Database Testing

SQL: Structure Query Language

- Definition: It is standard language to use fetch, store and Manipulate data in database.
- It is a language used to interact with the database, i.e. to create a database, to create a table in the database, to retrieve data or update a table in the database etc.

Types:

1) DDL (Data Definition Language)
2) DML (Data Manipulation language)
3) DCL (Data control language)
4) DQL (Data query Language)

For Testing require only two

1) DDL
-Create DB/Table, Alter DB/Table, Drop DB/Table, Truncate DB/Table

-In One Data base their will be multiple table

-In one Table there will be multiple row and columns

2) DML
-select, update, delete, insert into

Note:

1) There are multiple Databases like MySQL, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres, and other database systems.

But for Study & Testing we are going to use Oracle Developer.

2) SQL is not case sensitive means we can use capital as well as Small alphabets.

Steps for Download and Install Oracle Developer

1) Go to site for download Oracle Developer Application for

   A) Windows 64-bit Operating System: https://www.filehorse.com/download-oracle-databaseexpress-64/download/

Windows 32-bit Operating System: https://www.filehorse.com/download-oracle-databaseexpress-32/download/

2) Click on Start Download Button then One popup will open A) Oracle Account Sign In(If you already have account) B) Create New Account (If you don't have account) - Add Any Company Name (for ex. Shree Classes)

3) Once You provided all personal detail and clicked on Sign UP button then Mail will sent on your mail Id for Verification

4) Once Verification done then Sign In by using User name and password provide by you at the time Signup

5) Once you clicked on Sign In button then Automatically downloading will start below left site corner

6) For Downloading it will take some time. And once download completed go to download folder in File

7) Right click on file and Select Extract Files then SQL file will unzip

8) Then click on Set-up .exe file then it will be start Installing (It will ask click on checkbox for accept term and Condition so click on that checkbox)

9) Once Installation Finish then You can see Oracle Developer Icon on your Desktop then Restart laptop

Note: after Download and Installation completed then we need to create workspace for writing SQL Queries.

Steps for Creating work space

Open oracle developer from desktop

Click on Application Express

- Login Page will be open then enter username and password which was entered at the time installation
- Then create workspace window will be open
- Add required information and click on Create workspace
- Then click on I already have account Login here
- Then enter username and password which was created at the time of workspace creation and login
- Click on SQL workshop

Click on SQL command then workspace will be open then start write command

Commands/ SQL Query/SQL Statements/SQL Syntax:

1) Create: For Creating Table

Syntax:

CREATE TABLE table_name (column1 datatype (datatype value), column2 datatype (dataType value), column3 datatype (dataType value));

Example:

CREATE TABLE Shree_Classes (First_Name varchar (50), Last_Name varchar (50), Mobile_Number int, City varchar (20));

Note:
For Integer Value we do not need pass dataType Value

Output: Table will be created but need to add data in table by using insert into command.

Insert into: When we want to enter data in table then we use this

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

Syntax:

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

INSERT INTO Shree_Classes (First_Name, Last_Name, Mobile_Number, City) VALUES ('Ganesh', Raut', '1234567809', 'Solapur')

Note: We can add multiple rows through above commands just change the Columns Values

INSERT INTO Shree_Classes (First_Name, Last_Name, Mobile_Number, City) VALUES (‘Sachin’, ‘Sawant’, '1234567471', ‘Nashik’)

Way: 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.

Syntax:

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

INSERT INTO Shree_Classes
VALUES ('Vishal', 'Mohite', '1234578988', 'Pune');

Select: It is use to fetch data from table by using different combination of commands
For Fetching Specific column Data use below command

Syntax:
SELECT column_Name FROM table_Name

Example:
SELECT First_Name FROM Shree_Classes;

For fetching all table Data use below command

Syntax:
SELECT * FROM table_Name;

Example: SELECT * FROM Shree_Classes;

Distinct: When we don't want duplicate values then we use this command

Syntax:
SELECT DISTINCT column_Name FROM table_Name;

Example:
SELECT DISTINCT First_Name FROM Shree_Classes;

Where: The WHERE clause is used to filter records

.

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

Syntax:

SELECT column_Name FROM table_Name WHERE column_Name='Value';

Example:

SELECT First_Name FROM Shree _Classes WHERE City='Nashik';

Note: For Unique records (Duplicate not allowed) we can use below command

SELECT DISTINCT First_Name FROM Shree_Classes WHERE City='Nashik';

AND, OR

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.

Syntax: For AND

SELECT column_Name FROM table_Name

WHERE column_Name='Value' AND column_Name='Value';

Example: If both condition true then you will get output

SELECT City FROM Shree_Classes

WHERE First_Name='Vishal' AND Last_Name='Mohite';

Example:

If one condition true and one false then you will not get output

SELECT City FROM Shree_Classes WHERE First_Name='Vishal' AND Last_Name='Tendulkar';

Syntax for OR:

SELECT column_Name FROM table_Name

WHERE column_Name='Value' OR column_Name='Value';

Example: IF both condition true then you get output

SELECT City FROM Shree_Classes WHERE First_Name='Vishal' OR Last_Name='Mohite';

Example: IF one condition true and one false then also you get output

SELECT City FROM Shree_Classes WHERE First_Name='Vishal' AND Last_Name='Tendulkar';

Example: IF both condition false then you will not get output

SELECT City FROM Shree_Classes WHERE First_Name='Neha' AND Last_Name='Tendulkar';

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.

For Ascending Order: No need to write ASC Keyword

Syntax:
SELECT column_Name FROM table_name ORDER BY column_Name;

Example:
SELECT First_Name FROM Shree_Classes ORDER BY First_Name;

For Descending Order: Need to add DESC Keyword

Syntax:
SELECT column_Name FROM table_name ORDER BY column_Name DESC;
Example:

SELECT First_Name FROM Shree_Classes ORDER BY First_Name DESC;

Update:

Use for modifying existing records
Syntax:
UPDATE table_Name SET column_Name = 'Value'
WHERE column_Name='Value';

Example:
UPDATE Shree_Classes SET First_Name = 'Neha' WHERE City='Nashik';

Delete:

The DELETE statement is used to delete existing records in tables

Note: It is DML Command. We can retrieve / Recover data deleted by Delete Command

Below Command for deleting Specific row of the table

Syntax: DELETE FROM table_Name WHERE column_Name='Value';

Example:
DELETE FROM Shree_Classes WHERE First_Name='Vishal';

Below Command for deleting full table

Syntax: DELETE table_Name;
Example: DELETE Shree_Classes;

LIKE:

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

1) Ending with Specific Alphabets: (%Alphabet)

Syntax:
SELECT column_Name FROM table_Name WHERE column_Name LIKE '% Alphabet';

Example: For City Nashik

SELECT First_Name FROM Shree_Classes
WHERE City LIKE '%hik';

2) Start with Specific Alphabets (Alphabet %)

Syntax:
SELECT column_Name FROM table_Name WHERE column_Name LIKE 'Alphabet %';
Example:
SELECT First_Name FROM Shree_Classes
WHERE City LIKE 'Nas%';

3) By using underscore Sign (_)

Here we can substitute one or more than one Character in the particular stream to find out the data.
Syntax:
SELECT column_Name FROM table_Name WHERE column_Name LIKE '___Alphabet';
Example: 1
SELECT First_Name FROM Shree_Classes
WHERE City LIKE '__shik';

Example: 2
SELECT First_Name FROM Shree_Classes
WHERE City LIKE '__sh__';

Example: 3
SELECT First_Name FROM Shree_Classes
WHERE City LIKE 'Na____';

IN Operator

It is SQL statement use to select those multiple values/records which specify in queries.

Syntax:
SELECT column_Name FROM table_name WHERE column_Name IN ('value1', 'value2' ...);

Example:
SELECT Mobile_Number FROM Shree_Classes WHERE First_Name IN ('Vishal, 'Sachin' ...);

BETWEEN

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.

Syntax:
SELECT column_Name FROM table_Name WHERE column_Name BETWEEN value1 AND value2;

Example:
SELECT First_Name FROM Shree_Classes
WHERE Salary BETWEEN 20000 AND 80000;

Alter Table

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.

A) ADD:

For Adding New Column in Table

Syntax:
ALTER TABLE table_Name ADD column_Name datatype;

Example1:
ALTER TABLE Shree_Classes
ADD Salary int;

Example2:
ALTER TABLE Shree_Classes
ADD Company_Name Varchar (50);

B) DROP:

For Deleting Existing Column from Table

Syntax:
ALTER TABLE table_Name DROP COLUMN column_Name;

Example:
ALTER TABLE ShreeClasses
DROP COLUMN City;

Drop/Delete Full Table

Syntax: DROP TABLE table_Name;

Example: DROP TABLE Shree_Classes;

Note: We cannot retrieve/Recover data deleted/dropped by Drop DDL Command

14) TRUNCATE TABLE

>It is also a **Data Definition Language** Command (DDL).

>It is use to delete all the rows of a relation (table) in one go. With the help of"TRUNCATE" command

>we can't delete the single row as here **WHERE** clause is not used.

>By using this command the existence of all the rows of the table is lost.

Syntax: TRUNCATE TABLE table_Name*;*

Example: TRUNCATE TABLE Shree_Classes*;*

Note – Here we **can't restore** data. Only **Schema of the table** remains. Schema of table means only column names.

15) Command for **Retrieve/Recover** Data Deleted by **Delete Command**

Syntax: SELECT * from table_Name

AS OF TIMESTAMP(SYSTIMESTAMP-INTERVAL 'Time_Value' MINUTE)

Example;

SELECT * from Shree_Classes

AS OF TIMESTAMP(SYSTIMESTAMP-INTERVAL '**10'** MINUTE)

Note: By using above command we can **only see Deleted data** but this **data not save**. For saving this data need to **create new Table** and add this data in new Table.

Syntax:

CREATE TABLE new_Table_Name ASSELECT * from old_Table_Name
AS OF TIMESTAMP(SYSTIMESTAMP-INTERVAL 'Time_Value' MINUTE)

Example:

CREATE TABLE Shree_Classes2 ASSELECT * from Shree_Classes
AS OF TIMESTAMP (SYSTIMESTAMP-INTERVAL '30' MINUTE)

Note: IF you want to recover old Date Data then use below CommandSyntax:
INSERT INTO new_Table_Name

```
(SELECT * FROM old_Table_Name AS OF

TIMESTAMP TO_TIMESTAMP('2021-09-08 06:41:59', 'YYYY-MM-DD HH:MI:SS'));
```

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**.

The following constraints are commonly used in SQL:

1. **NULL**-We can keep that column values **Empty/Blank**
2. **NOT NULL** - Ensures that a column cannot have a NULL value
3. **UNIQUE** - Ensures that all values in a column are different
4. **PRIMARY KEY** - A combination of a **NOT NULL** and **UNIQUE**. Uniquely identifies each row in a table
5. **FOREIGN KEY** - Prevents actions that would destroy links between tables
6. **CHECK** - Ensures that the values in a column satisfies a specific condition

1. NULL: We can keep that column values **Empty/Blank**

Syntax:

CREATE TABLE table_Name

(Column1_Name dataType Constraint, Column2_Name dataType Constraint, Column3_Name dataType Constraint);

Example:

CREATE TABLE Shree_Classes

(First_Name varchar (50) NOT NULL, Last_Name Varchar (50) NOT NULL, City varchar
(50) NULL);

Note: In above example for City Column we can keep value as blank

2. NOT NULL

When we apply this constrains to specific column then we cannot keep those column Empty.We need provide value for this.

Syntax:

CREATE TABLE table_Name

(Column1_Name dataType Constraint, Column2_Name dataType Constraint, Column3_Name dataType Constraint);

Example:

CREATE TABLE Shree_Classes

(First_Name varchar (50) NOT NULL, Last_Name Varchar (50) NOT NULL, City varchar

(50) NULL);

Note: In above example for **First_Name** and **Last_Name** column values we **cannot keep** as blank because we applied **NOT NULL** Constraint for that column

3. UNIQUE:

>It is SQL Constraint which ensures that all the values in a column are UNIQUE/Different.

>Means duplicate values cannot allows in specified Column

Syntax:

CREATE TABLE table_Name

(Column1_Name dataType Constraint, Column2_Name dataType Constraint, Column3_Name dataType Constraint);

Example:

CREATE TABLE Shree_Classes

(ID int NOT NULL UNIQUE, First_Name varchar (50) NOT NULL, Last_Name Varchar (50) NOT NULL, City varchar (50) NULL);

Note:

1) In above example we applied **UNIQUE** Constraint for **ID Column** means we cannot enter **Duplicate ID numbe**r in that column.

2) If you tried to add duplicate ID means **already ID=1** is present in column and you again

tried to **add ID=1 for other row** then Getting below Error and action aborted

Error: Unique Constraint violated

4) PRIMARY KEY

The **PRIMARY KEY** constraint uniquely identifies each record in a table.

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

A table can have **only ONE primary key.**

Syntax:

```
CREATE TABLE table_Name

(Column1_Name dataType Constraint, Column2_Name dataType
Constraint,Column3_Name dataType Constraint);
```

Example:

```
CREATE TABLE
  Shree_Classes (ID int NOT
  NULL,
  Last_Name varchar (50)
  NOT NULL,First_Name
  varchar (50),
  Age int,
  PRIMARY KEY (ID)
);
```

Note: In above example for ID **Primary Key Constraint** applied so we cannot enter **Duplicate** and **Null Values** for ID column

FOREIGN KEY

The **FOREIGN KEY** constraint is used to prevent actions that would destroy links betweentables.

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.**

It is use to link two tables together. It is allow **duplicate** value

Syntax:

CREATE TABLE table_Name

(Column1_Name dataType Constraint, Column2_Name dataType Constraint, Column3_Name dataType Constraint, FOREIGN KEY (common_Column_Name) REFERENCES table_Name (common_Column_Name));

Note: Above query we give REFERENCE of Primary Key from Parent Table

Example:

```
CREATE TABLE Shree_Classes (
ID int NOT NULL,  First_Name varchar (50) NOT NULL,
Age int, FOREIGN KEY (ID) REFERENCES Shree_Classes1 (ID));
```

5) Check

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 thiscolumn.

Syntax:

CREATE TABLE table_Name

(Column1_Name dataType Constraint, Column2_Name dataType Constraint, Column3_Name dataType Constraint);

Example:

```
CREATE TABLE Shree_Classes (ID int NOT NULL,
Last_Name varchar (50) NOT NULL,First_Name varchar (50),
Age int,
CHECK (Age>=18)

);
```

Note: In above example we applied **Check Constraint** so if we tried to add **value less than 18** then below error will generated and action get aborted.

//Error: ORA-02290: check constraint (Shree.SYS_C00786715) violated

17) AGGREGATE FUNCTIONS

1) MIN: The **MIN ()** function returns the **smallest value of** the selected column.

Syntax:

```
SELECT MIN (column_Name)FROM table_Name;
```

Example:

```
SELECT MIN (Age)
FROM Shree_Classes;
```

2) MAX: The **MAX ()** function returns the largest value of the selected column.

Syntax:

```
SELECT MAX (column_Name)FROM table_name;
```

Example:

```
SELECT MAX (Age)
FROM Shree_Classes;
```

Mostly Asked Interview Question: How to find **2nd Highest** Salary/Age from column

Syntax 1: with **NOT IN** Function

```
SELECT MAX (column_Name) FROM table_Name WHERE

Column_Name NOT IN (SELECT MAX (column_Name) FROM table_Name));
```

Example 1: with **NOT IN** Function

```
SELECT MAX (Salary) FROM Shree_Classes WHERE
Salary NOT IN (SELECT MAX (Salary) FROM Shree_Classes));
```

Syntax: 2: with **Less than** Operator **(<)**

```
SELECT MAX (column_Name) FROM table_Name WHERE
Column_Name < (SELECT MAX (column_Name) FROM table_Name));
```

Example 2: with **Less than** Operator (<)

```
SELECT MAX (Salary) FROM Shree_Classes WHERE
Salary < (SELECT MAX (Salary) FROM Shree_Classes));
```

Note: In above Syntax 1st execute **inner query** which find highest salary then (<) less thanoperator find 2nd highest salary by executing **Outer Query**.

Example 3: Using below Syntax

```
SELECT MIN (Salary) FROM (SELECT Salary FROM employee ORDER BY Salary DESC) WHERE rownum<=2;
```

Note: By using above command we can find out any Salary 1st Highest, 2nd Highest, 3rdHighest just we need to change **rownum<=value**

Note: In above example we find out **2nd highest Salary** that's why added **rownum<=2** if we want **1st highest** or **3rd highest** then just add **rownum<= 1** or **3** respectively.

3) COUNT ():

The **COUNT ()** function returns the **number of rows** that matches a specified criterion.

Syntax:

```
SELECT COUNT (column_Name)
FROM table_name;
```

Example: SELECT COUNT (First_Name)

FROM Shree_Classes;

4) AVG ()

The **AVG ()** function returns the average value of a numeric column.

Syntax:

SELECT AVG (column_Name)FROM table_name;

Example:

```
SELECT AVG (Age)
FROM Shree_Classes;
```

5) SUM ()

Syntax:

SELECT SUM

(column_Name) FROM

table_name*;*

Example:

```
SELECT SUM (Age)
FROM Shree_Classes;
```

18) GROUP BY

In SQL, The Group By statement is used for organizing similar

data into groups. The data is further organized with the help of

Aggregate Functions

It means, if different rows in a precise column have the same values, it will arrange those rows in a group

>>**Note:** You must remember that **Group By** clause will always come at the end of the SQL

query, just like the **Order by** clause.

-The **SELECT** statement is used with the **GROUP BY** clause in the SQL query.

-**WHERE** clause is placed **before** the **GROUP BY** clause in SQL.

-**ORDER BY** clause is placed **after** the **GROUP BY** clause in SQL.

- **GROUP BY** Clause only return **UNIQUE Value** if column have **duplicate Value.**

Syntax:

```
SELECT column_Name FROM  table_name WHERE  condition GROUP BY
column_Name
ORDER BY column_Name;
```

Example: **1**: **Group by** Multiple Column with **Where Clause** and **Order BY Clause**

```
SELECT Age, First_Name FROM Shree_ClassesWHERE Age> 50
GROUP BY Age, First_NameORDER BY First_Name;
```

Example: 2: Group BY Clause for **Single Column** with Aggregate Function **Count**

```
SELECT Age, COUNT (First_Name)

FROM Shree_ClassesGROUP BY Age;
```

18) HAVING CLAUSE

Having clause is used with SQL Queries to give more precise condition for a statement. It is used to mention condition in Group by based SQL queries, just like WHERE clause is used with SELECT query

Syntax:

```
SELECT column_Name FROM  table_Name GROUP BY column_NameHAVING
condition ORDER BY column_Name;
```

Example: **1 Having Clause** for Multiple Columns with **GROUP BY** and **ORDER BY**

Clause

```
SELECT Salary, First_NameFROM  Shree_Classes
GROUP BY Salary, First_NameHAVING Salary>50000
ORDER BY First_Name;
```

Example: 2 Having Clause with Aggregate Function **SUM**

```
SELECT SUM (Age)

FROM Shree_ClassesGROUP BY Age
HAVING SUM (Age)>25;
```

Note: **Having Clause** was added to **SQL** because the **WHERE** Keyword **could not be used** with **Aggregate Function.**

18) UNION & UNION ALL

1) UNION:

>It is used to combine the result set of **two or more** SELECT statement.

>The Column must also have **similar dataType.**

>We can combine two different table column data in one column.

>It will fetch only unique records. It not allow duplicate values.

Syntax:

```
SELECT column_Name FROM table1UNION
SELECT column_Name FROM table2;
```

Example:

```
SELECT First_Name FROM Shree_ClassesUNION
SELECT First_Name FROM Shree_Classes2;
```

2) UNION ALL

>It will fetch all duplicate, unique records

Syntax:

```
SELECT column_Name FROM table1UNION
SELECT column_Name FROM table2;
```

Example:

```
SELECT First_Name FROM Shree_ClassesUNION ALL
SELECT First_Name FROM Shree_Classes2;
```

19) SQL JOINS

>JOIN is SQL Statements use to join two or more tables based on **Primary and Foreign Key.**

>A **JOIN** clause is used to combine rows from two or more tables, **based on a related column** between them.

Here we first crate two new table and then perform join functions;

1) Create new Employee Table with Primary Key (Parent Table)

```
CREATE table Employee (Emp_ID int, Emp_First_Name varchar (50), Emp_Last_Name varchar (50), Age int, EmailID varchar (50), Mobile_NO int, Address varchar (50), PRIMARY KEY (Emp_ID));
INSERT INTO Employee values (1, 'Nikhil', 'Jadhav', 23, 'nikhil@abc.com', 9863253524, 'Pune');
INSERT INTO Employee values (2, 'Nilesh', 'Mohite', 22, 'nilesh@abc.com', 9863253569, 'Mumbai');
INSERT INTO Employee values (3, 'Sachin', 'Sakpal', 25, 'sachin@abc.com', 9963253564, 'Delhi');
INSERT INTO Employee values (4, 'Nitin', 'Shinde', 29, 'nitin@abc.com', 9999253564, 'Bengaluru');
INSERT INTO Employee values (5, 'Kedar', 'Jadhav', 34, 'kedar@abc.com', 9977253564, 'Kolkata');
SELECT * from Employee;
```

2) Create new Project Table with Foreign Key (Child Table)

```
CREATE table Project (Project_ID int, Emp_ID int, Client_ID int, Project_Name varchar (50), Project_Start_Date int, FOREIGN KEY (Emp_ID) REFERENCES Employee (Emp_ID));
INSERT INTO Project values (111, 1, 3, 'Project1', 2020-01-01);

INSERT INTO Project values (222, 2, 1, 'Project2', 2020-02-02);

INSERT INTO Project values (333, 3, 5, 'Project3', 2020-03-03);

INSERT INTO Project values (444, 3, 2, 'Project4', 2020-04-04);

INSERT INTO Project values (555, 5, 4, 'Project5', 2020-05-05);
INSERT INTO Project values (555, 7, 4, 'Project5', 2020-05-05);
```

Note: In above syntax I tried to add Emp_ID=7 but getting below Error because this table isChild Table which referencing Parent Table where Emp_ID=7 not present so that's why Child table Emp_ID column not taking Emp_ID=7.

ORA-02291: integrity constraint (Shree.SYS_C007164) violated - parent key not found

INSERT INTO Project values (666, 5, 1, 'Project6', 2020-06-06);
INSERT INTO Project values (777, 5, 2, 'Project7', 2020-06-06);SELECT * from Project;

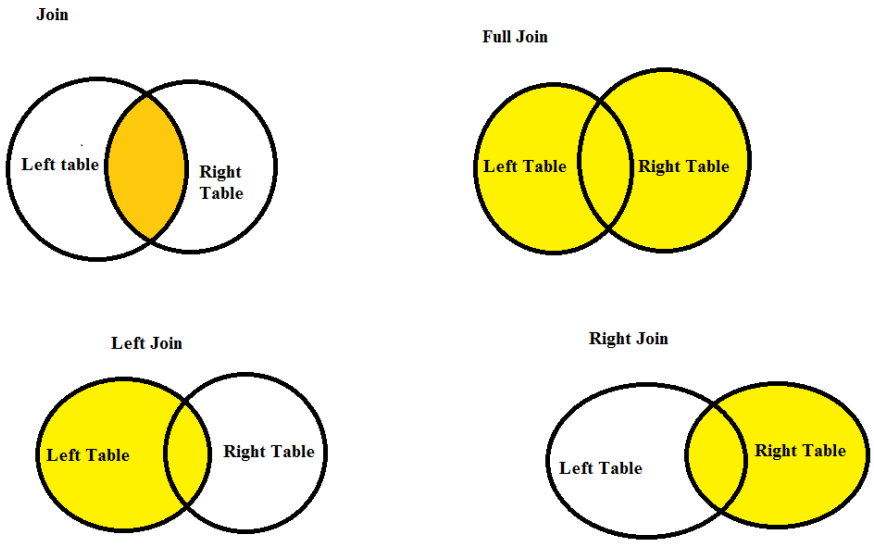
Types of SQL JOIN:

1) INNER JOIN or JOIN

2) LEFT JOIN

3) RIGHT JOIN

4) FULL JOIN



1) INNER JOIN or JOIN:

This type of join returns those records which have matching values in both tables. So, if you perform an INNER join operation between the Employee table and the Projects table, all the tuples which have

matching values in both the tables will be given as output.

Syntax:

```
SELECT Table1.Column1, Table1.Column2, Table2.Column1...FROM Table1INNER
JOIN Table2
ON Table1.MatchingColumnName = Table2.MatchingColumnName;
```

Example:

```
SELECT Employee. Emp_ID, Employee. Emp_First_Name, Employee. Emp_Last_Name,
Project. Project_ID, Project. Project_NameFROM Employee INNER JOIN Project ON
Employee.Emp_ID=Project.Emp_ID;
```

LEFT JOIN or LEFT OUTER JOIN

The **LEFT JOIN** or the **LEFT OUTER JOIN** returns all the records from the left table and also those records which satisfy a condition from the right table. Also, for the records having no matching values in the right table, the output or the result-set will contain the NULL values.

Syntax:

```
SELECT Table1.Column1, Table1.Column2, Table2.Column1...FROM Table1LEFT JOIN Table2
ON Table1.MatchingColumnName = Table2.MatchingColumnName;
```

Example:

```
SELECT Employee.Emp_ID, Employee. Emp_First_Name, Employee. Emp_Last_Name,
Project. Project_ID, Project. Project_NameFROM Employee LEFT JOIN Project
ON Employee.Emp_ID=Project.Emp_ID;
```

RIGHT JOIN or RIGHT OUTER JOIN

The RIGHT JOIN or the RIGHT OUTER JOIN returns all the records from the right table and also those records which satisfy a condition from the left table. Also, for the records having no matching values in the left table, the output or the result-set will **contain the NULL values.**

Syntax:

```
SELECT Table1.Column1, Table1.Column2, Table2.Column1...FROM Table1RIGHT JOIN Table2
ON Table1.MatchingColumnName = Table2.MatchingColumnName;
```

Example:

```
SELECT Employee. Emp_First_Name, Employee. Emp_Last_Name, Project. Project_ID,
Project. Project_Name

FROM Employee RIGHT JOIN Project ON Employee.Emp_ID=Project.Emp_ID
```

FULL JOIN or FULL OUTER JOIN

Full Join or the **Full Outer Join** returns all those records which either have a match in theleft (Table1) or the right (Table2) table.

Syntax:

```
SELECT Table1.Column1, Table1.Column2, Table2.Column1...FROM Table1
FULL JOIN Table2
ON Table1.MatchingColumnName = Table2.MatchingColumnName;
```

Example:

```
SELECT Employee. Emp_First_Name, Employee. Emp_Last_Name, Project. Project_ID FROM
Employee FULL JOIN Project
ON Employee.Emp_ID=Project.Emp_ID;
```

SQL INTERVIEW QUESTIONS

1) What is SQL?

SQL stands for Structured Query Language.

It is a language used to interact with the database, i.e to create a database, to create a table inthe database, to retrieve data or update a table in the database etc.

SQL is an ANSI (**American National Standards Institute**) standard. Using SQL, we cando many things,

For example –

>we can execute queries, we can insert records in a table,

>we can update records, we can create a database, we can create a table, we can delete a tableetc.

2) What is a Database?

A Database is defined as a structured form of data which is stored in a computer or data in anorganised manner and can be accessed in various ways.

It is also the **collection** of **schemas**, **tables**, **queries**, **views** etc. Database helps us in easilystoring, accessing and manipulation of data held in a computer.

The **Database Management System** allows a user to interact with the database.

3) What are the different subsets of SQL? Or Types SQL

- Data Definition Language (DDL) – It allows you to perform various operations on thedatabase such as **CREATE**, **ALTER**, and **DROP** objects.
- Data Manipulation Language (DML) – It allows you to access and manipulate data. Ithelps you to **insert**, **update**, **delete** and retrieve data from the database.
- Data Control Language (DCL) – It allows you to control access to the database.

1) Join in SQL What are joins in SQL?

A **JOIN** clause is used to combine rows from two or more tables, based on a related column between them. It is used to merge two tables or retrieve data from there. There are 4 types of joins, as you can refer to below:

- **Inner join:** Inner is the most common type of join. It is used to return all the rows from multiple tables where the join condition is satisfied.

- **Left Join:** Left Join in SQL is used to return all the rows from the left table but only the matching rows from the right table where the join condition is fulfilled.
- **Right Join:** Right Join in SQL is used to return all the rows from the right table but only the matching rows from the left table where the join condition is fulfilled.
- **Full Join:** Full join returns all the records when there is a match in any of the tables. Therefore, it returns all the rows from the left-hand side table and all the rows from the right-hand side table.

2) What are Constraints?

Constraints in SQL are used to specify the limit on the data type of the table. It can be specified while creating or altering the table statement. The sample of constraints is:

- NOT NULL
- CHECK
- UNIQUE
- PRIMARY KEY
- FOREIGN KE

3) What is the difference between DELETE and TRUNCATE statements?

DELETE	TRUNCATE
1) Delete command is used to delete a row in a table.	**1)** Truncate is used to delete all the rows from a table.
2) You can rollback data after using delete statement.	**2)** You cannot rollback data.

3) It is a DML command.	**3)** It is a DDL command.
4) It is slower than truncate statement.	**4)** It is faster.

1) What is a Foreign key in SQL?

- Foreign key maintains referential integrity by enforcing a link between the data in twotables.
- The foreign key in the child table references the primary key in the parent table.
- The foreign key constraint prevents actions that would destroy links between the childand parent tables.

2) What is the difference between primary key and unique constraints?

Primary key **cannot have NULL value;** the Unique constraints **can have NULL** values.There is **only one primary key** in a **table**, but there can be **multiple unique constrains**.

3) What is the difference between DROP and TRUNCATE commands?

DROP command **removes a table** and it cannot be rolled back from the database whereasTRUNCATE command removes all the rows from the table.

4) Are NULL values same as that of zero or a blank space?

A NULL value is not at all same as that of zero or a blank space. NULL value represents a value which is unavailable, unknown, assigned or not applicable whereas a zero is a number and blank space is a character.

5) What is the main difference between 'BETWEEN' and 'IN' conditionoperators?

BETWEEN operator is used to display rows based on a range of values in a row whereas the IN condition operator is used to check for values contained in a specific set of values.

Example of BETWEEN:

```
SELECT * FROM Students where ROLL_NO BETWEEN 10 AND 50;
```

Example of IN:

SELECT * FROM students where ROLL_NO IN (8, 15, 25);

What is the difference between 'HAVING' CLAUSE and a 'WHERE'CLAUSE?

>HAVING clause can be used only with SELECT statement. It is usually used in a GROUPBY clause and whenever GROUP BY is not used, HAVING behaves like a WHERE clause.

>Having Clause is only used with the GROUP BY function in a query whereas WHEREClause is applied to each row before they are a part of the GROUP BY function in a query.

How to change Table Name?

```
ALTER TABLE Employee

RENAME TO Employees_Details;
```

Output: Table Name will alter as an Employees_Details