reduced. To calculate the maximum buffer size, DB2 considers all other storage utilization, the operating system, and any other applications. Once the total available size is determined, this area can be divided into different buffer pools to improve utilization. If there are table spaces with different page sizes, there must be at least one buffer pool per page size.

Having more than one buffer pool can preserve data in the buffers. For example, you might have a database with many very-frequently used small tables, which would normally be in the buffer in their entirety to be accessible very quickly. You might also have a query that runs against a very...
large table that uses the same buffer pool and involves reading more pages than the total buffer size. When this query runs, the pages from the small, very frequently used tables are lost, making it necessary to re-read them when they are needed again. If the small tables have their own buffer pool, thereby making it necessary for them to have their own table space, their pages cannot be overwritten by the large query. This can lead to better overall system performance, albeit at the price of a small negative effect on the large query.

Often tuning is a trade-off between different functions of a system to achieve an overall performance gain. It is essential to prioritize functions and keep total throughput and usage in mind while making adjustments to the performance of a system.

With DB2 Version 8 and higher, you can change buffer pool sizes without shutting down the database. The ALTER BUFFERPOOL statement with the IMMEDIATE option takes effect right away, unless there is not enough reserved space in the database-shared memory to allocate new space. This feature can be used to tune database performance according to periodic changes in use, such as switching from daytime interactive use to nighttime batch work. DB2 Version 9.1 and higher enables fully automated size management of a bufferpool. DB2 self-tuning memory manager (STMM) controls this automation process.

**Physical storage organization**

Once tables are distributed among table spaces, you need to determine their physical storage. A table space can be stored in multiple containers and can be either SMS or DMS. SMS is easier to administer and might be a good choice for table spaces containing many small, diverse tables (such as a catalog table space), especially if these tables contain LOBs.

To reduce the overhead of extending the SMS containers one page at a time, run the `db2empfa` command. This sets the value of the database configuration parameter MULTIPAGE_ALLOC to YES.

DMS usually has better performance and provides the flexibility of storing indexes and LOB data separately. Multiple containers for a table space should typically be placed on separate physical volumes. This can improve the parallelism of some I/Os. When you have multiple user table spaces and multiple devices, consider the application logic to distribute the workload as evenly as possible among these devices.

RAID devices have their own special considerations. EXTENTSIZE should be equal to, or a multiple of, the RAID stripe size. PREFETCHSIZE should be the RAID stripe size multiplied by the number of RAID parallel devices (or a multiple of this product), and a multiple of the EXTENTSIZE. DB2 comes with its own registry variables, enabling you to enhance your specific environment. Enter the following command to enable I/O parallelism within a single container:

```
db2set DB2_PARALLEL_IO=*.
```

As in other areas of performance evaluation, the only sure way to know if a change has a beneficial effect is to conduct benchmarks. Benchmarks are somewhat more complicated to use for physical organization changes, because a comparatively large amount of effort is necessary to change table spaces. The most practical way is to minimize the number of cases during the design
phase so that fewer cases need to be benchmarked later. Perhaps the only situation that merits
the time and energy to rigorously benchmark competing designs is when performance is extremely
important, and when there is a likelihood of a significant performance difference between designs.
Emphasis should be placed on buffer pools, making sure that they are not allocated in virtual
memory and that they are utilized in the most efficient manner.

**Moving databases**

Always re-evaluate tuning parameters and physical organization of the database before moving it
to a different system, even when the systems are on the same kind of platform. The results can be
unexpected and require substantial work to resolve.

In a real-life situation, a DBA copied a well-tuned database from a Windows server with 1 GB of
storage to a laptop running Windows with 256 MB of storage. Connection, which was subsecond
on the server, took 45 minutes on the laptop. The DBA resolved the problem by reducing the buffer
pool size and other memory parameters.

The question becomes even more difficult if the platforms are different. Even between UNIX and
Windows, what is optimal on one system might not be on the other. Following are some tips to
consider:

- If the database copy is intended for production, repeat the tuning process.
- If the database has to be moved to the z/OS® platform, consult the appropriate manuals and
  IBM Redbooks®.
- On DB2 for i, physical setup and tuning is done outside of the database environment. Consult
  the IBM i system management manuals.

**Conclusion**

This article covered quite a bit of material, and it is by no means everything you should know about
database design and performance. The discussion focused on a couple of big issues of database
design without getting into the details of query optimization and application considerations.
Designing your database is most important, because everything else is layered on top, so your
initial planning should be comprehensive. For your convenience, other online references are
provided below so you can continue your education on this topic.

**Acknowledgments**

Thank you to Gabor Wieser and David J. Kline who wrote the original version of this article in
2002.
Related topics

- DB2 Express-C download
- DB2 for Linux, UNIX, and Windows Information Center
- DB2 Storage Trivia

© Copyright IBM Corporation 2002, 2010
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)