

Data Placement

Presented by DB2 Developer Domain

<http://www7b.software.ibm.com/dmdd/>

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

1. Introduction	2
2. Creating a database	4
3. Using schemas	10
4. Table space states	13
5. Creating and manipulating various DB2 objects	14
6. The DB2 storage triangle	24
7. SMS table spaces	26
8. DMS table spaces	29
9. Conclusion	32

Section 1. Introduction

What this tutorial is about

This tutorial discusses the creation of DB2® databases, as well as the various methods used for placing and storing objects within a database. This tutorial is the second in a series of six tutorials that you can use to help you prepare for the DB2 Database Administration Certification (Exam 701). The material in this tutorial primarily covers the objectives in Section 2 of the exam, entitled "Data Placement." You can view these objectives at: <http://www.ibm.com/certify/tests/obj701.shtml>.

In this tutorial, you will learn how to:

- Create a database
- Use schemas
- Determine the various table space states
- Create and manipulate the various DB2 objects
- Create an SMS table space and understand its characteristics
- Create an DMS table space and understand its characteristics

You do not need a copy of DB2 Universal Database™ to complete this tutorial. However, it is strongly recommended that you [download and install a copy](#).

Who should take this tutorial?

This tutorial is geared toward database administrators (DBAs) familiar with basic database concepts who are interested in becoming IBM-Certified Database Administrators for DB2 UDB V8 for Linux, UNIX®, and Windows™. This tutorial will help you prepare for the data placement section of the DB2 DBA Certification Test (Test 701).

You can prepare for this tutorial by completing [the tutorials for the DB2 Fundamentals Exam \(Exam 700\)](#).

About the author

Dwaine R. Snow is the product manager for DB2 UDB partitioned databases. In this

role, he is helping to determine the future direction of the product and decide what functional enhancements need to be made to it to ensure that it meets customer requirements. Dwaine has worked in a DB2 Universal Database services consultant role with IBM Canada for a number of years. In this role, he has provided on-site assistance with DB2 UDB database and application planning, project planning and execution, large complex OLTP and Decision Support system design, database and application design, and performance tuning, as well as client/server and legacy system integration.

Dwaine has written a number of books and articles on DB2, including the *DB2 Universal Database Advanced DBA Certification Guide and Reference*, and has presented at conferences around the world.

Acknowledgements

I would like to thank Raul F. Chong for his help in completing this tutorial and for reviewing the material to ensure that it will help you take and pass the data placement portion of the DBA Certification exam.

Section 2. Creating a database

Database directories

A *system database directory* file exists for each instance of the database manager, and contains one entry for each database that has been cataloged for this instance.

Databases are implicitly cataloged when the `create database` command is issued, and can be explicitly cataloged with the `catalog database` command.

A *local database directory* file exists in each drive or path in which a database has been defined. This directory contains one entry for each database that is accessible from that location.

Creating a database

When you create a database, the following tasks are done for you:

- The system catalog tables that are needed by the database are set up.
 - The database recovery log is allocated.
 - The database configuration file is created and the default values are set.
 - The database utilities are bound to the database.
-

The create database command

To create a database, issue the `create database` command.

You have the option to specify the following:

- Database partition number for the catalog partition
- Drive or path on which to create the database
- Codeset and territory
- Collating sequence
- Default extent size
- Whether the database should be automatically configured
- Table space definitions for the CATALOG, TEMPORARY, and USERSPACE1 table spaces

The default database

The `create database` command creates three default table spaces:

- SYSCATSPACE: For the system catalog tables
- TEMPSPACE1: For system-created temporary tables
- USERSPACE1: The default table space for user-created objects

SYSCATSPACE cannot be dropped. The TEMPSPACE1 table space can be dropped once another temporary table space has been created. The USERSPACE1 table space can be dropped once another user-created table space has been created.

The system catalogs

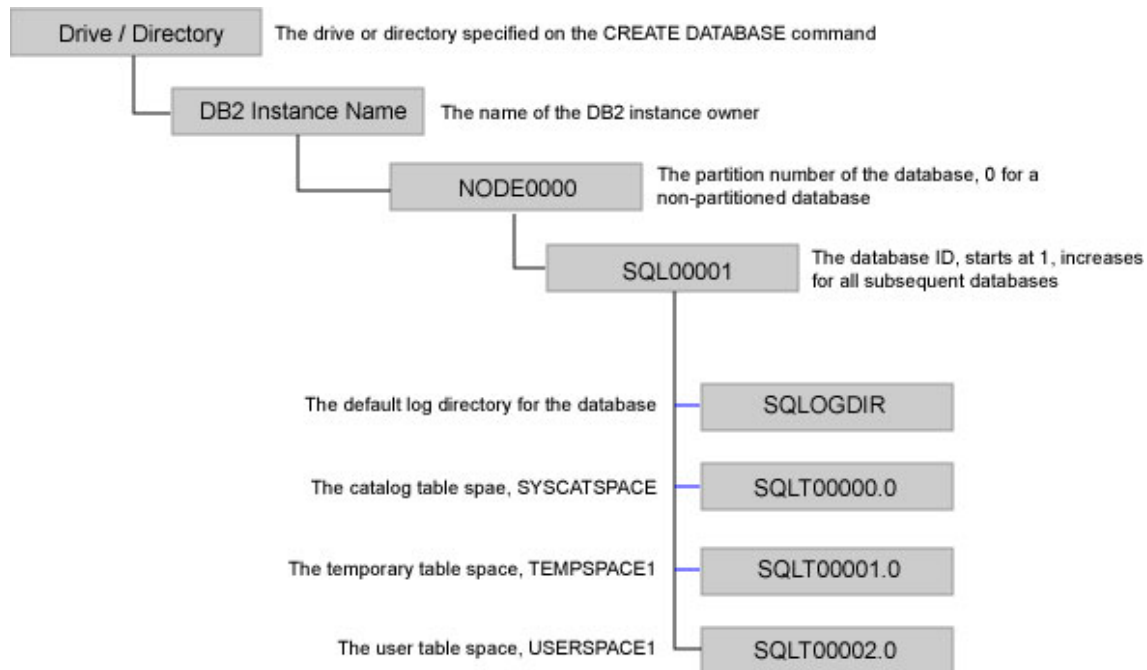
A set of system catalog tables is created and maintained for each database. These tables contain information about the definitions of the database objects (tables, views, indexes, and packages, for example) and security information about the type of access that users have to these objects. These tables are stored in the SYSCATSPACE table space.

The directory structure

The `create database` command allows you to specify the drive or directory on which to create the database, depending on the operating system.

- If no drive or directory is specified, the database is created on the path specified by the DFTDBPATH instance (database manager) configuration parameter.
- If no drive or directory is specified, and the DFTDBPATH instance level configuration parameter is not set, the database is created on the drive or path where the `create database` command was executed.

The `create database` command creates a series of subdirectories as illustrated in the figure below:



The first subdirectory is named after the instance owner for the instance in which the database was created. Under this subdirectory, DB2 creates a directory that indicates which database partition the database was created in.

For a nonpartitioned database the directory will be `NODE0000`. For a partitioned database, the directory will be named `NODExxxx`, where `xxxx` will be the four digit partition number for the database instance as designated in the `db2nodes.cfg` file. For example, for partition number 43, the directory would be `NODE0043`.

Note that in Windows, instances do not really have an instance owner, so the name of the instance (e.g., DB2) will be used in place of the instance owner's ID.

Since more than one database can be created on the same drive or directory, each database must have its own unique subdirectory. Under the `NODExxxx` directory, there will be an `SQLxxxxxx` directory for every database that was created on the drive/directory. For example, imagine that we have two databases, MYDB and SAMPLE, that were both created on the C: drive on Windows. There will be two directories: `SQL00001` and `SQL00002`.

In order to determine the directory under which the database was created, enter the command `list database directory on C:.` This will produce output like the following:

Database 1 entry:

Database alias	= MYDB
Database name	= MYDB
Database directory	=SQL00002
Database release level	= a.00
Comment	=
Directory entry type	= Home
Catalog database partition number	= 0

Database 2 entry:

Database alias	= SAMPLE
Database name	= SAMPLE
Database directory	= SQL00001
Database release level	= a.00
Comment	=
Directory entry type	= Home
Catalog database partition number	= 0
Database partition number	= 0

In the example above, the database SAMPLE was created in the SQL00001 directory and the database MYDB was created in the SQL00002 directory under the NODExxxx directory.

By default:

- The system catalog table space (SYSCATSPACE) will use the directory SQLT0000.0.
- The system temporary table space (TEMPSPACE1) will use the directory SQLT0001.0.
- The default user table space (USERSPACE1) will use the directory SQLT0002.0.

There is also a subdirectory named SQLOGDIR which holds the database log files. This is the default location for the log files, though you can change it once the database has been created.

Example Linux/UNIX create database commands

To create a database on the directory (file system) /database, use the following

command:

```
create database sample on /database
```

If this command were executed in the instance named dbinst, on the server where database partition 0 is defined, the following directory structures would be created:

- /database/dbinst/NODE0000/sqlldbidr
 - /database/dbinst/NODE0000/SQL00001
-

Example Windows create database commands

To create a database on the D: drive, use the following command:

```
create database sample on D:
```

If this command were executed in the instance named dbinst, on the server where database partition 0 is defined, the following directory structures would be created:

- D:\dbinst\NODE0000\sqlldbidr
 - D:\dbinst\NODE0000\SQL00001
-

Creating the USERSPACE1 table space as DMS

To create a database and define the USERSPACE1 table space to be DMS (see [DMS table spaces](#) on page 29) using two file containers, use the following commands:

On Linux or UNIX:

```
create database sample2 user table space managed by database
    using(file '/dbfiles/cont0' 5000, file '/dbfiles/cont1' 5000)
```

On Windows:

```
create database sample2 user table space managed by database
    using(file 'c:\dbfiles\cont0' 5000, file 'c:\dbfiles\cont1' 5000)
```

Creating the TEMPSPACE1 table space with user-defined containers

To create a database and define the TEMPSPACE1 table space to use two SMS containers (see [SMS table spaces](#) on page 26), use the following commands:

On Linux or UNIX:

```
create database sample3 temporary tablespace managed by system
    using('/dbfiles/cont0', '/dbfiles/cont1')
```

On Windows:

```
create database sample3 temporary tablespace managed by system
    using('c:\dbfiles\cont0', 'c:\dbfiles\cont1')
```

Changing the collating sequence for the database

The command (in UNIX):

```
creates a database SAMPLE on /mydbs collate using identity
```

or

```
creates a database SAMPLE on D: collate using identity
```

creates a database and compares strings byte for byte, since the collating sequence has been set to identity.

Section 3. Using schemas

What is a schema?

A *schema* is a high-level qualifier for database objects created within a database. It is a collection of database objects such as tables, views, indexes, or triggers. It provides a logical classification of database objects. While you are organizing your data into tables, it may also be beneficial to group tables and other related objects together.

This is done by defining a schema through the use of the `create schema` command. Information about the schema is kept in the system catalog tables of the database to which you are connected. As other objects are created, they can be placed within this schema.

System schemas

A set of system schemas are created with every database and placed into the SYSCATSPACE table space:

- **SYSIBM:**
 - The base system catalogs
 - Direct access is not recommended
 - **SYSCAT:**
 - SELECT authority granted to PUBLIC on this schema
 - Catalog read-only views
 - Recommended way to obtain catalog information
 - **SYSSTAT:**
 - Updatable catalog views -- influences the optimizer
 - **SYSFUN:**
 - User-defined functions
-

How is a schema used in DB2?

Use a schema to fully qualify a table or other object name, as follows:

```
schemaname.tablename
```

You can have multiple tables with the same name but different schema names. Thus, the table `user1.staff` is not the same as `user2.staff`. As a result, you can use schemas to create logical databases within a DB2 database.

To create a schema, use the `create schema` command.

Who can use a schema?

When you can create a schema, you can specify its owner using the `authorization` keyword. If you do not specify an owner, the authorization ID that executed the `create schema` statement will be the owner of the schema. Privileges on the schema can also be granted to users and/or groups at the same time. (See the [first tutorial](#) in this series for more information on privileges.)

Once a schema exists, the owner of the schema can grant `CREATE_IN` privilege on the schema to other users or groups.

Specifying the schema when creating an object

The schema name for an object can be explicitly specified as follows:

```
create table DWAINE.table1 (c1 int, c2 int)
```

If the user `DWAINE` connects to the database `SAMPLE`, and issues the following statement:

```
create table t2 (c1 int)
```

the schema `DWAINE` is created (as long as `IMPLICIT_SCHEMA` has not been revoked from the user `DWAINE`) as well as the table in the database.

Note that the ID used to connect to the database is known as the *authorization ID*.

Specifying the schema when using DML commands

When using DML commands (e.g., `select`, `insert`, `update`, `delete`) on database objects:

- The object schema can be explicitly specified on the object name, i.e., `schema1.table1`.
- The object schema can be specified using the `set current schema` or `set current sqlid` command.
- If no object schema is explicitly specified, the schema will be set to the current authorization ID.

For example, if the user *DWAINÉ* connects to the database *SAMPLE*, and issues the following statement:

```
select * from t2
```

DWAINÉ.T2 is selected from the table, if the table exists. Otherwise an error is returned.

Section 4. Table space states

Determining a table space's state

To find the state for the table spaces in a database, issue the following:

```
list tablespaces show detail
```

Table space states

A table space can have a number of different states, as illustrated in the figure below:

0x0	Normal
0x1	Quiesced: SHARE
0x2	Quiesced: UPDATE
0x4	Quiesced: EXCLUSIVE
0x8	Load pending
0x10	Delete pending
0x20	Backup pending
0x40	Roll forward in progress
0x80	Roll forward pending
0x100	Restore pending
0x100	Recovery pending (not used)
0x200	Disable pending
0x400	Reorg in progress
0x800	Backup in progress
0x1000	Storage must be defined
0x2000	Restore in progress
0x4000	Offline and not accessible
0x8000	Drop pending
0x20000000	Storage may be defined
0x40000000	StorDef is in 'final' state
0x80000000	StorDef was changed prior to rollforward
0x100000000	DMS rebalancer is active
0x200000000	TBS deletion in progress
0x400000000	TBS creation in progress
0x8	For service use only

Section 5. Creating and manipulating various DB2 objects

Introduction

This section discusses the purpose and use of:

- Buffer pools
 - Table spaces
 - Tables and indexes
 - Views
 - Identity columns
 - Temporary tables
 - Constraints
 - Triggers
-

Buffer pools

The database buffer pool area is a piece of memory used to cache a table's index and data pages as they are being read from disk to be scanned or modified. The buffer pool area helps to improve database system performance by allowing data to be accessed from memory instead of from disk. Because memory access is much faster than disk access, the less often that DB2 needs to read from or write to a disk, the better the system will perform.

When a database is created, one default buffer pool is created for the database. This buffer pool is named `IBMDEFAULTBP`, and it has a page size of 4 KB and is sized depending on the operating system. For Windows, the default buffer pool is 250 pages or 1 MB; for UNIX, the default buffer pool is 1,000 pages or 4 MB. The default buffer pool cannot be dropped, however, its size can be changed using the `alter bufferpool` command.

Creating a buffer pool

The `create bufferpool` command has options to specify the following:

- `buffer pool name`: Specifies the name of the buffer pool. The name cannot be used for any other buffer pools and cannot begin with the characters `SYS` or `IBM`.

- `immediate`: Specifies that the buffer pool will be created immediately if there is enough memory available on the system. If there is not enough reserved space in the database shared memory to allocate the new buffer pool, a warning is returned and buffer pool creation will be DEFERRED, as described below. (Note that `immediate` is the default.)
- `deferred`: Specifies that the buffer pool will be created the next time the database is stopped and restarted.
- `all dbpartitionnums`: Specifies that the buffer pool will be created on all partitions in the database. This is the default if no database partition group is specified.
- `database partition group`: Specifies the database partition group(s) on which the buffer pool will be created. The buffer pool will be created on all database partitions that are part of the specified database partition groups.
- `size`: Specifies the size of the buffer pool and is defined in number of pages. In a partitioned database, this will be the default size for all database partitions where the buffer pool exists.
- `numblockpages`: Specifies the number of pages to be created in the block-based area of the buffer pool. The actual value of `numblockpages` may differ from what was specified because the size must be a multiple of the `blocksize`. The block-based area of the buffer pool cannot be more than 98 percent of the total buffer pool size. Specifying a value of 0 will disable block I/O for the buffer pool.
- `blocksize`: Specifies the number of pages within a given block in the block-based area of the buffer pool. The block size must be between 2 and 256 pages; the default value is 32 pages.
- `pagesize`: Specifies the page size used for the buffer pool. The default page size is 4 KB or 4,096 bytes. The page size can be specified in either bytes or kilobytes.
- `extended storage/not extended storage`: Specifies whether or not buffer pool victim pages will be copied to a secondary cache called *extended storage*. Retrieving data from extended storage is more efficient than retrieving it from disk but less efficient than retrieving it from the buffer pool, so it is not applicable to 64-bit environments.

Note: Once a page size and name for a buffer pool have been defined, they cannot be altered.

Example create bufferpool statements

The following statement:

```
create bufferpool BP1 size 25000
```

creates a buffer pool named BP1 with a size of 100 MB (25,000 4 KB pages). Since the

page size is not specified, the buffer pool uses the default page size of 4 KB. Since the `immediate` option is the default, the buffer pool is allocated immediately and available for use as long as there is enough memory available to fulfill the request.

The following statement:

```
create bufferpool BP2 size 25000 pagesize 8 K
```

creates a buffer pool named BP2 with a size of 200 MB (25,000 8 KB pages). The buffer pool uses an 8 KB page size. Since the `immediate` option is the default, the buffer pool is allocated immediately and be available for use as long as there is enough memory available to fulfill the request.

The following statement:

```
create bufferpool BP3 deferred size 1000000
```

creates a buffer pool named BP3 with a size of 4 GB (1,000,000 4 KB pages). Since the page size is not specified, the buffer pool uses the default page size of 4 KB. Since the `deferred` option is specified, the buffer pool is not allocated until the database is stopped and restarted.

Creating tables

In order to create a table in a database, you must first be connected to the database. In addition, you must have SYSADM authority in the instance, or DBADM authority or CREATETAB privilege in the database.

When creating a table, you can specify the following:

- Schema
- Table name
- Column definitions
- Primary/foreign keys
- Table space for the data, index, and long objects

The figure below illustrates an example:

```
create table artists
(  artno          SMALLINT NOT NULL
,  name           VARCHAR (50)  WITH DEFAULT 'abc'
,  classification CHAR (1)  NOT NULL
,  bio            CLOB (100K)  LOGGED
,  article        DATALINK LINKTYPE URL FILE
                    LINK CONTROL MODE DB2OPTIONS,
,  picture        BLOB (2M)  NOT LOGGED COMPAT )
INDEX IN indtbsp
LONG IN longtbsp
IN datatbsp;
```

Where are tables created?

If a table is created without the `IN` clause, the table data (and its indexes and LOB data) is placed in the following order:

- In the `IBMDEFAULTGROUP` table space (if it exists and if the page size is sufficient)
- In a user-created table space which is of the smallest page size that is sufficient for the table
- In `USERSPACE1` (if it exists and has a sufficient page size)

The `IN`, `INDEX IN`, and `LONG IN` clauses specify the table spaces in which the regular table data, index, and long objects are to be stored. Note that this only applies to DMS table spaces.

Obtaining table information

You can obtain table information with the following commands:

- `list tables -- list tables for the current user`
- `list tables for all -- list all tables defined in the database`
- `list tables for schema schemaname -- list tables for the specified schema`
- `describe table tablename -- show the structure of the specified table`

For example, the following command:

describe table department

produces the following output:

Column name	Type schema	Type name	Length	Scale	Nulls
DEPTNO	SYSIBM	CHARACTER	3	0	No
DEPTNAME	SYSIBM	VARCHAR	29	0	No
MGRNO	SYSIBM	CHARACTER	6	0	Yes
ADMRDEPT	SYSIBM	CHARACTER	3	0	No
LOCATION	SYSIBM	CHARACTER	16	0	Yes

Indexes

An index can:

- Be ascending or descending (if not specified, the default is ascending)
- Be unique or non-unique (if not specified, the default is non-unique)
- Be compound
- Be used to enforce clustering
- Be bi-directional (controlled by `allow` or `disallow reverse scans`)
- Include additional columns (only applicable for unique indexes)

Here are a number of `create unique` statements that illustrate these options:

```
create unique index itemno on albums (itemno) desc
create index clx1 on stock (shipdate) cluster allow reverse scans
create unique index incidx on stock (itemno) include (itemname)
create index item on stock (itemno) disallow reverse scans collect detailed statistics
```

Identity columns

An *identity column* is a numeric column in a table that causes DB2 to automatically generate a unique numeric value for each row that is inserted into the table. A table can have a maximum of one identity column. The values for the column can be generated by DB2 *always* or *by default*:

- If values are always generated, the DB2 database always generates them, and applications are not allowed to provide an explicit value.
- If values are generated by default, then the values can be explicitly provided by an application; DB2 generates a value only if the application does not provide it. Thus, DB2 cannot guarantee the uniqueness of the values. This option is intended for data propagation or loading/unloading of a table.

Let's look at an example. Given the table created using the following command:

```
create table inventory (partno INTEGER GENERATED ALWAYS AS IDENTITY
                        (START WITH 100 INCREMENT BY 1), description CHAR(20) )
```

and the following statements:

Statement

```
insert into inventory VALUES (DEFAULT,'door')
insert into inventory (description) VALUES ('hinge')
insert into inventory VALUES (200,'windor')
insert into inventory (description) VALUES ('frame')
```

Result

```
inserts 100,door
inserts 101,hinge
error
inserts 102,frame
```

then the statement `SELECT * FROM inventory` gives the following:

```
100 door
101 hinge
102 frame
```

Views

Views are derived from one or more base tables, nicknames, or views, and can be used interchangeably with base tables when retrieving data. When changes are made to the data shown in a view, the data is changed in the table itself. A view can be created to limit access to sensitive data, while allowing more general access to other data.

The data for a view is not stored separately from the table. In other words, a view uses no space in the database, other than its definition in the system catalogs.

The creator of a view needs to have at least `SELECT` privilege on the base tables referenced in the view definition.

The information about all existing views is stored in:

- SYSCAT.VIEWS
- SYSCAT.VIEWDEP
- SYSCAT.TABLES

These `create view` statements illustrate how views work:

```
create view DEPTSALARY AS SELECT DEPTNO, DEPTNAME, SUM(SALARY)
      AS TOTALS FROM PAYROLL GROUP BY DEPTNO,DEPTNAME
create view EMPSALARY AS SELECT EMPNO, EMPNAME, SALARY FROM PAYROLL,
      PERSONNEL WHERE EMPNO=EMPNUMB
```

The with check option

`with check option` specifies the constraint that every row that is inserted or updated through a view must conform to the definition of the view. A row that does not conform to the definition of the view is a row that does not satisfy the search conditions of the view.

For example, consider this command:

```
create view emp_view2 (empno, empname, deptno) AS (SELECT id, name,
      dept FROM employee WHERE dept = 10)with check option
```

When this view is used to insert or update with new values, `with check option` restricts the input values for the `dept` column.

Constraints

There are a number of types of constraints in DB2:

- Referential integrity constraints
- Unique constraints
- Check constraints
- Informational constraints

You cannot directly modify a constraint; you must instead drop it and create a new constraint with the characteristics you want.

We'll consider each in the next few panels.

Referential integrity constraints

Referential integrity constraints are defined when a table is created, or subsequently using the `alter table` statement.

The clauses that establish referential integrity are:

- The primary key clause
- The unique constraint clause
- The foreign key clause
- The references clause

For example:

```
create table artists (artno INT, ... primary key (artno) foreign key dept (workdept)
references department on delete no action)
```

Let's take a look at the various referential integrity rules:

Insert rules:

- There is an implicit rule to back out of an insert if a parent is not found.

Delete rules:

- Restrict: A parent row is not deleted if dependent rows are found.
- Cascade: Deleting a row in a parent table automatically deletes any related rows in dependent tables.
- No Action (the default): Enforces presence of parent row for every child after all other referential constraints are applied.
- Set Null: Foreign key fields set to null; other columns left unchanged.

Update rules:

- Restrict: An update for a parent key will be rejected if a row in dependent table matches the original values of key.
 - No Action (the default): An update will be rejected for parent key if there is no matching row in the dependent table.
-

Unique constraints

A unique constraint can be used as the primary key for a foreign key constraint, just like an explicitly declared primary key. This allows RI constraints (see [Referential integrity constraints](#) on page 21) to be placed on different columns within the same table.

A unique constraint forces the values in the column to be unique; the column cannot contain null values.

Check constraints

A check constraint is used to enforce data integrity at the table level. It forces values in the table to conform to the constraint. All subsequent inserts and updates must conform to the defined constraint(s) on the table, or the statement will fail.

If existing rows in the table do not meet the constraint, the constraint cannot be defined. Constraint checking can be turned off to speed up the addition of a large amount of data, but the table will be placed in CHECK PENDING state.

Informational constraints

Informational constraints are rules that can be used by the optimizer, but are not enforced during run time. Because other constraints may result in the overhead for insert, update, or delete operations, informational constraints may be a better alternative if the application already verifies the data.

Informational constraints can be:

- **ENFORCED:** The constraint is enforced by the database manager during normal operations such as insert, update, or delete.
- **NOT ENFORCED:** When this is used, DB2 may return wrong results when any data in the table violates the constraint.
- **ENABLE QUERY OPTIMIZATION:** The constraint can be used for query optimization under appropriate circumstances.
- **DISABLE QUERY OPTIMIZATION:** The constraint can not be used for query optimization.

Triggers

A *trigger* defines a set of actions that are activated, or triggered, by an action on a specified base table. The actions triggered may cause other changes to the database or raise an exception. A trigger can be fired *before* or *after* inserts, updates, or deletes.

Triggers are used for:

- **Validation:** Used in this way, triggers are similar to constraints but more flexible.
- **Conditioning:** Used in this way, triggers allow new data to be modified/conditioned to a predefined value.
- **Integrity:** Used in this way, triggers are similar to referential integrity but more flexible.

Section 6. The DB2 storage triangle

The storage triangle

Within a DB2 database, the storage of the data and indexes is defined and controlled at four different levels. In order to accommodate partitioned databases, there is an abstract layer referred to as *partition groups*, as shown in the figure below. A partition group is a grouping or collection of one or more database partitions within a database. When a table space is created, it is assigned to a partition group and will only be created on the database partitions that are part of that partition group. Each table space must have one or more containers that define the physical storage for the table space. A container can be an operating system directory, a file with a predetermined size, a raw device such as an unformatted hard drive, a partition on the hard drive, or a logical volume.

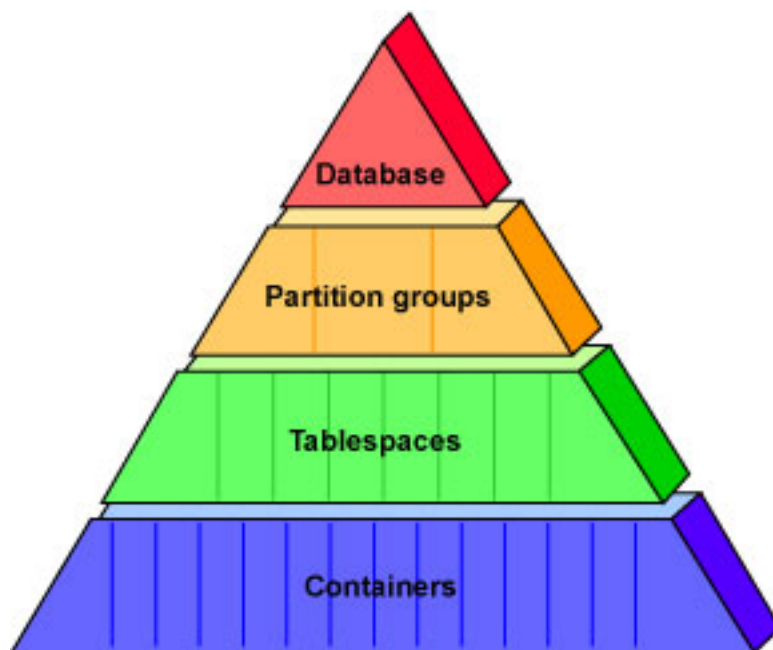


Table spaces

A *table space* is a logical entity used to define where tables and indexes will be stored within a database. As all DB2 tables and indexes reside in table spaces, this allows complete control over where the table and index data is physically stored.

A table space can be created using one or more underlying physical storage devices,

known as *containers*. This provides the ability to create a physical database design that will provide optimal performance in any physical environment.

Details about the table spaces in a database can be obtained using the following commands:

- `get snapshot for tablespaces`
- `list tablespaces`

Section 7. SMS table spaces

Introduction

System Managed Space (SMS) table spaces use the file system manager provided by the operating system to allocate and manage the space where the tables are stored. Within an SMS table space, each container is an operating system directory, and table objects are created as files within that directory. When creating an SMS table space, the user must specify the name of the directory for each of the containers. DB2 will create the tables within the directories used in the table space by using unique file names for each object.

If a table space is created with more than one container, DB2 will balance the amount of data written to the containers. Since containers cannot be dynamically added to an SMS table space once it has been created, it is important to know the size requirements of the table space and create all required containers when the table space is created.

SMS table spaces characteristics

With SMS table spaces:

- All table data and indexes share the same table space.
- Each table in a table space is given its own file name used by all containers. The file extension denotes the type of the data stored in the file.
- Dynamic file growth is possible, with an upper boundary on size governed by the number of containers, operating system limits governed by the size of the file system, and operating system limits governed by the size of individual files.
- When all space in a single container is allocated, the table space is considered full even if space remains in other containers.
- New containers can only be added to SMS on a partition that does not yet have any containers.
- On Linux or UNIX, the file system size may be increased.

SMS table spaces are very easy to administer, and are recommended for the TEMP table space.

Creating SMS table spaces

To create an SMS table space, use the following command:

```
create table space TS1 managed by system using ('path1', 'path2', 'path3')
```

When the path is specified for an SMS container, it can be either an absolute path or a relative path to the directory. If the directory does not exist, DB2 will create it. If the directory does exist, it cannot contain any files or subdirectories.

For example, this command:

```
create table space ts1 managed by system using ('D:\DIR1')
```

specifies the absolute path to the directory; therefore DB2 would create the DIR1 directory on the D: drive on the database server if it does not already exist.

This command:

```
create tablespace ts2 managed by system using ('DIR1')
```

specifies the relative path DIR1; therefore DB2 would create the DIR1 directory under the database home directory.

The following SQL statements create an SMS table space with three containers on three separate drives or file systems. Note that the table space name is the same; the examples are showing the differences between the UNIX/Linux and Windows table space definitions:

```
create tablespace smstbspc managed by system
    using ('d:\tbspc1', 'e:\tbspc2', 'f:\ tbspc3')
create tablespace smstbspc managed by system
    using ('/dbase/container1', '/dbase/container2', '/dbase/container3')
```

Altering SMS table spaces

SMS table spaces can only be altered to change the prefetch size. Containers *cannot* be added to an SMS table space using the `alter` command. However, containers can be redefined, added, or removed during a redirected restore.

Enabling multipage file allocation

In an SMS table space, by default object files are extended one page at a time as the object grows. When inserting a large number of rows into a table, this can result in a great deal of I/O, which can have a negative impact on the database performance. The `db2empfa` tool enables DB2 to allocate or extend the object file by a full extent at a time. This tool is enabled at the database level; when it is run, DB2 will allocate empty pages to fill up the last extent in all SMS table space containers within the specified database.

Note: Once the `db2empfa` tool has been run, its effects cannot be undone.

Section 8. DMS table spaces

Introduction

When using Database Managed Space (DMS) table spaces, the database manager controls the storage allocation within the table space. Within a DMS table space, a container can be either an operating system file, or a raw logical volume or disk partition. With DMS table spaces, the space is preallocated when the table space is created. When creating a DMS table space, the user must specify the name of the file, logical volume, or disk partition, as well as the size of the container(s).

DMS characteristics

With DMS table spaces:

- Space is allocated when the table space is created.
- Table space containers can be added or dropped using the `alter tablespace` command.
- When a container is added or dropped, the data is rebalanced automatically and asynchronously.
- Containers can be extended, reduced, or resized dynamically.
- The table space capacity is limited only by the physical storage.
- File system I/O is used for DMS file manipulation.
- Direct I/O is used for DMS raw manipulation.

DMS table spaces allow for the highest performance potential, especially for OLTP. They also allow the most flexible data placement, since you can split table objects (i.e., data, index, long varchar/LOB data) into different table spaces.

Creating a DMS table space with device containers

In Linux and UNIX, a device container is mapped to an underlying logical volume. In Windows, a device container is mapped to an unformatted disk partition. The device on which the container is created cannot be used for any other purpose -- in other words, it cannot contain any file systems and should not be formatted. When specifying the size of the container, make sure that all of the space on the device is used, since any unused space will not be available for any other purpose. However, this space can be

used at a later time if the table space containers are extended or resized. The size of the container can be specified in number of pages, KB, MB, or GB.

Some example commands that create DMS table spaces:

```
create tablespace mytbspc managed by database using (device '/dev/rmydisk1' 10000)
create tablespace mytbspc managed by database using (device '\\.\G:' 10000)
create tablespace mytbspc managed by database using (device '/dev/rmydisk1' 100M)
create tablespace mytbspc managed by database using (device '\\.\G:' 100M)
```

Creating a DMS table space with file containers

DB2 can also use files for DMS containers. When a file is specified, DB2 will preallocate the file with the specified size. With DMS files, you can have containers on the same file system or underlying volume, unlike device containers.

The size of the container can be specified in number of pages, KB, MB, or GB.

Some further example commands:

```
create tablespace mytbspc managed by database using (file '/dbfiles/ts1c1' 10000)
create tablespace mytbspc managed by database using (file 'G:\dbfiles\ts1c1' 10000)
create tablespace mytbspc managed by database using (file '/dbfiles/ts1c1' 100M)
create tablespace mytbspc managed by database using (file 'G:\dbfiles\ts1c1' 100M)
```

Altering a DMS table space

With DMS table spaces, the `alter database` command can be used to:

- Add containers
 - Drop containers
 - Extend containers
 - Resize containers, either larger or smaller
 - Change the prefetch size
-

SMS vs. DMS

The illustration below compares the merits of SMS and DMS:

	SMS	DMS
Striping	Yes	Yes
Object Management	Operating system (via unique file names)	DB2 (Object table and EMP extents)
Space Allocation	Grows/shrinks on demand	Preallocated
Ease of Administration	Best - Little/no tuning required (e.g. OS prefetching often very good) - Enlarge file systems(s) associated with containers	Good -Some tuning required (e.g. EXTENTSIZE PREFETCHSIZE) -Can enlarge table space via ALTER TABLESPACE ADD CONTAINER
Performance	Very Good	Best - Can achieve up to 5-10% advantage with raw containers - Index, LOBs, Data for a single table can be spanned across table spaces

Section 9. Conclusion

Summary

In this tutorial we discussed:

- Creating a database
- Using schemas
- Table space states
- Creating and manipulating the various DB2 objects
- Creating an SMS table space and understanding its characteristics
- Creating an DMS table space and understanding its characteristics

You are now better prepared to take the data placement portion of the DBA Certification test. Best of luck!

Resources

Check out the other parts of the DB2 V8.1 Database Administration certification prep series:

- [Server Management: DB2 V8.1 Database Administration certification prep, Part 1 of 6](#)
- [Database Access: DB2 V8.1 Database Administration certification prep, Part 3 of 6](#)
- [Monitoring DB2 Activity: DB2 V8.1 Database Administration certification prep, Part 4 of 6](#)
- [DB2 Utilities: DB2 V8.1 Database Administration certification prep, Part 5 of 6](#)
- [Backup and Recovery: DB2 V8.1 Database Administration certification prep, Part 6 of 6](#)

For more information on the DB2 V8.1 for Linux, UNIX, and Windows Database Administration Certification (Exam 701):

- Review the [IBM Data Management Skills information](#).
- Download a [self-study course for experienced Database Administrators \(DBAs\)](#) to quickly and easily gain skills in DB2 UDB.
- Download a [self-study course for experienced relational database programmers](#) who'd like to know more about DB2.

- See [General Certification information](#), including some book suggestions, exam objectives, and courses
 - Take a look at the [IBM Certification Exam Tool \(ICE\)](#), where you can use pre-assessment/sample tests to prepare for exams leading toward IBM certification.
 - Check out [developerWorks Toolbox](#) for one-stop access to over 1,000 IBM tools, middleware, and technologies from DB2, Lotus, Tivoli, and WebSphere for open standards-based Web services and application development.
-

Feedback

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT style sheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial [Building tutorials with the Toot-O-Matic](#) demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.