SQL - Overview

SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. SQL is an **ANSI** (American National Standards Institute) standard language, but there are many different versions of the SQL language.

What is SQL?

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.

SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

Also, they are using different dialects, such as –

- MS SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.

Why SQL?

SQL is widely popular because it offers the following advantages –

- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

A Brief History of SQL

- **1970** – Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
- **1974** – Structured Query Language appeared.

- **1978** – IBM worked to develop Codd's ideas and released a product named System/R.
- **1986** – IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software which later came to be known as Oracle.

SQL Process

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

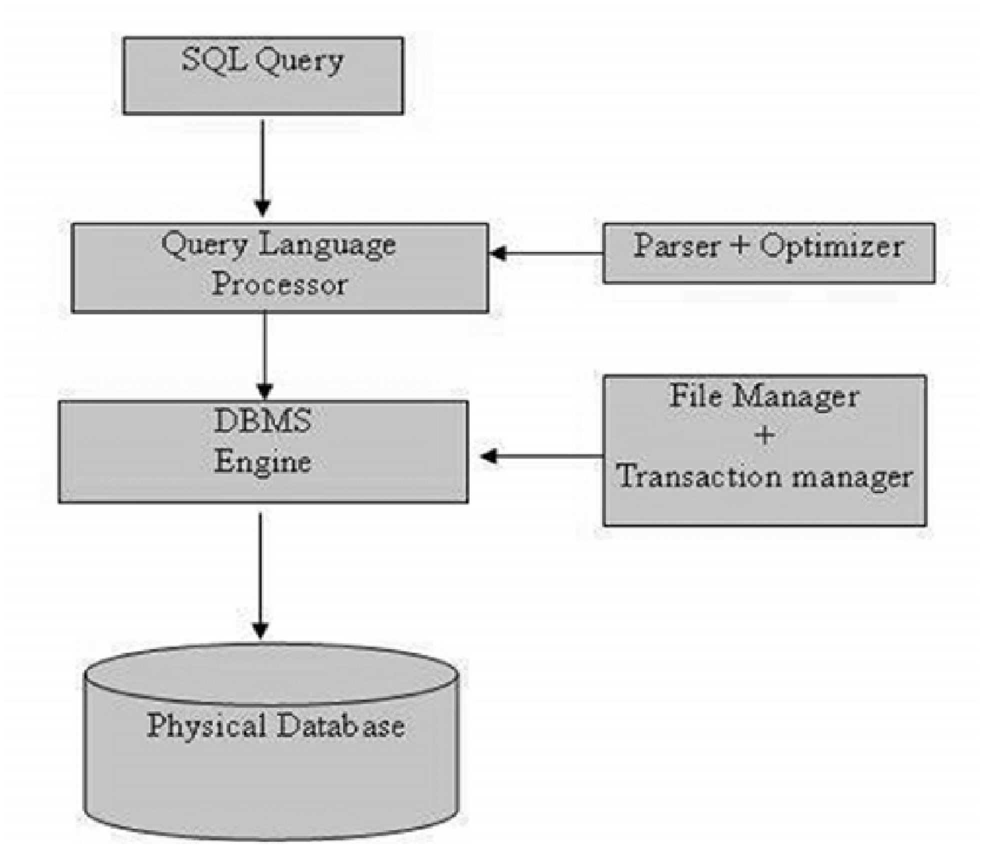
There are various components included in this process.

These components are –

- Query Dispatcher
- Optimization Engines
- Classic Query Engine
- SQL Query Engine, etc.

A classic query engine handles all the non-SQL queries, but a SQL query engine won't handle logical files.

Following is a simple diagram showing the SQL Architecture –



SQL Commands

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into the following groups based on their nature −

DDL - Data Definition Language

Sr.No.	Command & Description
1	**CREATE** Creates a new table, a view of a table, or other object in the database.
2	**ALTER** Modifies an existing database object, such as a table.
3	**DROP** Deletes an entire table, a view of a table or other objects in the database.

DML - Data Manipulation Language

Sr.No.	Command & Description
1	**SELECT** Retrieves certain records from one or more tables.
2	**INSERT** Creates a record.
3	**UPDATE** Modifies records.
4	**DELETE** Deletes records.

DCL - Data Control Language

Sr.No.	Command & Description
1	**GRANT** Gives a privilege to user.
2	**REVOKE** Takes back privileges granted from user.

SQL - RDBMS Concepts

What is RDBMS?

RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

What is a table?

The data in an RDBMS is stored in database objects which are called as **tables**. This table is basically a collection of related data entries and it consists of numerous columns and rows.

Remember, a table is the most common and simplest form of data storage in a relational database. The following program is an example of a CUSTOMERS table –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1|Ramesh|32|Ahmedabad|2000.00|
+----+----------+-----+-----------+----------+
```

What is a field?

Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

A field is a column in a table that is designed to maintain specific information about every record in the table.

What is a Record or a Row?

A record is also called as a row of data is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table –

```
+----+----------+-----+-----------+----------+
|1|Ramesh|32|Ahmedabad|2000.00|
+----+----------+-----+-----------+----------+
```

A record is a horizontal entity in a table.

What is a column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would be as shown below –

```
+-----------+

| ADDRESS   |

+-----------+

|Ahmedabad|

+----+------+
```

What is a NULL value?

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is the one that has been left blank during a record creation.

SQL Constraints

Constraints are the rules enforced on data columns on a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints can either be column level or table level. Column level constraints are applied only to one column whereas, table level constraints are applied to the entire table.

Following are some of the most commonly used constraints available in SQL –

- NOT NULL Constraint – Ensures that a column cannot have a NULL value.
- DEFAULT Constraint – Provides a default value for a column when none is specified.
- UNIQUE Constraint – Ensures that all the values in a column are different.
- PRIMARY Key – Uniquely identifies each row/record in a database table.
- FOREIGN Key – Uniquely identifies a row/record in any another database table.
- CHECK Constraint – The CHECK constraint ensures that all values in a column satisfy certain conditions.
- INDEX – Used to create and retrieve data from the database very quickly.

Data Integrity

The following categories of data integrity exist with each RDBMS −

- **Entity Integrity −** There are no duplicate rows in a table.
- **Domain Integrity −** Enforces valid entries for a given column by restricting the type, the format, or the range of values.
- **Referential integrity −** Rows cannot be deleted, which are used by other records.
- **User-Defined Integrity −** Enforces some specific business rules that do not fall into entity, domain or referential integrity.

Database Normalization

Database normalization is the process of efficiently organizing data in a database. There are two reasons of this normalization process −

- Eliminating redundant data, for example, storing the same data in more than one table.
- Ensuring data dependencies make sense.

Both these reasons are worthy goals as they reduce the amount of space a database consumes and ensures that data is logically stored. Normalization consists of a series of guidelines that help guide you in creating a good database structure.

Normalization guidelines are divided into normal forms; think of a form as the format or the way a database structure is laid out. The aim of normal forms is to organize the database structure, so that it complies with the rules of first normal form, then second normal form and finally the third normal form.

It is your choice to take it further and go to the fourth normal form, fifth normal form and so on, but in general, the third normal form is more than enough.

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)

SQL - CREATE Database

The SQL **CREATE DATABASE** statement is used to create a new SQL database.

Syntax

The basic syntax of this CREATE DATABASE statement is as follows −

```
CREATE DATABASE DatabaseName;
```

Always the database name should be unique within the RDBMS.

Example

If you want to create a new database <testDB>, then the CREATE DATABASE statement would be as shown below −

```
SQL> CREATE DATABASE testDB;
```

Make sure you have the admin privilege before creating any database. Once a database is created, you can check it in the list of databases as follows −

```
SQL> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| AMROOD             |
| testDB             |
+--------------------+
```

SQL - CREATE Table

Creating a basic table involves naming the table and defining its columns and each column's data type.

The SQL **CREATE TABLE** statement is used to create a new table.

Syntax

The basic syntax of the CREATE TABLE statement is as follows –

```
CREATE TABLE table_name(
column1 datatype,
column2 datatype,
column3 datatype,
   .....
columnN datatype,
   PRIMARY KEY( one or more columns )
);
```

CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement.

Then in brackets comes the list defining each column in the table and what sort of data type it is. The syntax becomes clearer with the following example.

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement. You can check the complete details at Create Table Using another Table.

Example

The following code block is an example, which creates a CUSTOMERS table with an ID as a primary key and NOT NULL are the constraints showing that these fields cannot be NULL while creating records in this table –

```
SQL> CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
AGE  INT              NOT NULL,
ADDRESS  CHAR(25),
   SALARY   DECIMAL (18,2),
   PRIMARY KEY (ID)
);
```

You can verify if your table has been created successfully by looking at the message displayed by the SQL server, otherwise you can use the **DESC**command as follows –

```
SQL> DESC CUSTOMERS;
+---------+---------------+------+-----+---------+-------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+-------+
| ID      | int(11)       | NO   | PRI |         |       |
| NAME    | varchar(20)   | NO   |     |         |       |
| AGE     | int(11)       | NO   |     |         |       |
| ADDRESS | char(25)      | YES  |     | NULL    |       |
| SALARY  | decimal(18,2) | YES  |     | NULL    |       |
+---------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

SQL - DROP or DELETE Table

The SQL **DROP TABLE** statement is used to remove a table definition and all the data, indexes, triggers, constraints and permission specifications for that table.

NOTE – You should be very careful while using this command because once a table is deleted then all the information available in that table will also be lost forever.

Syntax

The basic syntax of this DROP TABLE statement is as follows –

```
DROP TABLE table_name;
```

Example

Let us first verify the CUSTOMERS table and then we will delete it from the database as shown below –

```
SQL> DESC CUSTOMERS;

+---------+---------------+------+-----+---------+-------+

|Field|Type|Null|Key|Default|Extra|

+---------+---------------+------+-----+---------+-------+

| ID      |int(11)| NO   | PRI |||

| NAME    |varchar(20)| NO   ||||

| AGE     |int(11)| NO   ||||

| ADDRESS |char(25)| YES  || NULL    ||

|SALARY  |decimal(18,2)| YES  || NULL    ||

+---------+---------------+------+-----+---------+-------+

5 rows inset(0.00 sec)
```

This means that the CUSTOMERS table is available in the database, so let us now drop it as shown below.

```
SQL> DROP TABLE CUSTOMERS;

Query OK,0 rows affected (0.01 sec)
```

Now, if you would try the DESC command, then you will get the following error –

```
SQL> DESC CUSTOMERS;

ERROR 1146(42S02):Table'TEST.CUSTOMERS' doesn't exist
```

SQL - INSERT Query

The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

Syntax

There are two basic syntaxes of the INSERT INTO statement which are shown below.

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);
```

Here, column1, column2, column3,...columnN are the names of the columns in the table into which you want to insert the data.

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.

The **SQL INSERT INTO** syntax will be as follows –

```
INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);
```

Example

The following statements would create six records in the CUSTOMERS table.

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (1,'Ramesh',32,'Ahmedabad',2000.00);


INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (2,'Khilan',25,'Delhi',1500.00);
```

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (3,'kaushik',23,'Kota',2000.00);


INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (4,'Chaitali',25,'Mumbai',6500.00);


INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (5,'Hardik',27,'Bhopal',8500.00);


INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (6,'Komal',22,'MP',4500.00);
```

You can create a record in the CUSTOMERS table by using the second syntax as shown below.

```
INSERT INTO CUSTOMERS
VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );
```

All the above statements would produce the following records in the CUSTOMERS table as shown below.

```
+----+----------+-----+-----------+----------+

| ID | NAME     | AGE | ADDRESS   | SALARY   |

+----+----------+-----+-----------+----------+

|1|Ramesh|32|Ahmedabad|2000.00|

|2|Khilan|25|Delhi|1500.00|

|3|kaushik|23|Kota|2000.00|

|4|Chaitali|25|Mumbai|6500.00|

|5|Hardik|27|Bhopal|8500.00|

|6|Komal|22| MP        |4500.00|

|7|Muffy|24|Indore|10000.00|

+----+----------+-----+-----------+----------+
```

Populate one table using another table

You can populate the data into a table through the select statement over another table; provided the other table has a set of fields, which are required to populate the first table.

Here is the syntax –

```
INSERT INTO first_table_name [(column1, column2, ...columnN)]
   SELECT column1, column2, ...columnN
   FROM second_table_name
   [WHERE condition];
```

SQL - SELECT Query

The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

Syntax

The basic syntax of the SELECT statement is as follows –

```
SELECT column1, column2, columnN FROM table_name;
```

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

```
SELECT * FROM table_name;
```

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+

| ID | NAME     | AGE | ADDRESS   | SALARY   |

+----+----------+-----+-----------+----------+

|1|Ramesh|32|Ahmedabad|2000.00|

|2|Khilan|25|Delhi|1500.00|

|3|kaushik|23|Kota|2000.00|

|4|Chaitali|25|Mumbai|6500.00|

|5|Hardik|27|Bhopal|8500.00|

|6|Komal|22| MP        |4500.00|

|7|Muffy|24|Indore|10000.00|
+----+----------+-----+-----------+----------+
```

The following code is an example, which would fetch the ID, Name and Salary fields of the customers available in CUSTOMERS table.

```
SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;
```

This would produce the following result –

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  1 | Ramesh   |  2000.00 |
|  2 | Khilan   |  1500.00 |
|  3 | kaushik  |  2000.00 |
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

If you want to fetch all the fields of the CUSTOMERS table, then you should use the following query.

```
SQL> SELECT * FROM CUSTOMERS;
```

This would produce the result as shown below.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

SQL - WHERE Clause

The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. You should use the WHERE clause to filter the records and fetching only the necessary records.

The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc., which we would examine in the subsequent chapters.

Syntax

The basic syntax of the SELECT statement with the WHERE clause is as shown below.

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

You can specify a condition using the comparison or logical operators like >, <, =, **LIKE, NOT**, etc. The following examples would make this concept clear.

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+

| ID | NAME     | AGE | ADDRESS   | SALARY   |

+----+----------+-----+-----------+----------+

|1|Ramesh|32|Ahmedabad|2000.00|

|2|Khilan|25|Delhi|1500.00|

|3|kaushik|23|Kota|2000.00|

|4|Chaitali|25|Mumbai|6500.00|

|5|Hardik|27|Bhopal|8500.00|

|6|Komal|22| MP        |4500.00|

|7|Muffy|24|Indore|10000.00|

+----+----------+-----+-----------+----------+
```

The following code is an example which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 –

```
SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY >2000;
```

This would produce the following result –

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

The following query is an example, which would fetch the ID, Name and Salary fields from the CUSTOMERS table for a customer with the name **Hardik**.

Here, it is important to note that all the strings should be given inside single quotes (''). Whereas, numeric values should be given without any quote as in the above example.

```
SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE NAME ='Hardik';
```

This would produce the following result –

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  5 | Hardik   |  8500.00 |
+----+----------+----------+
```

SQL - AND and OR Conjunctive Operators

The SQL **AND** & **OR** operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called as the conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

The AND Operator

The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

Syntax

The basic syntax of the AND operator with a WHERE clause is as follows –

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] AND [condition2]...AND [conditionN];
```

You can combine N number of conditions using the AND operator. For an action to be taken by the SQL statement, whether it be a transaction or a query, all conditions separated by the AND must be TRUE.

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example, which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 and the age is less than 25 years –

```
SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY >2000 AND age <25;
```

This would produce the following result –

```
+----+-------+----------+
| ID | NAME  | SALARY   |
+----+-------+----------+
|  6 | Komal |  4500.00 |
|  7 | Muffy | 10000.00 |
+----+-------+----------+
```

The OR Operator

The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

Syntax

The basic syntax of the OR operator with a WHERE clause is as follows –

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] OR [condition2]...OR [conditionN]
```

You can combine N number of conditions using the OR operator. For an action to be taken by the SQL statement, whether it be a transaction or query, the only any ONE of the conditions separated by the OR must be TRUE.

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+

| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+

|1|Ramesh|32|Ahmedabad|2000.00|

|2|Khilan|25|Delhi|1500.00|

|3|kaushik|23|Kota|2000.00|

|4|Chaitali|25|Mumbai|6500.00|

|5|Hardik|27|Bhopal|8500.00|

|6|Komal|22| MP        |4500.00|

|7|Muffy|24|Indore|10000.00|

+----+----------+-----+-----------+----------+
```

The following code block hasa query, which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 and the age is less than 25 years.

```
SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY >2000 OR age <25;
```

This would produce the following result –

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  3 | kaushik  |  2000.00 |
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

SQL - UPDATE Query

The SQL **UPDATE** Query is used to modify the existing records in a table. You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

Syntax

The basic syntax of the UPDATE query with a WHERE clause is as follows –

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

You can combine N number of conditions using the AND or the OR operators.

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+

| ID | NAME     | AGE | ADDRESS   | SALARY   |

+----+----------+-----+-----------+----------+

|1|Ramesh|32|Ahmedabad|2000.00|

|2|Khilan|25|Delhi|1500.00|

|3|kaushik|23|Kota|2000.00|

|4|Chaitali|25|Mumbai|6500.00|

|5|Hardik|27|Bhopal|8500.00|

|6|Komal|22| MP        |4500.00|

|7|Muffy|24|Indore|10000.00|

+----+----------+-----+-----------+----------+
```

The following query will update the ADDRESS for a customer whose ID number is 6 in the table.

```
SQL> UPDATE CUSTOMERS

SET ADDRESS ='Pune'

WHERE ID =6;
```

Now, the CUSTOMERS table would have the following records –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | Pune      |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to modify all the ADDRESS and the SALARY column values in the CUSTOMERS table, you do not need to use the WHERE clause as the UPDATE query would be enough as shown in the following code block.

```
SQL> UPDATE CUSTOMERS

SET ADDRESS ='Pune', SALARY =1000.00;
```

Now, CUSTOMERS table would have the following records –

```
+----+----------+-----+---------+---------+
| ID | NAME     | AGE | ADDRESS | SALARY  |
+----+----------+-----+---------+---------+
|  1 | Ramesh   |  32 | Pune    | 1000.00 |
|  2 | Khilan   |  25 | Pune    | 1000.00 |
|  3 | kaushik  |  23 | Pune    | 1000.00 |
|  4 | Chaitali |  25 | Pune    | 1000.00 |
|  5 | Hardik   |  27 | Pune    | 1000.00 |
|  6 | Komal    |  22 | Pune    | 1000.00 |
|  7 | Muffy    |  24 | Pune    | 1000.00 |
+----+----------+-----+---------+---------+
```

SQL - DELETE Query

The SQL DELETE Query is used to delete the existing records from a table.

You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

Syntax

The basic syntax of the DELETE query with the WHERE clause is as follows –

```
DELETE FROM table_name
WHERE [condition];
```

You can combine N number of conditions using AND or OR operators.

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code has a query, which will DELETE a customer, whose ID is 6.

```
SQL> DELETE FROM CUSTOMERS

WHERE ID =6;
```

Now, the CUSTOMERS table would have the following records.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to DELETE all the records from the CUSTOMERS table, you do not need to use the WHERE clause and the DELETE query would be as follows –

```
SQL> DELETE FROM CUSTOMERS;
```

Now, the CUSTOMERS table would not have any record.

SQL - Using Joins

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Consider the following two tables –

Table 1 – CUSTOMERS Table

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|1|Ramesh|32|Ahmedabad|2000.00|
|2|Khilan|25|Delhi|1500.00|
|3|kaushik|23|Kota|2000.00|
|4|Chaitali|25|Mumbai|6500.00|
|5|Hardik|27|Bhopal|8500.00|
|6|Komal|22| MP        |4500.00|
|7|Muffy|24|Indore|10000.00|
+----+----------+-----+-----------+----------+
```

Table 2 – ORDERS Table

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
|102|2009-10-0800:00:00|3|3000|
|100|2009-10-0800:00:00|3|1500|
|101|2009-11-2000:00:00|2|1560|
|103|2008-05-2000:00:00|4|2060|
+-----+---------------------+-------------+--------+
```

Now, let us join these two tables in our SELECT statement as shown below.

```
SQL> SELECT ID, NAME, AGE, AMOUNT

   FROM CUSTOMERS, ORDERS

WHERE  CUSTOMERS.ID= ORDERS.CUSTOMER_ID;
```

This would produce the following result.

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
|  3 | kaushik  |  23 |   3000 |
|  3 | kaushik  |  23 |   1500 |
|  2 | Khilan   |  25 |   1560 |
|  4 | Chaitali |  25 |   2060 |
+----+----------+-----+--------+
```

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal to symbol.

There are different types of joins available in SQL –

- INNER JOIN – returns rows when there is a match in both tables.
- LEFT JOIN – returns all rows from the left table, even if there are no matches in the right table.
- RIGHT JOIN – returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN – returns rows when there is a match in one of the tables.
- SELF JOIN – is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- CARTESIAN JOIN – returns the Cartesian product of the sets of records from the two or more joined tables.

SQL Connection In Windows Form

```
*There are multiple way to create and execute sql connection through C#
and built in sql classes. Here one of them. But every system you need to:
1. Connection String
2. SqlConnetion
3. Query
4. SqlCommand

For Execution you need to:

      1. Connection String
      2. Sql Connection
      3. Query
      4. Sql Command
      5. Connection Open
      6. Execute Query
      7. Connection Close
```

Here is a simple example :

```
using System;
usingSystem.Windows.Forms;
usingSystem.Data.SqlClient;

namespace WindowsApplication1
{
public partial class Form1 : Form
    {
public Form1()
        {
InitializeComponent();
        }

private void button1_Click(object sender, EventArgs e)
        {
stringconnetionString = null;
SqlConnectionconnection ;
SqlCommandcommand ;
stringsql = null;

connetionString = "Data Source=ServerName;Initial
Catalog=DatabaseName;User ID=UserName;Password=Password";
sql = "Your SQL Statemnt Here";

connection = new SqlConnection(connetionString);
try
            {
connection.Open();
command = new SqlCommand(sql, connection);
command.ExecuteNonQuery();
command.Dispose();
connection.Close();
MessageBox.Show (" ExecuteNonQuery in SqlCommandexecuted !!");
            }
catch (Exception ex)
            {
MessageBox.Show("Can not open connection ! ");
            }
        }
    }
}
```