The basic categories of commands used in SQL to perform various functions are

* Data Definition Language (DDL)
* Data Manipulation Language (DML)
* Data Query Language (DQL)
* Data Control Language (DCL)
* Transactional control commands

**Data Definition Language** (DDL) is the part of SQL that enables a database user to **create and restructure database objects**, such as the creation or the deletion of a table. Some of the most fundamental DDL commands include:

* Create Table
* Alter Table
* Drop Table
* Create Index
* Alter Index
* Drop Index
* Create View
* Alter View
* Drop View

**Data Manipulation Language (DML)** is the part of SQL used to **manipulate data within objects** of a relational database. The three basic DML commands are:

* INSERT
* UPDATE
* DELETE

**Data Query Language (DQL)** - Though comprised of only one command, Data Query Language (DQL) is the most concentrated focus of SQL for modern relational database users. The base command is **SELECT**. This command, accompanied by many options and clauses, is used to compose queries against a relational database.

**Data Control Language (DCL)** - Data control commands in SQL enable you to **control access to data** within the database. These Data Control Language (DCL) commands are normally used to create objects related to user access and also control the distribution of privileges among users. Some data control commands are as follows:

* ALTER PASSWORD
* GRANT
* REVOKE
* CREATE SYNONYM

**Transactional Control Commands**:- In addition to the previously introduced categories of commands, there are commands that **enable the user to manage database transactions**:

* **COMMIT**—Saves database transactions
* **ROLLBACK**—Undoes database transactions
* **SAVEPOINT**—Creates points within groups of transactions in which to

**Database Objects**:- A database object is any defined object in a database that is used to store or reference data. Some examples of database objects include tables, views, clusters, sequences, indexes, and synonyms.

**What Is a Schema?**

🡺 Shema refers to the structure of a database object. Broadly**, schema can be a collection of database objects normally associated with one particular database username**. This username is called the schema owner, or the owner of the related group of objects. You may have one or multiple such schemas in a database.

Based on a user’s privileges within the database, the user has control over objects that are created, manipulated, and deleted. A schema can consist of a single table and has no limits to the number of objects that it may contain, unless restricted by a specific database implementation.

Say you have been issued a database username and password by the database administrator. Your **username is C##USER1**. Suppose you log on to the database and then **create a table called EMPLOYEE**. According to the database, **your table’s actual name is C##USER1.EMPLOYEE**. The schema name for that table is C##USER1, which is also the owner of that table. You have just created the first table of a schema.

The good thing about schemas is that when you access a table that you own (in your own schema), you do not have to refer to the schema name. For instance, you could refer to your table as either one of the following:

* EMPLOYEE
* C##USER1.EMPLOYEE

The first option is preferred because it requires fewer keystrokes. If **another user** were to query one of your tables, the user would have to specify the schema as **C##USER1.EMPLOYEE**



In this example, both users have a table called TEST. Tables can have the same names in a database as long as they belong to different users. If you look at it this way, table names are always unique in a database because the schema owner or username is actually part of the table name. For instance, **USER1.TEST is a different table than USER2.TEST**. If you do not specify a schema with the table name when accessing tables in a database, the database **server looks for a table that you own by default**. That is, if USER1 tries to access TEST, the database server looks for a USER1-owned table named TEST before it looks for other objects owned by USER1, such as synonyms to tables in another schema.

**Tables** :- The table is the **primary storage object for data** in a relational database. In its simplest form, a table consists of row(s) and column(s), both of which hold the data. A table **takes up physical space** in a database and can be permanent or temporary.

# Note:-

* When you create a table, you must specify the table name, name of each column, data type of each column, and size of each column
* The table and column names can be up to 30 characters long
* Table or column name must begin with a letter
* The names are not case sensitive
* Spaces and hyphens are not allowed in a table or a column name; but $, \_ and # are allowed

**Columns**:- A field, also called a column in a relational database, is part of a table that is **assigned a specific data type**. The data type determines what kind of data the column is allowed to hold. Every database table must consist of at least one column. Columns are those elements within a table that hold specific types of data, such as a person’s name or address. Generally, a column name must be one continuous string and can be limited to the number of characters used according to each implementation of SQL. It is typical to **use underscores** with names to provide separation between characters. For example, a column for the customer’s name can be named CUSTOMER\_NAME instead of CUSTOMERNAME. This is normally done to increase the readability of database objects. Columns also can be specified as **NULL** or **NOT NULL**, meaning that if a column is **NOT NULL, something must be entered**. If a column is specified as **NULL, nothing has to be entered.**

**Rows**:- A row is a **record of data** in a database table. For example, a row of data in a customer table might consist of a particular customer’s identification number, name, address, phone number, and fax number. A row is composed of fields that contain data from one record in a table. A table can contain as little as one row of data and up to as many as millions of rows of data or records.

**Data Types**:- Data types are characteristics of the data itself, whose attributes are placed on fields within a table. For example, you can specify that a field must contain numeric values, disallowing the entering of alphanumeric strings.

After all, you would not want to enter alphabetic characters in a field for a dollar amount. Defining each field in the database with a data type eliminates much of the incorrect data found in a database due to data entry errors. Field definition (data type definition) is a form of data validation that controls the type of data that may be entered into each given field.

Depending on the implementation of relational database management system (RDBMS), **certain data types can be converted automatically to other data types depending upon their format**. This type of conversion in known as an implicit conversion, which means that the database handles the conversion for you. An example of this is taking a numeric value of 1000.92 from a numeric field and inputting it into a string field. Other data types cannot be converted implicitly by the host RDBMS and therefore must undergo an explicit conversion. This usually involves the use of an SQL function, such as CAST or CONVERT. For example SELECT CAST(‘12/27/1974’ AS DATETIME) AS MYDATE

The very basic data types, as with most other languages, are

* String types
* Numeric types
* Date and time types

String Types

* **char(n)** – The **fixed-length character** with user specified length n. The full form character can be used instead.
* **varchar(n)/varchar2(n)** - The **variable-length character** with user specified maximum length n. The full form character varying is equivalent.

Numeric Types

Numeric values are stored in fields that are defined as some type of number, typically referred to as NUMBER, INTEGER, REAL, DECIMAL, and so on.

* BIT(n)
* BIT VARYING(n)
* DECIMAL(p,s)
* INTEGER
* SMALLINT
* BIGINT
* FLOAT(p, s)
* DOUBLE PRECISION(p, s)
* REAL(s)

Date and Time Types

Date and time data types are quite obviously used to keep track of information concerning dates and time. Standard SQL supports what are called DATETIME data types, which include the following specific data types:

* DATE
* TIME
* DATETIME
* TIMESTAMP

**Other Data Types**

NULL Types

NULL value is a missing value or a column in a row of data that has not been assigned a value. NULL values are used in nearly all parts of SQL, including the creation of tables, search conditions for queries, and even in literal strings.

LOB Types

Some variable-length data types need to **hold longer lengths of data** than what is traditionally reserved for a VARCHAR field. The **BLOB** and **TEXT** data types are two examples of such data types in modern database implementations. These data types are specifically made to hold large sets of data. The BLOB is a binary large object, so its data is treated as a large binary string (a byte string). A **BLOB** is especially useful in an implementation that needs to store binary media files in the database, such as images or MP3s. The **TEXT** data type is a large character string data type that can be treated as a large VARCHAR field. It is often used when an implementation needs to store large sets of character data in the database. An example of this would be storing HTML input from the entries of a blog site. Storing this type of data in the database enables the site to be dynamically updated.

**Domains**

A domain is a set of valid data types that can be used. A domain is associated with a data type, so only certain data is accepted.

**DDL - Table Creation**

The CREATE TABLE statement in SQL is used to create a table. Although the very act of creating a table is quite simple, much time and effort should be put into planning table structures before the actual execution of the CREATE TABLE statement. Carefully planning your table structure before implementation saves you from having to reconfigure things after they are in production. The basic syntax is:

CREATE TABLE EMPLOYEE

(EMP\_ID NUMBER(9) NOT NULL,

EMP\_NAME VARCHAR2 (40) NOT NULL,

EMP\_ST\_ADDR VARCHAR2 (20) NOT NULL,

EMP\_CITY VARCHAR2 (15) NOT NULL,

EMP\_ST VARCHAR(2) NOT NULL,

EMP\_ZIP INTEGER,

EMP\_PHONE INTEGER NULL,

EMP\_PAGER INTEGER NULL);

CREATE TABLE table\_name

( field1 data\_type [ not null ],

field2 data\_type [ not null ],

field3 data\_type [ not null ],

field4 data\_type [ not null ],

field5 data\_type [ not null ] );

**DDL - Table Description**

DESCRIBE statement is used for viewing table structure. The syntax is:

* **DESCRIBE table\_name;** or
* **DESC table\_name;**

For example: DESCRIBE EMPLOYEE;

**DDL - Creating Table from another Table**

The SQL statement populates the target table with data from the source table

**CREATE TABLE tablename (column1, column2) AS SELECT column1 ,column2 FROM tablename;**

CREATE TABLE Person(Roll, Name, Age) AS SELECT Roll, Name, Age FROM stud;

**DDL - Dropping a Table**

Deletes all the data including table itself

**DROP TABLE table\_name;**

**DDL – Altering a Table**

You can modify a table after the table has been created by using the **ALTER TABLE** command. You can add column(s), drop column(s), change column definitions, add and drop constraints.

Example

* ALTER TABLE EMPLOYEE MODIFY EMP\_ID VARCHAR(10);
* ALTER TABLE EMPLOYEE ADD EMP\_SEX VARCHAR(1);
* ALTER TABLE CUSTOMER DROP COLUMN CUSTOMER\_NAME;
* ALTER TABLE CUSTOMER ADD (CUSTOMER\_NAME VARCHAR2(45), CITY VARCHAR2(40));
* ALTER TABLE CUSTOMER MODIFY (CUSTOMER\_NAME VARCHAR2(100) NOT NULL, CITY

VARCHAR2(75));

* ALTER TABLE CUSTOMER RENAME COLUMN CUSTOMER\_NAME TO CNAME;
* ALTER TABLE CUSTOMERS RENAME TO CONTACTS;