SQL

1. **SQL** (*Structured Query Language*) is **used** to **perform operations** on the **records stored** in the **database** such as **updating** records, **deleting** records, **creating** and **modifying tables**, views, etc.
2. SQL is just a **query language**; it is **not a database**. To **perform** SQL queries, you **need** to **install** any **database**, for example, Oracle, MySQL, MongoDB, PostGre SQL, SQL Server, DB2, etc.

What is SQL?

1. SQL stands for **Structured Query Language**.
2. It is **designed** for **managing data** in a relational database management system (**RDBMS**).
3. It is pronounced as S-Q-L or sometime **See-Qwell**.
4. SQL is a database language, it is **used** for database creation, deletion, fetching rows, and modifying rows, etc.

- All **DBMS** like MySQL, Oracle, MS Access, Sybase, Informix, PostgreSQL, and SQL Server use SQL as **standard** database language.

Why SQL is required?

SQL is required:

1. To **create** new databases, tables and views
2. To **insert** records in a database
3. To **update** records in a database
4. To **delete** records from a database
5. To **retrieve** data from a database

What SQL does?

- With SQL, we can query our database in several ways, using English-like statements.
- With SQL, a user can access data from a relational database management system.
- It allows the user to describe the data.
- It allows the user to define the data in the database and manipulate it when needed.
- It allows the user to create and drop database and table.
- It allows the user to create a view, stored procedure, function in a database.
- It allows the user to set permission on tables, procedures, and views.

SQL Syntax

SQL **follows** some **unique** set of **rules** and **guidelines** called **syntax**. Here, we are providing all the basic SQL syntax.

- **SQL** is **not** case sensitive. Generally SQL **keywords** are **written** in **uppercase.**
- SQL statements are **dependent** on text lines. We can place a single SQL statement on one or multiple text lines.
- You can perform most of the **action** in a database **with** SQL statements.

SQL statement

SQL statements are **started** with any of the SQL **commands/keywords** like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP etc. and the **statement ends with a semicolon (;).**

Example of SQL statement:

1. **SELECT** *"column_name"* **FROM** *"table_name";*

Why semicolon is used after SQL statements:

Semicolon is **used** to **separate** SQL statements. It is a **standard way** to **separate** SQL statements in a **database system** in which more than one SQL statements are used in the same call.

SQL Commands

These are the some important SQL command:

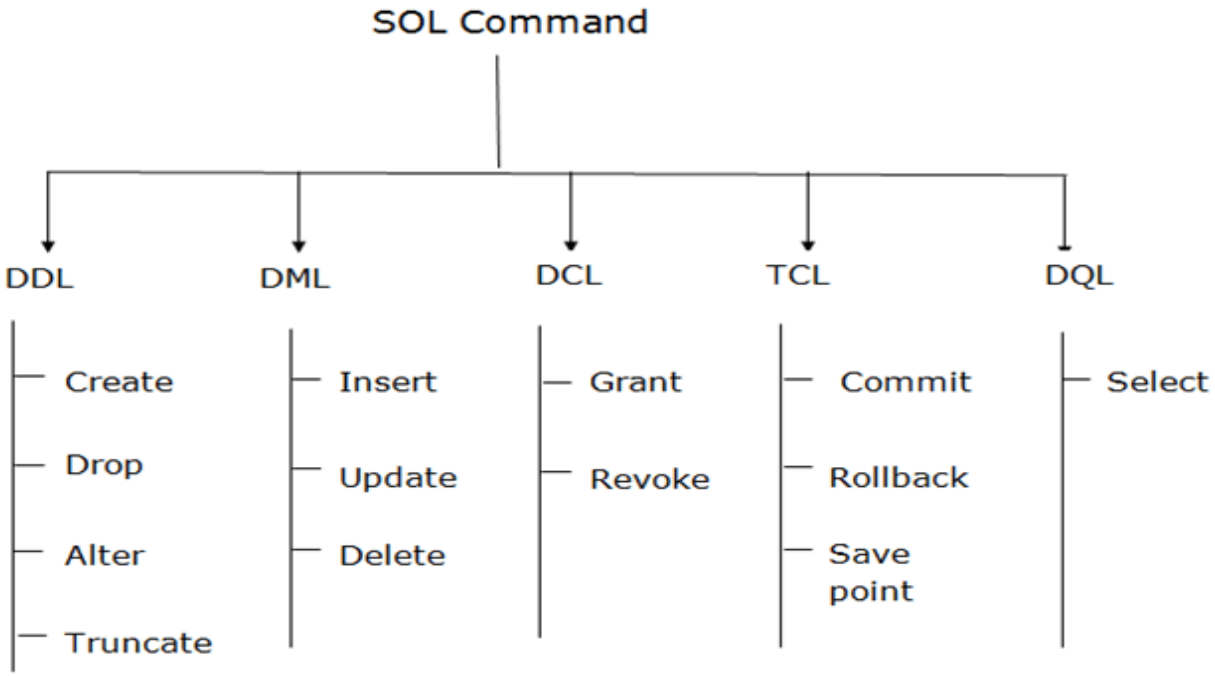
- **SELECT:** it extracts data from a database.
- **UPDATE:** it updates data in database.
- **DELETE:** it deletes data from database.
- **CREATE TABLE:** it creates a new table.
- **ALTER TABLE:** it is used to modify the table.
- **DROP TABLE:** it deletes a table.
- **CREATE DATABASE:** it creates a new database.
- **ALTER DATABASE:** It is used to modify a database.
- **INSERT INTO:** it inserts new data into a database.
- **CREATE INDEX:** it is used to create an index (search key).
- **DROP INDEX:** it deletes an index.

SQL Commands

- SQL commands are **instructions**. It is **used** to communicate **with** the database. It is also **used** to **perform** specific tasks, functions, and queries of **data**.
- SQL can **perform** various **tasks** like create a table, add data to tables, drop the table, modify the table, and set **permission for users**.

Types of SQL Commands

There are **five** types of SQL commands: **DDL, DML, DCL, TCL, and DQL.**



1. Data Definition Language (DDL)

- DDL **changes** the structure of the table **like** creating a table, deleting a table, altering a table, etc.
- **All** the **command** of DDL are **auto-committed** that **means** it permanently **save** all the **changes** in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

2. Data Manipulation Language

- DML commands are **used** to **modify** the database. It is responsible for all form of changes in the database.
- The command of DML is **not auto-committed** that means it **can't permanently save** all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

3. Data Query Language

DQL is **used** to **fetch** the data from the database.

It uses only one command:

- SELECT

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Sign Up" or "Login Credentials" or “Orders” or “Employ Info”). Tables contain records (rows) with data.

Table Name- **Sign Up**

ID	FN	LN	MobNo	EmailId	City
1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
3	Anna	Hajare	3333	3@gmail.com	Pune
4	Sharad	Pawar	4444	4@gmail.com	Baramati
5	Soniya	Gandhi	5555	5@gmail.com	Pune

The table above contains five records (one for each user) and 6 columns (ID, FN, LN, MobNo, EmailId, & City).

1. **SQL SELECT Statement**

The SELECT statement is used to select data from a database / table.

The following SQL statement selects or fetch all the records in the "SignUp" table:

The following SQL statement selects all the columns from the "SignUp" table:

SELECT Syntax

```
SELECT * FROM table_name;
```

Example

```
SELECT * FROM SignUp;
```

Output / Result

ID	FN	LN	MobNo	EmailId	City
1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
3	Anna	Hajare	3333	3@gmail.com	Pune
4	Sharad	Pawar	4444	4@gmail.com	Baramati
5	Soniya	Gandhi	5555	5@gmail.com	Pune

The following SQL statement selects or fetch the particular column data / records in the "Sign Up" table:

The following SQL statement selects the "FN" and "LN" columns from the "SignUp" table:

Syntax

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

Example

```
SELECT FN, LN
FROM SignUp;
```

Output / Result

	FN	**LN**
1	Rahul	Gandhi
2	Arvind	Kejriwal
3	Anna	Hajare
4	Sharad	Pawar
5	Soniya	Gandhi

2. **DISTINCT**

The SELECT DISTINCT statement is **used** to **return** / show only distinct (different) values or unique values in the particular column.

Inside a table, a column often **contains many** duplicate values; and sometimes you only want to list the different (distinct) values.

The following SQL statement selects only the DISTINCT values from the "LN" column in the "SignUp" table:

Syntax

```
SELECT DISTINCT column1, column2,...
FROM table_name;
```

Example

```
SELECT DISTINCT LN
FROM SignUp;
```

Output / Result

	LN
1	Gandhi
2	Kejriwal
3	Hajare
4	Pawar

SELECT Example Without DISTINCT

The following SQL statement selects all (including the duplicates) values from the "LN" column in the "SignUp" table:

Syntax

```
SELECT Column1
FROM table_name;
```

Example

```
SELECT LN
FROM SignUp;
```

Output / Result

	LN
1	Gandhi
2	Kejriwal
3	Hajare
4	Pawar
5	Gandhi

The following SQL statement lists the number of different (distinct) LN:

Syntax

```
SELECT COUNT (DISTINCT column1, column2,...)
FROM table_name;
```

Example

```
SELECT COUNT (DISTINCT LN)
FROM SignUp;
```

Output / Result

Number of Records-4

Number of Records-4

3. TOP

It is used to specify the number of records to return.

It is a SQL statement it used to show / display the top records in a particular column.

It is a SQL statement use to select top values in a particular table

The following SQL statement selects the first three records from the "Signup" table (for SQL Server/MS Access):

Syntax

```
SELECT TOP value / number * FROM table_name;
```

Example

```
SELECT TOP 3 * FROM SignUp;
```

Output

	ID	FN	LN	MobNo	EmailId	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
3	3	Anna	Hajare	3333	3@gmail.com	Pune

The following SQL statement selects the first 50% of the records from the "Sign Up" table (for SQL Server/MS Access):

Syntax

```
SELECT TOP 50 PERCENT * FROM table_name;
```

Example

```
SELECT TOP 50 PERCENT * FROM SignUp;
```

Output

	ID	FN	LN	MobNo	EmailId	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
3	3	Anna	Hajare	3333	3@gmail.com	Pune

4. ORDER BY

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

It is a SQL statement it used to display the records by ascending & descending order from selected column.

The following SQL statement selects all SignUp from the "SignUp" table, sorted by the "FN" column:

Syntax

```
SELECT * FROM table_name
ORDER BY column1;
```

Example

```
SELECT * FROM SignUp
ORDER BY FN;
```

Output

	ID	FN	LN	MobNo	EmailId	City
1	3	Anna	Hajare	3333	3@gmail.com	Pune
2	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
3	1	Rahul	Gandhi	1111	1@gmail.com	Pune
4	4	Sharad	Pawar	4444	4@gmail.com	Baramati
5	5	Soniya	Gandhi	5555	5@gmail.com	Pune

ORDER BY DESC Example

The following SQL statement selects all SignUp from the "SignUp" table, sorted DESCENDING by the "FN" column:

Syntax

```
SELECT * FROM table_name
ORDER BY column1 DESC;
```

Example

```
SELECT * FROM SignUP
ORDER BY FN DESC;
```

Output

	ID	FN	LN	MobNo	EmailId	City
2	5	Soniya	Gandhi	5555	5@gmail.com	Pune
1	4	Sharad	Pawar	4444	4@gmail.com	Baramati
3	1	Rahul	Gandhi	1111	1@gmail.com	Pune
4	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
5	3	Anna	Hajare	3333	3@gmail.com	Pune

5. **Where Clause**

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

Note: The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

The following SQL statement selects all the SignUp from the City "Pune", in the "SignUp" table:

Syntax

```
SELECT * FROM table_name
WHERE Condition or Column Name=Value;
```

Example

```
SELECT * FROM SignUp
WHERE City='Pune';
```

Output

	ID	FN	LN	MobNo	EmailId	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	3	Anna	Hajare	3333	3@gmail.com	Pune
3	5	Soniya	Gandhi	5555	5@gmail.com	Pune

Syntax

```
SELECT * FROM table_name
WHERE Condition;
```

Example

```
SELECT * FROM SignUp
WHERE ID=1;
```

Output

	ID	FN	LN	MobNo	EmailId	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune

Syntax

```
SELECT Column1, Column2
FROM table_name
WHERE Condition;
```

Example

```
SELECT EmailId
FROM SignUp
WHERE City='Pune';
```

Output

	Email Id
1	1@gmail.com
2	3@gmail.com
3	5@gmail.com

Types of Where Clause

1. OR
2. AND

The WHERE clause can be combined with AND & OR operators.

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

1. **OR 1 1=1 1 0 =1 0 1=1 0 0=0**

It is a SQL statement use to select records when both condition are true or one condition is true.

It is a SQL statement use to select records either one condition must be true then output will be true.

The following SQL statement selects all fields from "SignUp" where FN is "Rahul" OR LN is "Gandhi":

Syntax

```
SELECT * FROM table_name
WHERE Condition1 OR Condition 2;
```

Example

```
SELECT * FROM SignUp
WHERE FN='Rahul' OR LN='Gandhi';
```

1 1 = 1 0 1=1

Output

	ID	FN	LN	Mob. No	Email Id	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	5	Soniya	Gandhi	5555	5@gmail.com	Pune

Syntax

```
SELECT Column1, Column2
FROM table_name
WHERE Condition1 OR Condition 2;
```

Example

```
SELECT Email Id
FROM SignUp
WHERE FN='Rahul' OR LN='Gandhi';
```

Output

	Email Id
1	1@gmail.com
2	5@gmail.com

2. **AND**

It is a SQL statement in that select record when both the condition must be true then output will be true.

The following SQL statement selects all fields from "SignUp" where FN is "Rahul" AND LN is "Gandhi":

Syntax

```
SELECT * FROM table_name
WHERE condition1 AND condition2
```

Example

```
SELECT * FROM SignUp
WHERE FN='Rahul' AND LN='Gandhi';      1  1 =1
```

Output

	ID	FN	LN	Mob. No	Email Id	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune

Syntax

```
SELECT Column1, Column2
FROM table_name
WHERE Condition1 AND Condition 2;
```

Example

```
SELECT Mob. No
FROM SignUp
WHERE FN='Rahul' AND LN='Gandhi';
```

Output

	Mob. No
1	1111

6. LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

The following SQL statement selects all SignUp with a FN starting with "A":

- **For Starting with Specific alphabets**

Syntax

```
SELECT * FROM table_name
WHERE column LIKE pattern;
```

Example

```
SELECT * FROM SignUp
WHERE FN LIKE "A%";
```

Output

	ID	FN	LN	Mob. No	Email Id	City
1	3	Anna	Hajare	3333	3@gmail.com	Pune
2	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai

Syntax

```
SELECT column1,column2,...
FROM table_name
WHERE column LIKE pattern;
```

Example

```
SELECT LN
FROM SignUp
WHERE FN LIKE "A%";
```

Output

	LN
1	Kejriwal
2	Hajare

- **For Ending with Specific alphabets**

The following SQL statement selects all SignUp with a FN ending with "d":

Syntax

```
SELECT * FROM table_name
WHERE column LIKE pattern;
```

Example

```
SELECT * FROM SignUp
WHERE FN LIKE "%d";
```

Output

	ID	FN	LN	Mob. No	Email Id	City
1	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
2	4	Sharad	Pawar	4444	4@gmail.com	Baramati

Syntax

```
SELECT column1,column2,...
FROM table_name
WHERE column LIKE pattern;
```

Example

```
SELECT LN
FROM SignUp
WHERE FN LIKE "%d";
```

Output

	LN
1	Kejriwal
2	Pawar

Select all records where the value of the FN column does NOT start with the letter "A".

Syntax

```
SELECT * FROM table_name
WHERE column NOT LIKE Pattern;
```

Example

```
SELECT * FROM SignUp
WHERE FN  NOT LIKE "A%";
```

Output

	ID	FN	LN	Mob. No	Email Id	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	4	Sharad	Pawar	4444	4@gmail.com	Baramati
3	5	Soniya	Gandhi	5555	5@gmail.com	Pune

Select all records where the value of the column starts with the letter "A".

Syntax

```
SELECT * FROM table_name
WHERE column LIKE "A%";
```

Select all records where the value of the column *ends* with the letter "a".

Syntax

```
SELECT * FROM table_name
WHERE column LIKE "%a";
```

Select all records where the value of the column starts with letter "A" and ends with the letter "b".

Syntax

```
SELECT * FROM table_name
WHERE column LIKE "A%b";
```

Select all records where the value of the column contains the letter "a".

Syntax

```
SELECT * FROM table_name
WHERE column LIKE "%a%";
```

Select all records where the value of the column does NOT start with the letter "A".

Syntax

```
SELECT * FROM table_name
WHERE column NOT LIKE "A%";
```

For Starting with alphabets ABC

The following SQL statement selects all Sign Up with a FN starting with "A", "B", or "C":

Syntax

```
SELECT * FROM table_name
WHERE column LIKE "[ABC]%";
```

For Ending with alphabets ABC

The following SQL statement selects all Sign Up with a FN ending with "A", "B", or "C":

Syntax

```
SELECT * FROM table_name
WHERE column LIKE "%[ABC]";
```

LIKE Operator	Description
WHERE Column LIKE 'a%'	Finds any values that start with "a"
WHERE Column LIKE '%a'	Finds any values that end with "a"
WHERE Column LIKE '%or%'	Finds any values that have "or" in any position
WHERE Column LIKE '_r%'	Finds any values that have "r" in the second position
WHERE Column LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE Column LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE Column LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

7. WILDCARD

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

The following SQL statement selects all SignUp with a FN starting with any character, followed by "ahul":

Syntax

```
SELECT * FROM table_name
WHERE Column LIKE Specified pattern;
```

Example

```
SELECT * FROM SignUp
WHERE FN LIKE '_ahul';
```

Output

	ID	FN	LN	Mob. No	Email Id	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune

8. IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

It is used to fetch one or more data from the table.

It is use to select those records which specify in query.

The following SQL statement selects all SignUp that are Present in "Rahul", "Soniya":

Syntax

```
SELECT * FROM table_name
WHERE Column IN ('Value1', 'Value2', 'Value3'…)
```

Example

```
SELECT * FROM SignUp
WHERE FN IN ('Rahul', 'Soniya');
```

Output

	ID	FN	LN	Mob. No	Email Id	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	5	Soniya	Gandhi	5555	5@gmail.com	Pune

The following SQL statement selects all Sign Up that are NOT present in "Rahul", "Soniya":

Syntax

```
SELECT * FROM table_name
WHERE Column NOT IN ('Value1', 'Value2', 'Value3'...);
```

Example

```
SELECT * FROM SignUp
WHERE FN NOT IN ('Rahul', 'Soniya');
```

Output

	ID	FN	LN	Mob. No	Email Id	City
1	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
2	3	Anna	Hajare	3333	3@gmail.com	Pune
3	4	Sharad	Pawar	4444	4@gmail.com	Baramati

9. BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

It is a SQL statement use to select value between ranges which specified.

Syntax

```
SELECT * FROM table_name
WHERE Column_name  BETWEEN Value 1 AND Value 2;
```

Example

```
SELECT * FROM SignUp
WHERE ID BETWEEN 2 AND 4;
```

Output

	ID	FN	LN	Mob. No	Email Id	City
1	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
2	3	Anna	Hajare	3333	3@gmail.com	Pune
3	4	Sharad	Pawar	4444	4@gmail.com	Baramati

Example

```
SELECT * FROM SignUp
WHERE Email Id  BETWEEN '1@gmail.com' AND '3@gmail.com';
```

Output

ID	FN	LN	Mob. No	Email Id	City
1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
3	Anna	Hajare	3333	3@gmail.com	Pune

Example

```
SELECT * FROM Sign Up
WHERE FN BETWEEN Arvind AND Soniya;
```

Output

	ID	FN	LN	Mob. No	Email Id	City
1	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
2	3	Anna	Hajare	3333	3@gmail.com	Pune
3	4	Sharad	Pawar	4444	4@gmail.com	Baramati
4	5	Soniya	Gandhi	5555	5@gmail.com	Pune

NOT BETWEEN Example

To display the products outside the range of the previous example, use NOT BETWEEN:

It is a SQL statement use to select value between range which specified.

Syntax

```
SELECT * FROM table_name
WHERE Column_name  NOT BETWEEN Value 1 AND Value 2;
```

Example

```
SELECT * FROM SignUp
WHERE ID NOT BETWEEN 2 AND 4;
```

Output

	ID	FN	LN	Mob. No	Email Id	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	5	Soniya	Gandhi	5555	5@gmail.com	Pune

10. INSERT IN TO

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO *table_name* (*column1*, *column2*, *column3*,…)
VALUES (*value1*, *value2*, *value3*, ...);

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

INSERT INTO *table_name*
VALUES (*value1*, *value2*, *value3*, ...);

The following SQL statement inserts a new record in the "SignUp" table:

Syntax

INSERT INTO *table_name*
VALUES (*value1*, *value2*, *value3*, ...);

Example

INSERT INTO *SignUp*
VALUES (6,"*Jayant", "Patil", 6666, 6@gmail.com, "Sangli");*

Output

	ID	FN	LN	Mob. No	Email Id	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
3	3	Anna	Hajare	3333	3@gmail.com	Pune
4	4	Sharad	Pawar	4444	4@gmail.com	Baramati
5	5	Soniya	Gandhi	5555	5@gmail.com	Pune
6	6	Jayant	Patil	6666	6@gmail.com	Sangli

It is also possible to only insert data in specific columns.

The following SQL statement will insert a new record, but only insert data in the "FN", "LN", and "MobNo" columns (Sr.No will be updated automatically):

Syntax

```
INSERT INTO table_name (column1, column2, column3,…)
VALUES (value1, value2, value3, ...);
```

Example

```
INSERT INTO SignUp(FN,LN,Mob.No)
VALUES (“Amit”,”Deshmukh”,7777);
```

Output

	ID	FN	LN	Mob. No	Email Id	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
3	3	Anna	Hajare	3333	3@gmail.com	Pune
4	4	Sharad	Pawar	4444	4@gmail.com	Baramati

5	5	Soniya	Gandhi	5555	5@gmail.com	Pune
6	6	Jayant	Patil	6666	6@gmail.com	Sangli
7		Amit	Deshmukh	7777		

Did you notice that we did not insert any number into the Sr.No field?

The Sr.No column is in auto-increment field and will be generated automatically when a new record is inserted in to the table.

11. UPDATE

The UPDATE statement is used to modify the existing records in a table.

The following SQL statement updates the FN with a new Mob.No, Email Id & City.

Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Example

```
UPDATE SignUp
SET Mob.No = 0000, Email Id = 0@gmail.com, City = Delhi
WHERE FN = Rahul;
```

Output

	ID	FN	LN	Mob. No	Email Id	City
1	1	Rahul	Gandhi	0000	0@gmail.com	Delhi
2	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai

3	3	Anna	Hajare	3333	3@gmail.com	Pune
4	4	Sharad	Pawar	4444	4@gmail.com	Baramati
5	5	Soniya	Gandhi	5555	5@gmail.com	Pune

Update Warning!

Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

Example

UPDATE *SignUp*

SET *Mob.No = 0000, Email Id = 0@gmail.com, City = Delhi*

Output

	ID	FN	LN	Mob. No	Email Id	City
1	1	Rahul	Gandhi	0000	0@gmail.com	Delhi
2	2	Arvind	Kejriwal	0000	0@gmail.com	Delhi
3	3	Anna	Hajare	0000	0@gmail.com	Delhi
4	4	Sharad	Pawar	0000	0@gmail.com	Delhi
5	5	Soniya	Gandhi	0000	0@gmail.com	Delhi

Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

12. DELETE

The DELETE statement is used to delete existing records in a table.

It is a SQL Statement used to delete the particular row in to the table.

The following SQL statement deletes the Mob.No "2222" from the "SignUp" table:

Syntax

DELETE FROM *table_name* WHERE *condition*;

Example

DELETE FROM *SignUp* WHERE *Mob.No = 2222*;

Output

ID	FN	LN	Mob. No	Email Id	City
1	Rahul	Gandhi	1111	1@gmail.com	Delhi
3	Anna	Hajare	3333	3@gmail.com	Pune
4	Sharad	Pawar	4444	4@gmail.com	Baramati
5	Soniya	Gandhi	5555	5@gmail.com	Pune

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

Syntax

DELETE FROM *table_name*;

The following SQL statement deletes all rows in the "SignUp" table, without deleting the table:

Example

DELETE FROM *SignUp;*

Output

Delete all the records only

ID	FN	LN	Mob. No	Email Id	City

13. SELECT IN TO

The SELECT INTO statement copies data from one table into a new table.

Syntax

SELECT * INTO *newtable* FROM *oldtable;*

Example

SELECT * INTO *Register* FROM *SignUp;*

Output

Table Name- **Register**

ID	FN	LN	Mob. No	Email Id	City
1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
3	Anna	Hajare	3333	3@gmail.com	Pune
4	Sharad	Pawar	4444	4@gmail.com	Baramati
5	Soniya	Gandhi	5555	5@gmail.com	Pune

14. ALIAS

It is a SQL statement it is used for to change the temporary column name & table name without changing in database.

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

Alias Column Syntax

```
SELECT column_name AS alias_name
FROM table_name;
```

Alias Table Syntax

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

Alias for Columns Examples

The following SQL statement creates two aliases, one for the FN column and one for the LN column:

Example

```
SELECT FN AS FirstName, LN AS LastName
FROM SignUp;
```

Output

Table Name- SignUp

	ID	FirstName	LastName	Mob. No	Email Id	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
3	3	Anna	Hajare	3333	3@gmail.com	Pune
4	4	Sharad	Pawar	4444	4@gmail.com	Baramati
5	5	Soniya	Gandhi	5555	5@gmail.com	Pune

Alias for Columns & Table Examples

Example

```
SELECT FN AS FirstName, LN AS LastName
FROM SignUp AS Register;
```

Output

Table Name- Register

	ID	FirstName	LastName	Mob. No	Email Id	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
3	3	Anna	Hajare	3333	3@gmail.com	Pune
4	4	Sharad	Pawar	4444	4@gmail.com	Baramati
5	5	Soniya	Gandhi	5555	5@gmail.com	Pune

Aliases can be useful when:

- There are more than one table involved in a query

- Functions are used in the query
- Column names are big or not very readable
- Two or more columns are combined together

15. UNION

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

UNION Syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

Example

Table Name- **Sign up**

ID	FN	LN	Mob. No	Email Id	City	Seats
1	Rahul	Gandhi	1111	1@gmail.com	Pune	45
2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai	49
3	Anna	Hajare	3333	3@gmail.com	Pune	45
4	Sharad	Pawar	4444	4@gmail.com	Baramati	55
5	Soniya	Gandhi	5555	5@gmail.com	Pune	49

Table Name- **Register**

ID	FN	LN	Mob. No	Email Id	City	Seats
1	Rahul	Gandhi	1111	1@gmail.com	Pune	45
2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai	49
3	Anna	Hajare	3333	3@gmail.com	Pune	45
4	Sharad	Pawar	4444	4@gmail.com	Baramati	55
5	Soniya	Gandhi	5555	5@gmail.com	Pune	49
6	Sanjay	Raut	66666	6@gmail.com	Mumbai	53
7	Uddhav	Thakre	7777	7@gmail.com	Mumbai	54
8	Rohit	Pawar	8888	8@gmail.com	Baramati	56
9	Dhiraj	Deshmukh	0000	9@gmail.com	Latur	57

Example

```
SELECT LN FROM SignUp
UNION
SELECT LN FROM Register;
```

Output

	LN
1	Deshmukh
2	Gandhi
3	Hajare
4	Kejriwal
5	Pawar
6	Raut
7	Thakre

Example

```
SELECT Seats FROM SignUp
UNION
SELECT Seats FROM Register;
```

Output

	Seats
1	45
2	49
3	53
4	54
5	55
6	56
7	57

UNION ALL

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

Syntax

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

Example

```
SELECT LN FROM SignUp
UNION ALL
SELECT LN FROM Register;
```

Output

	LN
1	Gandhi
2	Kejriwal
3	Hajare
4	Pawar
5	Gandhi
6	Raut
7	Thakre
8	Pawar
9	Deshmukh

16. CREATE TABLE

The CREATE TABLE statement is used to create a new table in a database.

Syntax

```
CREATE TABLE table_name
(
   column1 datatype,
   column2 datatype,
   column3 datatype,
  ....
);
```

The column parameters specify the names of the columns of the table.

The data type parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

Example

The following example creates a table called "SignUp" that contains six columns: ID, FN, LN, Mob.No, Email Id and City:

```
CREATE TABLE SignUp
 (
    ID int,
    FN varchar(255),
    LN varchar(255),
   Mob.No int,
   Email Id varchar(255),
   City varchar(255)
);
```

The ID, Mob.No, column is of type int and will hold an integer.

The FN, LN, Email Id and City columns are of type varchar and will hold characters, and the maximum length for these fields is 255 characters.

The empty "Sign Up" table will now look like this:

Output

Table name- SignUp

ID	FN	LN	Mob. No	Email Id	City

17. DROP

The DROP TABLE statement is used to drop an existing table in a database.

It is used to delete both the structure & records stored in the table

Syntax

DROP TABLE *table_name*;

Example

DROP TABLE *SignUp*;

Output

Delete both the structure & records stored in the table

Note: Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!

18. TRUNCATE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

It is used to delete all the rows from the table means to records stored in the table & the structure remain same. Not delete the structure of the table.

Syntax

TRUNCATE TABLE *table_name*;

Example

TRUNCATE TABLE *SignUp*;

Output

ID	FN	LN	Mob. No	Email Id	City

No.	DELETE	TRUNCATE
1)	DELETE is a **DML command.**	TRUNCATE is a **DDL command.**
2)	We **can use WHERE** clause in DELETE command.	We **cannot use WHERE** clause with TRUNCATE
3)	DELETE statement is used **to delete a row** from a table	TRUNCATE statement is used **to remove all the rows** from a table.
4)	DELETE is **slower** than TRUNCATE statement.	TRUNCATE statement is **faster** than DELETE statement.
5)	You **can rollback** data after using DELETE statement.	It is **not possible to rollback** after using TRUNCATE statement.

19. ALTER

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

The following SQL adds an "Address" column to the "SignUp" table:

Syntax

ALTER TABLE *table_name*
ADD *column_name datatype*;

Example

ALTER TABLE *Signup*

ADD *Address varchar(255)*;

Output

	ID	FN	LN	Mob. No	Email Id	City	Address
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune	
2	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai	
3	3	Anna	Hajare	3333	3@gmail.com	Pune	
4	4	Sharad	Pawar	4444	4@gmail.com	Baramati	
5	5	Soniya	Gandhi	5555	5@gmail.com	Pune	

ALTER TABLE - DROP COLUMN

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

The following SQL deletes the "Address" column from the "SignUp" table:

Syntax

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example

```
ALTER TABLE SignUp
DROP COLUMN Address;
```

Output

	ID	FN	LN	Mob. No	Email Id	City
1	1	Rahul	Gandhi	1111	1@gmail.com	Pune
2	2	Arvind	Kejriwal	2222	2@gmail.com	Mumbai
3	3	Anna	Hajare	3333	3@gmail.com	Pune
4	4	Sharad	Pawar	4444	4@gmail.com	Baramati
5	5	Soniya	Gandhi	5555	5@gmail.com	Pune

ALTER TABLE - ALTER/MODIFY COLUMN

To change the data type of a column in a table, use the following syntax:

SQL Server / MS Access:

ALTER TABLE *table_name*
ALTER COLUMN *column_name datatype*;

My SQL / Oracle (prior version 10G):

ALTER TABLE *table_name*
MODIFY COLUMN *column_name datatype*;

Example

ALTER TABLE *table_name*
MODIFY COLUMN *Date of Birth year/int*;

Notice that the "Date of Birth" column is now of data type year and is going to hold a year in a two- or four-digit format.

Output

"Date of Birth" column is now of data type year and is going to hold a year in a two- or four-digit format.

20. CONSTRAINTS

SQL constraints are used to specify rules for the data in a table.

Constraints 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 table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

Syntax

CREATE TABLE T.N.
(
  C.N.1 *datatype constraint*,
  C.N *2 datatype constraint*,
  C.N.*3 datatype constraint*, ...
);

Example

CREATE TABLE *VCTC*
(

Sr No	int	Primary Key,
Fname	varchar(255)	Not Null,
Lname	varchar(255)	Not Null,
Mockresult	int	Check (Mockresult < 10),
Location	varchar(255)	Default “Punevctc”
Mobileno	int	Unique

);

The following constraints are commonly used in SQL:

1. **NOTNULL**

Ensures that a column cannot have a NULL value

The NOT NULL constraint enforces a column to NOT accept NULL values.

SQL NOT NULL on CREATE TABLE

Syntax

```
CREATE TABLE table_name
(
   column1 datatype NOT NULL,
   column2 datatype NOT NULL,
   column3 datatype NOT NULL,
   ....
);
```

The following SQL ensures that the "ID", "FN", "LN", “Mob. No”, “Email Id” and “City” columns will NOT accept NULL values when the "SignUp" table is created:

Example

```
CREATE TABLE SignUp
(
   ID int NOT NULL,
   FN varchar(255) NOT NULL,
   LN varchar(255) NOT NULL,
  Mob.No int NOT NULL,
  Email Id varchar(255) NOT NULL,
  City varchar(255) NOT NULL
);
```

2. UNIQUE KEY

Ensures that all values in a column are different

Unique Constraint allow only one NULL value.

Each table can have more than one / multiple Unique Constraint / value.

Syntax

```
CREATE TABLE table_name

(
   column1 datatype NOT NULL UNIQUE,
   column2 datatype NULL,
   column3 datatype NOT NULL,
   Column4 datatype NOT NULL UNIQUE,
  ....
);
```

The following SQL creates a UNIQUE constraint on the "ID" & “Mob. No” column when the "SignUp" table is created:

Example

```
CREATE TABLE SignUp

 (
    ID int NOT NULL/UNIQUE,
    FN varchar(255) NOT NULL,
    LN varchar(255) NOT NULL,
   Mob.No int NOT NULL/NULL/UNIQUE,
   Email Id varchar(255) NOT NULL,
   City varchar(255) NOT NULL
);
```

3. PRIMARY KEY

The PRIMARY KEY constraint uniquely identifies each record (rows) in a table.

A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

Syntax

```
CREATE TABLE table_name

(
   column1 datatype NOT NULL UNIQUE,
   column2 datatype NOT NULL,
   column3 datatype NOT NULL,
   column4 datatype NOT NULL UNIQUE,
  PRIMARY KEY (column)
   ....
);
```

The following SQL creates a PRIMARY KEY on the "ID" column when the "SignUp" table is created:

Example

```
CREATE TABLE SignUp

(
   ID int NOT NULL UNIQUE,
   FN varchar(255) NOT NULL,
```

LN varchar(255) NOT NULL,
Mob.No int NOT NULL/NULL/UNIQUE,
Email Id varchar(255) NOT NULL,
City varchar(255) NOT NULL,
PRIMARY KEY *(ID)*
);

4. FORGIEN KEY

A forgien key is SQL constraints use to link two tables together

Foreign key can accept multiple null value.

We can have more than one foreign key in a table.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Syntax

CREATE TABLE *table_name*

(
column1 datatype NOT NULL/NULL,
column2 datatype NOT NULL/NULL,
column3 datatype NOT NULL/NULL,
column4 datatype NOT NULL/NULL,
FOREIGN KEY *(column)*
....
);

Example

```
CREATE TABLE UIDAI
(
AADHAR NO int(25),
FIRSTNAME varchar(256)  NOT NULL,
LASTNAME varchar(256),
MOBILE int(20) UNIQUE,
EMAIL ID varchar(256) UNIQUE,
PAN CARD varchar(25) UNIQUE,
CITY varchar(25),
PRIMARY KEY (AADHAR NO)
)
CREATE TABLE ICICI
(
FIRSTNAME varchar(256),
LASTNAME varchar(256),
BALANCE int (20),
ACCOUNT NO int (20),
AADHAR NO int (20),
FOREIGN KEY (AADHAR NO) REFERENCE UIDAI (AADHAR NO)
)
```

CREATE TABLE *IDBI*

(

AADHAR NO. int (20),

BALANCE int (20),

FOREIGN KEY *(AADHAR NO)* REFERENCE *UIDAI (AADHAR NO)*

)

Notice that the "*AADHAR NO*" column in the "*ICICI*", "*AADHAR NO*" column in the "*IDBI*" table points to the "*AADHAR NO*" column in the "*UIDAI*" table.

The "*AADHAR NO*" column in the "*UIDAI*" table is the PRIMARY KEY in the "*UIDAI*" table.

The "*AADHAR NO*" column in the "*ICICI*" table is a FOREIGN KEY in the "*ICICI*" table.

The "*AADHAR NO*" column in the "*IDBI*" table is a FOREIGN KEY in the "*IDBI*" table.

5. DEFAULT

Sets a default value for a column if no value is specified

SQL DEFAULT on CREATE TABLE

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

Syntax

```
CREATE TABLE table_name

(
   column1 datatype NOT NULL/NULL,
   column2 datatype NOT NULL/NULL,
   column3 datatype NOT NULL/NULL DEFAULT "Enter Name/City",
   column4 datatype NOT NULL/NULL,
  ....
);
```

Example

```
CREATE TABLE Persons

(
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int NOT NULL,
   City varchar(255) DEFAULT 'Latur'

);
```

Output

Table Name- Persons

ID	LastName	FirstName	Age	City
1			20	Latur
2			22	Latur
3			23	Latur

6. CHECK

Ensures that the values in a column satisfies a specific condition

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK on CREATE TABLE

Syntax

```
CREATE TABLE table_name
(
  column1 datatype NOT NULL/NULL,
  column2 datatype NOT NULL/NULL,
  column3 datatype NOT NULL/NULL,
  column4 datatype NOT NULL/NULL,
  CHECK (column) <, >, = Value
  ....
);
```

Example

```
CREATE TABLE Persons
(
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int NOT NULL,
  CHECK (Age>21)
);
```

Output

Table Name- Persons

PersonID	LastName	FirstName	Age
1			22
2			23

Difference between Primary key, Foreign key & Unique Key

Primary Key	Foreign Key	Unique Key
The PRIMARY KEY constraint uniquely identifies each record in a table. Primary keys must contain UNIQUE values	A FOREIGN KEY is a key used to link two tables together. A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.	The UNIQUE constraint ensures that all values in a column are different.
Primary key cannot have a NULL value.	Foreign key can accept multiple null value.	Unique Constraint may have a NULL value.
Each table can have only one primary key.	We can have more than one foreign key in a table.	Each table can have more than one Unique Constraint.

21. JOIN

Joins are statements use to joint two or more tables on primary key & foregin key

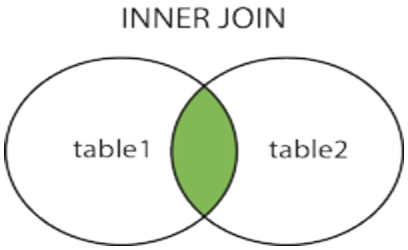
Join is an SQL statement it is used to join the two or more tables based on the basic of relationship between **the primary key & foregin key**

Types of JOIN

1. INNER JOIN

The INNER JOIN keyword selects records that have matching values in both tables.

Inner join is a SQL statement which is use to show records in columns which are related to common values



Syntax

SELECT *column_name(s)*
FROM *table1* INNER JOIN *table2*
ON *table1.column_name* = *table2.column_name*;

or

SELECT *Table1.Column1*
Table1.Column2

Table2.Column1

FROM *Table1* INNER JOIN *Table2*

ON *Table1.PK = Table2.FK*;

Example

Table1- UIDAI

Aadhar (PK)	**First Name**	**Last Name**	**City**
1	Sharad	Pawar	Baramati
2	Ajit	Pawar	Pune
3	Sanjay	Raut	Mumbai

Table2- IDBI

Account No	**Balance**	**Aadhar(FK)**
1234	50000	2
5898	40000	3
9872	30000	2
8796	90000	5

Example

SELECT *UIDAI. First Name*

UIDAI. Last Name

IDBI. Balance

FROM *UIDAI* INNER JOIN *IDBI*

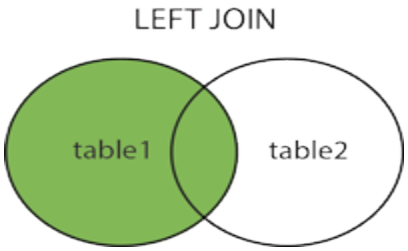
ON *UIDAI. Aadhar = IDBI. Aadhar*;

Output

First Name	**Last Name**	**Balance**
Ajit	Pawar	50000
Ajit	Pawar	30000
Sanjay	Raut	40000

2. LEFT JOIN

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.



Syntax

```
SELECT Table1.Column1
       Table1.Column1
       Table2.Column1
FROM Table1 LEFT JOIN Table2
ON Table1.PK = Table2.FK;
```

Example

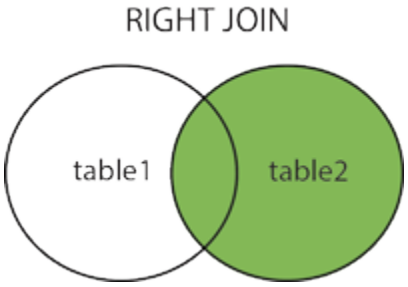
```
SELECT UIDAI. First Name
       UIDAI. Last Name
       IDBI. Balance
FROM UIDAI LEFT JOIN IDBI
ON UIDAI. Aadhar = IDBI. Aadhar;
```

Output

First Name	Last Name	Balance
Sharad	Pawar	
Ajit	Pawar	50000
Ajit	Pawar	30000
Sanjay	Raut	40000

3. RIGHT JOIN

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.



Syntax

```
SELECT Table1.Column1
       Table1.Column1
```

Table2.Column1

FROM *Table1* RIGHT JOIN *Table2*

ON *Table1.PK = Table2.FK*;

Example

SELECT *UIDAI. First Name*

UIDAI. Last Name

IDBI. Balance

FROM *UIDAI* RIGHT JOIN *IDBI*

ON *UIDAI. Aadhar = IDBI. Aadhar*;

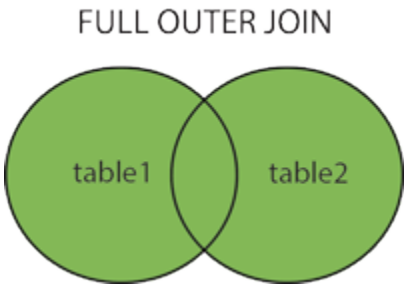
Output

First Name	**Last Name**	**Balance**
Ajit	Pawar	50000
Ajit	Pawar	30000
Sanjay	Raut	40000
		90000

4. **FULL JOIN**

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Tip: FULL OUTER JOIN and FULL JOIN are the same.



Syntax

SELECT *Table1.Column1*
Table1.Column2
Table2.Column1
FROM *Table1* FULL JOIN *Table2*
ON *Table1.PK = Table2.FK*;

Example

SELECT *UIDAI. First Name*
UIDAI. Last Name
IDBI. Balance
FROM *UIDAI* FULL JOIN *IDBI*
ON *UIDAI. Aadhar = IDBI. Aadhar*;

Output

First Name	**Last Name**	**Balance**
Sharad	Pawar	
Ajit	Pawar	50000
Ajit	Pawar	30000
Sanjay	Raut	40000

		90000

22. GROUP BY

The GROUP BY statement group's rows that have the same values into summary rows.

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

Table Name- ICICI

Account No	Withdrawal
1	**6000**
2	**7000**
1	**7000**
3	**6000**
1	**2000**
2	**4000**
3	**9000**
4	**3000**

Syntax

SELECT AGGREGATE FUNCTIONS *(Column)* FROM *table_name*
GROUP BY *Column;*

Example

```
SELECT SUM (Withdrawal) FROM ICICI
GROUP BY Account No.;
```

Output

Withdrawal
14000
11000
15000
3000

Syntax

```
SELECT Column, AGGREGATE FUNCTIONS (Column) FROM table_name
GROUP BY Column;
```

Example

```
SELECT Account No., SUM (Withdrawal) FROM ICICI
GROUP BY Account No.;
```

Output

Account No	Withdrawal
1	14000
2	11000
3	15000
4	3000

23. HAVING

It is a SQL statement use to satisfy the given condition & it is use with the aggregate function.

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

Syntax

```
SELECT AGGREGATE FUNCTIONS (Column) FROM table_name
GROUP BY Column;
HAVING AGGREGATE FUNCTIONS (Column) >, <, =, Value;
```

Example

```
SELECT SUM (Withdrawal) FROM ICICI
GROUP BY Account No.;
HAVING SUM (Withdrawal) <= 12000;
```

Output

Withdrawal
11000
3000

Syntax

SELECT *Column*, AGGREGATE FUNCTIONS *(Column)* FROM *table_name*

GROUP BY *Column;*

HAVING AGGREGATE FUNCTIONS *(Column)* >, <, =, *Value;*

Example

SELECT *Account No.*, SUM *(Withdrawal)* FROM *ICICI*

GROUP BY *Account No.;*

HAVING SUM *(Withdrawal)* <= *12000;*

Output

Account No	Withdrawal
2	11000
4	3000

24. AGGREGATE FUNCTIONS

-Its returns the single value based on the particular column

-An aggregate function performs calculations on set of values and **returns a single value.**

-To perform calculation on multiple rows of a single column and return single value.

Table-Students Marks

Sr.No	FN	LN	Marks
1	Virat	Kohli	65
2	Rohit	Sharma	75
3	Ravindra	Jadeja	95
4	Shikhar	Dhawan	85
5	Rishabh	Pant	55

FIRST():

- Will returns first value in the selected column **except NULL**

- Returns first value of a numeric column

Syntax:

```
SELECT FIRST(column_name)
FROM table_name;
```

Example:

SELECT FIRST*(Marks)*

FROM *Student Marks;*

Output

	FIRST(Marks)
1	65

LAST():

- Will returns last value in the selected column **except NULL**

- Returns last value of a numeric column

Syntax:

SELECT LAST*(column_name)*

FROM *table_name;*

Example:

SELECT LAST*(Marks)*

FROM *Student Marks;*

Output:

	LAST(Marks)
1	55

MIN():

- Will returns minimum value in the selected column **except NULL**

- Returns minimum value of a numeric column

Syntax:

SELECT MIN*(column_name)*

FROM *table_name;*

Example:

SELECT MIN*(Marks)*

FROM *Student Marks;*

Output:

	MIN(Marks)
1	55

MAX():

- Will returns Maximum value in the selected column **except NULL**
- Returns Maximum value of a numeric column

Syntax:

SELECT MAX*(column_name)*

FROM *table_name;*

Example:

SELECT MAX*(Marks)*

FROM *Student Marks;*

Output:

	MAX(Marks)
1	95

SUM():

- Sum of all **Non-Null** Values of a numeric column.
- It returns the total sum of a numeric column

Syntax:

SELECT SUM*(column_name)*

FROM *table_name;*

Example:

SELECT SUM*(Marks)*

FROM *Student Marks;*

Output:

	SUM(Marks)
1	375

AVG():

- This function returns average value of a numeric column

- It will apply for Non-Null Values

- **AVG**=**SUM***(column_name)*/**COUNT***(column_name);*

Syntax:

SELECT AVG*(column_name)*

FROM *table_name;*

Example:

SELECT AVG*(Marks)*

FROM *Student Marks;*

Output:

	AVG(Marks)
1	75

COUNT():

- Returns total number of records

- Return Non-Null Values of a column

Syntax:

SELECT COUNT*(column_name)*

FROM *table_name;*

Example:

SELECT COUNT*(Marks)*

FROM *Student Marks;*

Output:

	COUNT(Marks)
1	5

Find the highest Salary

Table Name – Employ Info

Id. No	FN	LN	Salary
1	Virat	Kohli	1000
2	Rohit	Sharma	2000
3	Ravindra	Jadeja	3000
4	Shikhar	Dhawan	4000
5	Rishabh	Pant	5000
6	Suryakumar	Yadav	6000
7	Ishan	Kishan	7000
8	Hardik	Pandya	8000
9	Jasprit	Bumrah	9000
10	Bhuvneshwar	Kumar	10000

To show 2nd max salary / How to get the 2nd max salary

- **Syntax**

  SELECT MAX (*Column*) FROM *table_name*

  WHERE *Column* NOT IN (SELECT MAX *(Column)* FROM *table_name*);

- **Example**

  SELECT MAX (*Salary*) FROM *Employ Info*

  WHERE *Salary* NOT IN (SELECT MAX *(Salary)* FROM *Employ Info*);

To show 2nd min salary / How to get the 2nd min salary

- **Syntax**

  SELECT MIN (*Column*) FROM *tabale_name*

  WHERE *Column* NOT IN (SELECT MIN *(Column)*) FROM *table_name*);

- **Example**

  SELECT MIN (*Salary*) FROM *Employ Info*

  WHERE *Salary* NOT IN (SELECT MIN *(Salary)*) FROM *Employ Info*);

How to get nth highest salary

- **Syntax**

  SELECT TOP 1 *Column*

  FROM (SELECT TOP N *Column* FROM *table_name*

  ORDER BY *Column* DESC)

  ORDER BY *Column* ASC;

To show 3rd max salary / How to get the 3rd max salary

- **Syntax**

  SELECT TOP 1 *Column*

  FROM (SELECT TOP 3 *Column* FROM *table_name* ORDER BY *Column* DESC)

  ORDER BY *Column* ASC;

- **Example**

```
SELECT TOP 1 Salary
FROM (SELECT TOP 3 Salary FROM Employ Info
        ORDER BY Salary DESC)
ORDER BY Salary ASC;
```

To show 7th highest salary / How to get the 7th highest salary

- **Syntax**

```
SELECT MIN (Column) FROM table_name
WHERE Column IN (SELECT TOP 7 Column FROM table_name
                    ORDER BY Column DESC);
```

- **Example**

```
SELECT MIN (Salary) FROM Employ Info
WHERE Salary IN (SELECT TOP 7 Salary FROM Employ Info
                    ORDER BY Salary DESC);
```

To show 7th min salary / How to get the 7th min salary

- **Syntax**

  SELECT MAX (*Column*) FROM *table_name*

  WHERE *Column* IN (SELECT TOP 7 *Column* FROM *table_name*

  ORDER BY *Column*);

- **Example**

  SELECT MAX (*Salary*) FROM *Employ Info*

  WHERE *Salary* IN (SELECT TOP 7 *Salary* FROM *Employ Info*

  ORDER BY *Salary*);

SQL Questions

1. Explain DML and DDL?
2. How many Aggregate functions are available in SQL?
3. What is the difference in BETWEEN and IN condition operators?
4. What is the difference between the HAVING clause and WHERE clause?
5. What is the difference between DELETE, TRUNCATE & DROP?
6. What are different Clauses used in SQL?
7. What are different SQL constraints?
8. What is the difference between UNIQUE key, PRIMARY KEY & FORGIN KEY constraints?
9. What are different JOINS used in SQL?
10. How to write a query to show the details of a student from Students table whose name start with K?
11. What is the syntax to add a record to a table?
12. What is the syntax of GROUP BY in SQL?
13. Define the SQL DELETE statement.
14. Write a SQL SELECT query that only returns each name only once from a table?
15. Write an SQL query to get the first maximum salary of an employee from a table named employee_table.
16. Write an SQL query to get the second maximum salary of an employee from a table named employee_table.
17. Write an SQL query to get the third maximum salary of an employee from a table named employee_table.
18. Write an SQL query to fetch unique values from a table?
19. Write an SQL query to fetch data from table whose name start with Vipul & Krishna?
20 Write an SQL query to print details of the Workers whose SALARY lies between 100000 and 500000.
21. Write an SQL query to print details of the Workers who have joined in Feb'2014.
22. Write an SQL query to fetch the count of employees working in the department 'Admin'.
23. Write an SQL query to fetch worker names with salaries >= 50000 and <= 100000.