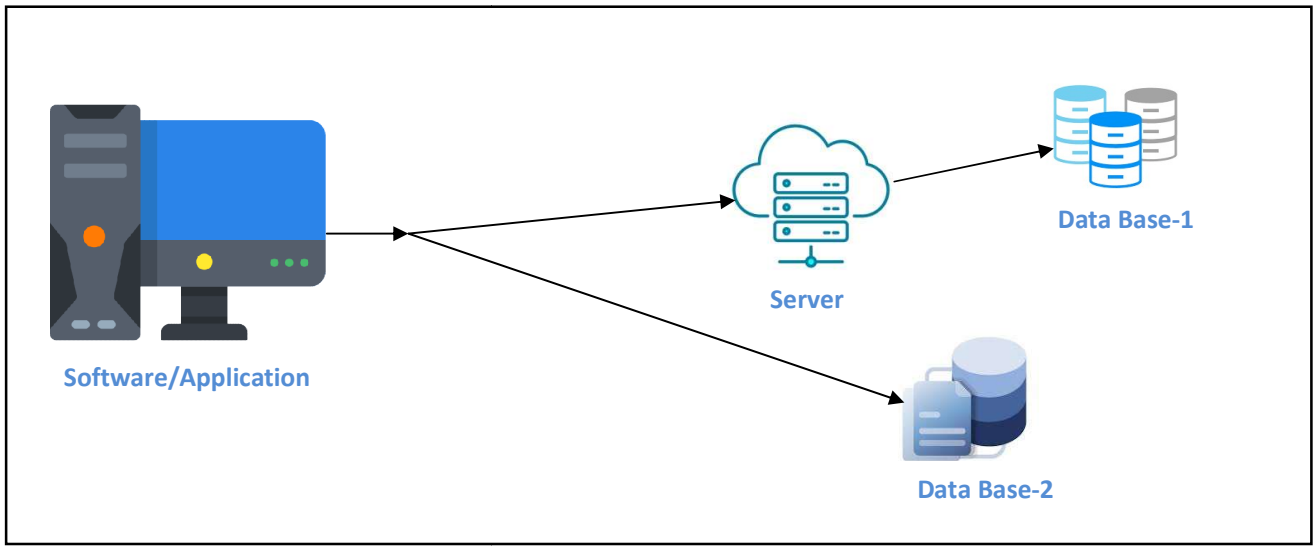
Database Testing/ Backend Testing

Till now we have seen how to test the application from the **Front-End side(UI side)** in that we have covered Functional and Non Functional part.

Now will see how to test the data which were entered from UI side and that will be stored in the **Data Base.**

Data loading into Database-

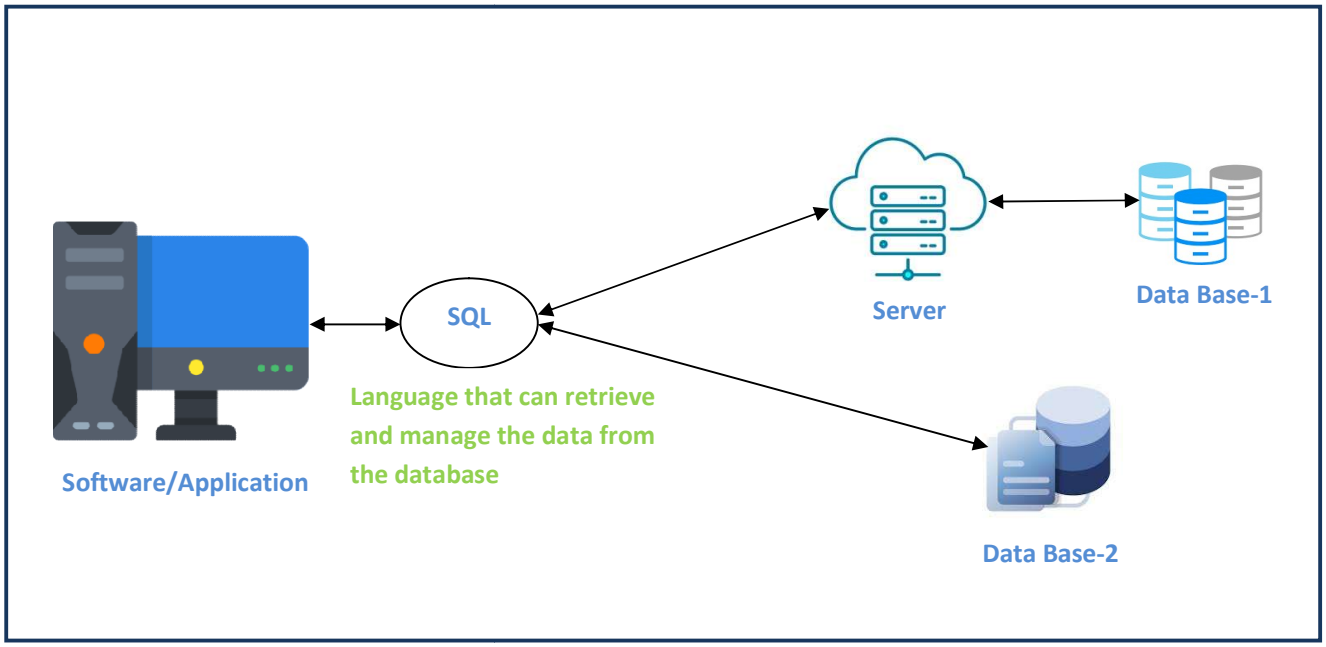


Database:

- A **database** is an **organized collection of data**, so that it can be easily accessed and managed.
- **The main purpose** of the database is to **operate a large amount of information by storing, retrieving, and managing data.**
- We can **organize data** into **tables, rows, columns, and index** it to make it easier to find relevant information. While **manipulating the data.**
- **So for accessing the data we need to understand special language which is known as SQL that can be easily manipulate the data with the predefine set of syntax.**

SQL:

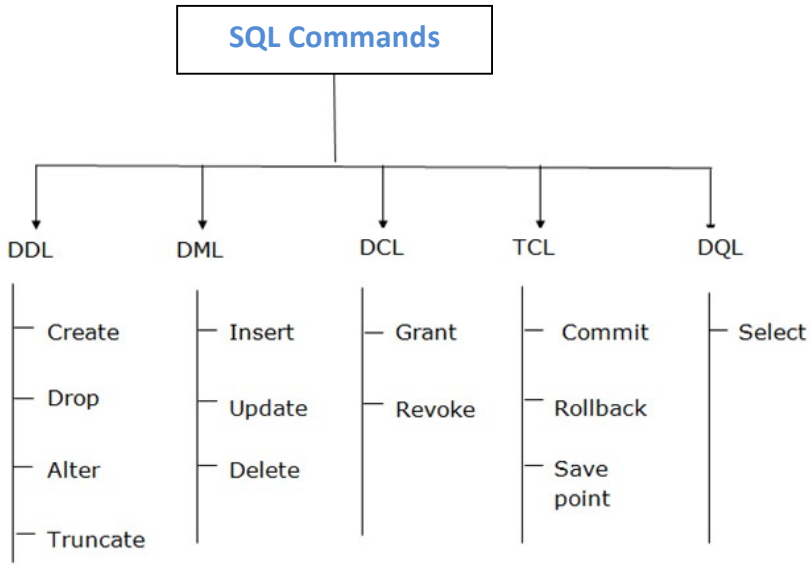
- **SQL** stands for **Structured Query Language**.
- It is a database computer language designed for the **retrieval and management** of data from the database. Also it is **not a Case Sensitive Language**.
- SQL is the standard language for **Relational Database System**. All the Relational Database Management Systems (RDMS) like **MySQL**, MS Access, **Oracle**, Sybase, Informix, Postgres and **SQL Server** use SQL as their standard database language.



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 etc.**

Types of SQL Commands-



DDL (Data Definition Language):

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.**
Some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

DML(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.**
Some commands that come under DML:

- INSERT
- UPDATE
- DELETE

DCL(Data Control Language):

DCL commands are used to **grant and take back authority from any database user.**
Some commands that come under DCL:

- Grant
- Revoke

TCL(Transaction Control Language)

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.
These operations are **automatically committed** in the database **that's why they cannot be used while creating tables or dropping them.**
Some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

DQL(Data Query Language):

DQL is used to **fetch the data from the database.**
It uses only **one** command:

- SELECT

SQL Queries

1. **CREATE-** It is a SQL statement used to create new table in database.
   SYNTAX-
   CREATE TABLE <TABLE NAME> (COLUMN1 DATATYPE (DATASIZE).....);

   Ex- If you want to crate a table for the Students Record.
   Table Name= **Student**
   Column in Student tables = ID, FirstName, LastName, MobileNumber.

   Syntax- CREATE TABLE Student (ID int, FirstName varchar2(10), LastName varchar2(10), MobileNumber int);

---

2. **INSERT INTO -** It is a SQL statement used to insert records in the table.
   SYNTAX-
   INSERT INTO <TABLE NAME> VALUES ('VALUE 1', 'VALUE 2', 'VALUE 3', 'VALUE4');
   Or
   INSERT INTO <TABLE NAME> < COLOUMN NAME> (COL-1, COL-2, COL-3, COL-4) VALUES ('VALUE 1', 'VALUE 2', 'VALUE 3', 'VALUE4');

   Ex- If you want to crate a table for the Students Record.
   Table Name= **Student**
   Column in Student tables = ID, FirstName, LastName, MobileNumber.
   Values = 1, Ashish, Tayade, 1234567890

   INSERT INTO Student VALUES ('1', 'Ashish', 'Tayade', '1234567890');
   Or
   INSERT INTO Student (ID, FirstName, LastName, MobileNumber) VALUES ('1', 'Ashish', 'Tayade', '1234567890');

   Note: If we don't provide the column name with its value then the value for that column is set to **NULL.**

   Task : Create your own database which consist of all the student information from 12Feb 2022 morning batch.

3. **DESCRIBE–** it is a SQL statement used to **describe the structure** of the table.
   SYNTAX- DESC <TABLE NAME>;
   EX- DESC student;

---

4. **SELECT-** It is a SQL statement used to **fetch a data**, columns data/whole table data from database/table.
   SELECT * = Select all data from the table.
   SYNTAX- SELECT *FROM <TABLE NAME>;
   EX- SELECT * FROM Student;

---

5. **SELECT COLUMN –** It is a SQL statement used to fetch data of **particular column** data from the table.
   SYNTAX-
   SELECT <COLUMN NAME> FROM <TABLE NAME>;
   Ex- SELECT FirstName FROM Student; **SELECT** ID, FirstName FROM Student;

---

6. **SELECT WITH (WHERE CLAUSE)** – It is a SQL statement used to select particular record from table using where clause.
   WHERE CLAUSE- It is SQL statement used to extract those records which fulfil the condition.
   SYNTAX-
   SELECT *FROM <TABLE NAME> WHERE <COLUMN NAME>= 'DATA VALUE';
   Ex- SELECT * FROM STUDENT WHERE FirstName = 'Ashish';

---

7. **SELECT COLUMN WITH (WHERE CLAUSE)-** It is a SQL statement used to fetch data of **particular column** data from the table along with it extract those records which fulfil the desired condition.
   SYNTAX-
   SELECT <Column>FROM <TABLE NAME> WHERE <COLUMN NAME>= 'DATA VALUE';
   Ex- SELECT ID, FirstName FROM STUDENT WHERE FirstName = 'Ashish';

---

8. **SELECT WITH (WHERE CLAUSE + AND)-** It is a SQL statement used to fetch data of **particular** data from the table along with it extract those records which fulfil the desired condition (i.e AND Condition).

SYNTAX-

SELECT * FROM <TABLE NAME> WHERE <COLUMN NAME>= 'DATA VALUE' **AND** <COLUMN NAME>= 'DATA VALUE';

SELECT <Column>FROM <TABLE NAME> WHERE <COLUMN NAME>= 'DATA VALUE' **AND** <COLUMN NAME>= 'DATA VALUE';

Ex-

SELECT ID, * FROM STUDENT WHERE FirstName = 'Test' **AND** LastName = 'Demo';

SELECT **ID, FirstName** FROM STUDENT WHERE FirstName = 'Test' **AND** LastName = 'Demo';

---

9. **SELECT COLUMN WITH (WHERE CLAUSE + OR)-** It is a SQL statement used to fetch data of **particular** data from the table along with it extract those records which fulfil the desired condition. (i.e OR Condition).

   SYNTAX-

   SELECT * FROM <TABLE NAME> WHERE <COLUMN NAME>= 'DATA VALUE' **OR** <COLUMN NAME>= 'DATA VALUE';

   SELECT <Column>FROM <TABLE NAME> WHERE <COLUMN NAME>= 'DATA VALUE' **OR** <COLUMN NAME>= 'DATA VALUE';

   Ex-

   SELECT * FROM STUDENT WHERE FirstName = 'Test' **OR** LastName = 'Demo';

   SELECT **ID, FirstName** FROM STUDENT WHERE FirstName = 'Test' **OR** LastName = 'Demo';

---

10. **DELETE –** It is SQL statement used to delete **all records** from table. Excluding the columns.

    SYNTAX- DELETE FROM <TABLE NAME>;

    EX- DELETE FROM Student;

---

11. **DELETE WITH (WHERE CLAUSE)- –** It is a SQL statement used to Delete the data from the table which fulfil the desired condition.

    SYNTAX- DELETE FROM <TABLE NAME>WHERE <COLUMN NAME>= 'DATA VALUE';

    EX- DELETE FROM Student WHERE FirstName = 'Test';

12. **DELETE WITH (WHERE CLAUSE + AND)-** – It is a SQL statement used to Delete the data from the table which fulfil the desired condition (i.e AND Condition).

SYNTAX- DELETE FROM <TABLE NAME>WHERE <COLUMN NAME>= 'DATA VALUE' **AND** <COLUMN NAME>= 'DATA VALUE';
EX- DELETE FROM Student WHERE FirstName = 'Test' **AND** LastName = 'Demo';

---

13. **DELETE WITH (WHERE CLAUSE + OR)-** – It is a SQL statement used to Delete the data from the table which fulfil the desired condition(i.e OR Condition).

SYNTAX- DELETE FROM <TABLE NAME>WHERE <COLUMN NAME>= 'DATA VALUE' **OR** <COLUMN NAME>= 'DATA VALUE';
EX- DELETE FROM Student WHERE FirstName = 'Test' **OR** LastName = 'Demo';

---

14. **TRUNCATE-** It is a SQL statement used to deletes the data inside a table, **but not the table itself. (Column remains same/exist).**

SYNTAX- TRUNCATE TABLE <TABLE NAME>;
EX- TRUNCATE TABLE STUDENT;

---

15. **DROP TABLE–** It is SQL statement used to delete structure of table with all records.

SYNTAX- DROP TABLE <TABLE NAME>;
EX- DROP TABLE Student;

---

16. **ALTER-** This command use to adds, deletes, or modifies columns in a table.
ALTER ADD- This command use to add the columns in a table.

SYNTAX-

```
ALTER TABLE <TABLE NAME>
ADD <COLUMN_NAME> <DATATYPE> <DATASIZE>;
```

EX-

```
ALTER TABLE STUDENT
ADD AGE VARCHAR2(30);
```

17. **ALTER MODIFY-** This command use to **Modify the column Data Type** from the table.

SYNTAX-

```
ALTER TABLE <TABLE NAME>
MODIFY <COLUMN_NAME> <DATA TYPE> <SIZE>;
```

EX-

```
ALTER TABLE STUDENT
MODIFY RollNo VARCHAR2(20);

ALTER TABLE STUDENT
MODIFY RollNo INT;
```

---

18. **ALTER RENAME-** This command use to **RENAME the columns** from the table.

SYNTAX-

```
ALTER TABLE <TABLE NAME>
RENAME COLUMN <COLUMN_NAME> TO <COLUMN_NAME>;
```

EX-

```
ALTER TABLE STUDENT
RENAME COLUMN RollNo TO STUDENT_ROLLNO;
```

---

19. **UPDATE –** It is a SQL statement used to **update the data values of particular record** in the table.

SYNTAX- UPDATE <TABLE NAME> SET <COLUMN NAME> = 'DATA VALUE' WHERE <COLUMN NAME> = 'DATA VALUE';

EX : UPDATE STUDENT SET FNAME='TEST' WHERE FIRSTNAME='DEMO';

AGGREGATE FUNCTIONS

In database, an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

SQL allows different aggregate function

- **Sum()** == This fun() calculate the sum of the values from the mentioned column.
  Syntax - SELECT SUM <COLUMN_NAME> FROM <TABLE_NAME>;

- **Avg()** == This fun()calculate the average value from the mentioned column.
  Syntax- SELECT AVG <COLUMN_NAME> FROM <TABLE_NAME>;

- **Min()** == This fun() fetch the minimum value from the mentioned column.
  Syntax - SELECT MIN <COLUMN_NAME> FROM <TABLE_NAME>;

- **Max()** == This fun() fetch the maximum value from the mentioned column.
  Syntax- SELECT MAX <COLUMN_NAME> FROM <TABLE_NAME>;

- **Count()** == This function shows the count of values.
  SELECT COUNT <COLUMN_NAME> FROM <TABLE_NAME>;

ORDER BY

The ORDER BY keyword is used to **sort the result-set** in **ascending** or **descending** order. ORDER BY keyword sorts the records in **ascending order by default**. To sort the records in descending order, use the DESC keyword.

ASC == Sort result in Ascending order .

SYNTAX- SELECT * FROM <TABLE NAME > ORDER BY <COLUM NAME> ASC;

- SELECT * FROM EMPLOYEE ORDER BY ID ASC;
- SELECT * FROM EMPLOYEE ORDER BY FIRSTNAME ASC;

DESC == Sort result in Descending order.

SYNTAX- SELECT * FROM <TABLE NAME > ORDER BY <COLUM NAME> DESC;

- SELECT * FROM EMPLOYEE ORDER BY ID DESC;
- SELECT * FROM EMPLOYEE ORDER BY FIRSTNAME DESC;

OPERATORS

AND: TRUE if all the conditions separated by AND is TRUE
SELECT * FROM EMPLOYEE WHERE FIRST ID = ‘1’ AND SALARY = ‘25000’;

OR: TRUE if any of the conditions separated by OR is TRUE
SELECT * FROM EMPLOYEE WHERE FIRST ID = ‘1’ OR SALARY = ‘25000’;

BETWEEN: TRUE if the operand is within the range of comparisons
SELECT * FROM EMPLOYEE WHERE SALARY BETWEEN ‘35000’ AND ‘60000’;

IN: TRUE if the operand is equal to one of a list of expressions
SELECT * FROM EMPLOYEE WHERE ID IN (1,2,3,8);

SELECT * FROM EMPLOYEE WHERE FIRSTNAME in (‘RAM’, ‘KIRAN’, ‘RISHI’, ‘VIVEK’);

LIKE: TRUE if the operand matches a pattern
It is a Wildcard Characters used in SQL, A wildcard character is used to substitute one or more characters in a string.

LIKE 'a%' Finds any values that **starts** with "a"
LIKE '%a' Finds any values that **ends** with "a"
LIKE '%or%' Finds **any values that have "or" in any position**

SELECT * FROM EMPLOYEE WHERE FIRSTNAME LIKE ‘K%’;
SELECT * FROM EMPLOYEE WHERE FIRSTNAME LIKE ‘%K’;
SELECT * FROM EMPLOYEE WHERE FIRSTNAME LIKE ‘%K%’;

NOT: Displays a record if the condition(s) is NOT TRUE

SELECT * FROM EMPLOYEE
WHERE FIRSTNAME NOT LIKE '%AR%';

SELECT * FROM EMPLOYEE
WHERE FIRSTNAME NOT LIKE 'K%';

= : Equal to

Syntax - SELECT * FROM EMPLOYEE WHERE SALARY = 45000;

< : Less than-

Syntax - SELECT * FROM EMPLOYEE WHERE SALARY < 45000;

> : Greater than-

Syntax - SELECT * FROM EMPLOYEE WHERE SALARY > 45000;

>= : Greater than or equal to

Syntax - SELECT * FROM EMPLOYEE WHERE SALARY >= 45000;

<= : Less than or equal to

Syntax - SELECT * FROM EMPLOYEE WHERE SALARY <= 45000;

<>or != : Not equal to

Syntax -

SELECT * FROM EMPLOYEE WHERE SALARY <> 45000;

SELECT * FROM EMPLOYEE WHERE SALARY!= 45000;

SQL ALIAS

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

TIME SPAN - only exists for the duration of that query.

HOW TO USE – It is created with the **AS** keyword.

SYNTAX-

1. **FOR COLUMN –**

   SELECT <COLUMN NAME> **AS ALIAS_NAME** FROM <TABLE NAME>

   EX- SELECT SALARY AS EMP_SALARY FROM EMPLOYEE;

2. **FOR TABLE-**

   SELECT <COLUMN NAME> FROM <TABLE NAME> **AS ALIAS_NAME**

   EX- SELECT SALARY FROM EMPLOYEE AS ALIAS_NAME;

GROUP BY

GROUP BY clause is used with SELECT statement to **arrange identical data into groups.** This GROUP BY clause follows the WHERE clause in a SELECT statement.

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

Syntax-
SELECT A.F. (*) FROM <TABLE NAME> GROUP BY <COLUMN NAME>

Ex- Employee table

ID	FIRSTNAME	LASTNAME	DEPARTMENT	SALARY
1	RAM	MARATHE	ADMIN	25000
2	KIRAN	SHARMA	IT-HELP	12500
3	RISHI	KALE	IT-HELP	12500
4	KIRAN	PANDEY	QA	47500
5	VARSHA	PATIL	DEV	52000
6	RENUKA	KHATRI	QA	45000
7	PRITAM	VARMA	DEV	53000
8	SACHIN	TAMBE	DEV	25000
9	UMA	TRIVEDI	HR	37500
10	PRADNYA	PAWAR	HR	30000

Que 1: FIND COUNT OF EMPLOYEE WHICH ARE WORKING IN THE SAME DEPARTMENT ALONG WITH THE DEPARTMENT NAME? (EX- HR-2, Admin- 1, Dev -3 etc..)

ANS : SELECT COUNT(*) AS EMP_COUNT , department AS DEPT_NAME FROM employee GROUP BY DEPARTMENT;

O/P:

EMP_COUNT	DEPT_NAME
1	ADMIN
3	DEV
2	QA
2	HR
2	IT-HELP

Que 2: Find MINIMUM SALARY FROM EACH DEPARTMENT?

ANS: SELECT MIN(SALARY) AS MIN_SALARY, DEPARTMENT AS DEPT_NAME FROM EMPLOYEE GROUP BY DEPARTMENT;

O/P:

MIN_SALARY	DEPT_NAME
25000	ADMIN
25000	DEV
45000	QA
30000	HR
12500	IT-HELP

TASK - Que 3: Find MAXIMUM SALARY FROM EACH DEPARTMENT ?

TASK - Que 4: Find AVRAGE SALARY FROM EACH DEPARTMENT?

TASK - Que 5: Find SUM OF THE SALARY FROM EACH DEPARTMENT?

TASK - Que 6: Find MINIMUM SALARY WITH EMPLOYEE COUNT FROM HR DEPARTMENT?

TASK - Que 7: Find MAXIMUM SALARY WITH EMPLOYEE COUNT FROM DEV DEPARTMENT?

HINT FOR QUE- 6 : DESIERED O/P -

EMP_COUNT	E_MIN_SALARY	E_DEPT
2	30000	HR

HINT FOR QUE- 7 : DESIERED O/P -

EMP_COUNT	E_MAX_SALARY	E_DEPT
3	53000	DEV

DISTINCT

DISTINCT statement is used to return only distinct (different) values. Which means it **hide repeated or duplicate records** from table.

SYNTAX – SELECT DISTINCT<COLUMN NAME> FROM <TABLE NAME>;

EX- SELECT DISTINCT DEPARTMENT FROM EMPLOYEE;

SELECT DISTINCT FIRSTNAME FROM EMPLOYEE;

TASK: FIND COUNT OF NON-REPETATING DEPARTMENT FROM THE TABLE?

SQL CONSTRAINTS

CONSTRAINT: CONSTRAINTS ARE USED TO **SPECIFY RULES FOR DATA** IN A TABLE. CONSTRAINTS CAN BE USED **WHEN THE TABLE IS CREATED** WITH CREATE TABLE STATEMENT.

- **NOT NULL**
- **UNIQUE**
- **PRIMARY KEY**
- **FOREIGN KEY**
- **CHECK -** Ensures that the values in a column satisfies a specific condition
- **DEFAULT -** Sets a default value for a column if no value is specified
- **CREATE INDEX -** Used to create and retrieve data from the database very quickly

---

NOT NULL: Ensures that a column cannot have a NULL value.

SYNTAX - CREATE TABLE <TABLE NAME> (COLUMN1 DATATYPE(DATASIZE) NOT NULL, COLUMN2 DATATYPE(DATASIZE), COLUMN3 DATATYPE(DATASIZE) NOT NULL, ……);

EX- CREATE TABLE TEMP_TABLE (ID INT NOT NULL, F_NAME VARCHAR2(10), L_NAME VARCHAR2(10));

UNIQUE: Ensures that all values in a column are different.

SYNTAX- CREATE TABLE <TABLE NAME>(COLUMN1 DATATYPE(DATASIZE) UNIQUE, COLUMN2 DATATYPE(DATASIZE),……)

EX- CREATE TABLE TEMP_TABLE (ID INT UNIQUE, F_NAME VARCHAR2(10), L_NAME VARCHAR2(10));

PRIMARY KEY: WHICH CONTAINS UNIQUELY IDENTIFIES EACH RECORD IN A TABLE. **PRIMARY KEY MUST CONTAINS UNIQUE VALUE AND CAN NOT CONTAIN NULL VALUE. ALSO WE CAN SAY IT IS COMBINATION OF UNIQUE AND NOT NULL.**

SYNTAX- CREATE TABLE <TABLE NAME>(COLUMN1 DATATYPE(DATASIZE), COLUMN2 DATATYPE(DATASIZE), COLUMN3 DATATYPE(DATASIZE), PRIMARY KEY(COLUMN1));

Ex- CREATE TABLE TABLE_NAME(ID INT , FN VARCHAR2(20), LN VARCHAR2(20), PRIMARY KEY (ID)):

FOREIGN KEY: IT IS USED TO **LINK TWO TABLES TOGETHER**. FOREIGN KEY IS A FIELD IN **ONE TABLE THAT REFFERS 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.

IMP

- **Primary key cannot be null on the other hand foreign key can be null.**
- **Primary key is always unique while foreign key can be duplicated.**

SYNTAX-

CREATE TABLE <TABLE NAME> (COLUMN1 DATATYPE(DATASIZE), COLUMN2 DATATYPE(DATASIZE), COLUMN3 DATATYPE(DATASIZE), FOREIGN KEY(COLUMN1) REFERENCES TABLENAME1 (COLUMN1));

EX-

PRIMARY KEY-

CREATE TABLE TEMP_TB(ID INT , FN VARCHAR2(20), LN VARCHAR2(20), PRIMARY KEY (ID)):

FOREIGN KEY-

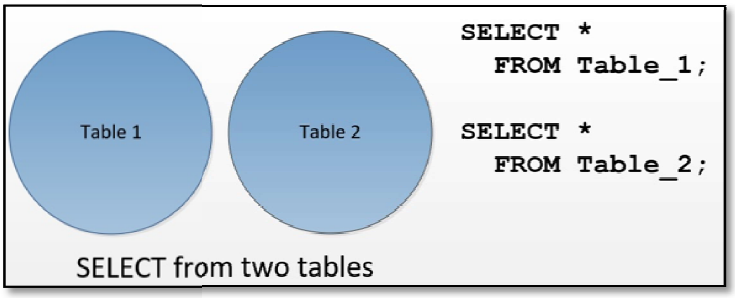
CREATE TABLE TEMP_TB1 (CO1 INT, CO2 INT, CO3 INT, FOREIGN KEY(CO1) REFERENCES <TABLE_NAME>(ID));

TASK- FIND OUT THE DIFFERENCE BETWEEN PRIMARY KEY AND FOREIGN KEY.

SQL- JOINS

As the name shows, JOIN means to combine something. In case of SQL, JOIN means **"to combine two or more tables"**.

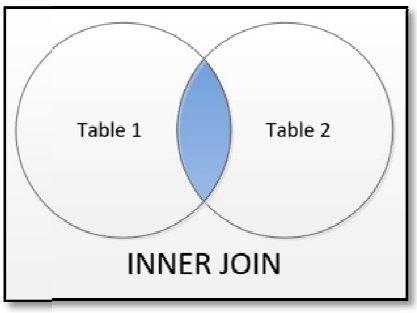
The SQL JOIN clause **takes records from two or more tables** in a database and **combines it together.**



TYPES OF JOINS USED IN SQL:

- **INNER JOIN**
- **RIGHT JOIN**
- **LEFT JOIN**
- **FULL JOIN**
- RIGHT OUTER JOIN
- LEFT OUTER JOIN
- FULL OUTER JOIN
- ANTISEMI JOIN

INNER JOIN: Inner Joins **Combine Records From Two Tables Whenever There Are Matching Values In A Field Common To Both Tables.**



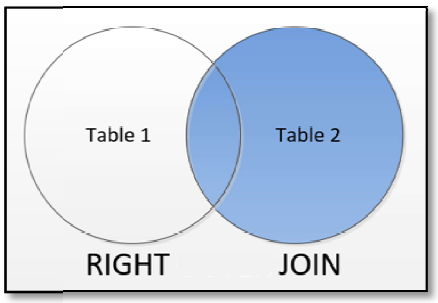
SYNTAX:

```
SELECT * FROM <TABLE_1>
INNER JOIN <TABLE_2>
ON TABLE_1.ID = TABLE_2.ID;
```

EX:

```
SELECT * FROM S_INFO
 INNER JOIN S_LOCATION
ON S_INFO.ID = S_LOCATION.ID;
```

RIGHT JOIN: Right join returns **all rows from the right table, and the matching rows from the left table. The result is NULL from the right side, if there is no match.**



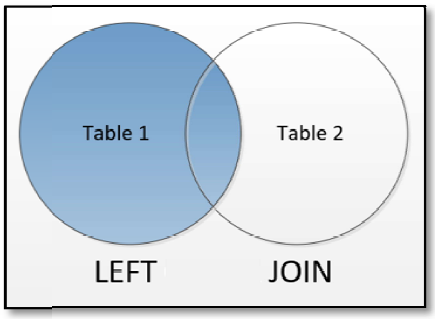
SYNTAX:

```
SELECT * FROM <TABLE_1>
RIGHT JOIN <TABLE_2>
ON TABLE_1.ID = TABLE_2.ID;
```

EX:

```
SELECT * FROM S_INFO
RIGHT JOIN S_LOCATION
ON S_INFO.ID = S_LOCATION.ID;
```

LEFT JOIN: Left join returns **all rows from the left table, and the matching rows from the right table. The result is NULL from the right side, if there is no match.**



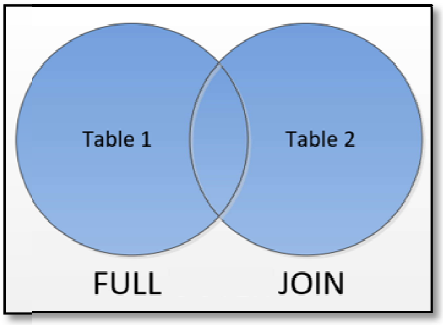
SYNTAX:

```
SELECT * FROM <TABLE_1>
LEFT JOIN <TABLE_2>
ON TABLE_1.ID = TABLE_2.ID;
```

EX:

```
SELECT * FROM S_INFO
LEFT JOIN S_LOCATION
ON S_INFO.ID = S_LOCATION.ID;
```

FULL JOIN: Full join returns all records from right and left table. **The result is NULL from the both table, if there is no match.**



SYNTAX:

```
SELECT * FROM <TABLE_1>
FULL JOIN <TABLE_2>
ON TABLE_1.ID = TABLE_2.ID;
```

EX:

```
SELECT * FROM S_INFO
FULL  JOIN S_LOCATION
ON S_INFO.ID = S_LOCATION.ID;
```

IMP:

SELECT 2ND MAX SALARY FROM TABLE-

SELECT MAX(SALARY) FROM EMPLOYEE WHERE SALARY NOT IN (SELECT MAX(SALARY) FROM EMPLOYEE);

SELECT 2ND MIN SALARY FROM TABLE-

SELECT MIN(SALARY) FROM EMPLOYEE WHERE SALARY NOT IN (SELECT MIN(SALARY) FROM EMPLOYEE);

SELECT Nth HIGHEST SALARY-

SELECT MIN(SALARY) FROM (SELECT DISTINCT SALARY FROM EMPLOYEE ORDER BY SALARY DESC) WHERE ROWNUM<=n;

EX- 3RD HIGHEST SALARY

SELECT MIN(SALARY) FROM (SELECT DISTINCT SALARY FROM EMPLOYEE ORDER BY SALARY DESC) WHERE ROWNUM<=3;

SELECT Nth LOWEST SALARY-

SELECT MAX(SALARY) FROM (SELECT DISTINCT SALARY FROM EMPLOYEE ORDER BY SALARY ASC) WHERE ROWNUM<=n;

EX- 3RD LOWST SALARY-

SELECT MAX(SALARY) FROM (SELECT DISTINCT SALARY FROM EMPLOYEE ORDER BY SALARY ASC) WHERE ROWNUM<=3;