There are 3 types of tiers in an application:

Presentation Tier (Frontend/UI)

- This is the part of the application users see and interact with.
- **HTML**: Used for the structure of the webpage.
- **CSS**: Used for styling the webpage (colors, layout, fonts, etc.).
- **JavaScript**: Used for adding functionality (e.g., buttons that work, animations, etc.).

Application Tier (Middleware/Server)

- This is where the logic of the application runs (backend).
- Examples of programming languages used here: **C, C++, Java, Python, C#**.

Data Tier (Database)

- This is where data is stored and managed.
- Two main types of databases:
- **SQL (Structured Data)**: Examples: MySQL, PostgreSQL, Oracle.
- **NoSQL (Unstructured Data)**: Examples: MongoDB, Cassandra, Redis.

What is Data?

- **Data**: Raw facts that describe the properties of an object or entity.
    - **Example**: A person's name is "Romeo," age is 22, and place is "Jspider."

Terms:

1. **Raw Facts**: Examples: "Romeo," "22," "Jspider."
2. **Properties**: Characteristics of an object. Example: Name, Age, Place.
3. **Object/Entity**: The main subject. Example: "Romeo."

What is a Database?

- A **database** is a system for storing and organizing data in a structured manner.
- Inside a database, we can perform the following operations (**CRUD**):

    - **C - Create/Insert**: Add new data.
    - **R - Read/Retrieve**: View or fetch data.
    - **U - Update/Modify**: Change existing data.
    - **D - Delete/Drop**: Remove data.

DBMS (Database Management System)

- A software that helps maintain and manage databases effectively.
- **Two Main Features**:
    1. **Security**: Controls who can access the data.
    2. **Authorization**: Grants specific permissions to users.
- **Storage**: Data is stored in files.
- **Communication**: We interact with DBMS using **Query Language**.

RDBMS (Relational Database Management System)

- An advanced version of DBMS where data is stored in **tables** (rows and columns).
- **Three Main Features**:

    1. **Security and Authorization**: Similar to DBMS.
    2. **Data Storage**: Organized in tables.
    3. **Communication**: Uses **SQL (Structured Query Language)** to interact with the database.

- **Why RDBMS?**

    - Suitable for large data volumes.
    - Supports complex queries and data relationships.
    - Follows **ACID** properties for data reliability.

Difference Between DBMS and RDBMS

Feature	DBMS	RDBMS
Data Storage	Stores data in files.	Stores data in tables (rows and columns).
Communication	Uses Query Language.	Uses SQL.
Data Volume	Handles small amounts of data.	Handles large amounts of data.
Data Integration	Does not support data integration.	Supports data integration.
ACID Properties	Does not strictly follow ACID properties.	Strictly follows ACID properties.
Data Redundancy	Cannot avoid redundancy entirely.	Redundancy is reduced with functional dependency.
Normalization	Normalization is not supported.	Normalization is supported to organize data.

Relational Model (E.F. Codd)

- Proposed by **E.F. Codd**, this model suggested that data must be stored in **tables** (rows and columns).
- Any database software that follows this model is an **RDBMS**.

Table

- A logical structure used to organize data into rows and columns.
  - **Row**: A single record (example: one student's details).
  - **Column**: An attribute or property (example: Name, Age)

Rules of E.F Codd:-

1) **Single Value in a Cell:**

Each cell in a database table should contain only one value, not multiple values or lists.

Example: If you have a table for storing student details:

Student_ID	Name	Subjects
101	Alice	Math, Science
101	Alice	Math

In the correct table design, each subject should go in a separate row if needed.

2) **Using Multiple Tables and Keys:**

1. Data can be stored across multiple tables, and connections (relationships) can be established using **keys** (like primary and foreign keys).
2. **Example:**

Students Table:

Student_ID	Name
101	Alice
102	Bob

Subjects Table:

Subject_ID	Subject_Name
1	Math
2	Science

Enrollments Table: (Linking Students and Subjects using keys)

Student_ID	Subject_ID
101	1
101	2
102	1

Here, Student_ID in the **Enrollments Table** connects to the **Students Table**, and Subject_ID connects to the **Subjects Table**

3) In R.D.B.M.S the data must be store in the form of tables that include meta data.

Meta data:- Data inside a data or details about a data is known as meta data. Meta data are auto generated and stored inside the meta table.

4) We can validat the data in two different ways:-

a) By assigning data types
b) By assigning constraints

Note:- data types are mandatory where as constraints are optional.

DataType:-

It is used to determine what kind of data to be stored in one platform memory location.

Types:-

1) Char
2) VarChar/VarChar2
3) Number
4) Date
5) Large Object
    a) Character large object
    b) Binary large object

1) Char Data Type:-
    a) Char data type can accept alphabets, numbers, special character and alpha numeric value.
    b) Char(Size) is the syntax.
    c) Size specify the maximum number of character that it can hold for single data.
    d) In char data type it is mandatory to mention size.
    e) The maximum size of char is 2000.
    f) In char the data must be enclosed inside single quotes.
    g) Char follow fixed length memory allocation.

Example:

```
CREATE TABLE Employee (
    EmployeeID CHAR(5), -- Fixed 5 characters
    Name CHAR(10)           -- Fixed 10 characters
);INSERT INTO Employee (EmployeeID, Name) VALUES ('E123', 'John');
```

Note: If "John" is stored, it will still take 10 characters of space.

2) VarChar Data Type:-
    a) It also accept alphabets, numbers, special character and alpha numeric value.
    b) VarChar(size) is the syntax.
    c) Size specify the maximum number of characacter that it can hold for one single data.
    d) In VarChar also it is important to mention size.
    e) Maximum size for varchar is 2000.

f) It also enclosed inside single quotes.
g) It follow variable length memory allocation.

i. VarChar2:-
   1. VarChar2 is an updated version of VarChar datatype.
   2. VarChar2(size) is the syntax.
   3. Maximum size for varchar2 is 4000.
   4. Whenever we use varchar data type that will get automatically converted into varchar2.

Example:

```
CREATE TABLE Product (
    ProductID VARCHAR2(10), -- Maximum 10 characters
    ProductName VARCHAR2(50)
);INSERT INTO Product (ProductID, ProductName) VALUES ('P001', 'Laptop');
```

Note: If "Laptop" is stored, it only takes 6 characters of space, unlike CHAR.

3) Number Data Type:-
a) It is used to store only the numerical value.
b) Number(Precision,[Scale]) is the syntax.
c) Precision specify the number of digits used to store the integer value.
d) Range of precision is 1 to 38.
e) Scale specify the number of digit used to store the decimal value within the precision.
f) The range of scale is -84 to 127.
g) The default value of scale is 0.

Example:

```
CREATE TABLE Sales (
    SalesID NUMBER(5),              -- Integer, max 5 digits
    Amount NUMBER(10, 2)            -- 10 digits, 2 after decimal
);INSERT INTO Sales (SalesID, Amount) VALUES (12345, 56789.34);
```

Note: Amount stores up to 10 digits, with 2 reserved for the decimal.

4) Date Data Type:-
a) It is used to store dates in specific formats.
b) Syntax- Date
c) The oracle specified format to store the data are:-
   i. ‘DD-Mon-YYYY’
   ii. ‘DD-Mon-YY’

Example:

```
CREATE TABLE Events (
    EventID NUMBER(5),
    EventDate DATE
);INSERT INTO Events (EventID, EventDate) VALUES (10001, '01-Jan-2024');
```

Note: You can format the output using TO_CHAR:

```
SELECT TO_CHAR(EventDate, 'DD-MM-YYYY') AS FormattedDate FROM Events;
```

5) Large Object Data Type:-

a) It is used to store a huge amount of data.
   i. Character large object(CLOB):- it is used to store huge amount of character base data upto the size of 4GB.
   ii. Binary large object(BLOB):- it is used to store huge amount of binary data upto the size of 4GB. Ex: Photos,video,audio,etc.

Examples:

```
CREATE TABLE Documents (
    DocID NUMBER,
    DocText CLOB,              -- Large text data
    DocFile BLOB               -- Binary file data
);INSERT INTO Documents (DocID, DocText) VALUES (1, 'This is a large document text.');
```

Note: Use functions or tools to handle uploading binary files for BLOB.

Constraints:-

The rules given to column for extra validation.

Types of Constraints:-

1) Unique
2) Not Null
3) Check
4) Primary Key
5) Foreign Key
6) Default

1) Unique:-

a) It is used to avoid duplicate or repeated value.

```
CREATE TABLE Users (
    userID INT,
    email VARCHAR(255) UNIQUE
);
```

2) Not Null:-

a) It is used to avoid null value.

```
CREATE TABLE Employees (
    empID INT NOT NULL,
    name VARCHAR(100) NOT NULL
);
```

3) Check:-

a) It is an extra validation given to column with the condition if the condition got satisfy it accept the data otherwise reject.

```
CREATE TABLE Orders (
```

```
orderID INT,
quantity INT CHECK (quantity > 0));
```

4) Primary Key:-
   a) It is used to identify the record uniquely from table.
   b) Characteristics of primary key:-
      i. It can not accept duplicate value.
      ii. It can not accept null value.
      iii. It must be a combination of unique and not null.
      iv. In a table we can have only one primary key column.
      v. Primary key is not mandatory, but highly recommended.

5) Foreign Key:-
   a) It is used to establish the connection between the table.
   b) Characteristics of Foreign Key:-
      i. It can accept duplicate value.
      ii. It can accept null value.
      iii. It can not be combination of unique and not null.
      iv. In a table we can have multiple foreign key column.
      v. If an attribute want to become foreign key then it must be a Primary key in its own table.
      vi. Foreign key will always present in the child table but it belongs to parent table.

Emp Table:-

	Eid	Ename	Sal	Doj	PhoneNo	DeptNo	Cid
Datatype	Varchar(6)	VarChar(18)	Number(6)	Date	Number(10)	FK	FK
	J001	Aditya	5000	5-12-24	9876543210	10	1A
	J002	Kumar	6000	6-12-24	9876543211	10	2C
	J003	Singh	7000	5-12-24	9876543212		
	J004	Rajput	8000	9-12-24	9987654321	20	2C

Dept Table:-

Foreign Key(FK)

DeptNo	Dname	Loc
10	Accountant	Banglore
20	Sale	Mysore

Customer Table

FK

Cid	CName

1A PQR

2C XYZ

Overview Of SQL

1. Data Definition Language (DDL)

Used to define and manage database structure (e.g., creating, altering, or deleting tables). Examples: CREATE, ALTER, DROP.

---

2. Data Manipulation Language (DML)

Used to modify data in the database. Examples: INSERT, UPDATE, DELETE.

---

3. Transaction Control Language (TCL)

Used to manage database transactions, ensuring data integrity. Examples: COMMIT, ROLLBACK, SAVEPOINT.

---

4. Data Control Language (DCL)

Used to control access to the database. Examples: GRANT, REVOKE.

---

5. Data Query Language (DQL)

Used to fetch data from the database. The primary command is SELECT.

DQL in Detail

Purpose:

DQL is used to retrieve records from the database.

Key Concepts in DQL:

Select

1. Used to display data fetched from the database.
2. Example: SELECT Sname FROM Students;
   This retrieves only the student names.

Projection

1. Used to fetch specific columns from a table.

   SELECT column_name(s) FROM table_name;

2. Execution Order:
   1. First, the FROM clause determines the table to fetch data from.
   2. Then, the SELECT clause retrieves the specified columns.

   SELECT Sname, Branch FROM Students;

Selection

1. Used to fetch specific rows and columns based on a condition.

   SELECT * FROM Students WHERE Branch = 'IT';

Join

1. Used to fetch records from multiple tables simultaneously.

   SELECT Students.Sname, Marks.Score FROM Students JOIN Marks ON Students.Sid = Marks.Sid;.

Example Table: Students

Sid	Sname	Branch	Percentage
J01	Aditya	IT	87
J02	Kumar	CSE	88
J03	Singh	CSSE	89

Query Examples

Retrieve student names:

SELECT Sname FROM Students;

Retrieve student names and branches:

SELECT Sname, Branch FROM Students;

Retrieve all student details:

SELECT * FROM Students;

Additional Notes

- **Asterisk (*)**: Fetches all columns in a table.
- **Semicolon (;)**: Marks the end of a SQL query

Setting Up SQL Environment

Adjust Table Display:

1. SET PAGES 100 LINES 100: Configures the output to display 100 lines per page and set a line width of 100 characters for better table alignment.

Clear the Screen:

1. CL SCR: Clears the SQL*Plus screen.

Basic Commands

View All Tables:

1. SELECT * FROM TAB;: Displays all available tables in the current schema (e.g., EMP, DEPT, BONUS, SALGRADE).

Recycle Bin:

1. Dropped tables may appear as BIN$... in the recycle bin. To view and permanently remove them:
    1. SELECT * FROM RECYCLEBIN;: Lists all items in the recycle bin.
    2. PURGE TABLE table_name;: Permanently deletes a table from the recycle bin (e.g., marks, students, etc.).

Connect to Another Database:

1. CONN username/password@database;: Connects to a different database schema (e.g., from SCOTT to HR).

Distinct Clause

- The DISTINCT keyword is used to remove duplicate values from the results of a query.
- It must be the **first argument** in the SELECT clause.
- If multiple columns are used with DISTINCT, duplicates are removed based on the **combination of those columns**.

Example 1:
Find all unique job roles from the employees:

SELECT DISTINCT job FROM emp;

Example 2:
Find unique combinations of job and manager:

SELECT DISTINCT job, mgr FROM emp;

Expression

- An **expression** is any calculation or operation that produces a result.
  Example: 1 + 2 = 3
- SQL expressions can include arithmetic operations, functions, or column references.

Practical Examples

Annual Salary of Employees
Calculate the yearly salary of each employee:

SELECT sal * 12 AS annual_salary FROM emp;

Half-term Salary
Calculate the 6-month salary of each employee:

SELECT sal * 6 AS half_term_salary FROM emp;

Quarter-term Salary
Show employee names and their 3-month salary:

SELECT ename, sal * 3 AS quarter_term_salary FROM emp;

Employee Details with Annual Salary
Include annual salary along with all employee details:

SELECT empno, ename, job, mgr, hiredate, sal, sal * 12 AS annual_salary FROM emp;

Salary After a 10% Hike
Show the salary after a 10% increase:

SELECT sal + (sal * 0.10) AS salary_with_hike FROM emp;

Salary After a 25% Hike
Calculate the salary with a 25% hike:

SELECT sal + (sal * 0.25) AS salary_with_25_percent_hike FROM emp;

Annual Salary After Monthly 15% Deduction
Calculate the annual salary after deducting 15% each month:

SELECT (sal * 0.85) * 12 AS annual_salary_after_deduction FROM emp;

Alias

- An **alias** is a temporary name for a column or table, used to make results easier to read.
- Aliases can be written with or without the keyword AS.
- Use quotes (" ") for aliases with spaces.

Examples:

Using alias without AS:

SELECT sal * 12 annual_salary FROM emp;

Using alias with AS:

SELECT sal * 12 AS annual_salary FROM emp;

Using alias with quotes for spaces:

SELECT sal * 12 AS "Annual Salary" FROM emp;

Example:
Show employee names and their annual salary with clear labels:

SELECT ename AS name, sal AS Salary, sal * 12 AS "Annual Salary" FROM emp;

Selection in SQL

- **Selection** is the process of retrieving specific data (columns) from a table.
- You decide which columns to retrieve by specifying their names in the SELECT statement.
  - Example: SELECT ENAME, SAL FROM EMP;
  - This retrieves the ENAME (employee name) and SAL (salary) columns from the EMP table.

WHERE Clause in SQL

- The **WHERE** clause is used to filter rows in a table based on a specific condition.
- It operates row by row and evaluates the condition for each row, returning only those rows where the condition is true.
  - Syntax: SELECT column_names FROM table_name WHERE condition;
  - Example: SELECT ENAME, SAL FROM EMP WHERE SAL > 2000;

- This retrieves the names and salaries of employees where the salary is greater than 2000.

Key Points About WHERE Clause

1. **Filters rows:** Retrieves only the rows that meet the specified condition.
2. **Returns Boolean values:** Evaluates each condition as TRUE or FALSE.
3. **Cannot use multi-row functions:** It works with single-row conditions.
4. **Order of execution:** The WHERE clause is executed before the SELECT statement.

Examples from Your Practice Questions

Retrieve employees hired after January 1, 1981:

```
SELECT * FROM EMP WHERE HIREDATE > '01-JAN-81';
```

Fetch employees with an annual salary greater than 12,000:

```
SELECT ENAME, (SAL * 12) AS ANNUAL_SALARY FROM EMP WHERE SAL * 12 > 12000;
```

Find employee numbers of those in department 30:

```
SELECT EMPNO FROM EMP WHERE DEPTNO = 30;
```

List employee names and hire dates before January 1, 1981:

```
SELECT ENAME, HIREDATE FROM EMP WHERE HIREDATE < '01-JAN-1981';
```

Retrieve all managers:

```
SELECT * FROM EMP WHERE JOB = 'MANAGER';
```

Find employees earning exactly 1400 commission:

```
SELECT ENAME, SAL FROM EMP WHERE COMM = 1400;
```

List employees whose commission is greater than their salary:

```
SELECT * FROM EMP WHERE COMM > SAL;
```

Get employee numbers of those hired before 1987:

```
SELECT EMPNO FROM EMP WHERE HIREDATE < '01-JAN-1987';
```

Retrieve all analysts:

```
SELECT * FROM EMP WHERE JOB = 'ANALYST';
```

Find employees earning more than 2000:

SELECT * FROM EMP WHERE SAL > 2000;

Operator:-

Eight different types of operator:

1) Arithmetic Operator (+,-,*,/)
2) Concatenation Operator(||)
3) Relational Operator(>,<,>=,<=)
4) Logical Operator(AND,OR,NOT)
5) Comparisson Operator(=,!= or <>)
6) Special Operator(IN, NOT IN, IS , IS NOT, LIKE, NOT LIKE, BETWEEN, NOT BETWEEN)
7) Subquery Operators(ALL, ANY, EXISTS, NOT EXISTS)
8) Set Operator(UNION, UNION ALL, INTERSECT, MINUS)

Sure! Let's break down the **SQL Operators** you have studied so far into simple and easy-to-understand English.

---

1. Arithmetic Operators:

- These are used to perform basic **mathematical operations** on numbers.
- Examples:
  - + → Adds two values.
  - - → Subtracts one value from another.
  - * → Multiplies two values.
  - / → Divides one value by another.

Example:

SELECT 10 + 5 AS Sum, 10 - 5 AS Difference, 10 * 5 AS Product, 10 / 2 AS Division;

Output:

Sum: 15, Difference: 5, Product: 50, Division: 5

---

2. Concatenation Operator (||):

- It is used to **merge (combine)** two or more strings together.
- Think of it like joining pieces of text.

Example:

SELECT first_name || ' ' || last_name AS FullName FROM emp;

If first_name is John and last_name is Doe, the output will be:

FullName: John Doe

Here:

- || is the **concatenation operator**.
- ' ' adds a space between the first_name and last_name.

---

3. Relational Operators:

- These operators are used to **compare values**.
- Examples:
    - > → Greater than.
    - < → Less than.
    - >= → Greater than or equal to.
    - <= → Less than or equal to.

Example:

SELECT * FROM emp WHERE salary > 5000;

- This will show all employees whose salary is **greater than 5000**.

---

4. Logical Operators:

Logical operators are used to **combine multiple conditions**.

a) AND Operator

- It returns **TRUE** only if **all conditions** are satisfied.

Example:

SELECT * FROM emp WHERE salary > 5000 AND department = 'HR';

- This will show employees whose salary is greater than 5000 **AND** work in the HR department.

---

b) OR Operator

- It returns **TRUE** if **any one condition** is satisfied.

Example:

SELECT * FROM emp WHERE salary > 5000 OR department = 'HR';

- This will show employees who either have a salary greater than 5000 **OR** work in the HR department.

---

c) NOT Operator

- It is a **Unary Operator**, meaning it works with **one condition**.
- It returns **TRUE** only if the condition is **not satisfied**.

Example:

SELECT * FROM emp WHERE NOT department = 'HR';

- This will show all employees who **do not work** in the HR department.

---

5)Special Operators in SQL (Simplified Explanation)

1. IN Operator

- **What it Does**: Checks if a value matches any value in a given list.
- **How it Works**:
    1. Think of it as a simpler way to use multiple OR conditions.
    2. It selects rows where the column value is **equal to any value** in the list.
    - **Syntax**:

    column_name IN (value1, value2, value3, ...)

    -
    - **Example**:

    SELECT * FROM Students WHERE Grade IN ('A', 'B', 'C');

    - **Explanation**: This query fetches all students with grades 'A', 'B', or 'C'.

2. NOT IN Operator

- **What it Does**: Checks if a value does **not match** any value in a given list.
- **How it Works**:
    - Similar to IN, but instead of selecting the matching values, it **rejects** them.
    - **Syntax**:

    column_name NOT IN (value1, value2, value3, ...)

    -
    - **Example**:

    SELECT * FROM Students WHERE Grade NOT IN ('F', 'D');

- **Explanation**: This query fetches all students who did not get grades 'F' or 'D'.

3. BETWEEN Operator

- **What it Does**: Filters rows where a value lies **within a range** (inclusive of the range boundaries).
- **How it Works**:
  - Useful when checking if a value falls between a minimum and maximum range.
  - **Syntax**:

column_name BETWEEN lower_value AND upper_value

  -
  - **Example**:

SELECT * FROM Products WHERE Price BETWEEN 100 AND 500;

  - **Explanation**: This query fetches all products with prices ranging from 100 to 500, including 100 and 500.

1. IS Operator

- **What it Does**: Used to compare a column's value with NULL.
- **How it Works**:
  - In SQL, = cannot be used to check for NULL values because NULL represents "unknown" or "no value."
  - Use IS to test whether a column's value **is NULL**.

Syntax: column_name IS NULL

SELECT * FROM Employees WHERE ManagerID IS NULL;

Explanation: Fetches all employees who do not have a manager (ManagerID is NULL).

---

2. IS NOT Operator

- **What it Does**: Checks if a column's value **is not NULL**.
- **How it Works**:
  - Used to exclude rows where a column's value is NULL.

Syntax: column_name IS NOT NULL

SELECT * FROM Employees WHERE ManagerID IS NOT NULL;

Explanation: Fetches all employees who have a manager (ManagerID is not NULL).

Key Points to Remember

IS **and** IS NOT **are specifically used with** NULL **values**:

- IS NULL: Checks for rows where the column is NULL.
- IS NOT NULL: Checks for rows where the column has a value.

Why = Does Not Work with NULL:

- NULL in SQL means "unknown." Comparing NULL = NULL does not return TRUE; it returns UNKNOWN.
- Hence, we need the IS and IS NOT operators to handle NULL.

---

4. LIKE Operator

- **What it Does**: Filters rows based on **pattern matching**.
- **How it Works**:
  - Matches text values using wildcard characters (% and _).
- **Wildcards**:
  - %: Represents **zero or more characters**.
  - _: Represents **exactly one character**.
  - **Syntax**:

```
column_name LIKE 'pattern'
```

- **Examples**:

1. **Using** %:

```
SELECT * FROM Employees WHERE Name LIKE 'A%';
```

2. **Explanation**: Fetches all employees whose names start with 'A' (e.g., 'Alice', 'Adam').
3. **Using** _:

```
SELECT * FROM Employees WHERE Name LIKE 'A_';
```

4. **Explanation**: Fetches all employees whose names start with 'A' and are exactly 2 characters long (e.g., 'Al').

Here's a simplified explanation of the concepts:

NOT LIKE Operator

The NOT LIKE operator does the opposite of LIKE.

Instead of matching values, it **excludes values** based on a pattern.

Syntax:

column_name NOT LIKE 'pattern';

Set operator:

It is used to merge the result table of two or more select statement.
No. Of column present in both result table must be equal.
Datatype should also be relavant.
There are 4 types:-
Union all
Intersect
minus

Functions

Functions are reusable blocks of code that perform specific tasks. In SQL, they are used to manipulate and analyze data.

Types of Functions:

User-Defined Functions:

- Functions you create yourself for specific tasks.

Inbuilt Functions:

- Provided by SQL, they come in two types:
  - **Single-Row Functions:** Work on one row at a time (e.g., UPPER(), LOWER()).
  - **Multi-Row Functions:** Work on multiple rows and return one result for the group.

Multi-Row Functions

These functions operate on multiple rows and return a single value.

Types:

1. MAX()**:** Finds the largest value.
2. MIN()**:** Finds the smallest value.
3. AVG()**:** Calculates the average value.
4. SUM()**:** Adds up all the values.

5. COUNT()**:** Counts the number of rows.

Characteristics of Multi-Row Functions:

1. They **cannot accept multiple arguments**.
2. You **cannot use them with a column name or expression** in the SELECT clause.
3. They **ignore NULL values** (except COUNT() when used with COUNT(*)).
4. The WHERE clause **cannot use multi-row functions**.

GROUP BY Clause

The GROUP BY clause groups rows that have the same values in specific columns.

After grouping, you can apply **multi-row functions** like SUM(), AVG(), etc.

Syntax:

```
SELECT column_name, group_function(column_name)
FROM table_name
WHERE <condition>
GROUP BY column_name;
```

Example:

```
SELECT Department, AVG(Salary)
FROM Employees
GROUP BY Department;
```

- Groups employees by department and calculates the average salary for each department.

Execution Order:

1. FROM (fetch the table data)
2. WHERE (filter rows)
3. GROUP BY (group rows)
4. SELECT (choose what to display)

HAVING Clause

The HAVING clause filters groups after grouping is done (similar to WHERE but for groups).

It's often used with **aggregate functions** like SUM() or COUNT().

Syntax:

```
SELECT column_name, group_function(column_name)
FROM table_name
WHERE <condition>
GROUP BY column_name
HAVING <group_condition>;
```

Example:

```
SELECT Department, AVG(Salary)
FROM Employees
GROUP BY Department
HAVING AVG(Salary) > 50000;
```

- Groups employees by department, calculates the average salary for each, and displays only departments with an average salary greater than 50,000.

Execution Order:

1. FROM (fetch the data)
2. WHERE (filter rows)
3. GROUP BY (group rows)
4. HAVING (filter groups)
5. SELECT (display the result)

Difference Between WHERE and HAVING Clauses

WHERE Clause:

- Filters rows based on a condition before any grouping is done.
- It checks each row one by one.
- Cannot use functions that work on multiple rows, like SUM() or AVG().
- Execution happens in this order: FROM -> WHERE -> SELECT.

HAVING Clause:

- Filters groups of rows (after GROUP BY is applied).
- It checks each group instead of individual rows.
- Can use functions that calculate values from multiple rows, like SUM() or AVG().
- Execution happens in this order: FROM -> GROUP BY -> HAVING -> SELECT.

ORDER BY Clause

What it does:

- Arranges the rows in the result table in a specific order.
- By default, it sorts in **ascending order** (smallest to largest).

- To sort in descending order (largest to smallest), use DESC.

Execution order:

- The ORDER BY clause is applied after all the other steps: FROM -> WHERE -> GROUP BY -> HAVING -> SELECT -> ORDER BY.

Single Row Functions in SQL

Single row functions are used to perform operations on individual rows of data in a table. They return one result for each row.

String Functions

Length:

1. Counts the number of characters in a string.
2. **Syntax:** SELECT LENGTH(column_name) FROM table_name;

SELECT LENGTH('string') FROM table_name;

Concat:

1. Joins (merges) two strings together.
2. **Syntax:** SELECT CONCAT('string1', 'string2') FROM table_name;

Upper:

1. Converts a string to uppercase.
2. **Syntax:** SELECT UPPER('string') FROM table_name;

Lower:

1. Converts a string to lowercase.
2. **Syntax:** SELECT LOWER('string') FROM table_name;

Initcap:

1. Converts the first character to uppercase and the rest to lowercase.
2. **Syntax:** SELECT INITCAP('string') FROM table_name;

Reverse:

1. Reverses the characters in a string.
2. **Syntax:** SELECT REVERSE('string') FROM table_name;

Substring:

1. Extracts a part of a string.
2. **Syntax:** SELECT SUBSTR('string', position, length) FROM table_name;

Replace:

1. Replaces a part of a string with another string.
2. **Syntax:** SELECT REPLACE('string', 'old_part', 'new_part') FROM table_name;

Instr:

1. Finds the position of a character or substring in a string.
2. Returns 0 if not found.
3. **Syntax:** SELECT INSTR('string', 'search_string', position, occurrence) FROM table_name;

Lpad:

1. Pads the left side of a string with a specified character to a certain length.
2. **Syntax:** SELECT LPAD('string', length, 'pad_char') FROM table_name;

Rpad:

1. Pads the right side of a string with a specified character to a certain length.
2. **Syntax:** SELECT RPAD('string', length, 'pad_char') FROM table_name;

Trim:

1. Removes unwanted characters or spaces from both ends of a string.
2. **Syntax:** SELECT TRIM('char' FROM 'string') FROM table_name;

Ltrim:

1. Removes unwanted characters or spaces from the left side of a string.
2. **Syntax:** SELECT LTRIM('string', 'unwanted_char') FROM table_name;

Rtrim:

1. Removes unwanted characters or spaces from the right side of a string.
2. **Syntax:** SELECT RTRIM('string', 'unwanted_char') FROM table_name;

Mathematical Functions

Mod:

1. Finds the remainder when one number is divided by another.
2. **Syntax:** SELECT MOD(num1, num2) FROM table_name;

Power:

1. Calculates one number raised to the power of another.
2. **Syntax:** SELECT POWER(base, exponent) FROM table_name;

Round:

1. Rounds a number to a specified number of decimal places.
2. Default is 0 decimal places.
3. **Syntax:** SELECT ROUND(num, scale) FROM table_name;

Trunc:

1. Truncates a number to the nearest lower value based on the scale.
2. **Syntax:** SELECT TRUNC(num, scale) FROM table_name;

Date and Time Functions

Sysdate:

1. Returns the current date from the local system.
2. **Syntax:** SELECT SYSDATE FROM dual;

Current_date:

1. Returns the current date from the server.
2. **Syntax:** SELECT CURRENT_DATE FROM dual;

Systimestamp:

1. Returns the current date, time, and time zone from the system.
2. **Syntax:** SELECT SYSTIMESTAMP FROM dual;

To_date:

1. Adds or subtracts days from a date.
2. **Syntax:** SELECT TO_DATE + days FROM dual;

Last_day():

1. Finds the last date of the month for a given date.
2. **Syntax:** SELECT LAST_DAY('date') FROM dual;

Add_months():

1. Adds a specified number of months to a date.
2. **Syntax:** SELECT ADD_MONTHS(date, months) FROM dual;

Months_between():

1. Calculates the number of months between two dates.
2. **Syntax:** SELECT MONTHS_BETWEEN(date1, date2) FROM dual;

To_char:

1. Converts a date to a specific format.
2. **Syntax:** SELECT TO_CHAR(date, 'format_model') FROM dual;

Common Format Models:

- YYYY → 2024
- YY → 24
- YEAR → TWENTY TWENTY FOUR
- MON → DEC
- MM → 12
- MONTH → DECEMBER
- DD → 30
- DAY → MONDAY
- DY → MON
- D → Day of the week (e.g., 2 for Monday if Sunday is 1)

NVL (NULL Value Logic)

What it does:

1. NVL is used to replace NULL values with something else to avoid problems caused by NULL.

Syntax: NVL(arg1, arg2)

1. arg1: The column or value that might be NULL.
2. arg2: The value to use if arg1 is NULL.

Example:

If an employee's bonus is NULL, replace it with 0:

SELECT NVL(bonus, 0) FROM employees;

NVL2

What it does:

- NVL2 checks if a column has a NULL value and performs an action based on the result.

Syntax: NVL2(arg1, arg2, arg3)

- arg1: The column to check.
- arg2: The value to return if arg1 is **not NULL**.
- arg3: The value to return if arg1 is **NULL**.

Example:
If salary is not NULL, return "Has Salary", otherwise return "No Salary":

```
SELECT NVL2(salary, 'Has Salary', 'No Salary') FROM employees;
```

CASE Function

What it does:

- CASE lets you return different results based on conditions.

Syntax:

```
CASE
   WHEN condition1 THEN result1
   WHEN condition2 THEN result2
   ...
   ELSE default_result
END
```

Example:
If salary is above 50000, return "High Salary"; otherwise, return "Low Salary":

```
SELECT
   CASE
      WHEN salary > 50000 THEN 'High Salary'
      ELSE 'Low Salary'
   END AS Salary_Status
FROM employees;
```

-

Subquery

What it does:

- A **subquery** is a query inside another query. It is used to get data that helps the main query.

When to use it:

Unknown Data: When you don't know the value you need for your condition.
Example:
Find employees who earn more than James:

1.

```
SELECT ename
FROM emp
WHERE sal > (SELECT sal FROM emp WHERE ename = 'JAMES');
```

Data in Different Tables: When the required data is in another table.
Example:
Find the department name of Allen:

```
SELECT dname
FROM dept
WHERE deptno = (SELECT deptno FROM emp WHERE ename = 'Allen');
```

Types of Subqueries

Single-row Subquery

- Returns one value as the result.
- Can use normal comparison operators like =, <, or >.
    - Example:

```
SELECT ename
FROM emp
WHERE sal > (SELECT AVG(sal) FROM emp);
```

Multi-row Subquery

- Returns multiple values as the result.
- Requires special operators like IN, ANY, or ALL.
    - Example:
      Find employees whose salaries match those of people in department 10:

```
SELECT ename
FROM emp
WHERE sal IN (SELECT sal FROM emp WHERE deptno = 10);
```

Key Takeaways

1. Use **NVL** to handle NULL values by replacing them with a default.
2. Use **NVL2** to check for NULL and take different actions based on whether a value exists.
3. Use **CASE** for condition-based results, similar to if-else.
4. Use **subqueries** to fetch unknown or related data, and remember:

    - Single-row subqueries return one value.
    - Multi-row subqueries return multiple values.

---

1. ALL Operator:

- The ALL operator is used with a subquery (a query inside another query) and allows multiple values on the right-hand side (RHS).
- It **returns** TRUE **only if the condition is satisfied for all the values** returned by the subquery.
- You need to use it with a relational operator (like >, <, =).

Example:

```
SELECT ename
FROM emp
WHERE sal > ALL (SELECT sal FROM emp WHERE dept = 20);
```

- This query will list the names (ename) of employees whose salary (sal) is greater than **all** the salaries of employees in department 20.

---

2. ANY Operator:

- The ANY operator is also used with a subquery and allows multiple values on the RHS.
- It **returns** TRUE **if the condition is satisfied for at least one value** returned by the subquery.
- Like ALL, it must also be used with a relational operator.

Example:

```
SELECT ename
FROM emp
WHERE sal > ANY (SELECT sal FROM emp WHERE dept = 20);
```

- This query will list the names (ename) of employees whose salary (sal) is greater than **at least one** salary of employees in department 20.

---

3. Nested Subquery:

- A **nested subquery** is when you write one subquery inside another subquery.
- These are used to perform more complex operations, like finding specific ranks of values (e.g., 2nd highest salary).

Example:
To find the 2nd highest salary:

```
SELECT MAX(sal)
FROM emp
WHERE sal < (SELECT MAX(sal) FROM emp);
```

- The inner query finds the highest salary (MAX(sal)).
- The outer query finds the maximum salary that is less than the highest, i.e., the 2nd highest salary.

To find the nth highest or nth lowest salary, you can keep nesting queries in a similar way.

What is a JOIN?

A **JOIN** is used to combine data from multiple tables in a database, allowing you to fetch related records from those tables.

Types of JOINS:

1. **Cartesian Join (Cross Join):**
    1. Combines **every row** from the first table with **every row** from the second table.
    2. This can result in a lot of rows, as the total number of rows is the product of rows in both tables.
    3. **Example:** If Table 1 has 3 rows and Table 2 has 3 rows, the result will have **3 × 3 = 9 rows**.
    4. Total columns in the result = Columns in Table 1 + Columns in Table 2.

Syntax:

Oracle Syntax:

```
SELECT column_name OR expression
FROM table1, table2;
```

ANSI Syntax:

```
SELECT column_name OR expression
FROM table1 CROSS JOIN table2;
```

Example:

Table 1 (A):

ID	Name
1	John
2	Jane
3	Jim

Table 2 (B):

Dept_ID	Dept_Name
10	HR
20	Sales
30	IT

Result of a Cartesian Join:

ID	Name	Dept_ID	Dept_Name
1	John	10	HR
1	John	20	Sales
1	John	30	IT
2	Jane	10	HR
2	Jane	20	Sales
2	Jane	30	IT
3	Jim	10	HR
3	Jim	20	Sales
3	Jim	30	IT

Here:

- Total rows = 3 (from Table 1) × 3 (from Table 2) = 9 rows.
- Total columns = 2 (from Table 1) + 2 (from Table 2) = 4 columns.

So, **Cartesian Join** is like matching **every combination of rows** from the two tables, even if the data doesn't make logical sense. It is often used as a base step for more meaningful joins like Inner or Outer Joins.

2) Inner Join:

- **Purpose**: Retrieves only the matching records from two tables.
- **Key Rule**: A join condition is mandatory.
- **Syntax**:
    - **ANSI Syntax**: SELECT column_name FROM table1 INNER JOIN table2 ON <join_condition> WHERE <filter_condition>;
    - **Oracle Syntax**: SELECT column_name FROM table1, table2 WHERE <join_condition> AND <filter_condition>;

---

3) Outer Join:

- **Purpose**: Retrieves unmatched records along with the matched ones.

Types of Outer Joins:

Left Outer Join:

- Gets all records from the left table, including unmatched records, and the matching records from the right table.
- **Syntax**:

    ANSI Syntax: SELECT column_name FROM table1 LEFT JOIN table2 ON <join_condition>;

    Oracle Syntax: SELECT column_name FROM table1, table2 WHERE table1.column_name = table2.column_name(+);

Right Outer Join:

- Gets all records from the right table, including unmatched records, and the matching records from the left table.
- **Syntax**:

    ANSI Syntax: SELECT column_name FROM table1 RIGHT JOIN table2 ON <join_condition>;

    Oracle Syntax: SELECT column_name FROM table1, table2 WHERE table1.column_name(+) = table2.column_name;

Full Outer Join:

- Gets unmatched records from both tables, along with the matched ones.
- **Syntax**:

    ANSI Syntax: SELECT column_name FROM table1 FULL JOIN table2 ON <join_condition>;

Natural Join:

- **Behavior**:

- Acts like an **Inner Join** when there is a relationship (common columns) between the tables.
- Acts like a **Cross Join** (cartesian product) when there is no relationship.

Syntax: SELECT column_name FROM table1 NATURAL JOIN table2;

Self Join

What is it?
A **self join** happens when a table is joined to itself. This is useful when you need to compare rows in the same table.

When and Why to Use It?
Use a self join when the data you want to display is in the same column but stored in **different rows**.
For example: In an **employee table**, you might want to find out who reports to whom. Both employees and their managers are stored in the same table.

How to Write It?

Oracle Syntax:

```
SELECT col_name
FROM table1 t1, table1 t2
WHERE <join_condition>;
```

ANSI Syntax:

```
SELECT col_name
FROM table1 t1
JOIN table1 t2
ON <join_condition>;
```

Correlated Subquery

What is it?
A **correlated subquery** is a query inside another query, where the **inner query depends on the outer query** to execute. Similarly, the outer query depends on the result of the inner query.

How Does It Work?
The inner query runs **once for every row** in the outer query.
It's like a conversation where one question (inner query) depends on the answer to another question (outer query).

Why Use It?
Use a correlated subquery when you need to compare data in the same table or find specific values that need row-by-row processing. For example, finding the **Nth maximum or minimum value**.

EXISTS Operator

- **What Does It Do?**
  It checks if a subquery returns any data.
    - If the subquery returns **at least one row**, EXISTS returns **TRUE**.
    - If the subquery returns **no rows**, EXISTS returns **FALSE**.

NOT EXISTS Operator

- **What Does It Do?**
  It checks if a subquery returns no data.
    - If the subquery returns **no rows**, NOT EXISTS returns **TRUE**.
    - If the subquery returns **any rows**, NOT EXISTS returns **FALSE**.

Finding Nth Max or Min Value

What is it?

You can use a correlated subquery to find the **Nth highest or lowest value** in a table.
For example, finding the 2nd highest salary or 3rd lowest price.

How to Write It?

Nth Maximum:

```
SELECT column_name
FROM table_name t1
WHERE (
   SELECT COUNT(DISTINCT column_name)
   FROM table_name t2
   WHERE t2.column_name > t1.column_name
) IN (N-1);
```

Nth Minimum:

```
SELECT column_name
FROM table_name t1
WHERE    (
   SELECT COUNT(DISTINCT column_name)
   FROM table_name t2
   WHERE t2.column_name < t1.column_name
) IN (N-1);
```

Pseudo Columns

They are the false columns present in every table but we can not see the column until we call it explicitly.

Pseudo columns are rowid and rownum.

Select rowid,rownum,emp.* from emp;

RowId

It is a 18 digit address, assigned to every record in a table.

Rowid are generate at the time insertion of the record.

Rowid are unique.

We can not do any changes or modification in rowid.

Rowid are static in nature.

Rownum

It is the serial number assigned to every record in the table.

It is generated at time of execution.

It is dynamic in nature.

It always start with 1.

To make the rownum static:

Assign rownum to a table and rename it as serial number.

```
Select rownum as slno ,emp.* from emp;
```

Pass the result table of step 1 inside a from clause and use the slno as condition to get the expected output.

```
Select * from (select rownum as slno,emp.* from emp) where slno = 3;
```

To find Nth max value by pseudo column:

Step 1 - take the distinct sal from table and arrange it in descending order.

Select distinct sal from emp order by sal desc;

Step 2 - assign rownum and rename it as slno for the result table of step 01

Select rownum as slno,sal from (Select distinct sal from emp order by sal desc);

Step 3 - pass the result table of step 02 inside a from clause to make rownum static, and use the slno as condition to get expected result.

Select sal from (Select rownum as slno,sal from (Select distinct sal from emp order by sal desc)) where slno in 3;

To find nth min using pseudo column

Step 1 - take the distinct sal from table and arrange it in descending order.

Select distinct sal from emp order by sal;

Step 2 - assign rownum and rename it as slno for the result table of step 01

Select rownum as slno,sal from (Select distinct sal from emp order by sal);

Step 3 - pass the result table of step 02 inside a from clause to make rownum static, and use the slno as condition to get expected result.

Select sal from (Select rownum as slno,sal from (Select distinct sal from emp order by sal)) where slno in 3;

Data Definition Language:

It is used to create or modify the structure of an object inside the database.

Statement of DDL:-

1) Create - it is used to crate a new object inside the database. If you create a table in database first check is there any table present with the same name like you creating emp then first check emp table is created already or not.

create table stu (

sid varchar(5) primary key,

sname varchar(20) not null,

sage number(2) not null,

sloc varchar(15) not null,

sdob date,

sphno number(10) not null unique check(length(sphno)= 10))

create table Attendance_Details (

sno number(3) not null,

att_date date not null,

att_status varchar(10),

student_id varchar(5),

foreign key(student_id) references stu(sid))

Sure! Let me break down your SQL notes into simpler terms so it’s easier to understand:

Copying a Table

1. **Copying a table with records**:

- You can create a new table and copy data from an existing table.

- Example: `Create table d10 as select * from emp where deptno = 10;`

- This creates a new table `d10` and copies all rows from the `emp` table where `deptno` is 10.

2. **Copying a table without records**:

- You can create a new table with the same structure but without copying any data.

- Example: `Create table emp_1 as select * from emp where 1 = 2;`

  - This creates a new table `emp_1` with the same structure as `emp`, but no rows are copied because `1 = 2` is always false.

Renaming a Table

- You can change the name of an existing table.

- Syntax: `Rename table_name to new_name;`

  - Example: `Rename emp to employee;` changes the table name from `emp` to `employee`.

Altering a Table

Alter is used to modify the structure of an existing table.

1. **Adding a column**:

   - Syntax: `Alter table table_name Add col_name datatype constraints;`

     - Example: `Alter table emp Add age number;` adds a new column `age` to the `emp` table.

2. **Deleting a column**:

   - Syntax: `Alter table table_name Drop column column_name;`

     - Example: `Alter table emp Drop column age;` removes the `age` column from the `emp` table.

3. **Changing a column name**:

- Syntax: `Alter table table_name Rename column old_column to new_column;`

- Example: `Alter table emp Rename column age to employee_age;` renames the `age` column to `employee_age`.

4. **Modifying a column's datatype**:

- Syntax: `Alter table table_name Modify col_name new_datatype;`

- Example: `Alter table zombies Modify sname varchar(25);` changes the `sname` column to a `varchar(25)` datatype.

5. **Adding constraints**:

- Syntax: `Alter table table_name Add constraints ref_name constr_name(col_name);`

- Example: `Alter table stn Add constraints cid_fk foreign key(cid) References courses(cid);` adds a foreign key constraint to the `cid` column.

6. **Deleting constraints**:

- Syntax: `Alter table table_name Drop constraint ref_name;`

- Example: `Alter table stn Drop constraint cid_fk;` removes the `cid_fk` constraint.

Truncate

- Truncate is used to delete all records from a table permanently, but it keeps the table structure intact.

- Syntax: `Truncate table table_name;`

- Example: `Truncate table emp;` deletes all rows from the `emp` table.

Drop

- Drop is used to delete the entire table (structure and data) from the database.

- Syntax: `Drop table table_name;`

- Example: `Drop table emp;` deletes the `emp` table and moves it to the recycle bin.

Recovering a Dropped Table

1. Check the recycle bin: `Select * from recyclebin;`

2. Recover the table: `Flashback table table_name to before drop;`

- Example: `Flashback table emp to before drop;` restores the `emp` table.

Purge

- Purge is used to permanently delete a table from the recycle bin.

- Syntax: `Drop table table_name purge;`

- Example: `Drop table emp purge;` permanently deletes the `emp` table.

Data Manipulation Language (DML)

DML is used to manipulate data in a table.

1. **Insert**:
    - Add new rows to a table.
    - Syntax: `Insert into table_name values(v1, v2, v3…);`
        - Example: `Insert into emp values(101, 'John', 30);`

2. **Update**:
    - Modify existing data in a table.
    - Syntax: `Update table_name Set col_name = value Where condition;`
        - Example: `Update emp Set age = 31 Where id = 101;`

3. **Delete**:
    - Remove specific rows from a table.
    - Syntax: `Delete from table_name Where condition;`
        - Example: `Delete from emp Where id = 101;`

Difference Between Truncate and Delete

Truncate	**Delete**
Removes all rows permanently.	Removes specific rows.

| Does not support `Where` clause. | Supports `Where` clause. |

| Belongs to DDL. | Belongs to DML. |

| Auto-commits (cannot rollback). | Can be rolled back. |

| Does not activate triggers. | Activates triggers. |

Transaction Control Language (TCL)

TCL is used to manage transactions (changes made by DML statements).

1. **Commit**:
    - Saves changes permanently.
    - Syntax: `Commit;`

2. **Rollback**:
    - Undoes changes made after the last commit.
    - Syntax: `Rollback;`

3. **Savepoint**:
    - Marks a point in a transaction to which you can rollback.
    - Syntax: `Savepoint savepoint_name;`
        - Example: `Savepoint s1; Rollback to s1;`

Data Control Language (DCL)

DCL is used to control access to data.

1. **Grant**:
    - Gives permissions to users.
    - Syntax: `Grant sql_statement On table_name To username;`
        - Example: `Grant select On emp To user1;`

2. **Revoke**:
    - Takes back permissions from users.
    - Syntax: `Revoke sql_statement On table_name From username;`
        - Example: `Revoke select On emp From user1;`

Attributes

Attributes are properties that define an entity (like columns in a table).

1. **Key Attribute**:
    - Uniquely identifies a record (e.g., `id`).

2. **Non-Key Attribute**:
    - All other attributes (e.g., `name`, `age`).

3. **Prime Key Attribute**:

- The main key attribute used to uniquely identify a record (e.g., `id`).

4. **Non-Prime Key Attribute**:

- Other key attributes besides the prime key.

5. **Super Key Attribute**:

- A combination of key attributes that uniquely identifies a record.

6. **Composite Key Attribute**:

- A combination of two or more non-key attributes used to uniquely identify a record.

7. **Foreign Key Attribute**:

- An attribute that links to another table's primary key (e.g., `dept_id` in `emp` table linking to `dept` table).

Functional dependency: there exist a dependency in such a way that an attribute in a relation determines another attribute is known as functional dependency.