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).

TODAYS QUESTION:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20

7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

1) WRITE A QUERY TO DISPLAY ALL THE DETAILS FROM THE EMPLOYEE TABLE;

```
SQL> SELECT * FROM EMP;
```

2) WAQTD NAMES OF ALL THE EMPLOYEES;

```
SQL> SELECT ENAME FROM EMP;
```

3) WAQTD NAME AND SALARY GINVEN TO ALL THE EMPLOYEES.

```
SQL> SELECT ENAME,SAL FROM EMP;
```

4) WAQTD NAME AND COMMISSION GIVEN TO ALL THE EMPLOYEES;

```
SQL> SELECT ENAME,COMM FROM EMP;
```

5) WAQTD EMPLOYEE ID AND DEPARTMENT NUMBER OF ALL THE EMPLOYEES IN EMP TABLE;

```
SQL> SELECT EMPNO,DEPTNO FROM EMP;
```

6) WAQTD ENAME AND HIREDATE OF ALL THE EMPLOYEES

```
SQL> SELECT ENAME,HIREDATE FROM EMP;
```

7) WAQTD NAME AND DESIGNATION OF ALL THE EMPLOYEES;

SQL> SELECT ENAME,JOB FROM EMP;

8) WAQTD NAME, JOB AND SALARY GIVEN ALL THE EMPLOYEES.

SQL> SELECT ENAME,JOB,SAL FROM EMP;

9) WAQTD DNAMES PRESENT IN DEPARTMENT TABLE;

SQL> SELECT * FROM DEPT;

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

SQL> SELECT DNAME FROM DEPT;

10) WAQTD DNAME AND LOCATION PRESENT IN DEPT TABLE.

SQL> SELECT DNAME,LOC FROM DEPT;

ANOTHER 25 PRACTICE QUESTIONS

SQL> select * from tab;

```
TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
COUNTRIES                      TABLE
DEPARTMENTS                    TABLE
EMPLOYEES                      TABLE
EMP_DETAILS_VIEW               VIEW
JOBS                           TABLE
JOB_HISTORY                    TABLE
LOCATIONS                      TABLE
REGIONS                        TABLE

8 rows selected.

SQL> desc employees;

Name                          Null?                    Type
EMPLOYEE_ID                   NOT NULL                 NUMBER(6)
FIRST_NAME                                             VARCHAR2(20)
LAST_NAME                     NOT NULL                 VARCHAR2(25)
EMAIL                         NOT NULL                 VARCHAR2(25)
PHONE_NUMBER                                           VARCHAR2(20)
HIRE_DATE                     NOT NULL                 DATE
JOB_ID                        NOT NULL                 VARCHAR2(10)
SALARY                                                 NUMBER(8,2)
```

COMMISSION_PCT NUMBER(2,2)

MANAGER_ID NUMBER(6)

DEPARTMENT_ID NUMBER(4)

SQL> --q1(1. Display the employee ID and first name of all employees.)

SQL> select employee_id,first_name from employees;

SQL> --(Q2. Show the last name and hire date of all employees.)

SQL> select last_name,hire_date from employees;

SQL> --(Q3. List the job ID and salary for all employees.)

SQL> select job_id,salary from employees;

SQL> --(Q4. Display the email and phone number of all employees.)

SQL> select email,phone_number from employees;

SQL> --(Q5. Show the department name and location ID of all departments)

SQL> desc departments;

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(4)
LOCATION_ID		NUMBER(4)

SQL> select department_name,location_id from departments;

```
SQL> --(Q6. Retrieve the employee ID, first name, and last name of all employees.)

SQL> select employee_id,first_name,last_name from employees;



SQL> --(Q7. Show the job title and minimum salary from the jobs table.)

SQL> desc jobs;

Name                                      Null?    Type

JOB_ID                                    NOT NULL VARCHAR2(10)

JOB_TITLE                                 NOT NULL VARCHAR2(35)

MIN_SALARY                                         NUMBER(6)

MAX_SALARY                                         NUMBER(6)

SQL> select job_title,min_salary from jobs;



SQL> --(Q8. List the commission percentage and salary of all employees)

SQL> select commission_pct,salary from employees;



SQL> --(Q9. Display the department name and manager ID for all departments.)

SQL> select department_name,manager_id from departments;



SQL> --(Q10. Retrieve the first name and job ID of all employees.)

SQL> select first_name,job_id from employees;
```

SQL> --(Q11. Show the job ID and maximum salary from the jobs table.)

SQL> select job_id,max_salary from jobs;

SQL> --(Q12. List the department ID and department name from the departments table.)

SQL> select department_id,department_name from departments;

SQL> --(Q13. Display the employee ID and department ID for all employees.)

SQL> select employee_id,department_id from employees;

SQL> --(Q14. Show the city and state province)

SQL> desc REGION;

Name	Null?	Type
REGION_ID	NOT NULL	NUMBER(2)
REGION_NAME		VARCHAR2(25)

SQL> desc locations;

Name	Null?	Type
LOCATION_ID	NOT NULL	NUMBER(4)
STREET_ADDRESS		VARCHAR2(40)
POSTAL_CODE		VARCHAR2(12)
CITY	NOT NULL	VARCHAR2(30)
STATE_PROVINCE		VARCHAR2(25)
COUNTRY_ID		CHAR(2)

SQL> select city,state_province from locations;

SQL> --(Q15. List the employee ID and start date)

SQL> select employee_id,hire_date from employees;

SQL> --(Q16. Retrieve the first name, last name, and email of all employees.)

SQL> select first_name,last_name,email from employees;

SQL> --(Q17. Display the job title and department ID of all employees.)

SQL> select job_id,department_id from employees;

18. Show the department ID and location ID

SQL> select department_id,location_id from departments;

19. List the job ID and job title

SQL> select job_id,job_title from jobs;

20. Retrieve the location ID and city name

SQL> select location_id,city from locations;

21. Show the employee ID and salary from the employees table.

SQL> select employee_id,salary from employees;

22. Display the country ID and region ID from the countries table.

SQL> select country_id,region_id from countries;

23. List the employee ID, start date, and end date of the job.

SQL> select employee_id,start_date,end_date from job_history;

24. Show the first name and last name from the employees table.

SQL> select first_name,last_name from employees;

25. Retrieve the location ID and postal code from the locations table.

SQL> select location_id,postal_code from locations;

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;

Question Related to Expression[Alias]:

---

1. WAQTD NAME OF THE EMPLOYEE ALONG WITH THEIR ANNUAL SALARY.

SELECT ENAME, (SAL * 12) AS ANNUAL_SALARY FROM EMP;

---

2. WAQTD ENAME AND JOB FOR ALL THE EMPLOYEES WITH THEIR HALF TERM SALARY.

SELECT ENAME, JOB, (SAL * 6) AS HALF_SALARY FROM EMP;

---

3. WAQTD ALL THE DETAILS OF THE EMPLOYEES ALONG WITH AN ANNUAL BONUS OF 2000.

SELECT EMP.*, (SAL + 2000) AS AFTER_ANNUAL_BONUS FROM EMP;

---

4. WAQTD NAME, SALARY, AND SALARY WITH A HIKE OF 10%.

SELECT ENAME, SAL, (SAL * 1.1) AS SALARY_AFTER_HIKE FROM EMP;

---

5. WAQTD NAME AND SALARY WITH A DEDUCTION OF 25%.

SELECT ENAME, (SAL * 12 * 0.75) AS SALARY_AFTER_DEDUCT FROM EMP;

---

6. WAQTD NAME AND SALARY WITH A MONTHLY HIKE OF 50.

SELECT ENAME, (SAL + 12 * 50) AS SALARY_WITH_MONTHLY_HIKE FROM EMP;

---

7. WAQTD NAME AND ANNUAL SALARY WITH A DEDUCTION OF 10%.

SELECT ENAME, (SAL * 12 * 0.9) AS SALARY_AFTER_DEDUCT FROM EMP;

---

8. WAQTD TOTAL SALARY GIVEN TO EACH EMPLOYEE (SAL + COMM).

SELECT ENAME, (SAL + NVL(COMM, 0)) AS TOTAL_SALARY FROM EMP;

---

9. WAQTD DETAILS OF ALL THE EMPLOYEES ALONG WITH ANNUAL SALARY.

SELECT EMP.*, (SAL * 12) AS ANNUAL_SALARY FROM EMP;

---

10. WAQTD NAME AND DESIGNATION ALONG WITH 100 PENALTY IN SALARY.

SELECT ENAME, JOB, (SAL - 100) AS SALARY_AFTER_PENALTY FROM EMP;

---

1. Employee ID and salary after adding $1000 bonus.

SELECT EMPLOYEE_ID, (SALARY + 1000) AS BONUS_AFTER_SALARY FROM EMPLOYEES;

---

2. Employee ID and annual salary (monthly salary * 12).

SELECT EMPLOYEE_ID, (SALARY * 12) AS ANNUAL_SALARY FROM EMPLOYEES;

---

3. Employee ID and salary after a 10% hike.

SELECT EMPLOYEE_ID, (SALARY * 1.1) AS SALARY_AFTER_HIKE FROM EMPLOYEES;

---

4. Employee ID and salary after 5% tax deduction.

SELECT EMPLOYEE_ID, (SALARY * 0.95) AS SALARY_AFTER_TAX FROM EMPLOYEES;

---

5. Employee ID and salary after a 20% reduction.

SELECT EMPLOYEE_ID, (SALARY * 0.8) AS SALARY_AFTER_TEMPORARY_CUT FROM EMPLOYEES;

---

6. Employee ID and salary after a 15% raise.

SELECT EMPLOYEE_ID, (SALARY * 1.15) AS SALARY_RAISE FROM EMPLOYEES;

---

7. Employee ID and half-year salary (monthly salary * 6).

SELECT EMPLOYEE_ID, (SALARY * 6) AS HALF_SALARY FROM EMPLOYEES;

---

8. Employee ID and salary divided by 2 (part-time role).

SELECT EMPLOYEE_ID, (SALARY / 2) AS PART_TIME_SALARY FROM EMPLOYEES;

---

9. Employee ID and salary after a 15% increase.

SELECT EMPLOYEE_ID, SALARY, (SALARY * 1.15) AS INCREASED_SALARY FROM EMPLOYEES;

---

10. Employee ID and salary after a 25% raise.

SELECT EMPLOYEE_ID, SALARY, (SALARY * 1.25) AS INCREASED_SALARY FROM EMPLOYEES;

---

11. Employee ID and salary after a $500 deduction.

SELECT EMPLOYEE_ID, SALARY, (SALARY - 500) AS DEDUCTED_SALARY FROM EMPLOYEES;

---

12. Employee ID and three-month salary.

SELECT EMPLOYEE_ID, SALARY, (SALARY * 3) AS THREE_MONTH_SALARY FROM EMPLOYEES;

---

13. Employee ID and salary after a 30% reduction.

SELECT EMPLOYEE_ID, SALARY, (SALARY * 0.7) AS REDUCED_SALARY FROM EMPLOYEES;

---

14. Employee ID and salary after a 10% cut.

SELECT EMPLOYEE_ID, SALARY, (SALARY * 0.9) AS CUT_SALARY FROM EMPLOYEES;

---

15. Employee ID and difference between salary and $50,000.

SELECT EMPLOYEE_ID, SALARY, (50000 - SALARY) AS DIFFERENCE_SALARY FROM EMPLOYEES;

---

16. Employee ID and total salary over 18 months.

SELECT EMPLOYEE_ID, SALARY, (SALARY * 18) AS TOTAL_SALARY FROM EMPLOYEES;

---

17. Employee ID and monthly salary (salary / 12).

SELECT EMPLOYEE_ID, SALARY, (SALARY / 12) AS MONTHLY_SALARY FROM EMPLOYEES;

---

18. Employee ID and salary after $2500 bonus and $500 deduction.

SELECT EMPLOYEE_ID, SALARY, (SALARY + 2500 - 500) AS BONUS_SALARY FROM EMPLOYEES;

---

19. Employee ID and salary after 10% hike and $200 deduction.

SELECT EMPLOYEE_ID, SALARY, (SALARY * 1.1 - 200) AS CURRENT_SALARY FROM EMPLOYEES;

---

20. Employee ID and weekly salary (salary / 52).

SELECT EMPLOYEE_ID, SALARY, (SALARY / 52) AS WEEKLY_SALARY FROM EMPLOYEES;

---

21. Employee ID and salary after $2000 deduction.

SELECT EMPLOYEE_ID, SALARY, (SALARY - 2000) AS DIFFERENCE_SALARY FROM EMPLOYEES;

---

22. Employee ID and salary multiplied by 1.05, then divided by 2.

SELECT EMPLOYEE_ID, SALARY, (SALARY * 1.05 / 2) AS DIFFERENCE_SALARY FROM EMPLOYEES;

---

23. Employee ID and doubled salary.

SELECT EMPLOYEE_ID, SALARY, (SALARY * 2) AS DOUBLE_SALARY FROM EMPLOYEES;

---

24. Employee ID and quarterly salary (monthly salary * 3).

SELECT EMPLOYEE_ID, SALARY, (SALARY * 3) AS THREE_MONTH_SALARY FROM EMPLOYEES;

---

25. Employee ID and salary after $5000 deduction and 15% raise.

SELECT EMPLOYEE_ID, SALARY, ((SALARY - 5000) * 1.15) AS ADJUSTED_SALARY FROM EMPLOYEES;

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;
```
