**SQL**

SQL stands for Structured Query Language

1. Basic SQL Syntax:

* SQL keywords
* Data types
* Operators
* SQL statements (SELECT, INSERT, UPDATE, DELETE)

1. Data Definition Language (DDL):

* CREATE TABLE
* ALTER TABLE
* DROP TABLE
* Truncate table

1. Data Manipulation Language (DML):

* SELECT statement (SELECT, FROM, WHERE, ORDER BY, GROUP BY, HAVING, JOINs)
* INSERT statement
* UPDATE statement
* DELETE statement

1. Aggregate Functions:

* SUM, AVG, COUNT, MIN, MAX
* GROUP BY clause
* HAVING clause

1. Data Constraints:

* Primary Key
* Foreign Key
* Unique
* NOT NULL
* CHECK

1. Joins:

* INNER JOIN
* LEFT JOIN
* RIGHT JOIN
* FULL OUTER JOIN
* Self Join
* Cross Join

1. Subqueries:

* Types of subqueries (scalar, column, row, table)
* Nested subqueries
* Correlated subqueries

1. Advanced SQL Functions:

* String functions (CONCAT, LENGTH, SUBSTRING, REPLACE, UPPER, LOWER)
* Date and time functions (DATE, TIME, TIMESTAMP, DATEPART, DATEADD)
* Numeric functions (ROUND, CEILING, FLOOR, ABS, MOD)
* Conditional functions (CASE, COALESCE, NULLIF)

1. Views:

* Creating views
* Modifying views
* Dropping views

1. Indexes:

* Creating indexes
* Using indexes for query optimization

1. Transactions:

* ACID properties
* Transaction management (BEGIN, COMMIT, ROLLBACK, SAVEPOINT)
* Transaction isolation levels

1. Data Integrity and Security:

* Data integrity constraints (referential integrity, entity integrity
* GRANT and REVOKE statements (granting and revoking permissions)
* Database security best practices

1. Stored Procedures and Functions:

* Creating stored procedures
* Executing stored procedures
* Creating functions
* Using functions in queries

1. Performance Optimization:

* Query optimization techniques (using indexes, optimizing joins, reducing subqueries)
* Performance tuning best practices

1. Advanced SQL Concepts:

* Recursive queries
* Pivot and unpivot operations
* Window functions (Row\_number, rank, dense\_rank, lead & lag)
* CTEs (Common Table Expressions)
* Dynamic SQL

# SQL Commands:

1. SELECT - extracts data from a database
2. UPDATE - updates data in a database
3. DELETE - deletes data from a database
4. INSERT INTO - inserts new data into a database
5. CREATE DATABASE - creates a new database
6. ALTER DATABASE - modifies a database
7. CREATE TABLE - creates a new table
8. ALTER TABLE - modifies a table
9. DROP TABLE - deletes a table
10. CREATE INDEX - creates an index (search key)
11. DROP INDEX - deletes an index

**Test table**:



1. **CREATE:** The CREATE Keyword is used to create a database, table, views, and index. We can create the table CUSTOMER as below.

create table CUSTOMER (

id INT PRIMARY KEY,

name VARCHAR(50),

state VARCHAR(20) );

1. **PRIMARY KEY:** This keyword uniquely identifies each of the records.

create DATABASE User;

1. **INSERT:** The INSERT Keyword is used to insert the rows of data into a table. We can insert the rows below to the already created CUSTOMER table using the queries below.

insert into CUSTOMER values (121,'Rajesh','Maharashtra');

insert into CUSTOMER values (256,'Leela','Punjab');

## SELECT Statement:

SELECT column1, column2 FROM table\_name;

Example:

1. SELECT CustomerName, City FROM Customers;
2. SELECT \* FROM Customers;

## SELECT DISTINCT Statement:

SELECT DISTINCT column1, column2 FROM table\_name;

Example:

1. SELECT DISTINCT Country FROM Customers;
2. SELECT COUNT (DISTINCT Country) FROM Customers;
3. SELECT Count (\*) AS DistinctCountries FROM ( SELECT DISTINCT Country FROM Customers );

## WHERE Clause

SELECT column1, column2 FROM table\_name WHERE condition;

Example:

1. SELECT \* FROM Customers WHERE Country='Mexico';
2. SELECT \* FROM Customers WHERE CustomerID=1;

Operators in The WHERE Clause

1. = Equal
2. > Greater than
3. < Less than
4. >= Greater than or equal
5. <= Less than or equal
6. **< >** / !**=** Not equal.
7. BETWEEN Between a certain range
8. LIKE Search for a pattern
9. IN to specify multiple possible values for a column

## AND, OR and NOT Operators:

1. The AND operator displays a record if all the conditions separated by AND are TRUE.
2. The OR operator displays a record if any of the conditions separated by OR is TRUE.
3. The NOT operator displays a record if the condition(s) is NOT TRUE.

AND:

SELECT column1, column2 FROM table\_name

WHERE condition1 AND condition2 AND condition3;

SELECT \* FROM Customers  
WHERE Country='Germany' AND City='Berlin';

OR:

SELECT column1, column2 FROM table\_name

WHERE condition1 OR condition2 OR condition3;

SELECT \* FROM Customers  
WHERE City='Berlin' OR City='Munched';

NOT:

SELECT column1, column2 FROM table\_name WHERE NOT condition;

SELECT \* FROM Customers  
WHERE NOT Country='Germany';

Combining AND, OR and NOT

SELECT \* FROM Customers  
WHERE Country='Germany' AND (City='Berlin' OR City='München');

SELECT \* FROM Customers  
WHERE NOT Country='Germany' AND NOT Country='USA';

## ORDER BY

SELECT column1, column2   
FROM table\_name  
ORDER BY column1, column2,  ASC|DESC;

**code:**

SELECT \* FROM Customers  
ORDER BY Country DESC;

SELECT \* FROM Customers  
ORDER BY Country ASC, CustomerName DESC;

## INSERT INTO Statement

INSERT INTO table\_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country) VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

## NULL Value

SELECT column\_namesFROM table\_name  
WHERE column\_name IS NULL;

SELECT CustomerName, ContactName, Address  
FROM Customers  
WHERE Address IS NULL;

## IS NOT NULL

SELECT column\_namesFROM table\_name  
WHERE column\_name IS NOT NULL;

SELECT CustomerName, ContactName, Address  
FROM Customers  
WHERE Address IS NOT NULL

## UPDATE Statement

UPDATE table\_name  
SET column1 = value1, column2 = value2  
WHERE condition;s

UPDATE Customers  
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'  
WHERE CustomerID = 1;

## DELETE Statement

DELETE FROM table\_name WHERE condition; DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

DELETE FROM table\_name;

## SQL Aggregate Function:

SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.

1. MIN ()
2. MAX ()
3. COUNT (),
4. AVG ()
5. SUM ()

**MIN ():**

SELECT MIN(column\_name)  
FROM table\_name  
WHERE condition;

SELECT MIN(Price) AS SmallestPrice  
FROM Products;

**MAX ():**

SELECT MAX(column\_name)  
FROM table\_name  
WHERE condition;

SELECT MAX(Price) AS LargestPrice  
FROM Products;

**COUNT ():**

1. COUNT with WHERE:

SELECT COUNT (column\_name) FROM table\_name WHERE condition;

SELECT COUNT (\*) FROM PRODUCT\_MAST WHERE RATE>=20;

1. COUNT() with DISTINCT

SELECT COUNT(DISTINCT COMPANY)  FROM PRODUCT\_MAST;

1. COUNT() with GROUP BY

SELECT COMPANY, COUNT(\*)  FROM PRODUCT\_MAST  GROUP BY COMPANY;

1. COUNT() with HAVING

SELECT COMPANY, COUNT(\*)  FROM PRODUCT\_MAST GROUP BY COMPANY  HAVING COUNT(\*)>2;

**AVG ():**

SELECT AVG(column\_name) FROM table\_name WHERE condition;

SELECT AVG(Price) FROM Products;

**SUM ():**

SELECT SUM(column\_name) FROM table\_name WHERE condition;

SELECT SUM(Quantity) FROM OrderDetails;

## LIKE Operator

SELECT column1, column2, FROM table\_name WHERE columnN  LIKE pattern ;

SELECT \* FROM Customers WHERE CustomerName LIKE 'a%';

1. LIKE **'a%'** = Finds any values that start with "a"
2. LIKE **'%a'** = Finds any values that end with "a"
3. LIKE  **%or%’** = Finds any values that have "or" in any position
4. LIKE **'\_r%**' = Finds any values that have "r" in the second position
5. LIKE ‘**a\_%'** = Finds any values that start with "a" and are at least 2 characters
6. LIKE **'a \_ \_%'** = Finds any values that start with "a" and are at least 3 characters
7. LIKE **'a%o'** = Finds any values that start with "a" and ends with "o"

## Wildcard Characters

1. % = Represents zero or more characters
2. \_ = Represents a single character
3. [] = Represents any single character within the brackets
4. ^ = Represents any character not in the brackets
5. - = Represents any single character within the specified range

## IN Operator:

IN operator is a shorthand for multiple OR conditions;

SELECT column\_name(s) FROM table\_name WHERE column\_name IN (value1, value2 );

SELECT \* FROM Customers WHERE Country IN ('Germany', 'France', 'UK');

SELECT column\_name(s) FROM table\_nameHERE column\_name IN (SELECT STATEMENT);

SELECT \* FROM Customers WHERE Country IN (SELECT Country FROM Suppliers);

## BETWEEN Operator:

SELECT column\_name(s) FROM table\_name WHERE column\_name BETWEEN value1 AND value2;

SELECT \* FROM Products WHERE Price BETWEEN 10 AND 20;

SELECT \* FROM Products WHERE Price NOT BETWEEN 10 AND 20;

SELECT \* FROM Products  
WHERE Price BETWEEN 10 AND 20  
AND CategoryID NOT IN (1,2,3);

SELECT \* FROM Products  
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'  
ORDER BY ProductName;

SELECT \* FROM Orders  
WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;

## SQL Aliases:

SELECT column\_name AS alias\_name  
FROM table\_name;

SELECT CustomerID AS ID, CustomerName AS Customer  
FROM Customers;

SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) AS Address  
FROM Customers;

## SQL JOIN

Here are the different types of the JOINs in SQL:

1. (INNER) JOIN:
2. LEFT (OUTER) JOIN:
3. RIGHT (OUTER) JOIN:
4. FULL (OUTER) JOIN:



## INNER JOIN:

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2  
ON table1.column\_name = table2.column\_name;

SELECT Orders.OrderID, Customers.CustomerName  
FROM Orders  
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

**JOIN Three Tables**

SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName  
FROM (

(

Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID

)  
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);

## LEFT JOIN

SELECT column\_name(s)  
FROM table1  
LEFT JOIN table2ON table1.column\_name = table2.column\_name;

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID  
ORDER BY Customers.CustomerName;

## RIGHT JOIN

SELECT column\_name(s)  
FROM table1  
RIGHT JOIN table2ON table1.column\_name = table2.column\_name;

SELECT Orders.OrderID, Employees.LastName, Employees.FirstName  
FROM Orders  
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
ORDER BY Orders.OrderID;

## FULL OUTER JOIN

SELECT column\_name(s)  
FROM table1  
FULL OUTER JOIN table2ON table1.column\_name = table2.column\_nameWHERE condition;

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;

## CROSS JOIN

SELECT column\_name(s) FROM table1 CROSS JOIN table2;

SELECT Customers.CustomerName, Orders.OrderID FROM Customers CROSS JOIN Orders;

## UNION Operator

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

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2; SELECT City FROM Customers  
UNION  
SELECT City FROM Suppliers  
ORDER BY City;

## GROUP BY Statement

SELECT column\_name(s) FROM table\_name WHERE condition GROUP BY column\_name(s)  
ORDER BY column\_name(s);

SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country;

SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country ORDER BY COUNT(CustomerID) DESC;

## HAVING Clause

The SQL HAVING clause is used if we need to filter the result set based on aggregate functions such as MIN() and MAX(), SUM() and AVG(), and COUNT().

SELECT column\_name(s) FROM table\_name WHERE condition GROUP BY column\_name(s)  
HAVING condition ORDER BY column\_name(s);

SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country   
HAVING COUNT(CustomerID) > 5;

## EXISTS Operator

SELECT column\_name(s) FROM table\_name WHERE EXISTS (SELECT column\_name FROM table\_name WHERE condition);

SELECT SupplierName FROM Suppliers WHERE EXISTS (SELECT ProductName FROM Products  WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20);

## ANY and ALL Operators

Any Operator:

1. Return a Boolean value as a result;
2. Return true if any of the subquery values meet tha condition.

SELECT column\_name(s) FROM table\_name WHERE column\_name operator ANY (SELECT column\_name   FROM table\_name   WHERE condition);

ALL Operator:

1. returns a boolean value as a result
2. returns TRUE if ALL of the subquery values meet the condition
3. is used with SELECT, WHER and HAVING statements

SELECT column\_name(s) FROM table\_name WHERE column\_name operator ALL ( SELECT  column\_name   FROM table\_name   WHERE condition);

## SELECT INTO Statement

SELECT \*  
INTO newtable [IN externaldb]  
FROM oldtableWHERE condition;

SELECT \* INTO CustomersBackup2017  
FROM Customers;

## INSERT INTO SELECT Statement

INSERT INTO table2  
SELECT \* FROM table1WHERE condition;

INSERT INTO table2 (column1, column2, column3, ...)  
SELECT column1, column2, column3, ...  
FROM table1  
WHERE condition;