Defining Database Objects

Objectives

Defining Database Objects

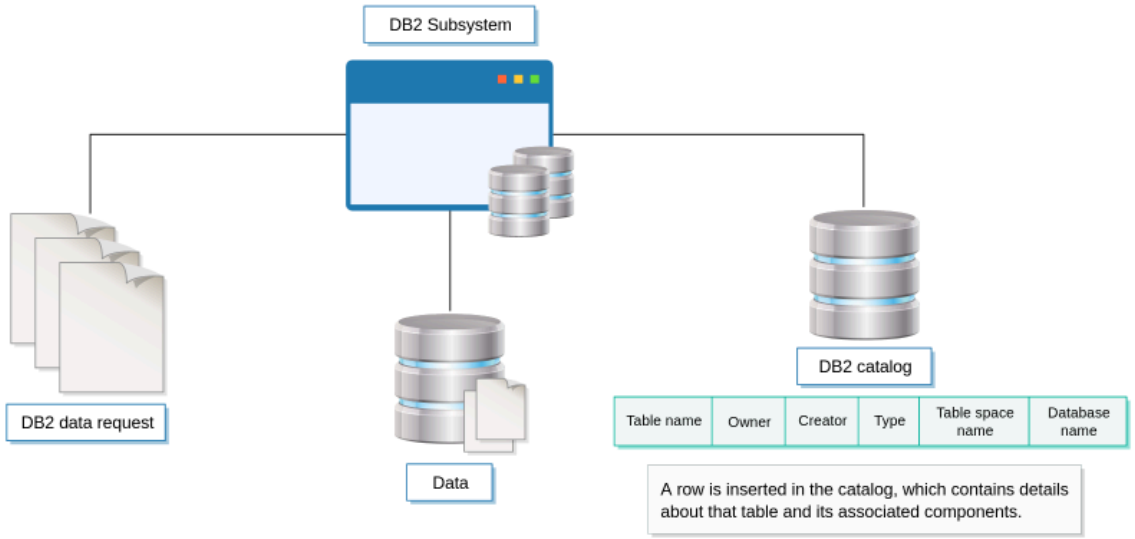
Db2 consists of a number of elements that are used to store and organize your data. In this module you will begin with the basics and see how a database and its related table spaces and tables are created using SQL statements. You will also look at the type of security used to manage Db2.

After completing the module you will be able to describe:

- The Types of Tables That Can be Used Within Db2
- Commonly Used SQL Statements Used to Manage Db2 Objects
- Security That Can be Configured for Db2 Objects
- The SQL Statements Used to Create a Database, Table Space and Table

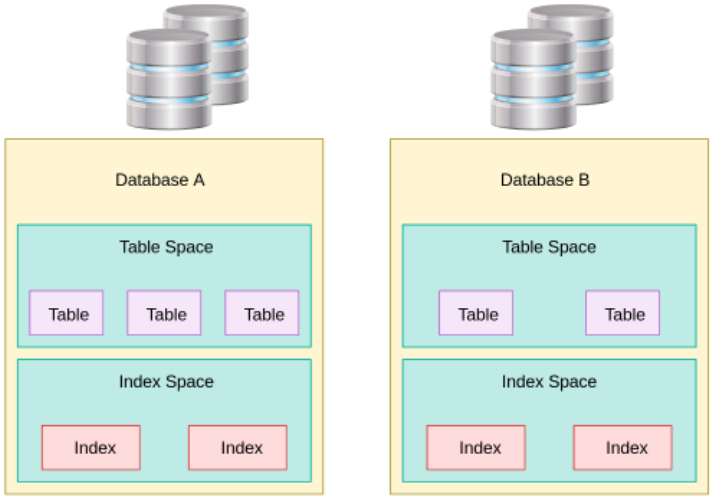
There are many aspects to consider when needing to store your data within a Db2 database; storage requirements, security considerations, and backup and recovery to name a few. Following these decisions, comes the more specific building of your database and its components.

This module will look at these items and in the case of the database creation, show the code that can be used to perform this task.

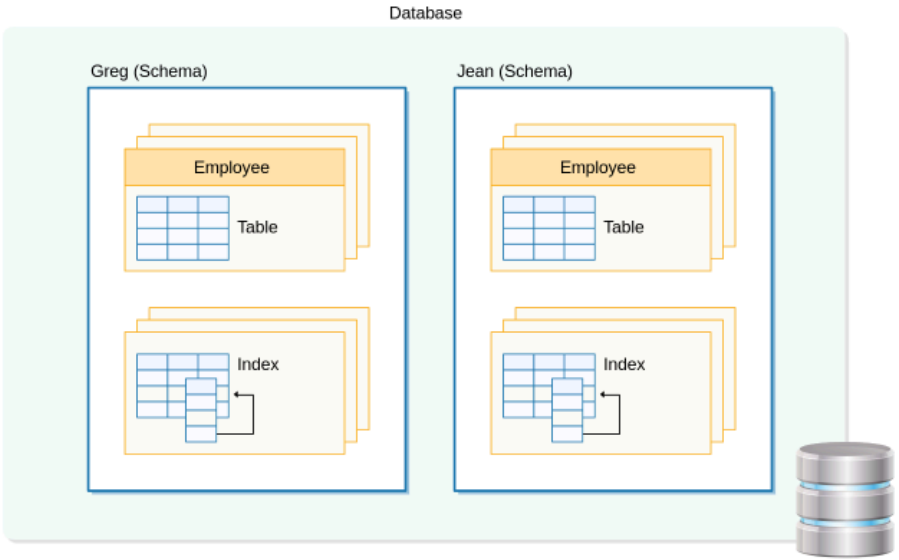


Within a Db2 environment, data is constantly being created, modified, or deleted. For Db2 to keep track of changes to objects such as table spaces, indexes, tables, and storage groups, it maintains the Db2 catalog (which itself contains a set of tables) that describe the attributes of these items.

Click Play to see how the Db2 catalog performs this task.



A Db2 subsystem often contains a number of databases. These databases consist of a logical collection of table spaces and index spaces, whose details are stored in the Db2 catalog. Even though there are individual components that comprise a database, a number of commands can be invoked to control it as a single entity. For example, an entire database can be stopped and started as a unit, and the status of all the database objects can be shown using a single database command.



Specific objects within a Db2 database can be logically grouped together to form a schema. The types of objects you may find in a schema includes tables, indexes, table spaces, distinct types, functions, stored procedures, and triggers. Some of these are discussed in this module, while others are discussed in later courses.

The benefit of schemas is that it allows you to create multiple versions of an object that is used for the same purpose. For example, if an EMPLOYEE table is created under a schema name called GREG, then the table name would become GREG.EMPLOYEE. This type of Db2 object naming standard allows other EMPLOYEE tables to be created within the database, whilst enforcing the naming uniqueness of each.

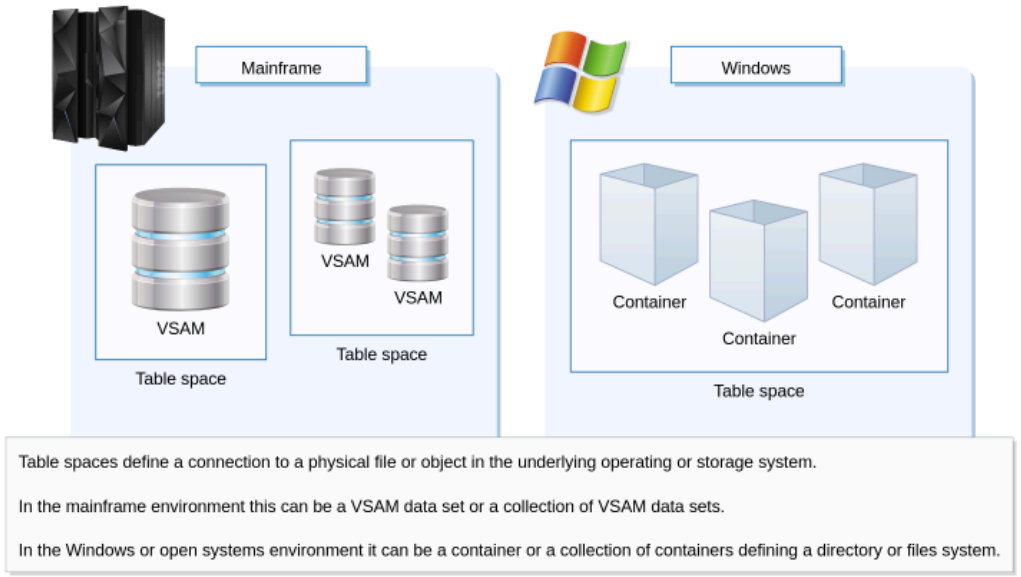
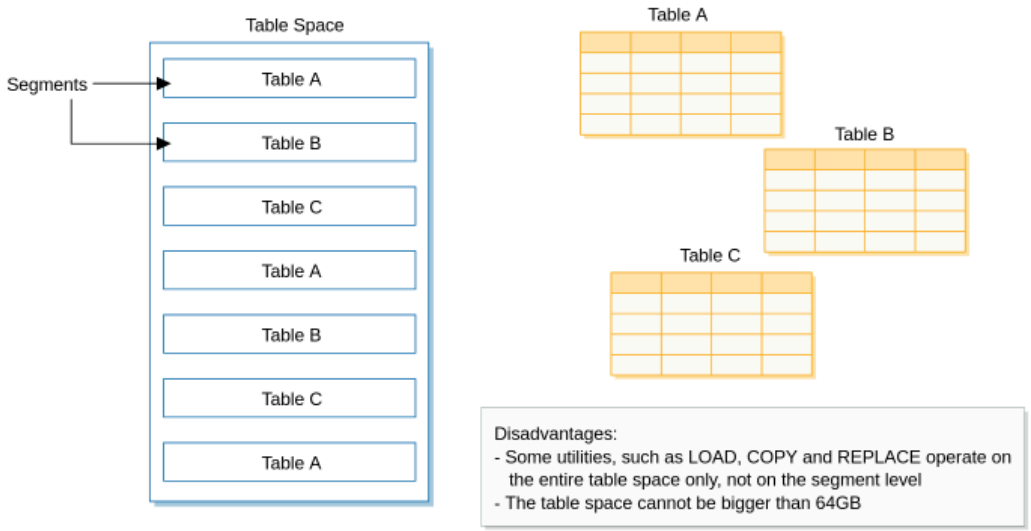


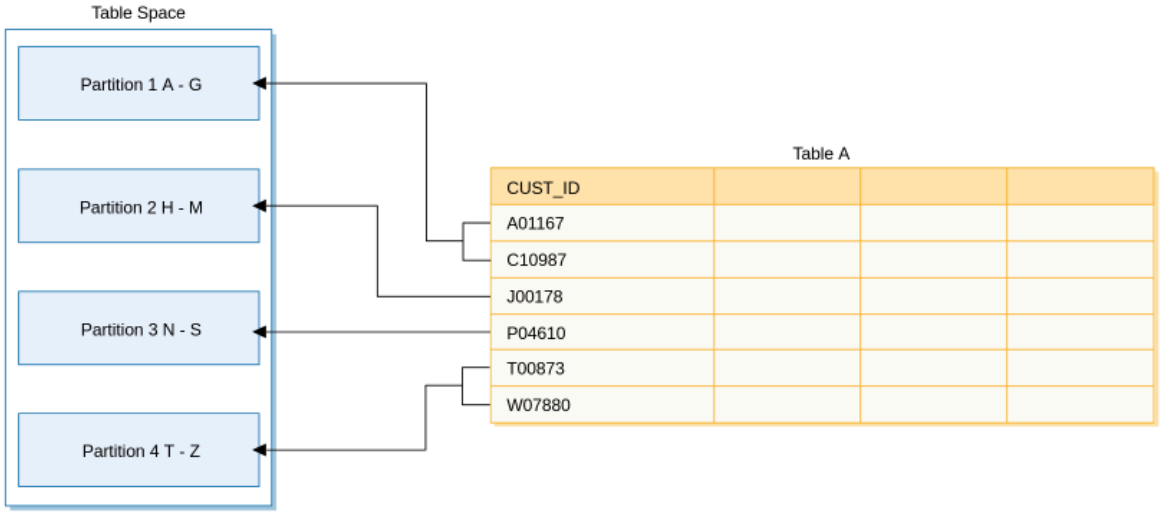
Table spaces are units of storage which Db2 uses to house one of more tables of data. The structure of a table space is different between a z/OS version and one for Linux, UNIX and Windows. In z/OS, a table space exists on one or more VSAM data sets, while in a Linux, UNIX and Windows environment a table space is a collection of containers.

There are different types of table spaces that can be created depending on the type of data that needs to be stored in them, each having their own advantages and disadvantages. These are discussed on the following pages.

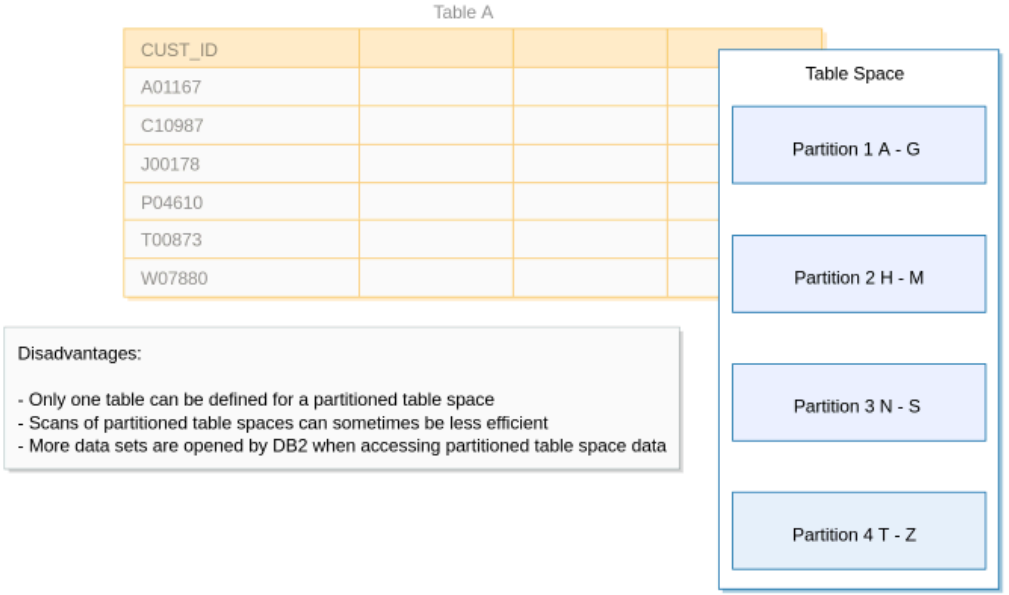


A segmented table space is one that is divided into equal-sized groups of 4K, 8K, 16K, 32K, or 64K pages called segments. Each segment only holds data associated with one table, which provides the following benefits:

- when a scan of all rows is requested for a table, only those segments relating to that table are scanned
- when Db2 locks a table it does not affect other segments in the same table space, that hold other table data
- when a table is removed, its segments become available for use immediately

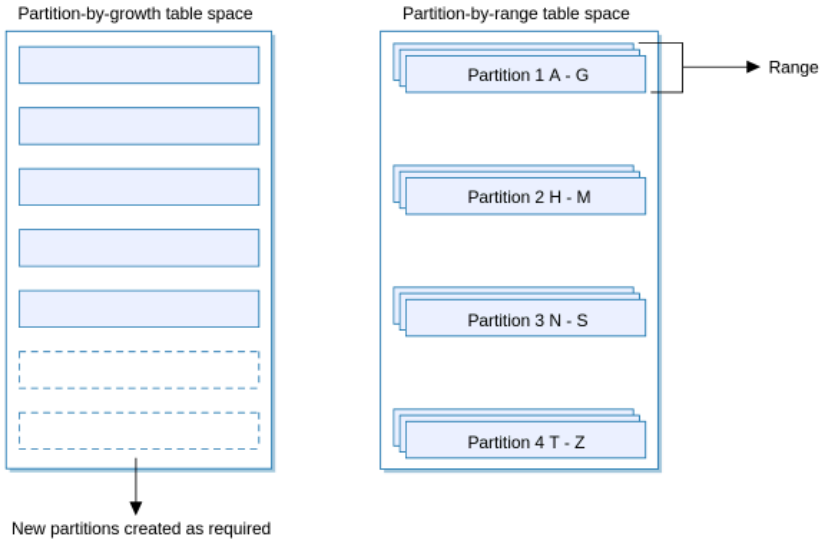


A partitioned table space is one that is divided into a number of partitions. The difference between this and the segmented table space is that a partitioned table space only stores information for a single table. The configuration of the partitions is based on boundary values of specific table columns.



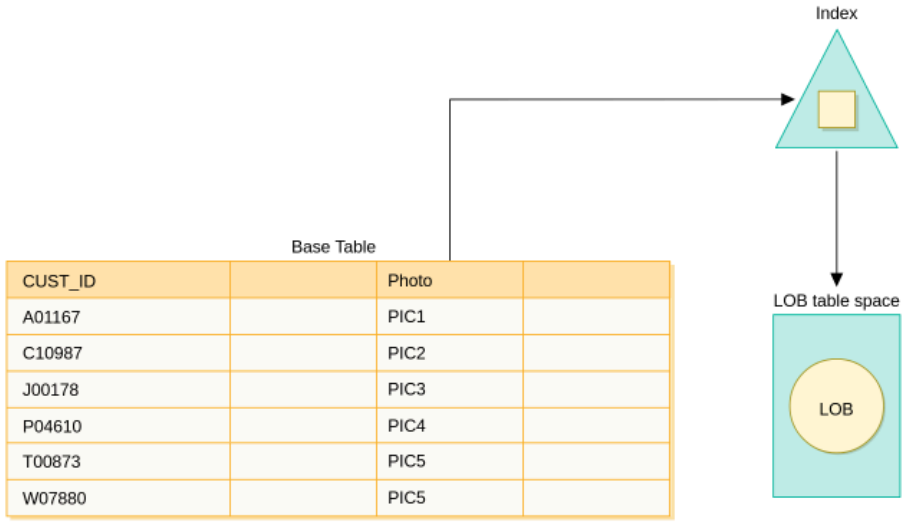
Partitioned table spaces provide a number of advantages over segmented table spaces:

- as data is added or deleted, partition boundaries can be easily redefined if required and partitions can be added with little impact to availability
- a large partitioned table space can be spread over several volumes, which increases access efficiency
- a partitioned table space can be greater than 64GB (if it is enabled for extended addressability)
- utilities and SQL statements can work on different partitions at the same time



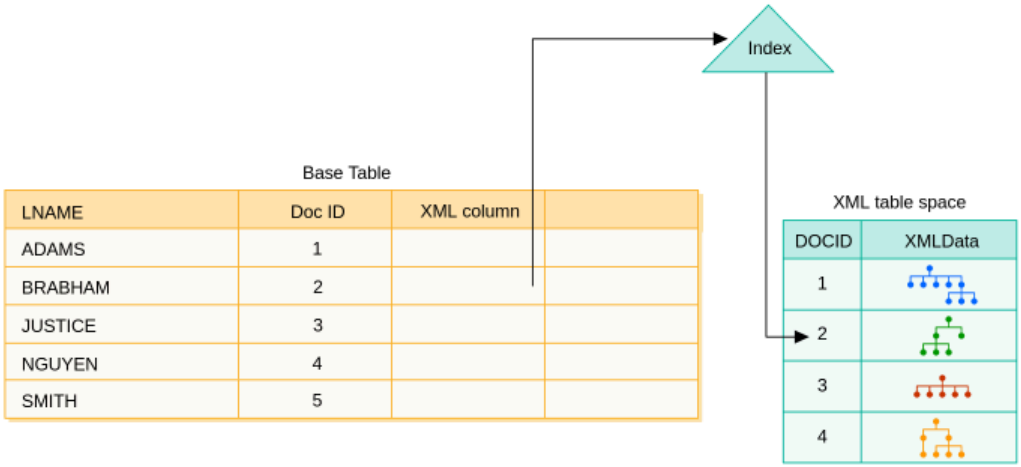
In more recent Db2 releases the universal table space has emerged, which combines the advantages of partitioned and segmented table spaces. There are two types of universal table spaces:

- Partition-by-growth, which are segmented tables that provide for the automatic creation of partitions as the table grows
- Partition-by-range, which uses segmented table space organization but is based on partitioning ranges

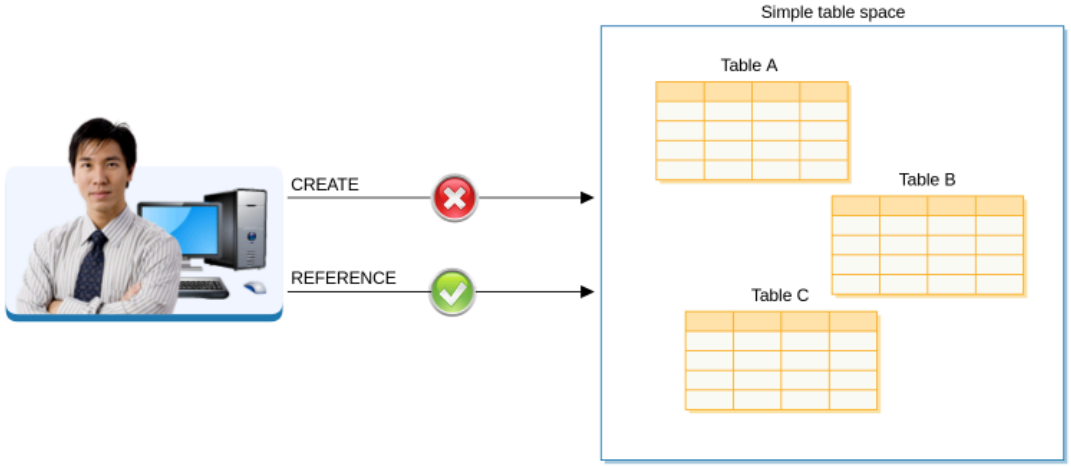


Large object (LOB) table spaces are used to store data such as graphics, video or large text strings. These types of table spaces will have a link to a base table that contains the LOB column values.

In the example here, the base table contains a LOB column referencing ID photographs. These photographs are stored in a LOB table, which are accessed through an index.

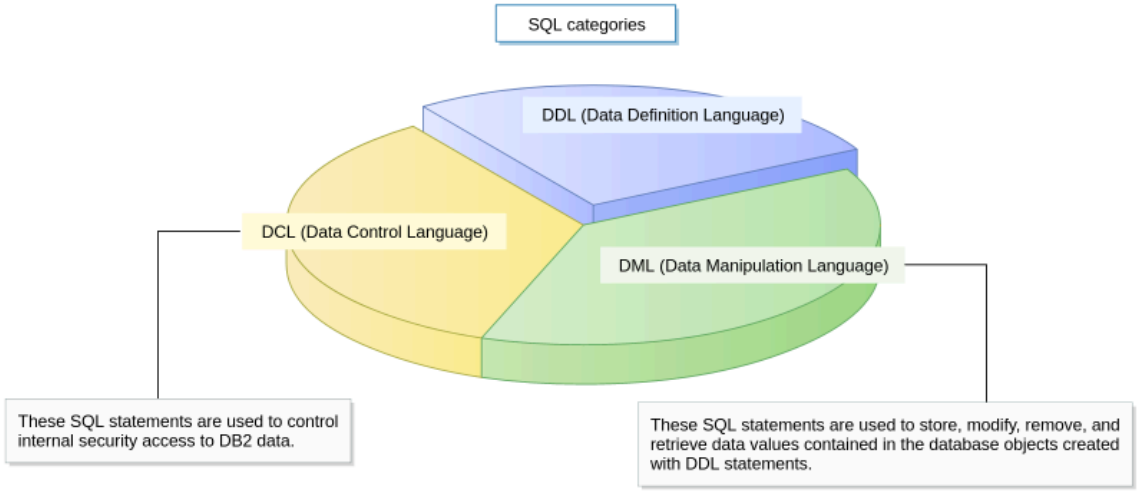


XML Table Spaces are used to store an XML table. Like the LOB table space, a base table contains a reference, which in this instance is to the XML table space where that data is stored.



Depending on the age of your Db2 environment, you may encounter simple table spaces. These types of table spaces are neither segmented or partitioned and were used in earlier versions of Db2, to store data.

You cannot create simple table spaces with today's Db2, but if you have retained them through migration you can still alter, update and retrieve data from them. IBM recommends that any simple table spaces be converted to one of the table spaces discussed previously.



All objects in a Db2 database can be maintained using a subset of Structured Query Language (SQL). This subset of SQL is commonly called DDL which stands for Data Definition Language.

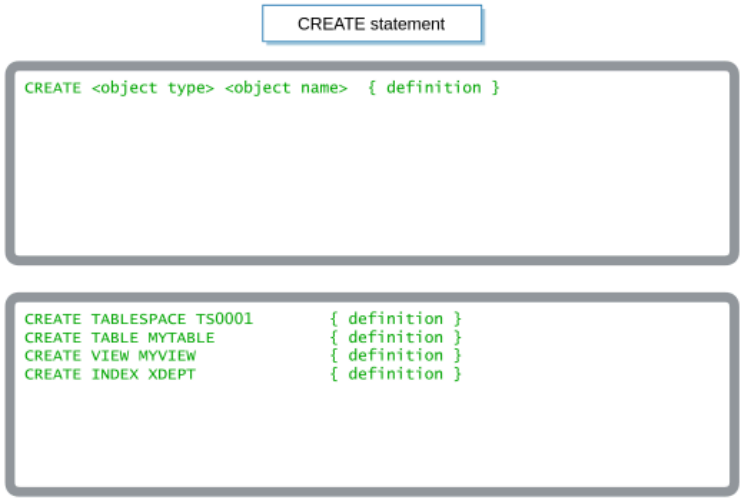
Usually it is the job of the DBA to create and delete tables, table spaces, views, and other objects. When a system is designed for Db2 the table names, column data types, and expected volumes of data are given to the DBA who then generates the necessary DDL to create the objects.

The three main DDL statements are CREATE, ALTER, and DROP.

Tables, table spaces, views, indexes and many other database objects can all be created and dropped (removed). After being created some can also be altered in limited ways.

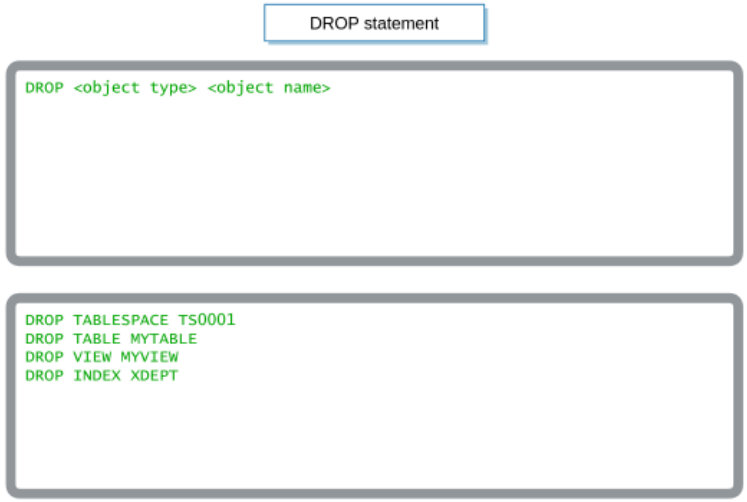
Click each of the DDL statements in order for a graphical representation of their purpose.

Next you will look at each statement in more detail.



The CREATE statement has a different syntax depending on the object that is to be created. An example of this syntax is shown above. Some objects, such as plans, are not created by using the CREATE statement, but are created as part of a database process.

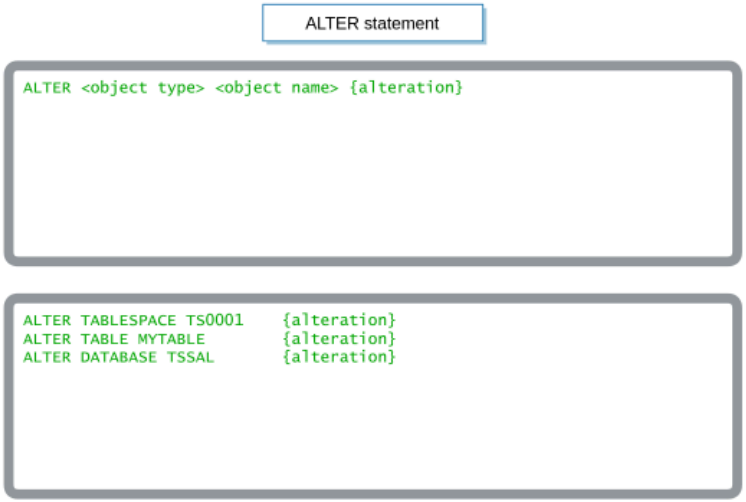
In most cases the CREATE statement builds a definition for the object but does not store any data. For example, the CREATE TABLE statement creates an empty table but the CREATE INDEX statement on a populated table builds the index.



The DROP statement syntax is very similar to the CREATE statement. In most cases you just need to replace the CREATE statement with DROP and leave the object definition out.

The DROP statement not only deletes the definition of an object, it also deletes any data stored by the object. It may also drop any dependent objects. For example, dropping a table drops any dependent indexes.

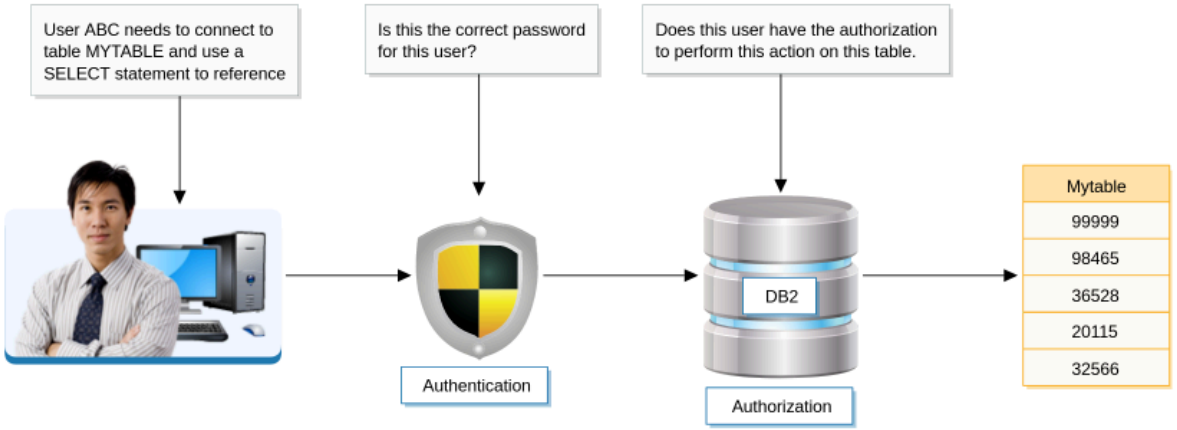
In some situations Db2 will not allow you to drop an object if other objects depend on it.



The ALTER statement is used to change the definition of a component associated with the specified Db2 object. It isn't used to change the actual data values stored in the object.

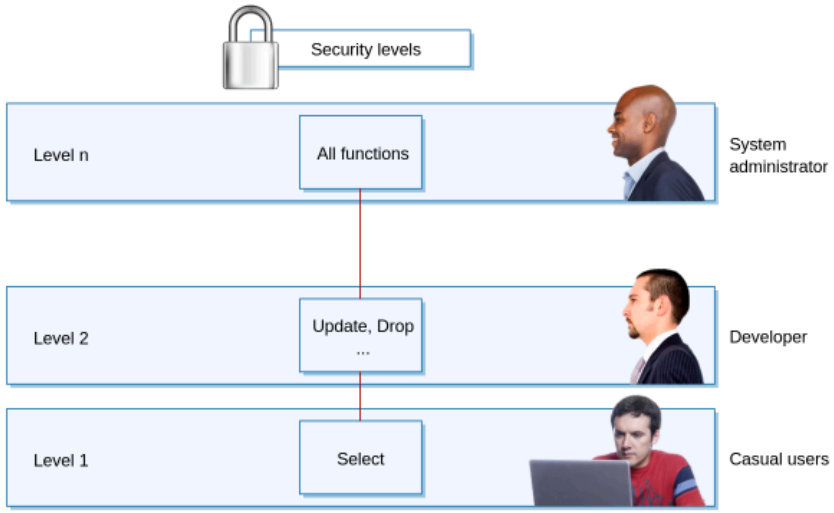
For example, the ALTER TABLE statement can be used to add, drop or rename columns in the table, modify the data type assigned to a column, or change other table attributes, but will not modify any data stored in the table.

The syntax used for the ALTER statement is similar to the CREATE statement.



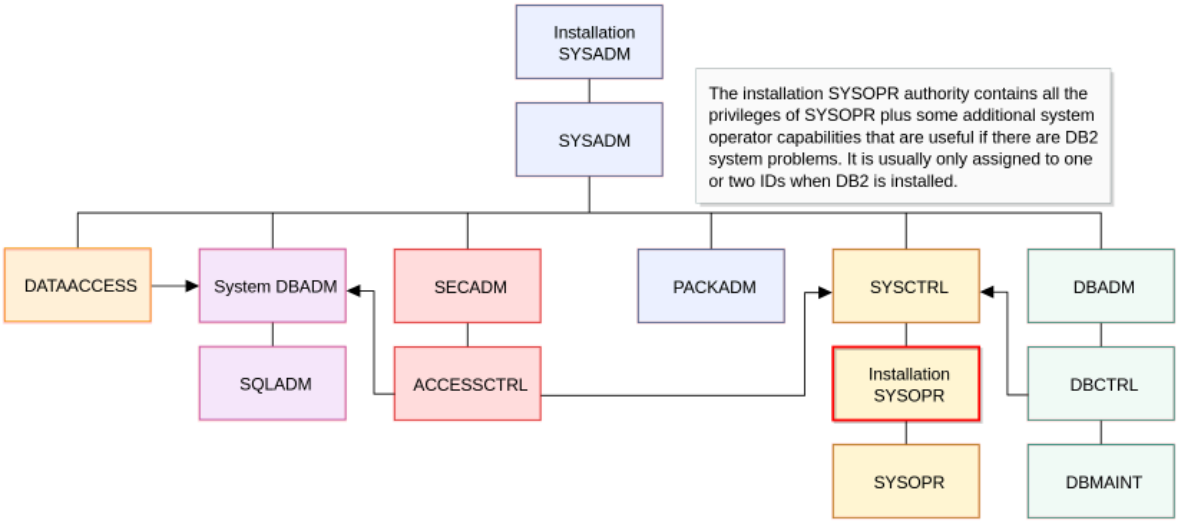
Before creating your Db2 environment you will need to carefully consider your security requirements. In today's world where Db2 can be accessed from a number of devices, you will need to properly authenticate users. This could be through the operating system security software (for example, RACF), via a Lightweight Directory Access Protocol (LDAP) server, Kerberos, or through a customized authentication plug-in.

Once a user has been authenticated, Db2 performs its own authorization check to determine the actions that the user is allowed to perform. These authorization rules will have been defined by your system administrator and security specialist and are discussed in detail in this section of the module.



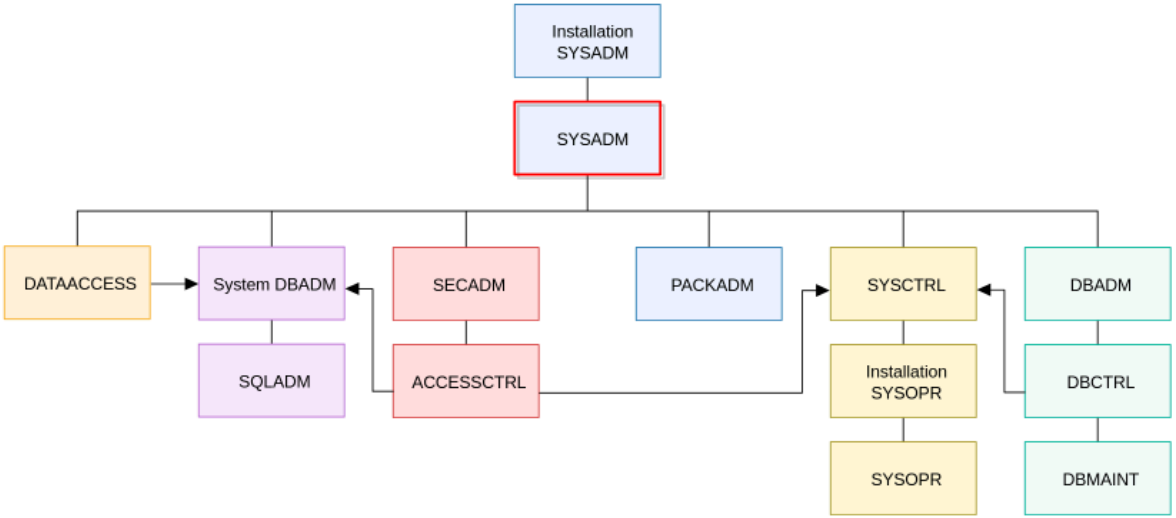
Access to user data is managed by Db2 through the use of authorities and privileges. For example, if a user needs to update data in a table, they will need to have that privilege for that table. These privileges are then grouped together to form administrative authorities.

In the example shown here, the system administrator has access to all Db2 privileges, while the developer has access to a subset of privileges. Casual users in this scenario have only a set of limited privileges.



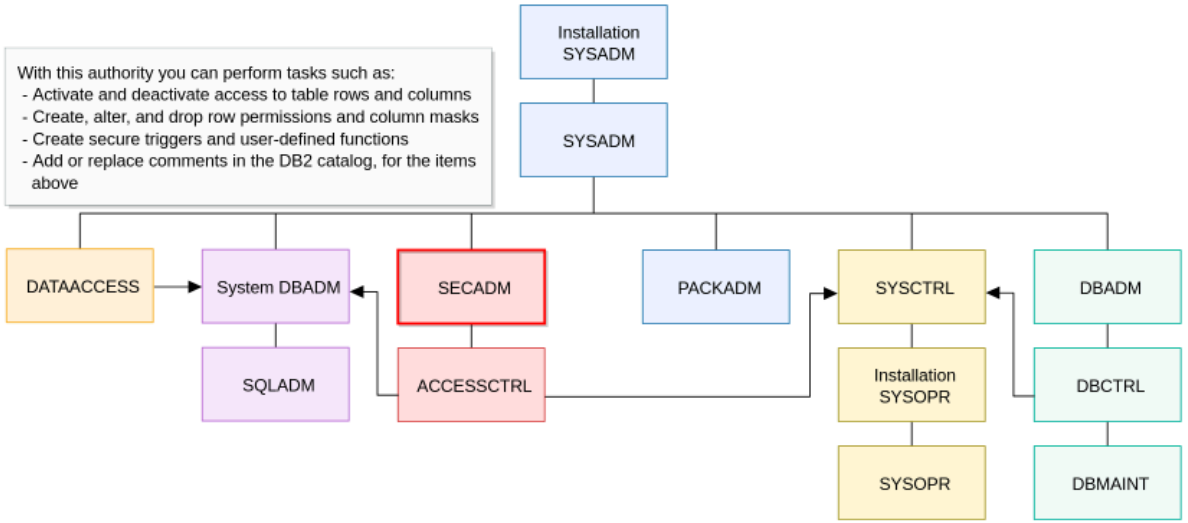
Authorities are configured in a hierarchy structure, so that levels at the top will encompass all the authorities immediately below them and will also contain additional privileges. For example, if you were assigned the SYSADM authority it would automatically mean that you have all the privileges from the SYSCTRL, DBADM, Installation SYSOPR, SYSOPR, PACKADM, System DBADM, DBCTRL, DBMAINT, SECADM, SQLADM, ACCESSCTRL, and DATAACCESS.

Mouse-over the authorities for a brief description of their purpose.



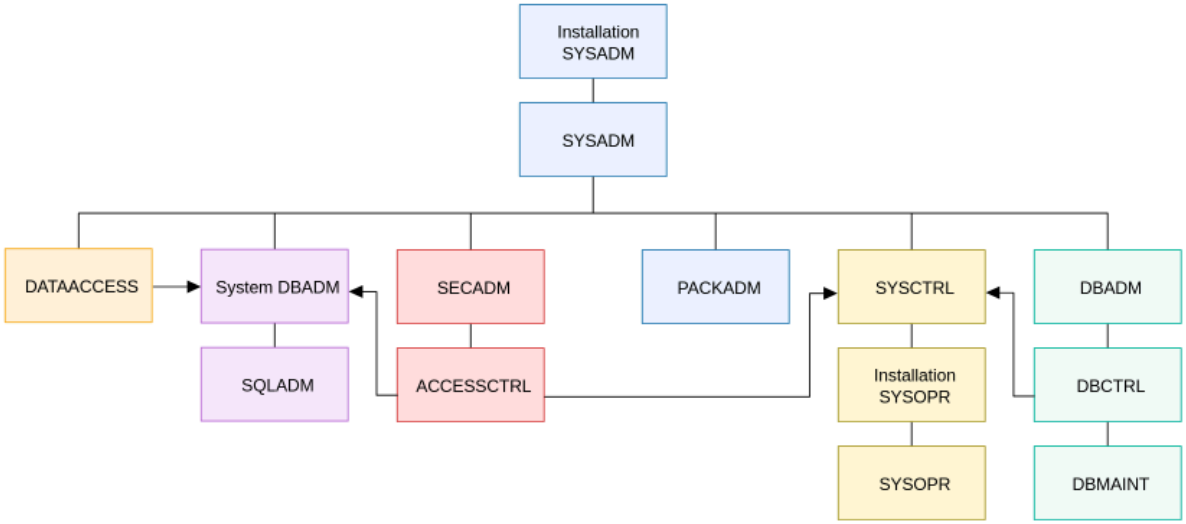
We will now discuss in more detail some of these Db2 authorities and their capabilities. Prior to Db2 V10 the SYSADM authority needed to be assigned to a number of people in the organization to allow them to perform their tasks. This authority was quite powerful as it allows almost complete control over Db2, from system configuration, and security management, to data access control.

In Db2 V10 the privileges in SYSADM were split into a number of new authorities: SECADM, ACCESSCTRL, DATAACCESS, System DBADM and SQLADM. This means that more granular authority privileges can now be made available to Db2 users. For example, an SQL performance analyst can now be given SQLADM authority to issue SQL EXPLAIN statements (that provides them with access path information), instead of SYSADM authority which had more power than they required.



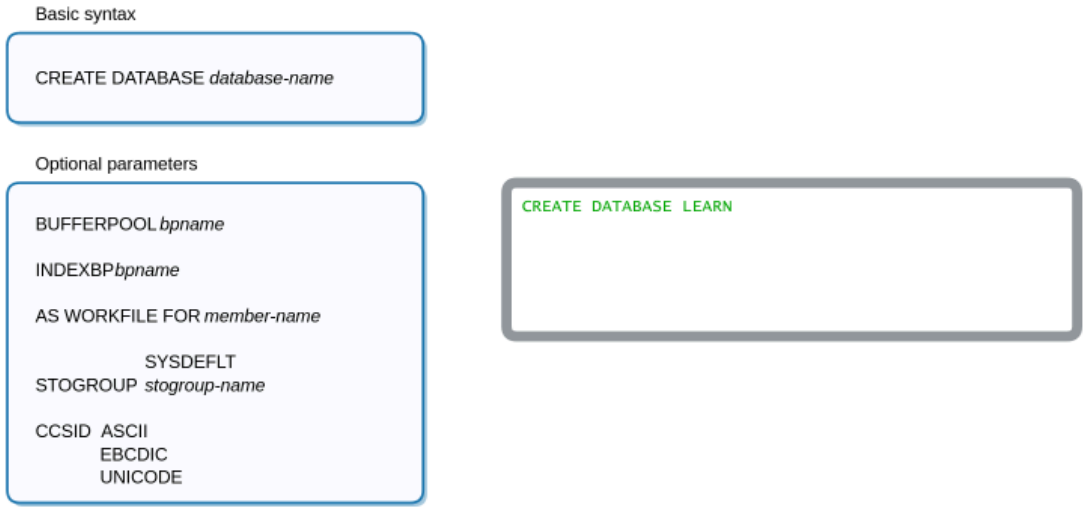
SECADM authority enables you to manage security-related objects in Db2 and control access to all database resources. It does not have any inherent privilege to access data stored in the objects, such as tables. ACCESSCTRL allows you to grant explicit privileges to authorization IDs or roles by issuing SQL GRANT statements and revoke privileges by issuing SQL REVOKE statements with the BY clause. DATAACCESS authority allows you to access data in tables, views, and materialized query tables in a Db2 subsystem. It also allows you to execute plans, packages, functions, and procedures.

Mouse-over the SECADM authority for an example of the tasks that are allowed.



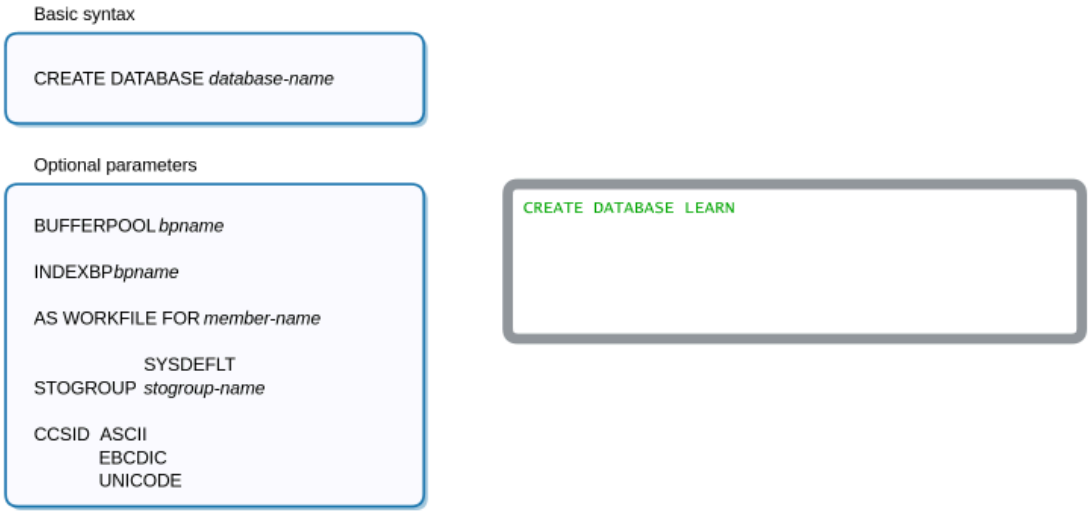
The System DBADM authority allows the user to manage all databases within a Db2 subsystem. By default, this authority also inherits the privileges from the DATAACCESS and ACCESSCTRL authorities, although this can be removed by the administrator if need be.

SQLADM authority allows you to issue the SQL EXPLAIN statements, execute the PROFILE commands, run the RUNSTATS and MODIFY STATISTICS utilities on all user databases, and execute system-defined routines, such as stored procedures or functions, and any packages that are executed within the routines.



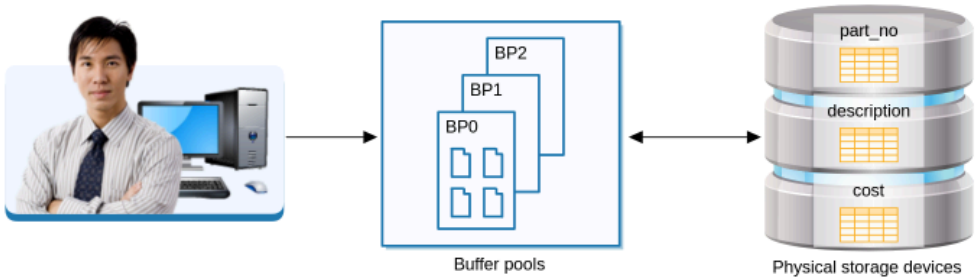
The first step in creating a database is to issue a CREATE DATABASE statement. This statement can be invoked through an application program, or interactively using an online facility such a SPUFI or QMF. Note that some of the parameters here are unique to the mainframe environment.

An example of a basic statement is shown above. The name used for the database must be unique to the Db2 subsystem, and in this scenario is called LEARN.



The CREATE DATABASE statement shown here is valid syntax as the optional parameters are defaulted if not specified.

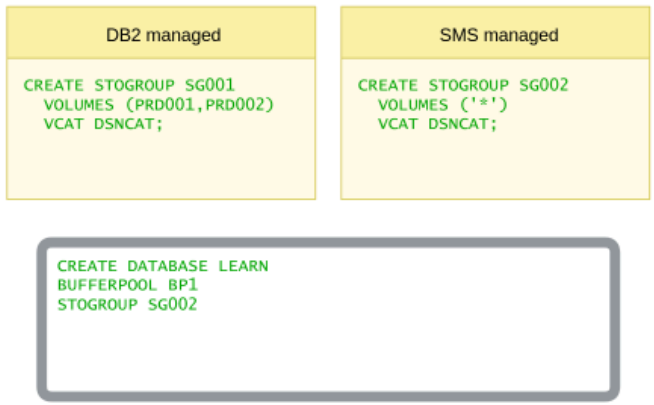
You will now see how these optional parameters are used to more specifically configure your database.




```
CREATE DATABASE LEARN
BUFFERPOOL BP1
```

The bufferpool parameter specifies the default bufferpool for table spaces created within the specified database. Similarly the INDEXBP specifies the default bufferpool for indexes created in the specified database.

Bufferpools define memory 'buffers' to handle and optimize data I/O. They can be sized to suit the data being accessed and tables can be distributed among them to optimize performance.



The STOGROUP parameter specifies the default storage group for table spaces and indexes within the database. Storage groups provide a means of identifying DASD requirements to Db2.

They can be managed by Db2 where the volumes are specified, or managed externally by SMS.

More information on storage groups can be found in other courses and modules.

Basic syntax

```
CREATE DATABASE database-name
```

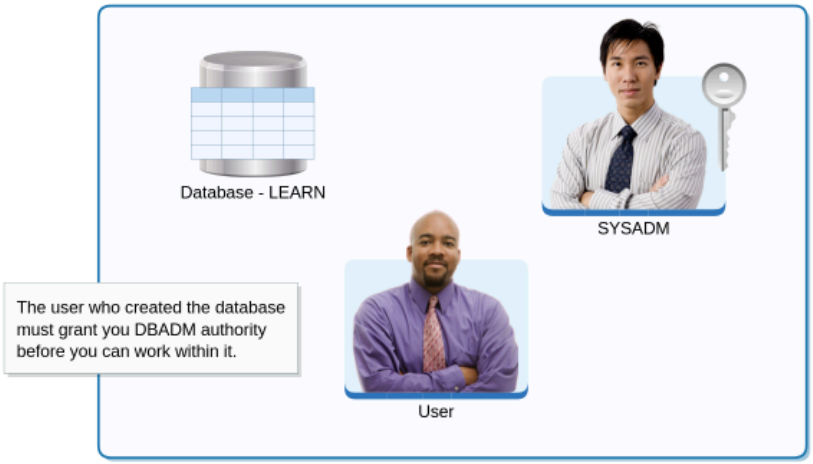
Optional parameters

```
BUFFERPOOL bpname

INDEXBP bpname

AS WORKFILE FOR member-name

              SYSDEFLT
STOGROUP  stogroup-name

CCSID  ASCII
       EBCDIC
       UNICODE
```

```
CREATE DATABASE SAMDEMO
CCSID EBCDIC;
```

The CCSID parameter is used to specify the default encoding scheme for all data stored in the database.

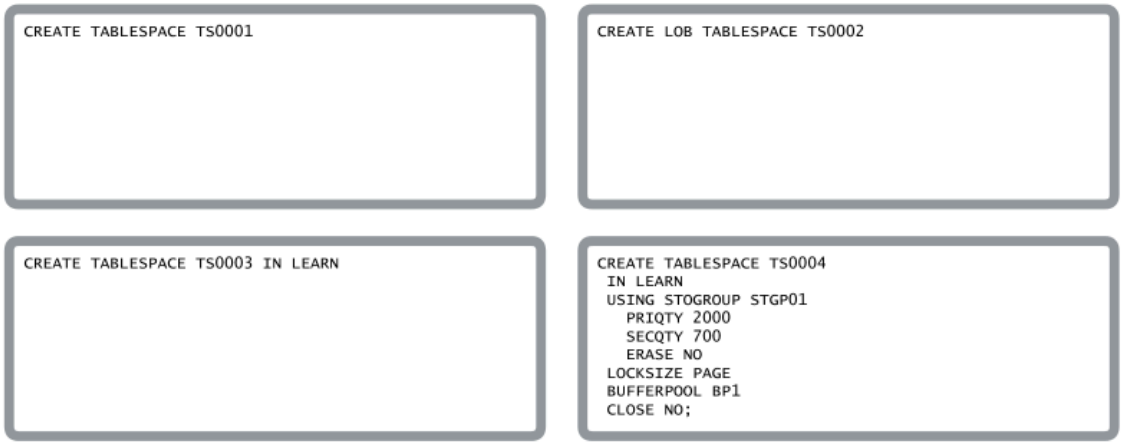
In the example shown here, database SAMDEMO is being created and default storage group and buffer pool names will be used. The EBCDIC encoding scheme will be used for data stored in the database.



In some of the previous examples, we provided the code to create a database called LEARN. If in reality the database was created by you, you would automatically have the authority required to work with it. If the database was created by someone else and you needed to work with it, the original creator can grant you DBADM authority to work with that database.

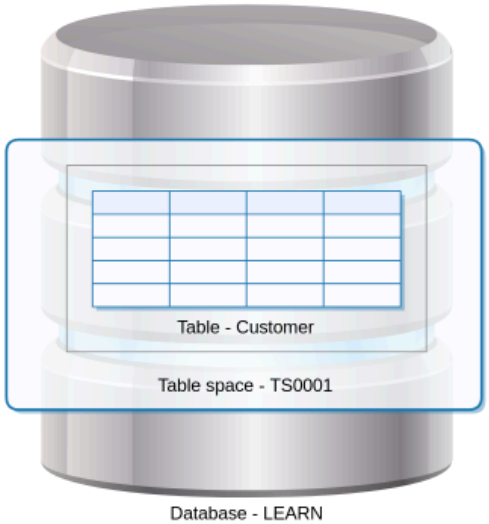
Before creating any data in your database, you will need to define a container for that data to reside in. This container is called a table space.

Click **Play** to see an example of this concept.



The CREATE TABLESPACE statement is used to define a segmented, partitioned, or universal table space for the database. In its simplest form, the statement identifies the name required for the table space.

The second example shows the creation of a large object (LOB) table space, which is used to hold LOB values. The third example specifies the database in which the table space will be created, while the last example shows a small number of optional parameters that can be specified when creating a table space. IBM's Db2 Knowledge Center website should be accessed for more information on these parameters.



Having created a table space called TS0001 you can now create the table CUSTOMER. When you create a table you must define at least one column.

You will now create a table with three simple columns. In the next module you will examine the data types that can be stored in Db2 and see how to define them.

Customer

```
CREATE TABLE CUSTOMER
 (
 NAME_ID CHAR(8) NOT NULL,
 OFFICIAL_NAME CHAR(35) NOT NULL,
 LAST_UPDATE TIMESTAMP NOT NULL WITH DEFAULT
 )
```

The CREATE TABLE command shown here is defining three columns, shown at the top of the page, for a new table named CUSTOMER.

In this example, the NAME-ID column will be 8 characters in length, OFFICIAL_NAME will be 35 characters and in the LAST_UPDATE column, the default number of digits for the TIMESTAMP parameter is assumed, which is 6.

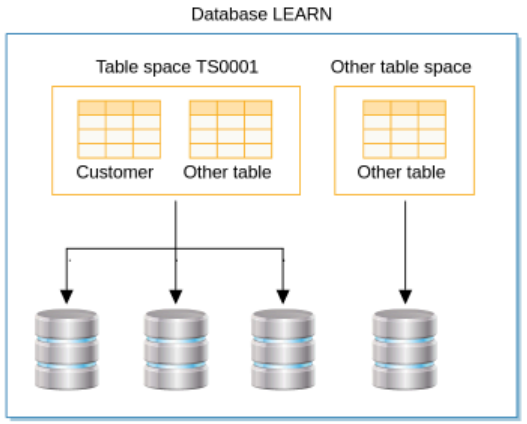
Name_ID	Official_Name	Last_Update

Customer

```
CREATE TABLE CUSTOMER
 (
 NAME_ID CHAR(8) NOT NULL,
 OFFICIAL_NAME CHAR(35) NOT NULL,
 LAST_UPDATE TIMESTAMP NOT NULL WITH DEFAULT
 ) IN LEARN.TS0001
```

The CREATE TABLE code on the previous page did not specify where to place the table, so Db2 would have implicitly created a database and table space to place it in.

In the example shown here, the IN statement on the last line shows how you can specify that the table is to be placed in the LEARN database and TS0001 table space.



You have just seen how to create a simple table in your database. In the next module you will look at the many options and parameters that can be specified when creating a table and the scenarios where they are required.

Summary

Defining Database Objects

Db2 consists of a number of elements that are used to store and organize your data. In this module you saw how a database and its related table spaces and tables are created using SQL statements, and the type of security used to manage Db2.

You should now be able to describe:

- The Types of Tables That Can be Used Within Db2
- Commonly Used SQL Statements Used to Manage Db2 Objects
- Security That Can be Configured for Db2 Objects
- The SQL Statements Used to Create a Database, Table Space and Table