SQL NOTES

BY : SANJAY ELUMALAI

DISCLAIMER AND CONTENT ACKNOWLEDGMENT

- I do not own the original content or intellectual rights. All credits for the actual content, teaching style, and structure go to the respective trainer.
- I have only compiled, formatted, and written this PDF to help learners understand SQL better.
- I have used the abbreviation "WAQTD" (Write A Query To Display) consistently in many SQL question formats for simplification.
- I sincerely apologize for any inadvertent spelling errors and truly appreciate your understanding.
- The PDF includes comprehensive coverage of SQL basics and core concepts like DDL, DML, DQL, DCL, and TCL, data types, constraints, and functions, clauses like SELECT, WHERE, GROUP BY, HAVING, ORDER BY, operators and expressions, joins and subqueries (nested, correlated, etc.), pattern matching, special operators, and normalization.
- I guarantee that learners will gain a solid understanding of foundational SQL concepts after going through this material.

TABLE OF CONTENTS

INTRODUCTION

WHAT IS SQL AND WHY WE NEED IT ?

- SQL (Structured Query Language) is a language used to work with data stored in a relational database. It helps you add new data, view existing data, update data, and remove data using simple commands. It is widely used in applications where data needs to be stored, managed, and accessed efficiently. SQL is used to manipulate the database .

S	Q	L
• **IN BUILT FUNCTION**	• **CODE / STATEMENTS**	• **COMMUNICATION MEDIUM**
• **IN BUILT METHODS**		
• **IN BUILT OPERATIONS**		

DATA

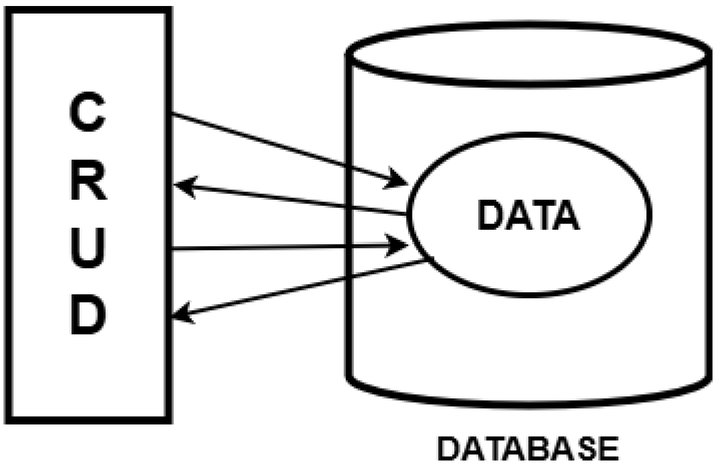
- Data is a **raw fact** which is used to describe **attributes of an entity.**
- **Attributes = Properties.**
- **Entity = Object.**

EXAMPLE:

Brand	One Plus
Cost	54,000
Color	White
RAM	12 GB
ROM	256 GB

DATABASE

- Database is a **place or medium** which is used to **store data** in a **systematic** and **organized manner.**

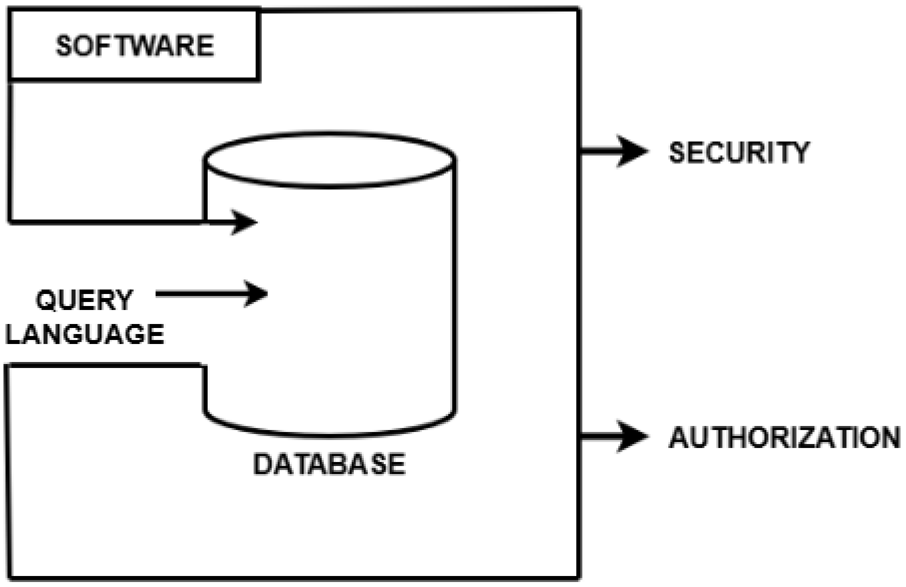


- **CREATE / INSERT**
- **READ / RETRIEVE**
- **UPDATE / MODIFY**
- **DELETE / DROP**

These operations are known as **“CRUD”** operations.

DATABASE MANAGEMENT SYSTEM (DBMS)

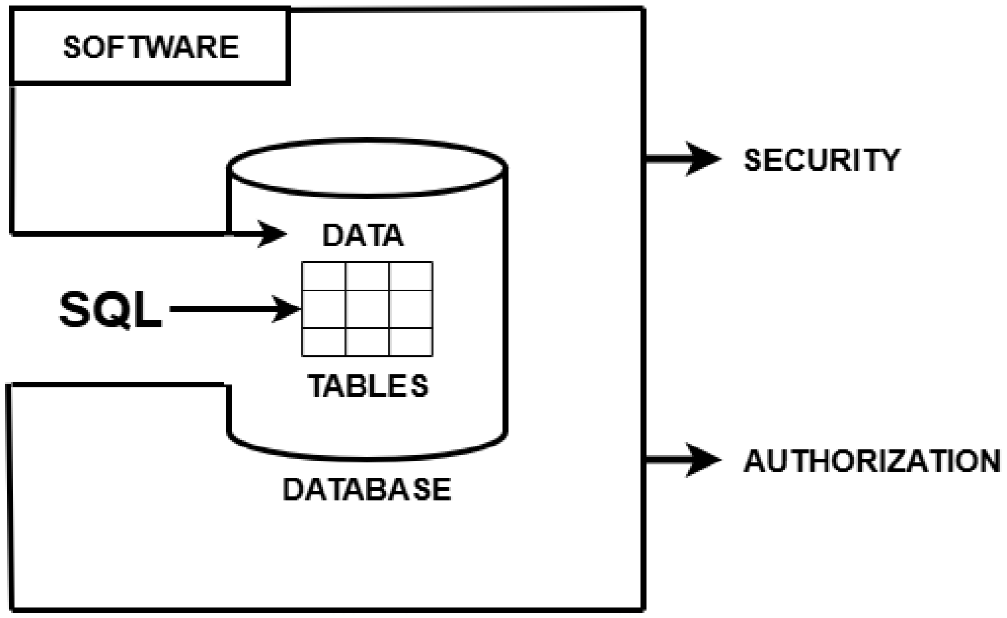
- Database management system is a software which is used to **maintain and manage** the database.



- In DBMS the data will be stored in **file format**.
- DBMS provides two important features such as **security** and **authorization.**
- **Query language** is used to interact between the software.

RELATIONAL DATABASE MANAGEMENT SYSTEMS (RDBMS)

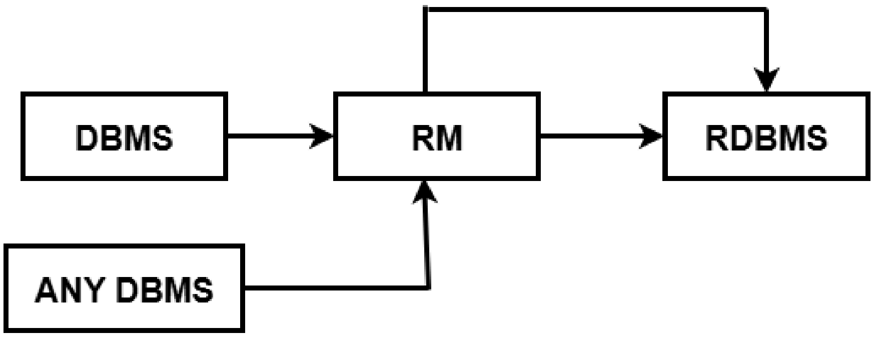
- RDBMS is a type of DBMS. It is a software which is used to **maintain and manage** the database.



- In RDBMS the **data will be stored in table format**.
- RDBMS too provides two important features **security and authorization.**
- **SQL** is used to interact between the software.

RELATIONAL MODEL (RM)

- Relational model was introduced by a data scientist coder “EF CODD”.
- The main rule of Relational Model is the data that we stored should be **stored in table model**.



- **COLUMNS / ATTRIBUTES / FIELD**
- **ROWS / RECORDS / TUPLES**

	CELL		

TABLE

CELL : The **logical intersection** of **rows and columns** which is used to **store data** is known as cell.

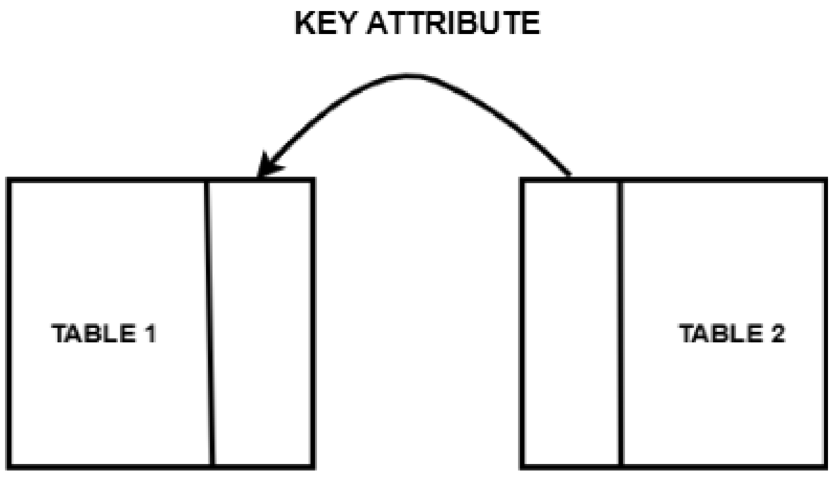
RULES OF EF CODD

RULE 1

- The data that we will store in a cell should be **single value data**. If **we store multi valued data** . The data will end up in **data loss.**

RULE 2

- According to EF CODD we can store data in multiple tables if we want . We can **establish connection between the table by using key attributes.**

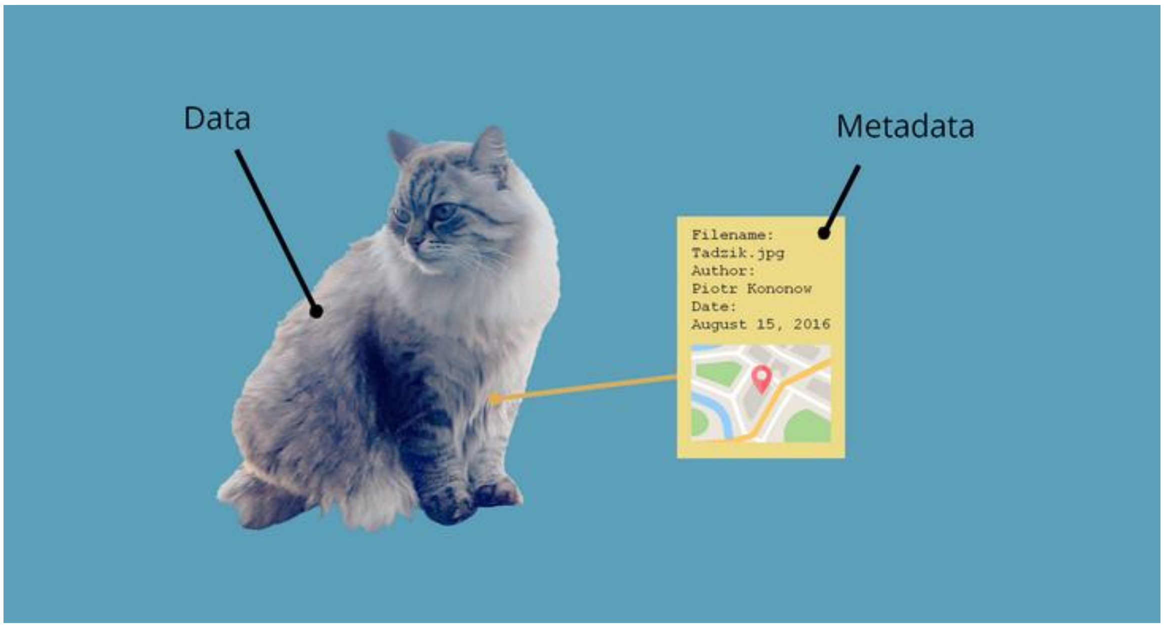


RULE 3

- The data that we store will be **stored in table format** including **meta data** and **meta table**.

META-DATA

- Meta-data is **detailed description of another data**.
- Meta-data is **automatically generated by operating system (OS).**
- Meta-data is **human readable data**.



META-TABLE

- Meta-table is **used to store meta-data**.
- Meta-table is **automatically generated by operating system (OS).**
- Meta-table is **not human readable data**.

RULE 4 :

- Here we can **verify / validate** the data that we are storing by 2 methods
    1. **By assigning DATATYPES.**
    2. **By assigning CONSTRAINTS.**

NOTE : **Datatypes are mandatory , constraints are optional**.

DATA-TYPES

- Data-types are used to determine **what kind of data** we are **going store inside a cell.**
- There are **5 types** of Data-types.

5 TYPES

1. **CHAR**
2. **VARCHAR / VARCHAR2**
3. **DATE**
4. **NUMBER**
5. **LARGE OBJECTS**
    a) **CHARACTER LARGE OBJECT [CLOB].**
    b) **BINARY LARGE OBJECT [BLOB].**

1. CHAR DATATYPE

- Char datatype accepts **'A-Z'** , **'a-z'** , **'0-9'** , **special characters (@ , $, #,…etc).**

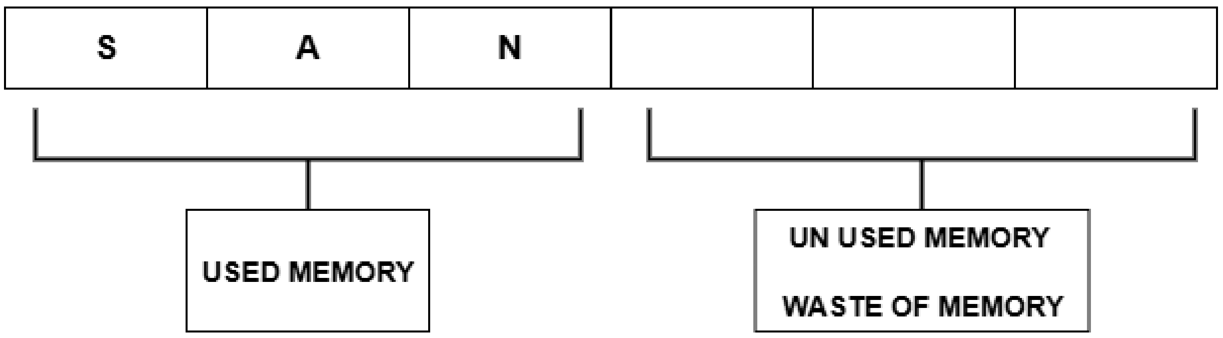
SYNTAX

CHAR(SIZE);

- **Only 2000 Characters are allowed.**

SIZE

- Size is used to **determine the number of character** we are **going to store.**
- CHAR data-type follows " **fixed length memory allocation** ".
- Example : **CHAR(8);**
- Value should be **alphanumeric**.



2. VARCHAR

- Varchar accepts **'A-Z' , 'a-z' , '0-9'** , **special characters (@ , $, # , ….etc)**.
- Varchar data-type can accepts **maximum of 2000** , characters.
- Varchar follows " **variable length memory allocation** ".

SYNTAX

VARCHAR(SIZE)

VARCHAR2

- Varchar2 is **updated version of varchar**. varchar2 can store up to **4000 characters**.

SYNTAX

VARCHAR2(SIZE)

- Example. **VARCHAR2(7)**

3. DATE

SYNTAX

DATE;

- In oracle we have 2 specified data formats.
- They are:
    1. **‘DD-MON-YY**’ (present / current running year).

       Example: ’**06-MAY-25’**
    2. ‘DD-MON-YYYY’ (past / future)

       Example: **’06-MAY-2025**’

4. NUMBER

- This data-type is used to store **only numeric values**.
- [] -> not mandatory.

SYNTAX

NUMBER(PRECISION , [SCALE]);

PRECISION

- Precision is used to **store total number of Integer values**.
- In precision we can store maximum of **38 digits.**

SCALE

- Scale is used to **store number of digits after the decimal point**.
- The Default value of scale is 0.
- Scale is not mandatory.
- In scale we can **store maximum of 127 digits** and **minimum of -84 digits.**
- Example. **NUMBER(5) +/- 9 9 9 9 9**

  a) P > S

  - **NUMBER(5,2) +/- 9 9 9 . 9 9**
  - **NUMBER(7,3) +/- 9 9 9 9 . 9 9 9**

  b) P < S

  - **NUMBER(2,5) +.0 0 0 9 9**

  c) P = S

  - **NUMBER(5,5) +/- .9 9 9 9 9**

5. LARGE OBJECT

1.CHARACTER LARGE OBJECT

- Syntax : **CLOB;**
- In character large object we should store **only characters** such as (‘**A-Z’ , ‘a-z’ , ‘0-9’ , Special characters**).

2.BINARY LARGE OBJECT

- Syntax : **BLOB;**
- In binary large object we should store **only binary values** such as (**mp3 / mp4 , pdf / doc , jpeg / png**).

Note : we can store up to 4GB of data in both.

CONSTRAINTS

- Constraints are set of rules or conditions Assigned for a column.
- In constraints we have 5 types they are :

  1.UNIQUE

  2.NOT NULL

  3.CHECK

  4.PRIMARY KEY

  5.FOREIGN KEY

1.UNIQUE

- Unique is a constraint which is used to **avoid repeated** or **duplicate values** from the column.
- If we assign unique for a column it **consist only unique values**.

2.NOT NULL

- It is a constraint which is used to avoid **null values.**
- If we assign not null for a column it becomes mandatory column.

3.CHECK

- Check is a constraint , it is an extra validation which comes with a condition , if the condition is satisfied , the values will be accepted , else rejected.

4.PRIMARY KEY

- Primary key is a constraint which is used to **identify a record uniquely from the table.**

BEHAVIOUR OF PRIMARY KEY

- One primary key is enough for one table.
- Primary key will **not accept repeated values and null values**.
- Primary key is always a **combination of unique and not null constraints.**
- Primary key is also called as **key attribute.**
- Primary key is not mandatory because primary key is a type of constraint but one primary key is highly recommended for one table.

5.FOREIGN KEY

- Foreign key is a constraint which is **used to establish connection between the tables.**

BEHAVIOUR OF FOREIGN KEY

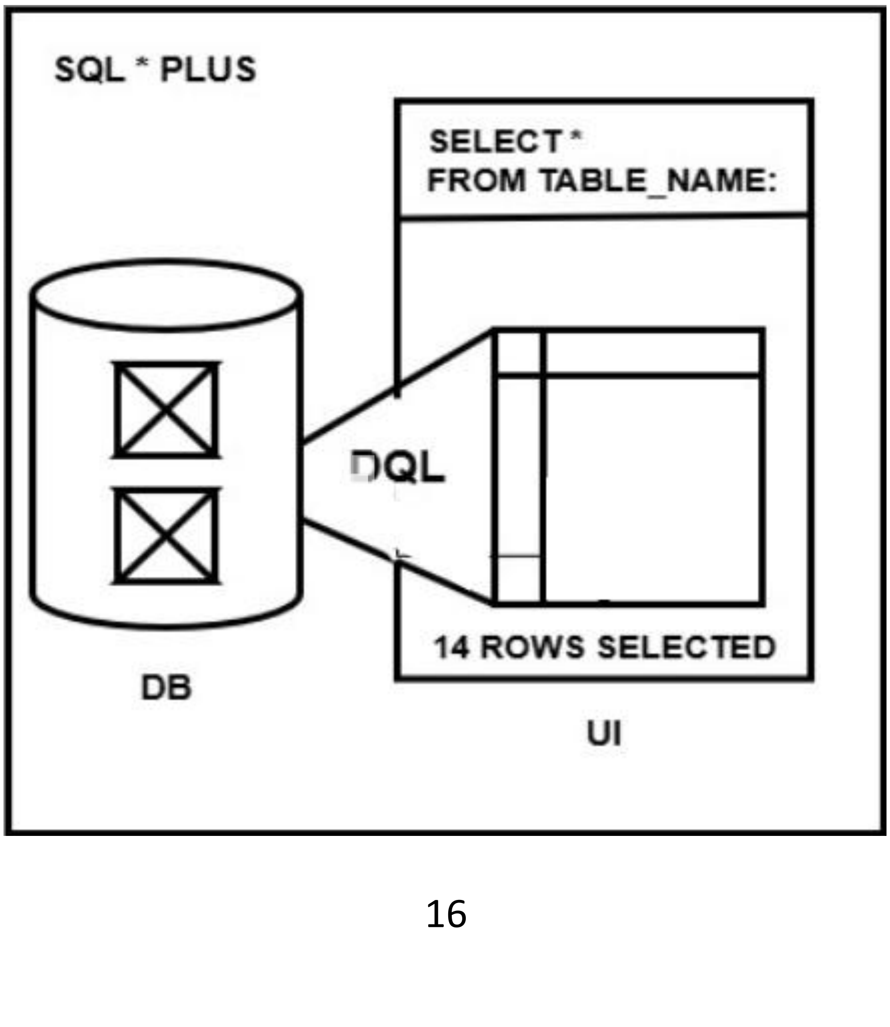
- We can have multiple foreign keys in a table.
- Foreign key **will accept repeated values and null values**.
- Foreign key is **not a combination of unique and not null constraints.**
- Foreign key is also called as **referential integrity constraint**.
- To be a foreign key , it should be primary key in its own or parent table.
- Foreign key is present in child table but it actually belongs to parent table.

OVERVIEW OF SQL

1. **DDL (DATA DEFINITION LANGUAGE)**
2. **DML (DATA MANIPULATION LANGUAGE)**
3. **DQL (DATA QUERY LANGUAGE)**
4. **DCL (DATA CONTROL LANGUAGE)**
5. **TCL (TRANSACTION CONTROL LANGUAGE)**

DATA QUERY LANGUAGE (DQL)

- Data query language is **used to retrieve the data from the database.**



DQL HAS 4 STATEMENTS :

1. **SELECT**
2. **PROJECTION**
3. **SELECTION**
4. **JOIN**

1. SELECT

- It is used to **retrieve the data from the table and display it** .

2. PROJECTION

- It is a **process of retrieving the data from the database by selecting only the columns** is known as projection.

3. SELECTION

- It is a **process of retrieving the data from the database by selecting both the columns and rows** is known as selection.

4. JOIN

- It is a **process of retrieving the data from multiple tables simultaneously** is known as joins.

PROJECTION

- It is a **process of retrieving the data from the database by selecting only the columns** is known as projection.
- In projection all the records / values present in a particular column are by default selected.

SYNTAX FOR PROJECTION

```
SELECT */ [ DISTINCT ] COLUMN_NAME / EXPRESSION [ ALIAS ]
FROM TABLE_NAME;
```

ORDER OF EXECUTION

1. FROM Clause
2. SELECT Clause

[**Clause are in-built function** available to us in SQL]

NOTES:

- FROM clause starts the execution.
- For FROM clause we should pass TABLE_NAME as an argument.
- The job of FROM clause is to go to database SELECT the mentioned TABLE_NAME and keep the table under execution or else end up throwing an error message.

- After the execution of FROM clause , SELECT clause starts the execution .
- For SELECT clause we can pass 3 argument they are:

  1. **ASTERISK (*)**
  2. **COLUMN_NAME**
  3. **EXPRESSION**

- SELECT clause is responsible for preparing RESULT TABLE.
- The output of SELECT clause is also known as RESULT TABLE or RESULT SET.

ASTERISK (*):

- * IS A SINGLE ARGUMENT.
- * IS USED TO DISPLAY ENTIRE TABLE.

SEMI-COLON (;) :

- ; SHOULD BE USED IN THE END OF THE QUERY.
- ; SEMI-COLON IS USED TO STATE THE END OF THE QUERY.

<u>IMPORTANT NOTE</u>

I HAVE USED THE ABBREVIATION " WAQTD " (WRITE A QUERY TO DISPLAY) CONSISTENTLY IN MANY SQL QUESTION FORMATS FOR SIMPLIFICATION.

EXAMPLE:
WAQTD EMPLOYEES NAME FROM EMPLOYEE TABLE .

```
SELECT ENAME
FROM EMP;
```

EMP:

EMPNO	ENAME	SAL	DEPTNO
01	ALLEN	3000	10
02	SMITH	4000	20
03	JONES	2000	30
04	SCOTT	1000	20
05	KING	5000	10

OUTPUT OF THE FROM CLAUSE

RESULT TABLE /SET:

ENAME
ALLEN
SMITH
JONES
SCOTT
KING

OUTPUT OF SELECT CLAUSE / FINAL OUTPUT.

WAQTD EMPLOYEE NO WITH THEIR DEPT NUM FROM EMPLOYEE TABLE.

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

RESULT TABLE / RESULT SET :

EMPNO	DEPTNO
01	10
02	20
03	30
04	40
05	50

```
SET LINES 100 PAGES 100
```

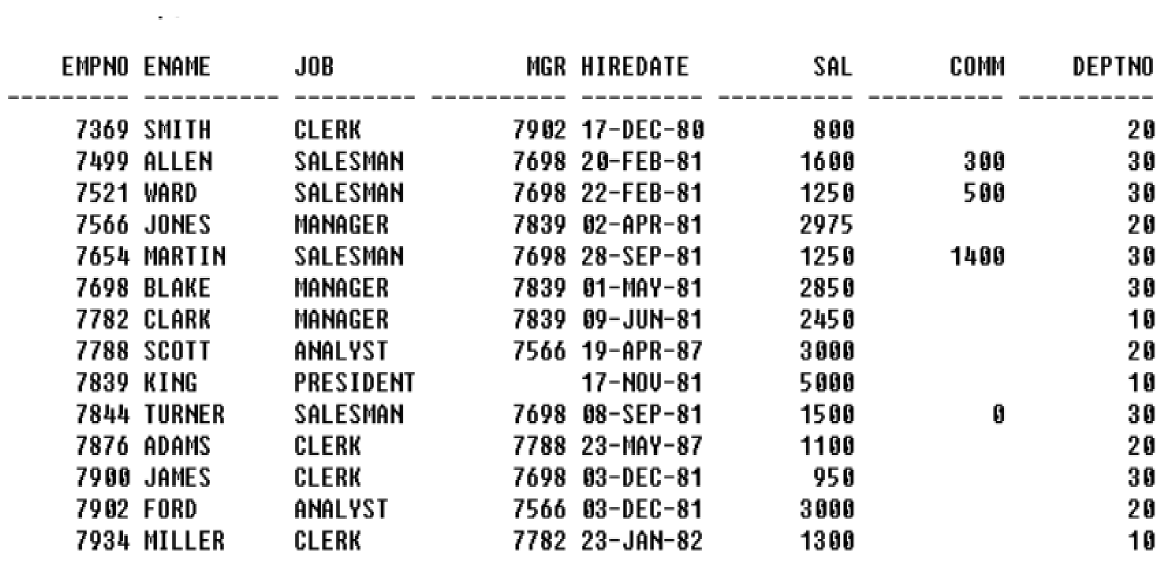
USE THIS COMMAND FIRST WHENEVER PERFORMING QUERIES .
IT WILL AVOID THE RECORDS BEING COLLAPSED.
ONLY USE ONCE WHEN YOU ENTER THE SOFTWARE.

```
CL SCR
```

USE THIS COMMAND FOR CLEARING THE SCREEN.

NOTE : SO , FROM HERE ON WE WILL BE WORKING WITH TABLES AND QUERIES. FOR THAT WE MUST REMEMBER THE TABLE NAMES AND COLUMNS NAMES FOR PRACTICING .THE RESPECTIVE NAMES WILL BE MENTIONED BELOW.

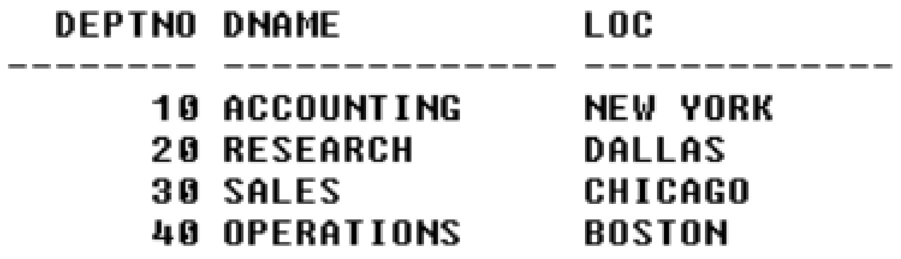
TABLE EMP AND COLUMN NAMES OF EMP TABLE



```
    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. **EMPNO**
2. **ENAME**
3. **JOB**
4. **MGR (REPORTING MANAGER)**
5. **HIREDATE**
6. **SAL**
7. **COMM**
8. **DEPTNO**

TABLE DEPT AND COLUMN NAMES OF DEPT TABLE



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

1. **DEPTNO**
2. **DNAME**
3. **LOC**

QUESTIONS ON EMP AND DEPT TABLE

1. WAQTD ALL THE DETAILS FROM THE EMPLOYEE TABLE.

```
SELECT *
FROM EMP;
```

2. WAQTD NAME OF ALL THE EMPLOYEES.

```
SELECT ENAME
FROM EMP;
```

3. WAQTD NAME AND SALARY GIVEN TO ALL THE EMPLOYEE.

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

4. WAQTD NAME AND COMMISSION GIVEN TO ALL THE EMPLOYEES.

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

5. WAQTD EMPLOYEE ID AND DEPARTMENT NUMBER OF ALL THE EMPLOYEE IN THE TABLE.

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

6. **WAQTD ENAME AND HIREDATE OF ALL THE EMPLOYEE.**

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

7. **WAQTD NAME AND DESIGNATION OF ALL THE EMPLOYEE.**

```
SELECT ENAME , JOB
FROM EMP;
```

8. **WAQTD NAME , JOB AND SALARY GIVEN TO ALL THE EMPLOYEE.**

```
SELECT ENAME , JOB , SALARY
FROM EMP;
```

9. **WAQTD DNAMES PRESENT IN DEPARTMENT TABLE.**

```
SELECT DNAME
FROM DEPT;
```

10. **WAQTD DNAME AND LOCATION PRESENT IN DEPT TABLE.**

```
SELECT DNAME , LOC
FROM DEPT;
```

DISTINCT CLAUSE

- DISTINCT clause is used to **avoid repeated or duplicated values** from the result table.
- DISTINCT clause **should be the first argument** in the select clause.
- For DISTINCT clause we can pass multiple COLUMN_NAMES , DISTINCT clause will check the combination of the columns to consider a repeated value.

QUESTIONS ON DISTINCT CLAUSE

1. **WAQTD DISTINCT SALARY FROM THE EMPLOYEE TABLE.**

```
SELECT DISTINCT SAL
FROM EMP;
```

2. **WAQTD DIFFERENT DEPARTMENT NUMBER FROM EMPLOYEE TABLE.**

```
SELECT DISTINCT DEPTNO
FROM EMP;
```

3. **WAQTD DISTINCT HIREDATE FROM EMPLOYEE TABLE.**

```
SELECT DISTINCT HIREDATE
FRPM EMP;
```

4. **WAQTD UNIQUE EMPS NAMES FROM EMPLOYEE TABLE.**

```
SELECT DISTINCT ENAME
FROM EMP;
```

5. **WAQTD DIFFERENT DESIGNATION FROM EMPLOYEE TABLE.**

```
SELECT DISTINCT JOB
FROM EMP;
```

6. **WAQTD DISTINCT SALARY ALONG WITH DEPTNO.**

```
SELECT DISTINCT SAL , DEPTNO
FROM EMP;
```

7. **WAQTD DIFFERENT EMPLOYEE NO ALONG WITH EMPLOYEE NAME.**

```
SELECT DISTINCT EMPNO , ENAME
FROM EMP;
```

8. **WAQTD DIFFERENT SALARY , COMMISSION ALONG WITH DEPARTMENT NUMBER.**

```
SELECT DISTINCT SAL , COMM , DEPTNO
FROM EMP;
```

9. **WAQTD DISTINCT EMPLOYEE NUMBER , DESIGNATION , EMPLOYEE NAME ALONG WITH THEIR SALARY.**

```
SELECT DISTINCT EMPNO , JOB , ENAME , SAL
FROM EMP;
```

EXPRESSION

- Expression is **responsible for giving an output or value.**
- Expression is a **combination of operands and operators**.

OPERANDS

- Operands are the values which we pass.
- Operands (0 , 1, 2, 3 ,…..,n)

OPERATORS

- Operators are the symbols which perform some operations on operands to generate an output
- Operators (+ , - , * , /)

EXAMPLE :

- **SAL * 12**
- *** IS OPERATOR , SAL AND 12 ARE OPERANDS . ALL THESE COMBINE AND FORM AN EXPRESSION (SAL * 12).**

FORMULA TO CALCULATE SALARY PERCENTAGE

1. **SAL + SAL * A / 100**
   100 + 100 * 10 / 100
   = > 100 + 10 = 110

2. **SAL - SAL * A / 100**
   100 - 100 * 10 / 100
   = > 100 - 10 = 90

FORMULA FOR PERCENTAGE SALARY HIKE

1. **NORMAL SALARY PERCENTAGE HIKE**
   SAL + SAL * A/100

2. **ANNUAL-SALARY PERCENTAGE HIKE**
   SAL * 12 + SAL * 12 * A/100

3. **HALF-TERM SALARY PERCENTAGE HIKE**
   SAL * 6+ SAL *6 * A/100

4. **QUARTER-TERM SALARY PERCENTAGE HIKE**
   SAL * 3 + SAL *3 * A/100

ALIAS

- Alias is used to assign an alternate name or temporary name for column or expression in result table.
- We can assign alias by using “**AS**” keyword.

KEYWORD => AS

- To assign alias as strings (multiple words) we have two formats .
- They are :

1. **SELECT SAL*12 AS ANNUAL_SALARY
FROM EMP;**

2. **SELECT SAL*12 “ANNUAL SALARY”
FROM EMP;**

NOTE:

- We assign alias to make result table COLUMN_NAME more readable and understandable .
- Alias is also known as identifiers.
- Keyword is optional.
- Alias is recommended only for expression. For COLUMN_NAME it is not recommended to use alias , better never use alias for COLUMN_NAME because changing the original COLUMN_NAME is not considered ideal.

TO DISPLAY A COLUMN ALONG WITH WHOLE TABLE

SYNTAX:

```
TABLE_NAME .* , COLUMN_NAME / EXPRESSION
```

- TABLE_NAME DOT ASTERISK COMMA COLUMN_NAME/EXP

EXAMPLE:

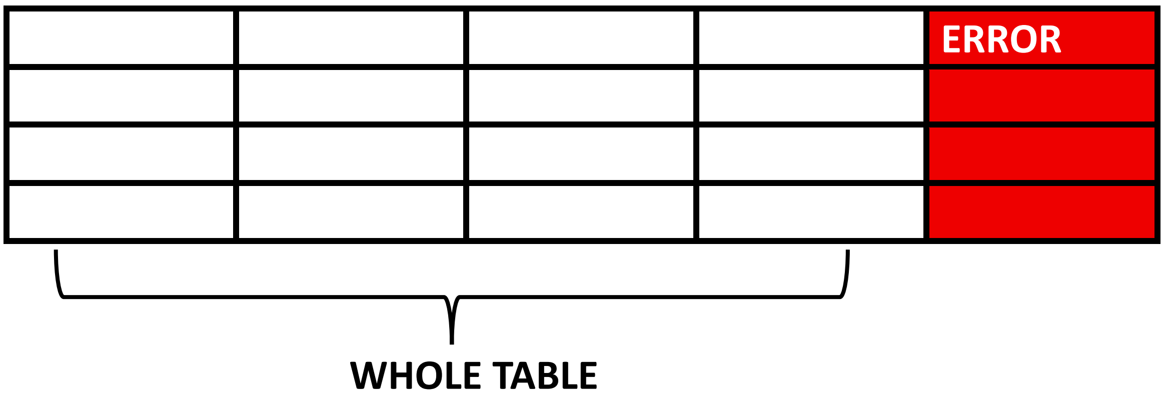
WAQTD DETAILS OF THE EMPS ALONG WITH THE ANNUAL SALARY.

```
SELECT EMP.* , SAL * 12 AS ANNUAL_SALARY
FROM EMP;
```

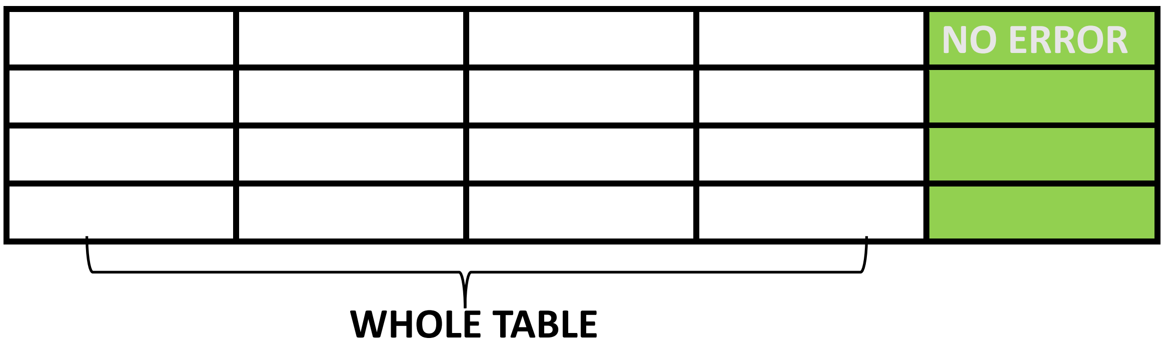
WHAT IF " SELECT * , SAL *12 " IS USED ?

- It will throw error.

1. **ASTERISK (*)**



2. **TABLE_NAME.***



QUESTIONS ON EXPRESSION

1. WAQTD ANNUAL SAL OF THE EMPLOYEE.

```
SELECT SAL*12 AS ANNUAL_SALARY
FROM EMP;
```

2. WAQTD HALF-TERM SALARY OF THE EMPLOYEE.

```
SELECT SAL * 6 AS HALFTERM_SALARY
FROM EMP;
```

3. WAQTD QUARTER-SALARY OF THE EMPLOYEES.

```
SELECT SAL*3 AS QUARTER_SALARY
FROM EMP;
```

4. WAQTD SALARY WITH THE HIKE OF 1000.

```
SELECT SAL+1000 AS SALARY_HIKE
FROM EMP;
```

5. WAQTD SALARY OF PENALTY OF 1000.

```
SELECT SAL-1000 AS SALARY_PENALTY
FROM EMP;
```

6. WAQTD SALARY WITH THE HIKE OF 1250 ALONG WITH ANNUAL-SALARY BONUS OF 2000

```
SELECT SAL+1250 AS SALARY_HIKE ,
 SAL*12+2000 AS ANNUAL_BONUS
FROM EMP;
```

7. WAQTD QUARTER-SALARY WITH THE DEDUCTION OF 1000 , SALARY WITH THE PENALTY 1440.

```
SELECT SAL*3-1000 AS QUARTER_SAL_PENALTY,
SAL-1440 AS SALARY_PENALTY
FROM EMP;
```

8. WAQTD SALARY WITH THE HIKE OF 2500 , ANNUAL-SALARY WITH DEDUCTION OF 1500.

```
SELECT SAL+2500 AS SALARY_HIKE ,
SAL*12-1500 AS ANNUAL_SALARY
FROM EMP;
```

9. WAQTD SALARY WITH THE DEDUCTION OF 9000 , HALF-TERM SALARY WITH THE BONUS OF 600 ALONG WITH ANNUAL HIKE OF 5000.

```
SELECT SAL-9000 AS SALARY_DEDUCTION,
SAL*6+600 AS HALFTERM_BONUS,
SAL*12+5000 AS ANNUAL_HIKE
FROM EMP;
```

10. WAQTD HALF-TERM SALARY WITH THE HIKE OF 1800 , ANNUAL-SALARY WITH THE PENALTY OF 1700 , SALARY WITH DEDUCTION OF 1600 ALONG WITH QUARTER-SALARY HIKE OF 1500.

```
SELECT SAL*6+1800 AS HALFTERM_HIKE,
SAL*12 – 1700 AS ANNUAL_PENALTY,
SAL-1600 AS SALARY_PENALTY,
SAL*3+1500 AS QUATER_HIKE
FROM EMP;
```

11. WAQTD SALARY WITH HIKE OF 15%.

```
SELECT SAL+SAL*15/100 AS SALARY_HIKE
FROM EMP;
```

12. WAQTD SALARY WITH THE DEDUCTION OF 10%.

```
SELECT SAL-SAL*10/100 AS SALARY_PENALTY
FROM EMP;
```

13. WAQTD ANNUAL-SALARY WITH THE HIKE OF 25% , SALARY WITH THE DEDUCTION OF 5%.

```
SELECT SAL*12+SAL*12*25/100 AS ANNUAL_HIKE,
SAL-SAL*5/100 AS SALARY_PENALTY
FROM EMP;
```

14. WAQTD HALF-TERM SALARY WITH THE HIKE OF 15 % , ANNUAL-SALARY WITH DEDUCTION OF 15%.

```
SELECT  SAL*6+SAL*6*15/100 AS  HALFTERM_HIKE,
SAL*12-SAL*12*15/100 AS ANNUAL_PENALTY
FROM EMP;
```

15. WAQTD QUARTER-SALARY WITH THE HIKE OF 6% , HALF-TERM SALARY WITH THE DEDUCTION OF 12% ALONG WITH SALARY HIKE OF 10%;

```
SELECT  SAL*3+SAL*3*6/100 AS QUARTER_HIKE,
SAL*6-SAL*6*12/100 AS HALFTERM_HIKE,
SAL+SAL*10/100 AS SALARY_HIKE
FROM EMP;
```

QUESTIONS ON PROJECTION

1. WAQTD DISTINCT JOB AND SALARY GIVEN TO THE EMPLOYEES

```
SELECT DISTINCT JOB , SAL
FROM EMP;
```

2. WAQTD NAME AND ANNUAL SALARY WITH THE HIKE OF 10%.

```
SELECT ENAME , SAL*12+SAL*12*10/100 AS
ANNUAL_HIKE
FROM EMP;
```

3. WAQTD ALL THE DETAILS OF THE EMPLOYEE ALONG WITH DEPTNO.

```
SELECT EMP.*,DEPTNO
FROM EMP;
```

4. WAQTD NAME , JOB , SALARY WITH A HIKE OF 20%.

```
SELECT ENAME,JOB,SAL+SAL*20/100 AS SALARY_HIKE
FROM EMP;
```

5. WAQTD NAME ,SAL AND SAL WITH DEDUCTION OF 35%.

```
SELECT ENAME,SAL,SAL-SAL*25/100 AS SALARY_PENALTY
FROM EMP;
```

6. **WAQTD DISTINCT NAME AND JOB, HIREDATE OF THE EMPLOYEES.**

```
SELECT DISTINCT ENAME , JOB,HIREDATE
FROM EMP;
```

7. **WAQTD NAME AND SAL WITH THE HIKE OF 500.**

```
SELECT ENAME , SAL +500 AS SALARY_HIKE
FROM EMP;
```

8. **WAQTD ALL THE DETAILS OF THE EMPLOYEE ALONG WITH EMPNO .**

```
SELECT EMP.*,EMPNO
FROM EMP;
```

9. **WAQTD NAME ANNUAL SALARY AND ANNUAL SALARY WITH A DEDUCTION OF 2500.**

```
SELECT ENAME , SAL*12 AS ANNUAL_SALARY,
SAL*12-2500 AS ANNUAL_PENALTY
FROM EMP;
```

10. **WAQTD DISTINCT NAME AND SALARY OF AN EMPLOYEE WITH A HIKE OF 3000.**

```
SELECT DISTINCT ENAME , SAL+3000 AS SALARY_HIKE
FROM EMP;
```

11. WAQTD NAME OF THE EMPLOYEE ALONG WITH THEIR ANNUAL SALARY BONUS OF 3000.

```
SELECT ENAME, SAL*12+3000 AS SALARY_HIKE
FROM EMP;
```

12. WAQTD NAME AND DESIGNATION ALONG WITH 1000 PENALTY IN SALARY.

```
SELECT ENAME,JOB,SAL-1000 AS SALARY_PENALTY
FROM EMP;
```

13. WAQTD ALL THE DETAILS OF THE EMPLOYEES ALONG WITH AN SALARY BONUS OF 5000.

```
SELECT EMP.*,SAL+5000 AS SALARY_HIKE
FROM EMP;
```

14. WAQTD DETAILS OF THE EMP AND SALARY WITH A HIKE OF 15%.

```
SELECT EMP.*,SAL+SAL*15/100
FROM EMP;
```

15. WAQTD DISTINCT DEPTNO AND SALARY WITH DEDUCTION OF 5%.

```
SELECT DISTINCT DEPTNO , SAL-SAL*5/100 AS
SALARY_PENALTY
FROM EMP;
```

16. **WAQTD NAME OF THE EMPLOYEE ALONG WITH THEIR ANNUAL SALARY.**

```
SELECT ENAME , SAL*12 AS ANNUAL_SALARY
FROM EMP;
```

17. **WAQTD ENAME AND JOB FOR ALL THE EMPLOYEE WITH THEIR HALF-TERM SALARY.**

```
SELECT ENAME , JOB , SAL*6 AS HALFTERM_SALARY
FROM EMP;
```

18. **WAQTD ALL THE DETAILS OF THE EMPLOYEES ALONG WITH THE ANNUAL BONUS OF 2000.**

```
SELECT EMP.*,SAL*12+2000 AS ANNUAL_BONUS
FROM EMP;
```

19. **WAQTD NAME SALARY AND SALARY WITH A HIKE OF 10%.**

```
SELECT ENAME , SAL , SAL+SAL*10/100 AS SALARY_HIKE
FROM EMP;
```

20. **WAQTD NAME AND SALARY WITH DEDUCTION OF 25%.**

```
SELECT ENAME , SAL-SAL*25/100 AS SALARY_PENALTY
FROM EMP;
```

21. WAQTD DISTINCT DEPTNO AND JOB FROM EMP TABLE

```
SELECT DISTINCT DEPTNO , JOB
FROM EMP;
```

22. WAQTD NAME AND ANNUAL SALARY WITH DEDUCTION OF 10%.

```
SELECT ENAME , SAL*12-SAL*12*10/100
FROM EMP;
```

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

```
SELECT SAL+COMM AS TOTAL_SALARY
FROM EMP;
```

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

```
SELECT EMP.* , SAL*12 AS ANNUAL_SALARY
FROM EMP;
```

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

```
SELECT ENAME , JOB , SAL-100 AS SALARY_PENALTY
FROM EMP;
```

26. WAQTD NAME OF THE EMP ALONG WITH THEIR HALF-TERM SALARY.

```
SELECT ENAME , SAL*6 AS HALFTERM_SALARY
FROM EMP;
```

27. WAQTD ENAME AND HIREDATE FOR ALL THE EMPLOYEES WITH THEIR ANNUAL SALARY.

```
SELECT ENAME , HIREDATE , SAL*12 AS ANNUAL_SALARY
FROM EMP;
```

28. WAQTD ALL THE DETAILS OF THE EMPLOYEE ALONG WITH AN SALARY BONUS OF 2000.

```
SELECT EMP.* , SAL+2000 AS SALARY-HIKE
FROM EMP;
```

29. WAQTD DETAILS OF THE EMP AND SALARY WITH A HIKE OF 12%.

```
SELECT EMP.* , SAL+SAL*12/100 AS SALARY_HIKE
FROM EMP;
```

30. WAQTD DETAILS OF THE EMP WITH MONTHLY HIKE OF 500.

```
SELECT EMP.* , SAL+500 SALARY_HIKE
FROM EMP;
```

31. WAQTD NAME , SALARY , JOB AND ANNUAL SALARY WITH DEDUCTION OF 15%.

```
SELECT ENAME , SAL , JOB ,
SAL*12-SAL*12*15/100 AS ANNUAL_PENALTY
FROM EMP;
```

32. WAQTD DETAILS OF ALL THE EMP ALONG WITH ANNUAL SALARY AND HAL-TERM SALARY.

```
SELECT EMP.* , SAL*12 AS ANNUAL_SALARY , SAL*6 AS
HALF_SAL
FROM EMP;
```

SELECTION

- It is a **process of retrieving the data from the database by selecting both columns and rows** is known as selection.

SYNTAX FOR SELECTION

```
SELECT */[DISTINCT] COLUMN_NAME / EXPRESSION [ALIAS]
FROM TABLE_NAME
WHERE <FILTER_CONDITION>;
```

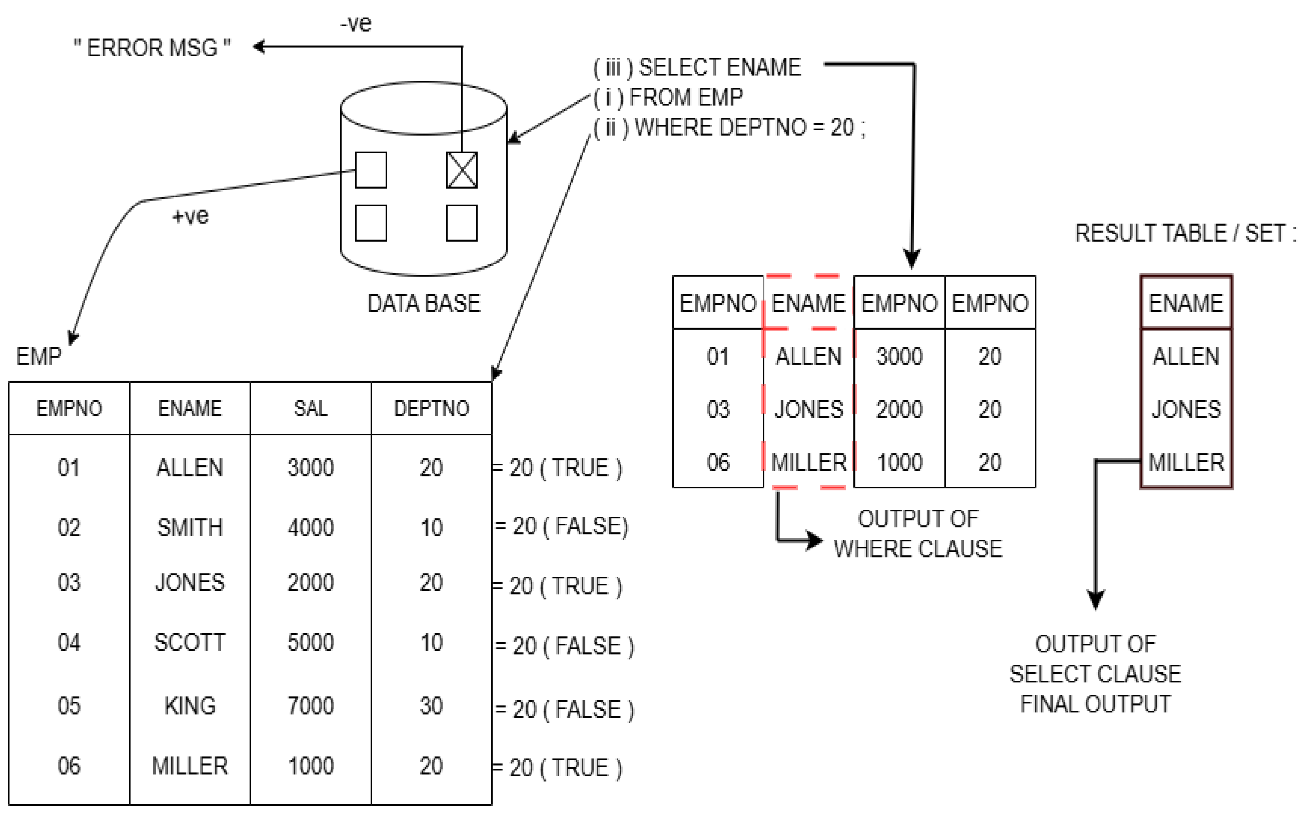
- Order of execution
  1. FROM
  2. WHERE
  3. SELECT

WHERE CLAUSE

- WHERE clause is used to filter the records.
- WHERE clause executes after the execution of FROM clause.
- WHERE clause executes row by row.
- In WHERE clause we should pass a filter condition as argument.
- In WHERE clause we can write conditions which returns a boolean value (true or false).
- In WHERE clause we can pass multiple conditions using logical operators.

EXAMPLE

1 . WAQTD NAMES OF THE EMPS WORKING IN DEPT 20 .



EMPNO	ENAME	SAL	DEPTNO
01	ALLEN	3000	20
02	SMITH	4000	10
03	JONES	2000	20
04	SCOTT	5000	10
05	KING	7000	30
06	MILLER	1000	20

EMPNO	ENAME	EMPNO	EMPNO
01	ALLEN	3000	20
03	JONES	2000	20
06	MILLER	1000	20

ENAME
ALLEN
JONES
MILLER

QUESTIONS ON SELECTION

1. WAQTD THE ANNUAL SALARY OF THE EMPLOYEES WHOSE NAME IS SMITH.

```
SELECT SAL * 12 AS ANNUAL_SALARY
FROM EMP
WHERE ENAME = 'SMITH';
```

2. WAQTD NAME OF THE EMPLOYEE WORKING AS CLERK.

```
SELECT ENAME
FROM EMP
WHERE JOB = 'CLERK';
```

3. WAQTD SALARY OF THE EMPLOYEES WHO ARE WORKING AS SALESMAN.

```
SELECT SAL
FROM EMP
WHERE JOB = 'SALESMAN';
```

4. WAQTD DETAILS OF THE EMPLOYEES WHO EARN MORE THAN 2000.

```
SELECT *
FROM EMP
WHERE SAL > 2000;
```

5. WAQTD DETAIL OF THE EMPLOYEE WHOSE NAME IS JONES.

```
SELECT *
FROM EMP
WHERE ENAME = 'JONES';
```

6. WAQTD DETAILS OF THE EMPLOYEES WHO WAS HIRED AFTER 01-JAN-81.

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

7. WAQTD NAME AND SALARY ALONG WITH HIS ANNUAL SALARY IF THE ANNUAL SALARY IS MORE THAN 12000.

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

8. WAQTD EMPNO OF THE EMPS WORKING IN DEPT 30.

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

9. WAQTD ENAME AND HIREDATE IF THEY ARE HIRED AFTER 1981.

```
SELECT ENAME , HIREDATE
FROM EMP
WHERE HIREDATE > '31-DEC-1981';
```

10. WAQTD DETAILS OF THE EMPLOYEES WORKING AS MANAGER.

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

11. WAQTD NAME AND SALARY GIVEN TO AN EMPLOYEE IF EMPLOYEE EARNS A COMMISSION OF RUPEES 1400.

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

12. WAQTD DETAILS OF THE EMPLOYEES HAVING COMMISSION MORE THAN SALARY

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

13. WAQTD EMPNO OF EMP HIRED BEFORE THE YEAR 87.

```
SELECT EMPNO
FROM EMP
WHERE HIREDATE < ’01-JAN-87’;
```

14. WAQTD DETAILS OF THE EMPLOYEES WORKING AS AN ANALYST

```
SELECT *
FROM EMP
WHERE JOB = ‘ANALYST’;
```

15. WAQTD DETAILS OF EMPLOYEES EARNING MORE THAN 2000 RUPEES PER MONTH.

```
SELECT *
FROM EMP
WHERE SAL > 2000;
```

16. WAQTD DETAILS OF THE EMPLOYEES ALONG WITH ANNUAL SALARY BONUS OF 2000 AND THE EMPLOYEES SHOULD BE HIRED AFTER 1981.

```
SELECT EMP.* , SAL*12 + 2000 AS ANNUAL_BONUS
FROM EMP
WHERE HIREDATE > '31-DEC-1981';
```

17. WAQTD DETAILS OF THE EMPLOYEES ALONG WITH HALF-TERM DEDUCTION OF 1000 AND THE EMPLOYEE SHOULD BE HIRED BEFORE 1995.

```
SELECT EMP.* , SAL*6-1000 AS HALFTERM_PENALTY
FROM EMP
WHERE HIREDATE < '01-JAN-1995';
```

18. WAQTD DETAILS OF THE EMPLOYEES ALONG WITH ANNUAL BONUS OF 2% AND THE EMPLOYEE SHOULD BE HIRED AFTER 2004.

```
SELECT EMP.*,SAL*12+SAL*12*2/100 AS ANNUAL_BONUS
FROM EMP
WHERE HIREDATE > '31-DEC-2004';
```

19. WAQTD DETAILS OF THE EMPLOYEES ALONG WITH ANNUAL DEDUCTION OF 2000 AND THE EMPLOYEES SHOULD BE HIRED BEFORE 1980

```
SELECT EMP.*, SAL*12-2000 AS ANNUAL_PENALTY
FROM EMP
WHERE HIREDATE < '01-JAN-1980';
```

20. WAQTD DETAILS OF THE EMPLOYEES ALONG WITH HALF-TERM HIKE OF 12% AND THE EMPLOYEES SHOULD BE HIRED AFTER 1991.

```
SELECT EMP.* , SAL*6 + SAL*6 * 12/100 AS ANNUAL_HIKE
FROM EMP
WHERE HIREDATE > '31-DEC-1991';
```

21. WAQTD DETAILS OF THE EMPLOYEES HIRED BEFORE 1987.

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

22. WAQTD DETAILS OF THE EMPLOYEES HIRED BEFORE 2015.

```
SELECT *
FROM EMP
WHERE HIREDATE < '01-JAN-2015' ;
```

23. WAQTD DETAILS OF THE EMPS HIRED AFTER 2020.

```
SELECT *
FROM EMP
WHERE HIREDATE  > '31-DEC-2020';
```

24. WAQTD DETAILS OF THE EMPLOYEES HIRED BEFORE 1993.

```
SELECT *
FROM EMP
WHERE HIREDATE < '01-JAN-1993';
```

<u>**OPERATORS IN SQL**</u>

1. ARITHMETIC OPERATORS (+ , - , * , /) .
2. CONCATENATION OPERATOR (||) .
3. COMPARISON OPERATOR (= , != or <>) .
4. RELATIONAL OPERATORS (> , < , >= , <=) .
5. LOGICAL OPERATORS(AND , OR , NOT) .
6. SPECIAL OPERATORS
    1. IN
    2. NOT IN
    3. BETWEEN
    4. NOT BETWEEN
    5. IS
    6. IS NOT
    7. LIKE
    8. NOT LIKE
7. SUB-QUERY OPERATORS
    - ALL
    - ANY

CONCATENATION OPERATOR

- This operator is used to concatenate (||) , a given string to a particular column.

SYMBOL : ||

EXAMPLE:

```
SELECT 'MR.' || ENAME AS EMPLOYEE NAME
FROM EMP;
```

```
SELECT SAL || ' RS ' AS SALARY
FROM EMP;
```

```
SELECT FIRST_NAME || ' ' || LAST _NAME AS FULL_NAME
FROM CUSTOMERS;
```

LOGICAL OPERATORS

- It is used to pass multiple conditions in WHERE clause .
- In logical operators we have 3 types:
    1. AND
    2. OR
    3. NOT

AND OPERATOR

- It is a logical operator which is used to pass multiple conditions in WHERE clause.
- It **consider all conditions** to generate an output.

OR OPERATOR

- OR operator is a logical operator which is used to pass multiple conditions in WHERE clause .
- OR operator **consider any one of the conditions** to generate an output.

NOT OPERATOR

- NOT operator is a logical operator which is used to pass multiple conditions in WHERE clause.
- NOT operator is a **negation operator** (neglecting records).

AND OPERATOR

CONDITION-1	CONDITION-2	RESULT
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE
TRUE	TRUE	TRUE

OR OPERATOR

CONDITION-1	CONDITION-2	RESULT
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE
TRUE	TRUE	TRUE

NOT OPERATOR

CONDITION	RESULT
TRUE	FALSE
FALSE	TRUE

QUESTIONS ON LOGICAL OPERATORS

1. WAQTD DETAILS OF THE EMPLOYEES WORKING AS CLERK AND EARNING LESS THAN 1500.

```
SELECT *
FROM EMP
WHERE JOB = ‘CLERK’ AND SAL < 1500 ;
```

2. WAQTD NAME AND HIREDATE OF THE EMPLOYEES WORKING AS MANAGER IN DEPT 30.

```
SELECT ENAME , HIREDATE
FROM EMP
WHERE JOB = ‘MANAGER’ AND DEPTNO = 30 ;
```

3. WAQTD DETAILS OF THE EMPLOYEES ALONG WITH ANNUAL SALARY IF THEY ARE WORKING IN DEPT 30 AS SALESMAN AND THEIR ANNUAL SALARY HAS TO BE GREATER THAN 14000.

```
SELECT EMP.* , SAL*12 AS ANNUAL_SALARY
FROM EMP
WHERE DEPTNO = 30 AND JOB = ‘SALESMAN’
AND SAL*12 > 14000 ;
```

4. WAQTD ALL THE DETAILS OF THE EMPLOYEES WORKING IN DEPT 10 OR AS ANALYSTS.

```
SELECT *
FROM EMP
WHERE DEPTNO = 10 OR JOB = ‘ANALYST’ ;
```

5. WAQTD NAMES OF THE EMPLOYEES WHOSE SALARY IS LESS THAN 1100 AND THEIR DESIGNATION IS CLERK.

```
SELECT ENAME
FROM EMP
WHERE SAL < 1100 AND JOB = ‘CLERK’ ;
```

6. WAQTD NAME AND SAL, ANNUAL SAL AND DEPTNO IF DEPTNO IS 20 EARNING MORE THAN 1100 AND ANNUAL SAL EXCEEDS 1200.

```
SELECT ENAME  , SAL  ,
SAL*12 AS ANNUAL_SALARY  ,  DEPTNO
FROM EMP
WHERE DEPTNO =20 AND SAL > 1100 AND SAL*12 > 1200;
```

7. WAQTD EMPNO AND NAMES OF THE EMPLOYEES WORKING AS MANAGER IN DEPT 20.

```
SELECT EMPNO , ENAME
FROM EMP
WHERE JOB = ‘MANAGER’ AND DEPTNO = 20;
```

8. WAQTD DETAILS OF EMPS WORKING IN DEPT 20 OR 30.

```
SELECT *
FROM EMP
WHERE DEPTNO = 20 OR DEPTNO = 30 ;
```

9. WAQTD DETAILS OF EMPS WORK AS ANALYST IN DEPT 10.

```
SELECT *
FROM EMP
WHERE JOB = ‘ANALYST’ AND DEPTNO = 10 ;
```

10. WAQTD DETAILS OF EMPLOYEES WORKING AS PRESIDENT WITH SALARY OF RUPEES 4000.

```
SELECT *
FROM EMP
WHERE JOB = ‘PRESIDENT’ AND SAL=4000;
```

11. WAQTD DETAILS OF EMPLOYEES EXCEPT THE EMPS WORKING AS PRESIDENT.

```
SELECT *
FROM  EMP
WHERE JOB <> ‘PRESIDENT’  ;
```

12. WAQTD DETAILS OF EMPLOYEES EXCEPT THE EMPLOYEES WORKING IN DEPT 10 OR 20.

```
SELECT *
FROM EMP
WHERE DEPTNO <> 10 AND DEPTNO <> 20 ;
```

13. WAQTD NAMES AND DEPTNO, JOB OF EMPLOYEES WORKING AS CLERK IN DEPT 10 OR 20.

```
SELECT ENAME , DEPTNO , JOB
FROM EMP
WHERE JOB = ‘CLERK’
AND (DEPTNO =10 OR DEPTNO = 20);
```

14. WAQTD DETAILS OF EMPLOYEES WORKING AS CLERK OR MANAGER IN DEPT 10.

```
SELECT *
FROM EMP
WHERE ( JOB = 'CLERK' OR JOB = 'MANAGER' )
AND DEPTNO = 10;
```

15. WAQTD NAMES OF EMPLOYEES WORKING IN DEPT 10, 20 , 30 , 40.

```
SELECT ENAME
FROM EMP
WHERE DEPTNO = 10 OR DEPTNO = 20
OR DEPTNO = 30 OR DEPTNO = 40;
```

16. WAQTD DETAILS OF EMPLOYEES WITH EMPNO 7902 OR 7839.

```
SELECT *
FROM EMP
WHERE EMPNO = 7902 OR EMPNO = 7839 ;
```

17. WAQTD DETAILS OF EMPLOYEES WORKING AS MANAGER OR SALESMAN OR CLERK.

```
SELECT *
FROM EMP
WHERE JOB = 'MANAGER' OR JOB = 'SALESMAN'
OR JOB = 'CLERK' ;
```

18. WAQTD NAMES OF EMPLOYEES HIRED AFTER 81 AND BEFORE 87.

```
SELECT ENAME
FROM EMP
WHERE HIREDATE > ’31-DEC-81’
AND HIREDATE < ’01-JAN-87’;
```

19. WAQTD DETAILS OF THE EMPLOYEES EARNING MORE THAN 1250 BUT LESS THAN 3000.

```
SELECT *
FROM EMP
WHERE SAL > 1250 AND SAL < 3000 ;
```

20. WAQTD NAMES OF THE EMPLOYEES HIRED AFTER 81 INTO DEPT 10 OR 30.

```
SELECT ENAME
FROM EMP
WHERE HIREDATE > ’31-DEC-81’
AND DEPTNO=10 OR DEPTNO=30;
```

21. WAQTD NAMES OF EMPLOYEES ALONG WITH ANNUAL SALARY OF THE EMPLOYEES WORKING AS MANAGER OR CLERK INTO DEPT 10 OR 30.

```
SELECT ENAME , SAL*12 AS ANNUAL_SALARY
FROM EMP
WHERE JOB =  ‘MANAGER’ OR JOB = ‘CLERK’
AND DEPTNO=10 OR DEPTNO=30 ;
```

SPECIAL OPERATORS

- Special operators are used to optimize or compress filter conditions in WHERE clause .
- In special operators we have 8 types :

  1. **IN**
  2. **NOT IN**
  3. **BETWEEN**
  4. **NOT BETWEEN**
  5. **IS**
  6. **IS NOT**
  7. **LIKE**
  8. **NOT LIKE**

IN OPERATOR

- IN operator is a special operator , it is a multi valued operator which can accept multiple values on RHS .

SYNTAX

COLUMN_NAME / EXPRESSION IN (V_1 , V_2 , V_3 , …., V_N) ;

NOT IN OPERATOR

- NOT IN operator is a special operator , it is similar to the IN operator , but instead of accepting multiple values it rejects multiple values .

SYNTAX

COLUMN_NAME / EXPRESSION NOT IN (V_1 , V_2 , V_3 , ..., V_N) ;

NOTE : THIS OPERATORS ARE SIMILAR AS COMPARISON OPERATOR (=);

QUESTIONS ON IN AND NOT IN OPERATORS

1. **WAQTD DETAILS OF THE EMPLOYEES WORKING IN DEPT 10 , 20.**

```
SELECT *
FROM EMP
WHERE DEPTNO IN (10,20);
```

2. **WAQTD DETAILS OF THE EMPLOYEES EXCEPT THE EMPLOYEES WORKING IN DEPTNO 10 , 20.**

```
SELECT *
FROM EMP
WHERE  DEPTNO NOT IN (10,20);
```

3. **WAQTD DETAILS OF THE EMPLOYEES WORKING AS CLERK OR SALESMAN.**

```
SELECT *
FROM EMP
WHERE JOB IN (‘CLERK’ , ‘SALESMAN’);
```

4. **WAQTD DETAILS OF THE EMPLOYEES EXCEPT THE EMPLOYEES WORKING AS CLERK OR SALESMAN.**

```
SELECT *
FROM EMP
WHERE JOB NOT IN (‘CLERK’ , ‘SALESMAN’);
```

5. **WAQTD DETAILS OF EMPLOYEES EXCEPT THE EMPLOYEES WORKING IN DEPT 10, 20 AND THE EMPLOYEES SHOULD BE WORKING AS CLERK , SALESMAN.**

```
SELECT *
FROM EMP
WHERE DEPTNO NOT IN (10 , 20 )
AND JOB IN(‘CLERK’ , ‘SALESMAN’);
```

6. **WAQTD DETAILS OF EMPLOYEES EXCEPT THE EMPLOYEES WORKING AS CLERK OR SALESMAN AND THE EMPLOYEES SHOULD BE WORKING IN DEPT 10, 20.**

```
SELECT *
FROM EMP
WHERE JOB NOT IN (‘CLERK’ , ‘SALESMAN’)
AND DEPTNO IN (10 , 20 );
```

BETWEEN OPERATOR

- BETWEEN operator is a special operator , whenever we have ranges of values we use BETWEEN operator .

SYNTAX

```
COLUMN_NAME BETWEEN ' LOWER_RANGE ' AND
'HIGHER_RANGE' ;
```

NOT BETWEEN

- NOT BETWEEN operator is a special operator, it is similar to the BETWEEN operator but instead of accepting the range , it excludes it .

SYNTAX

```
COLUMN_NAME NOT BETWEEN ' LOWER_RANGE ' AND '
HIGHER_RANGE ' ;
```

NOTE:

RANGE (STARTING VALUE AND ENDING VALUE)

S.NO	CASES	+1 AND -1 METHOD
01	BETWEEN	MANDATORY
02	FROM-TO	NOT MANDATORY
03	DURING	NOT MANDATORY

QUESTIONS ON BETWEEN AND NOT BETWEEN OPERATORS

BETWEEN CASES

1. WAQTD DETAILS OF EMPS HIRED BETWEEN 2004 AND 2008.

```
SELECT *
FROM EMP
WHERE HIREDATE BETWEEN '01-JAN-2005'
AND '31-DEC-2007';
```

2. WAQTD DETAILS OF EMPLOYEES EXCEPT THE EMPLOYEES HIRED AFTER 2004 AND BEFORE 2008.

```
SELECT *
FROM EMP
WHERE HIREDATE NOT BETWEEN '01-JAN-2005'
AND '31-DEC-2007';
```

3. WAQTD DETAILS OF THE EMPLOYEES EXCEPT THE EMPLOYEES HIRED BETWEEN 2012 AND 2016.

```
SELECT *
FROM EMP
WHERE HIREDATE NOT BETWEEN '01-JAN-2013'
AND '31-DEC-2015';
```

FORM-TO CASES

1. WAQTD DETAILS OF THE EMPLOYEES HIRED FROM 2004 TO 2008.

```
SELECT *
FROM EMP
WHERE HIREDATE BETWEEN ’01-JAN-2004’
AND ’31-DEC-2008’;
```

2. WAQTD DETAILS OF EMPLOYEES EXCEPT THE EMPLOYEES HIRED FROM 2004 TO 2008.

```
SELECT *
FROM EMP
WHERE HIREDATE NOT BETWEEN ’01-JAN-2004’
AND ’31-DEC-2008’;
```

3. WAQTD DETAILS OF THE EMPLOYEES HIRED FROM 2012 TO 2016.

```
SELECT *
FROM EMP
WHERE HIREDATE BETWEEN ’01-JAN-2012’
AND ’31-DEC-2016’;
```

4. WAQTD DETAILS OF THE EMPLOYEES EXCEPT THE EMPLOYEES HIRED FROM 2012 TO 2016.

```
SELECT *
FROM EMP
WHERE HIREDATE NOT BETWEEN ’01-JAN-2012’
AND ’31-DEC-2016’;
```

DURING CASES

1. WAQTD DETAILS OF THE EMPLOYEES HIRED DURING 2004.

```
SELECT *
FROM EMP
WHERE HIREDATE BETWEEN ’01-JAN-2004’
AND ’31-DEC-2004’ ;
```

2. WAQTD DETAILS OF THE EMPLOYEES EXCEPT THE EMPLOYEES HIRED DURING 2004.

```
SELECT *
FROM EMP
WHERE HIREDATE NOT BETWEEN ’01-JAN-2004’
AND ’31-DEC-2004’ ;
```

3. WAQTD DETAILS OF THE EMPLOYEES HIRED DURING 2020.

```
SELECT *
FROM EMP
WHERE HIREDATE BETWEEN ’01-JAN-2020’
AND ’31-DEC-2020’ ;
```

4. WAQTD DETAILS OF EMPLOYEES EXCEPT THE EMPLOYEES HIRED DURING PREVIOUS YEAR (2024).

```
SELECT *
FROM EMP
WHERE HIREDATE NOT BETWEEN ’01-JAN-2024’
AND ’31-DEC-2024’;
```

IS OPERATOR

- IS operator is a special operator , it is used to compare values with NULL .

SYNTAX

```
COLUMN_NAME IS NULL ;
```

IS NOT OPERATOR

- IS NOT operator is a special operator , it is used to check if a column is not NULL .

SYNTAX

```
COLUMN_NAME IS NOT NULL ;
```

QUESTIONS ON IS AND IS NOT OPERATORS

1. **WAQTD DETAILS OF THE EMPLOYEES WHO DON'T EARN COMMISSION.**

```
SELECT *
FROM EMP
WHERE COMM IS NULL;
```

2. WAQTD DETAILS OF THE EMPLOYEES WHO EARN COMMISSION.

```
SELECT *
FROM EMP
WHERE COMM IS NOT NULL;
```

3. WAQTD DETAILS OF THE EMPLOYEES WHO HAVE A REPORTING MANAGER.

```
SELECT *
FROM EMP
WHERE MGR IS NOT NULL;
```

4. WAQTD DETAILS OF THE EMPLOYEES WHO DON'T EARN COMMISSION, BUT THEY SHOULD HAVE A REPORTING MANAGER.

```
SELECT *
FROM EMP
WHERE COMM IS NULL AND MGR IS NOT NULL;
```

5. WAQTD DETAILS OF THE EMPLOYEES WHO EARN COMMISSION BUT THEY SHOULD NOT HAVE A REPORTING MANAGER.

```
SELECT *
FROM EMP
WHERE COMM IS NOT NULL AND MGR IS NULL ;
```

LIKE OPERATOR

- LIKE operator is a special operator , it is used to perform pattern matching .

SYNTAX

```
COLUMN_NAME LIKE ' PATTERN ' ;
```

NOT LIKE

- NOT LIKE operator is a special operator , it is similar to the LIKE operator , but instead of matching the pattern it rejects the pattern .

SYNTAX

```
COLUMN_NAME NOT LIKE ' PATTERN ' ;
```

- To achieve pattern matching we have to use wildcard characters.

1. Percent sign (%) :
    - It accepts any character (' A-Z ' , ' a-z ' ,' 0-9') .
    - It accepts any number of times and accepts blank spaces.
2. Under Score (_) :
    - It accepts only one character .
    - It accepts only one time and accepts blank space.

QUESTIONS ON LIKE AND NOT LIKE OPERATORS

1. WAQTD NAMES OF THE EMPLOYEES STARTING WITH THE CHARACTER ‘S’.

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE ‘S%’ ;
```

2. WAQTD NAMES OF THE EMPLOYEES, EXCEPT THE ONES STARTING WITH THE CHARACTER ‘S’.

```
SELECT ENAME
FROM EMP
WHERE ENAME NOT LIKE ‘S%’ ;
```

3. WAQTD NAME OF THE EMPLOYEES ENDING WITH ‘S’.

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE ‘%S’ ;
```

4. WAQTD NAMES OF THE EMPLOYEES EXCEPT THE NAMES ENDING WITH CHARACTER ‘S’.

```
SELECT ENAME
FROM EMP
WHERE ENAME NOT LIKE ‘%S’ ;
```

5. WAQTD NAME OF THE EMPLOYEES STARTING WITH CHARACTER ‘J’ AND ENDING WITH CHARACTER ‘S’.

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE ‘J%S’;
```

6. WAQTD NAME OF THE EMPLOYEES EXCEPT THE NAME STARTING WITH CHARACTER ‘J’ AND ENDING WITH ‘S’.

```
SELECT ENAME
FROM EMP
WHERE ENAME NOT LIKE ‘J%S’;
```

7. WAQTD NAMES OF THE EMPLOYEES HAVING ‘A’.

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE ‘%A%’ ;
```

8. WAQTD NAMES OF THE EMPLOYEES EXCEPT THE NAMES HAVING CHARACTER ‘A’.

```
SELECT ENAME
FROM EMP
WHERE ENAME NOT LIKE ‘%A%’ ;
```

9. WAQTD NAMES OF THE EMPLOYEES HAVING ‘A’ AS THE THIRD CHARACTER.

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE ‘__A’;
```

10. WAQTD NAMES OF EMPLOYEES EXCEPT THE NAMES HAVING ‘A’ AS THE THIRD CHARACTER.

```
SELECT ENAME
FROM EMP
WHERE ENAME NOT LIKE ‘__A’;
```

11. WAQTD HIREDATE OF THE EMPLOYEES HIRED IN THE FIRST MONTH OF THE YEAR.

```
SELECT HIREDATE
FROM EMP
WHERE HIREDATE LIKE ‘%JAN%’;
```

12. WAQTD HIREDATE OF THE EMPLOYEES EXCEPT THE EMPLOYEES HIRED IN THE FIRST MONTH OF THE YEAR.

```
SELECT HIREDATE
FROM EMP
WHERE HIREDATE NOT LIKE ‘%JAN%’;
```

13. WAQTD HIREDATE OF THE EMPLOYEES HIRED IN THE LAST MONTH OF THE YEAR.

```
SELECT HIREDATE
FROM EMP
WHERE HIREDATE LIKE ‘%DEC%’;
```

14. WAQTD HIREDATE OF THE EMPLOYEES HIRED IN THE PREVIOUS MONTH.

```
SELECT HIREDATE
FROM EMP
WHERE HIREDATE LIKE ‘%JUN%’;
```

15. WAQTD THREE-DIGIT SALARY.

```
SELECT SAL
FROM EMP
WHERE SAL LIKE ‘___’;
```

16. WAQTD SALARIES WHICH ARE NOT THREE-DIGIT.

```
SELECT SAL
FROM EMP
WHERE SAL NOT LIKE '___';
```

17. WAQTD FOUR-DIGIT SALARY WHICH IS ENDING WITH 50.

```
SELECT SAL
FROM EMP
WHERE SAL LIKE '__50';
```

18. WAQTD SALARY WHICH IS STARTING WITH 30.

```
SELECT SAL
FROM EMP
WHERE SAL LIKE '30%';
```

19. WAQTD SALARY OF THE EMPLOYEES WHICH SHOULD NOT END WITH 00.

```
SELECT SAL
FROM EMP
WHERE SAL NOT LIKE '%00';
```

ESCAPE CHARACTER

- It is used to **remove the special behaviour of wildcard character** and treat it as **normal** character .
- We have to define the escape character .
- It should be written before the special character .
- Recommended escape characters are (@ , $, & , # , /) .

SYNTAX

```
COLUMN_NAME / EXPRESSION LIKE / NOT LIKE '
PATTERN_TO_MATCH ' ESCAPE ' SPECIAL_CHARACTER' ;
```

EXAMPLE:

WAQTD NAMES OF THE EMPS HAVING PERCENTILE (%) IN THEM.

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE ‘%$%%’ ESCAPE ‘$’ ;
```

EXPLANATION FOR ‘%$%%’

```
% -> WILDCARD CHARACTER
$ -> ESCAPE CHARACTER / SPECIAL CHARACTER
% -> WILDCARD CHARACTER TO NORMAL CHARACTER
% -> WILDCARD CHARACTER
```

QUESTIONS ON SPECIAL OPERATORS

1. WAQTD DETAILS OF THE EMPLOYEES WORKING IN DEPARTMENT 10 OR 30.

```
SELECT *
FROM EMP
WHERE DEPTNO IN (10,30);
```

2. WAQTD NAME OF THE EMPLOYEES HIRED DURING 1982.

```
SELECT ENAME
FROM EMP
WHERE HIREDATE BETWEEN ’01-JAN-82’
 AND ’31-DEC-82’;
```

3. WAQTD NAME AND SALARY GIVEN TO EMPLOYEES EARNING COMMISSION.

```
SELECT ENAME , SAL
FROM EMP
WHERE COMM IS NOT NULL ;
```

4. WAQTD DETAILS OF THE EMPLOYEES WORKING AS CLERK IN DEPARTMENT 10 OR 30 HAVING CHARACTER ‘A’ IN THEIR NAMES.

```
SELECT *
FROM EMP
WHERE JOB IN ‘CLERK’ AND DEPTNO IN (10,30)
AND ENAME LIKE ‘%A%’;
```

5. WAQTD NAMES OF THE EMPLOYEES HAVING CHARACTER 'S' AS THEIR LAST CHARACTER.

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE '%S' ;
```

6. WAQTD NAME AND HIRE DATE OF THE EMPLOYEES HIRED AFTER 1982 BUT BEFORE 1987.

```
SELECT ENAME , HIREDATE
FROM EMP
WHERE HIREDATE BETWEEN '01-JAN-1983'
AND '31-DEC-1988' ;
```

7. WAQTD DETAILS OF THE EMPLOYEES WORKING AS ANALYST AND EARNING 4-DIGIT SALARY.

```
SELECT *
FROM EMP
WHERE JOB IN 'ANALYST' AND SAL LIKE '____';
```

8. WAQTD NAMES OF THE EMPS HIRED IN THE FIRST MONTH.

```
SELECT ENAME
FROM EMP
WHERE HIREDATE LIKE '%JAN%';
```

9. WAQTD DETAILS OF THE EMPLOYEES WORKING AS SALESMAN AND DO NOT EARN ANY COMMISSION.

```
SELECT *
FROM EMP
WHERE JOB IN 'SALESMAN' AND COMM IS NULL;
```

ORDER BY CLAUSE

- It is used to arrange the records either in ascending or descending order .

SYNTAX

```
SELECT GROUP_BY_EXPRESSION / GROUP_FUNCTION
FROM TABLE_NAME
WHERE <FILTER_CONDITION>
GROUP BY COLUMN_NAME / EXPRESSION ->( IF USED )
HAVING [ < GROUP_FILTER_CONDITIONS> ] -> ( IF USED )
ORDER BY COLUMN_NAME [ASC / DESC] ; -> (IF USED )
```

ORDER OF EXECUTION

1. **FROM -- ROW BY ROW**
2. **WHERE -- ROW BY ROW**
3. **GROUP BY (IF USED) -- GROUP BY GROUP**
4. **HAVING (IF USED) -- GROUP BY GROUP**
5. **SELECT -- GROUP BY GROUP**
6. **ORDER BY -- ROW BY ROW**

COMMENTS :

- It executes row by row .
- By default ORDER BY clause arranges the records in ascending order.
- ORDER BY clause is the only clause which is written at the ending of the query.
- ORDER BY clause is the only clause which executes at the last

QUESTIONS ON ORDER BY CLAUSE

1. WAQTD DETAILS OF EMPLOYEES HAVING 2 LS PRESENT IN THEIR NAMES AND WORKING AS MANAGERS, ARRANGED BY EMPLOYEE NUMBER.

```
SELECT *
FROM EMP
WHERE ENAME LIKE ‘%L%L’ AND JOB = ‘MANAGER’
ORDER BY EMPNO;
```

2. WAQTD NAMES OF EMPLOYEES WHO EARNED SALARY BUT NOT COMMISSION, ARRANGED BY DEPARTMENT NUMBER IN DESCENDING ORDER.

```
SELECT ENAME
FROM EMP
WHERE SAL IS NOT NULL AND COMM IS NULL
ORDER BY DEPTNO DESC;
```

3. WAQTD NAMES OF THE EMPLOYEES IF NAME STARTS WITH A OR J, ARRANGED BY EMPLOYEE NAME.

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE ‘A%’ OR ENAME LIKE ‘J%’
ORDER BY ENAME;
```

4. WAQTD LIST ALL THE EMPLOYEE NAMES EXCEPT FOR THE EMPLOYEES WHOSE NAME HAS ‘A’ AS THE THIRD CHARACTER, ARRANGED BY SALARY IN DESCENDING ORDER.

```
SELECT ENAME
FROM EMP
WHERE ENAME NOT LIKE ‘__A’
ORDER BY SAL DESC;
```

5. WAQTD LIST THE DETAILS OF THE EMPLOYEES WORKING AS MANAGER, HIRED AFTER 1984, AND HAVING A NAME WHICH ENDS WITH ‘S’, ARRANGED BY HIRE DATE.

```
SELECT *
FROM EMP
WHERE JOB=’MANAGER’ AND HIREDATE > ’31-DEC-84’
AND ENAME LIKE ‘%S’
ORDER BY HIREDATE;
```

QUESTIONS ON SPECIAL OPERATORS AND ORDER BY CLAUSE

1. WAQTD DETAILS OF EMPLOYEES HAVING CHARACTER 'N' PRESENT IN THEIR NAMES AND WORKING AS SALESMAN, ARRANGED BY NAME.

```
SELECT *
FROM EMP
WHERE ENAME LIKE ‘%N%’ AND JOB IN ‘SALESMAN’
ORDER BY ENAME;
```

2. **WAQTD DETAILS OF THE EMPLOYEES IF NAME STARTS WITH 'S' OR 'N', ARRANGED BY HIRE DATE.**

```
SELECT *
FROM EMP
WHERE ENAME LIKE ‘S%’ AND ENAME LIKE ‘N%’
ORDER BY HIREDATE;
```

3. **WAQTD LIST ALL THE EMPLOYEE NAMES EXCEPT FOR EMPLOYEES WHOSE NAME HAS 'L' AS THE SECOND CHARACTER, ARRANGED BY HIRE DATE IN DESCENDING ORDER.**

```
SELECT ENAME
FROM EMP
WHERE ENAME NOT LIKE ‘_L’
ORDER BY HIREDATE DESC ;
```

4. **WAQTD LIST THE DETAILS OF THE EMPLOYEES WORKING AS MANAGER, HIRED DURING 1981, AND HAVE 'A' IN THEIR NAMES, ARRANGED BY HIRE DATE.**

```
SELECT *
FROM EMP
WHERE JOB IN ‘MANAGER’
AND HIREDATE BETWEEN ’01-JAN-81’ AND ’31-DEC-81’
AND ENAME LIKE ‘%A%’
ORDER BY HIREDATE;
```

FUNCTIONS OR METHODS

- Functions are a list of instructions which is used to perform a specific task.
- There are 3 main components in functions:

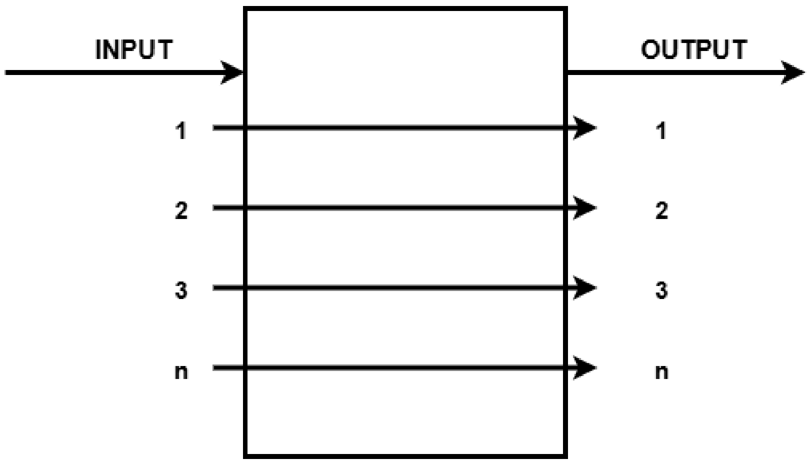
    1. Function name
    2. Number of arguments (input)
    3. Return type

- We can classify functions in 2 types.
- They are:

    1. Single row function ()
    2. Multi row function () or [group function ()] or [aggregate function ()] .

SINGLE ROW FUNCTION ()

- Single row function accepts first input , executes to generate an output , and so on.
- Single row function: If we pass n number of inputs, it returns n number of outputs.



1. Function name : LENGTH()
2. No.of argument (Input) : ENAME
3. Return type : No.of input = No.of output

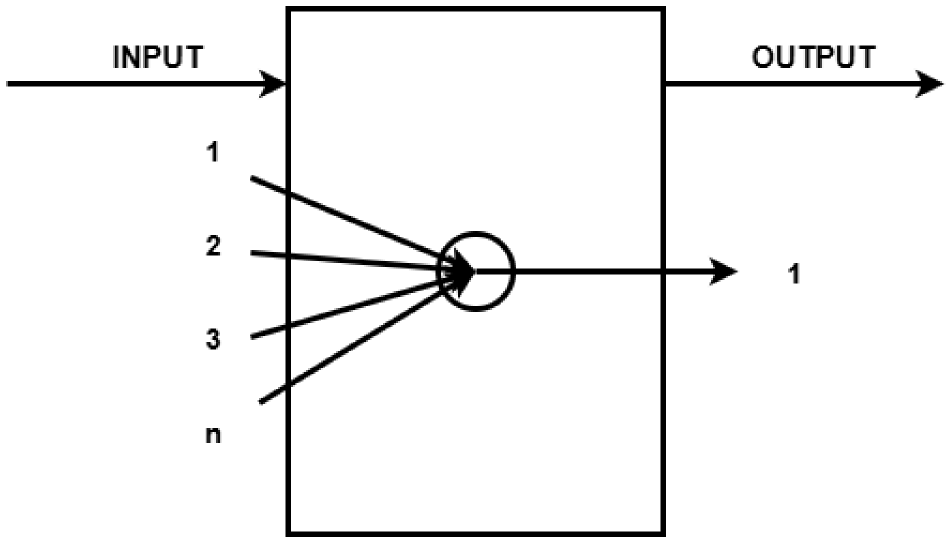
EXAMPLE :

**SELECT LENGTH (ENAME)
FROM EMP;**

EMP (INPUT)		OUTPUT
ENAME		**LENGTH(ENAME)**
MILLER	**----------------->**	**6**
KING	**----------------->**	**4**
SCOTT	**----------------->**	**5**
SMITH	**----------------->**	**5**

MULTI-ROW FUNCTION ()

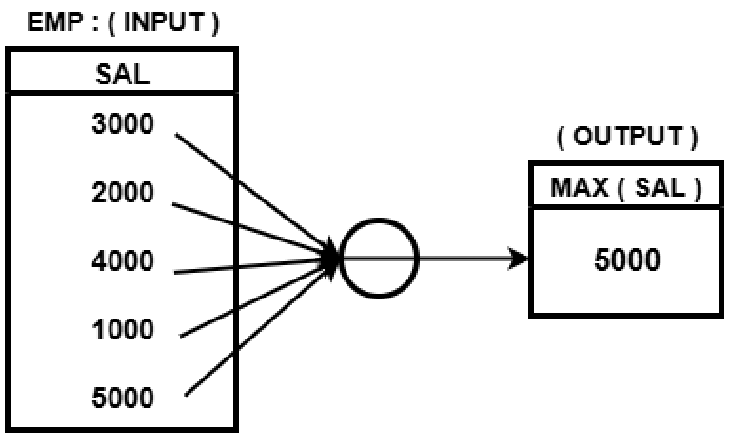
- Multi-row function accepts all the input at once and executes to generate a single output.
- For multi-row function, if you pass n number of inputs, it returns a single output.
- Multi-row function is also known as "group function" or "aggregate function" .



1. Function Name: MAX()
2. No of Arguments (Input) : SAL
3. Return type : SINGLE OUTPUT

EXAMPLE :

```
SELECT MAX ( SAL )
FROM EMP;
```



LIST OF MULTI ROW FUNCTION ()

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

- **MAX ()** – It is used to obtain the maximum value present in the column.
- **MIN ()** – It is used to obtain the minimum value present in the column.
- **SUM()** – It is used to obtain the (total) and summation of values present in the column.
- **AVG()** – It is used to obtain the average of values present in the column.
- **COUNT()** – It is used to obtain the number of values present in the column.

EXAMPLE:

```
SELECT MAX( SAL ) , MIN( SAL ) , AVG( SAL ) , SUM ( SAL ) ,
COUNT ( SAL )
FROM EMP;
```

OUTPUT :

MAX(SAL)	MIN(SAL)	AVG(SAL)	SUM(SAL)	COUNT(SAL)
5000	800	2073.21	29025	14

RULES OF MULTI-ROW FUNCTION ()

- Multi-row function can accept only " one argument " , i.e., a column name or an expression.

```
MULTI_ROW_FUNCTION ( COLUMN_NAME/EXPRESSION )
```

- Along with a MULTI-ROW FUNCTION () , " we are not supposed to use any other COLUMN_NAME " in the SELECT clause.
- Multi-row function () "ignores the NULL".
- We cannot use MULTI-ROW FUNCTION() in the " WHERE clause ".
- COUNT() is the only MULTI-ROW FUNCTION() which can accept ASTERISK(*) as an argument.

QUESTIONS ON MULTI-ROW FUNCTION()

1. WAQTD NUMBER OF EMPLOYEES GETTING SALARY LESS THAN 2000 IN DEPARTMENT 10

```
SELECT COUNT(*)
FROM EMP
WHERE SAL < 2000 AND DEPTNO = 10;
```

2. WAQTD TOTAL SALARY NEEDED TO PAY EMPLOYEES WORKING AS CLERK

```
SELECT SUM(SAL) AS TOTAL_SALARY
FROM EMP
WHERE JOB = ‘CLERK’ ;
```

3. WAQTD AVERAGE SALARY NEEDED TO PAY ALL EMPLOYEES

```
SELECT AVG(SAL) AS AVERAGE_SALARY
FROM EMP;
```

4. WAQTD NUMBER OF EMPLOYEES HAVING 'A' AS THEIR FIRST CHARACTER

```
SELECT COUNT(*)
FROM EMP
WHERE ENAME LIKE ‘A%’;
```

5. WAQTD NUMBER OF EMPLOYEES WORKING AS CLERK OR MANAGER

```
SELECT COUNT(*)
FROM EMP
WHERE JOB IN (‘CLERK’ , ‘MANAGER’);
```

6. WAQTD TOTAL SAL NEEDED TO PAY EMPS HIRED IN FEB

```
SELECT SUM(SAL) AS TOTAL_SALARY
FROM EMP
WHERE HIREDATE LIKE ‘%FEB%’;
```

7. WAQTD NUMBER OF EMPS REPORTING TO 7839 (MGR)

```
SELECT COUNT(*)
FROM EMP
WHERE MGR IN 7839;
```

8. WAQTD NUMBER OF EMPS GETTING COMM IN DEPT 30

```
SELECT COUNT(*)
FROM EMP
WHERE COMM IS NOT NULL AND DEPTNO IN 30;
```

9. WAQTD AVERAGE SALARY, TOTAL SALARY, COUNT OF EMPLOYEES, AND MAXIMUM SALARY GIVEN TO EMPLOYEES WORKING AS PRESIDENT

```
SELECT AVG(SAL) , SUM(SAL) , COUNT(*) , MAX(SAL)
FROM EMP
WHERE JOB IN ‘PRESIDENT’;
```

10. WAQTD NUMBER OF EMPLOYEES HAVING CHARACTER 'A' IN THEIR NAME

```
SELECT COUNT(*)
FROM EMP
WHERE ENAME LIKE ‘%A%’ ;
```

TYPES OF SINGLE ROW FUNCTIONS

- In Single Row Function, we have 14 types:

1. **LENGTH()**
2. **UPPER()**
3. **LOWER()**
4. **INITCAP()**
5. **REVERSE()**
6. **SUBSTR()**
7. **INSTR()**
8. **REPLACE()**
9. **MOD()**
10. **ROUND()**
11. **TRUNC()**
12. **ADD_MONTHS()**
13. **TO_CHAR()**
14. **NVL() — (Null value logic)**

NOTE:
DUAL IS A DUMMY TABLE WITH THE DATA-TYPES VARCHAR2(1).

LENGTH()

- LENGTH() is used to give the total number of characters present in a string.

SYNTAX

```
LENGTH ( ‘STRING‘ ) ;
```

EXAMPLE :

```
SELECT LENGTH ( ‘TAMILNADU‘ )
FROM DUAL ;
```

OUTPUT : 9

UPPER ()

- UPPER () function is used to convert the given strings into uppercase.

SYNTAX

```
UPPER ( ‘STRING‘ ) ;
```

LOWER ()

- LOWER () function is used to convert the given strings into lower case.

SYNTAX

```
LOWER ( 'STRING' ) ;
```

INITCAP ()

- INITCAP () function is used to convert the given string's first character into uppercase and the remaining to lowercase.

SYNTAX

```
INITCAP ( 'STRING' );
```

EXAMPLE :

```
SELECT UPPER ( 'tamil nadu' ) ,
LOWER ( 'TAMIL NADU' )  ,
INITCAP ( 'tAMIL nADU' )
FROM DUAL ;
```

OUTPUT :

UPPER	LOWER	INITCAP
TAMIL NADU	tamil nadu	Tamil Nadu

MOD()

- MOD() function is used to get the modulus of two given numbers.

SYNTAX

```
MOD( 8 , 5 );
```

EXAMPLE :

```
SELECT MOD( 8 , 5 )
FROM DUAL;
```

OUTPUT : 3

ADD_MONTHS()

- ADD_MONTHS() function is used to add the number of months to the given date format.

SYNTAX

```
ADD_MONTHS( 'DATE' , NUMBER_OF_MONTHS );
```

EXAMPLE :

```
SELECT ADD_MONTHS ( '20-MAY-2025' , 12 )
FROM DUAL ;
```

OUTPUT : 20-MAY-2026

REVERSE()

- REVERSE() function is used to reverse the given string.

SYNTAX

```
REVERSE( ‘STRING‘ );
```

EXAMPLE :

```
SELECT REVERSE( ‘SANJAY ER‘ )
FROM DUAL ;
```

OUTPUT : RE YAJNAS

REPLACE()

- REPLACE() function is used to replace a string from the original string with a new string.

SYNTAX

```
REPLACE ( ‘ORIGINAL_STRING‘ , ‘STRING’ , ‘NEW_STRING’ );
```

EXAMPLE :

```
SELECT REPLACE( ‘BANGALORE’ , ‘B’ , ‘M’ );
FROM DUAL;
```

OUTPUT : MANGALORE

ROUND()

- ROUND() unction is used to round off the given number to its nearest value.

SYNTAX

```
ROUND(NUMBER);
```

EXAMPLE :

```
SELECT ROUND(21.4) , ROUND(21.5) , ROUND(21.8)
FROM DUAL;
```

OUTPUT :

ROUND(21.4)	ROUND(21.5)	ROUND(21.8)
21	22	22

TRUNC()

- TRUNC() function is used to round off a given number to its lowest value.

SYNTAX

```
TRUNC(NUMBER);
```

EXAMPLE :

```
SELECT TRUNC(21.4) , TRUNC(21.7), TRUNC(21.9)
FROM DUAL;
```

OUTPUT :

TRUNC(21.4)	TRUNC(21.7)	TRUNC(21.9)
21	21	21

SUBSTR()

- SUBSTR() function is used to extract a part of a string from its original string by using position and length.

SYNTAX

```
SUBSTR('ORIGINAL_STRING' , POSITION , [LENGTH] );
```

ORIGINAL STRING :

C	H	E	N	N	A	I
1	2	3	4	5	6	7

EXAMPLE :

```
SELECT SUBSTR ( ' CHENNAI ' , 3 , 4 )
FROM DUAL ;
```

OUTPUT : ENNA

INSTR()

- INSTR() function is used to get the original position of a substring by using its occurrence.

SYNTAX

```
INSTR( ' ORIGINAL_STRING ' , ' SUBSTR ' , POSITION,
[ OCCURRENCE ] );
```

ORIGINAL STRING :

A	A	D	H	I	D	E	V	A	P	R	A	S	A	D	H	A	N
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

EXAMPLE :

```
SELECT INSTR( 'AADHIDEVAPRASADHAN' , 'D' , 1 , 3 )
FROM DUAL;
```

OUTPUT : 15

TOCHAR()

- TOCHAR() function is used to convert a given date format into a string format.

SYNTAX

```
TO_CHAR ( TO_DATE () , FORMAT_MODEL );
```

EXAMPLE :

```
SELECT TO_CHAR ( TO_DATE (‘ 20-MAY-2024 ‘) , ‘DAY’ )
FROM DUAL ;
```

OUTPUT : TUESDAY

FORMAT MODELS

1	YEAR : TWENTY TWENTY FIVE
2	YYYY : 2025
3	YY : 25
4	MONTH : MAY
5	MON : MAY
6	MM : 5
7	DAY : TUESDAY
8	DY : TUE
9	DD : 20
10	D : 2 (DAY OF THE WEEK)
11	HH24 : 17 HOURS
12	HH 12 : 5 HOURS
13	M1 : 22 MINUTES
14	SS : 53 SECONDS
15	‘HH:12 : M1:SS’ : 5 : 22 : 53
16	‘DD-MM-YY’ : 20-5-25
17	‘MM-DD-YYYY’ : 5-20-2025

NULL VALUE LOGIC () : NVL()

- NVL() function is used to overcome the drawbacks of null operations.

SYNTAX

```
NVL( ARG1 , ARG2 ) ;
```

ARGUMENT 1

- For argument 1, we can pass columns or expressions which can result in NULL.

ARGUMENT 2

- For argument 2, we can pass values (0, 1, 2 , , , , n). If the argument 1 is resulting in NULL , argument 2 will be substituted.

1. **WITHOUT NVL() :**

```
SELECT SAL + COMM
FROM EMP :
```

EMP :

SAL	COMM		SAL + COMM
100	50	100 + 50 = 150	150
200	NULL	200 + NULL = NULL	NULL
150	50	150 + 50 = 200	200
100	NULL	100 + NULL = NULL	NULL

2. **WITH NVL() :**

SAL + NVL(COMM , 0);

100 + NVL(50,0) = 150
200 + NVL(NULL,0) = 200
150 + NVL(50,0) = 200
100 + NVL(NULL,0) = 100

```
SELECT SAL+NVL( COMM , 0 )
FROM EMP;
```

SAL + NVL (COMM , 0)
150
200
200
100

GROUP BY CLAUSE

- GROUP BY CLAUSE is used to group the records.

SYNTAX

```
SELECT GROUP_BY_EXPRESSION / GROUP_FUNCTION
FROM TABLE_NAME
[ WHERE < FILTER_CONDITION > ]
GROUP BY COLUMN_NAME / EXPRESSION ;
```

NOTE :

GROUP_BY_EXPRESSION : ANY COLUMN_NAME OR EXPRESSION WRITTEN INSIDE THE GROUP BY CLAUSE IS KNOWN AS GROUP_BY_EXPRESSION.

ORDER OF EXECUTION

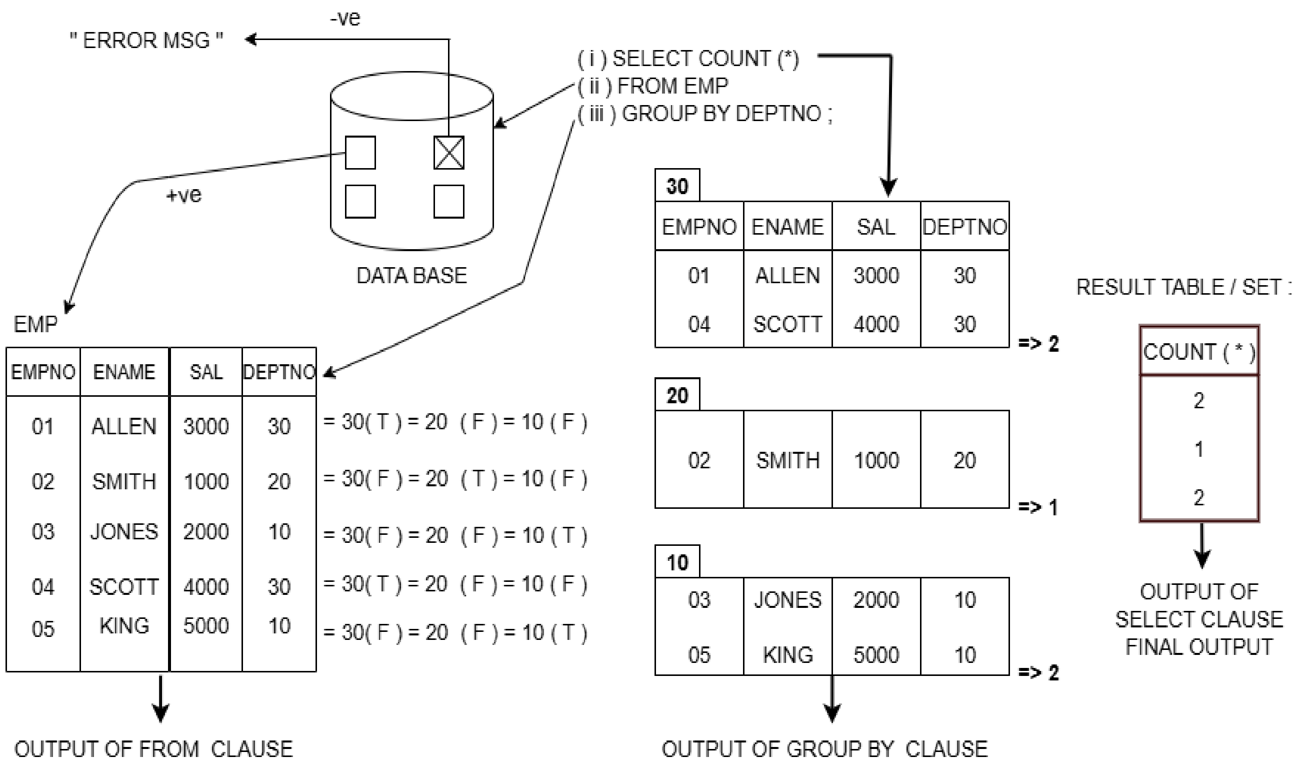
1. **FROM**
2. **WHERE (IF USED) [ROW BY ROW]**
3. **GROUP BY [ROW BY ROW]**
4. **SELECT [GROUP BY GROUP]**

COMMENTS

- GROUP BY CLAUSE is used to group the records.
- GROUP BY CLAUSE executes row by row.
- After the execution of GROUP BY CLAUSE, we get groups.
- Therefore, any CLAUSE that executes after GROUP BY must execute group by group.
- The COLUMN_NAME or expression used for grouping can be used in the SELECT CLAUSE.
- GROUP BY CLAUSE can be used without using WHERE CLAUSE.

EXAMPLE

1 . WAQTD NO OF EMP WORKING IN EACH DEPT .



```
( i ) SELECT COUNT (*)
( ii ) FROM EMP
( iii ) GROUP BY DEPTNO ;
```

EMPNO	ENAME	SAL	DEPTNO	
01	ALLEN	3000	30	= 30(T) = 20 (F) = 10 (F)
02	SMITH	1000	20	= 30(F) = 20 (T) = 10 (F)
03	JONES	2000	10	= 30(F) = 20 (F) = 10 (T)
04	SCOTT	4000	30	= 30(T) = 20 (F) = 10 (F)
05	KING	5000	10	= 30(F) = 20 (F) = 10 (T)

30

EMPNO	ENAME	SAL	DEPTNO
01	ALLEN	3000	30
04	SCOTT	4000	30

=> 2

20

02	SMITH	1000	20

=> 1

10

03	JONES	2000	10
05	KING	5000	10

=> 2

COUNT (*)
2
1
2

QUESTIONS ON GROUP BY CLAUSE

1. WAQTD NUMBER OF EMPLOYEES WORKING IN EACH DEPARTMENT EXCEPT PRESIDENT

```
SELECT COUNT(*)
FROM EMP
WHERE JOB <> ‘PRESIDENT’
GROUP BY DEPTNO;
```

2. WAQTD TOTAL SALARY NEEDED TO PAY ALL THE EMPLOYEES IN EACH JOB

```
SELECT SUM(SAL)
FROM EMP
GROUP BY JOB;
```

3. WAQTD NUMBER OF EMPLOYEES WORKING AS MANAGER IN EACH DEPARTMENT

```
SELECT COUNT(*)
FROM EMP
WHERE JOB = ‘MANAGER’
GROUP BY DEPTNO;
```

4. WAQTD AVERAGE SALARY NEEDED TO PAY ALL THE EMPLOYEES IN EACH DEPARTMENT EXCLUDING DEPARTMENT NO. 20

```
SELECT AVG(SAL)
FROM EMP
WHERE DEPTNO <> 20
GROUP BY DEPTNO;
```

5. WAQTD NUMBER OF EMPLOYEES HAVING CHARACTER 'A' IN THEIR NAMES IN EACH JOB

```
SELECT COUNT(*)
FROM EMP
WHERE ENAME LIKE ‘%A%’
GROUP BY JOB;
```

6. WAQTD NUMBER OF EMPLOYEES AND AVERAGE SALARY NEEDED TO PAY THE EMPLOYEES WHOSE SALARY IS GREATER THAN 2000 IN EACH DEPARTMENT

```
SELECT COUNT(*) , AVG(SAL)
FROM EMP
WHERE SAL > 2000
GROUP BY DEPTNO ;
```

7. WAQTD TOTAL SALARY NEEDED TO PAY AND THE NUMBER OF SALESMEN IN EACH DEPARTMENT

```
SELECT SUM(SAL),COUNT(*)
FROM EMP
WHERE JOB = ‘SALESMAN’
GROUP BY DEPTNO;
```

8. WAQTD NUMBER OF EMPLOYEES WITH THEIR MAXIMUM SALARIES IN EACH JOB

```
SELECT COUNT(*) , MAX(SAL)
FROM EMP
GROUP BY JOB ;
```

9. **WAQTD MAXIMUM SALARIES GIVEN TO AN EMPLOYEE WORKING IN EACH DEPARTMENT**

```
SELECT MAX(SAL)
FROM EMP
GROUP BY DEPTNO ;
```

10. **WAQTD NUMBER OF TIMES THE SALARIES HAVE BEEN PRESENT IN EACH DEPARTMENT FROM THE EMP TABLE**

```
SELECT COUNT(SAL)
FROM EMP
GROUP BY DEPTNO;
```

HAVING CLAUSE

➢ HAVING CLAUSE is used to filter the groups.

SYNTAX

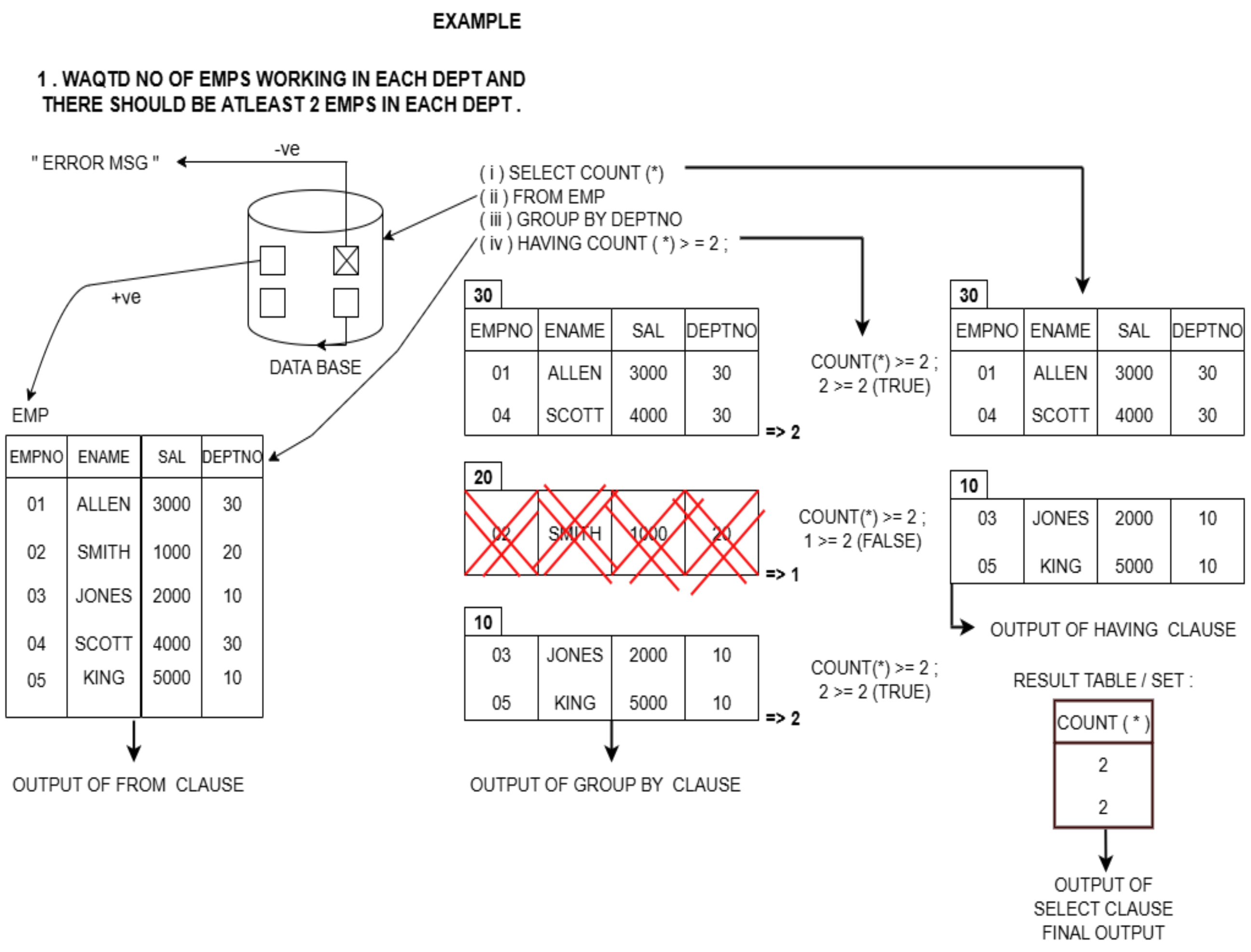
```
SELECT GROUP_BY_EXPRESSION / GROUP_FUNCTION
FROM TABLE_NAME
[ WHERE < FILTER_CONDITION > ]
GROUP BY COLUMN_NAME / EXPRESSION
HAVING < GROUP_FILTER_CONDITION > ;
```

ORDER OF EXECUTION

1. **FROM**
2. **WHERE (IF USED) [ROW BY ROW]**
3. **GROUP BY [ROW BY ROW]**
4. **HAVING [GROUP BY GROUP]**
5. **SELECT [GROUP BY GROUP]**

COMMENTS

- HAVING CLAUSE executes group by group.
- HAVING CLAUSE executes after the execution of GROUP BY CLAUSE.
- Since HAVING CLAUSE executes after GROUP BY CLAUSE, we have to write group filter conditions.
- HAVING CLAUSE can be used without GROUP BY CLAUSE.
- HAVING CLAUSE is used to pass all multi-row function conditions.



QUESTIONS ON HAVING CLAUSE

1. WAQTD DEPARTMENT NUMBER AND NUMBER OF EMPLOYEES WORKING IN EACH DEPARTMENT IF THERE ARE AT LEAST 2 CLERKS IN EACH DEPARTMENT

```
SELECT DEPTNO , COUNT(*)
FROM EMP
WHERE JOB IN ‘CLERK’
GROUP BY DEPTNO
HAVING COUNT(*) > 1;
```

2. WAQTD DEPARTMENT NUMBER AND TOTAL SALARY NEEDED TO PAY ALL EMPLOYEES IN EACH DEPARTMENT IF THERE ARE AT LEAST 4 EMPLOYEES IN EACH DEPARTMENT

```
SELECT DEPTNO , SUM(SAL)
FROM EMP
GROUP BY DEPTNO
HAVING COUNT(*) > 3;
```

3. WAQTD NUMBER OF EMPLOYEES EARNING SALARY MORE THAN 1200 IN EACH JOB AND THE TOTAL SALARY NEEDED TO PAY EMPLOYEE OF EACH JOB MUST BE EXCESS 3800

```
SELECT COUNT(*)
FROM EMP
WHERE SAL>1200
GROUP BY JOB
HAVING SUM(SAL) > 3800;
```

4. **WAQTD DEPARTMENT NUMBER AND NUMBER OF EMPLOYEES WORKING ONLY IF THERE ARE 2 EMPLOYEES WORKING IN EACH DEPARTMENT AS MANAGER**

```
SELECT DEPTNO , COUNT(*)
FROM EMP
WHERE JOB = ‘MANAGER’
GROUP BY DEPTNO
HAVING COUNT(*) = 2 ;
```

5. **WAQTD JOB AND MAX SALARY OF EMPLOYEE IN EACH JOB IF THE MAX SAL EXCEEDS 2600**

```
SELECT JOB , MAX(SAL)
FROM EMP
GROUP BY JOB
HAVING MAX(SAL) > 2600 ;
```

6. **WAQTD THE SALARIES WHICH ARE REPEATED IN EMPLOYEE TABLE**

```
SELECT SAL
FROM EMP
GROUP BY SAL
HAVING COUNT(*) > 1;
```

7. **WAQTD THE HIGHER RATE WHICH ARE DUPLICATE IN EMPLOYEE TABLE**

```
SELECT HIREDATE
FROM EMP
GROUP BY HIREDATE
HAVING COUNT(*) > 1;
```

8. WAQTD AVERAGE SALARY OF EACH DEPARTMENT IF AVERAGE SALARY IS LESS THAN 3000

```
SELECT AVG(SAL)
FROM EMP
GROUP BY DEPTNO
HAVING AVG(SAL) < 3000;
```

9. WAQTD DEPARTMENT NUMBER IF THERE ARE AT LEAST 3 EMPLOYEES IN EACH DEPARTMENT WHOSE NAME HAS CHARACTER 'A' OR 'S'

```
SELECT DEPTNO
FROM EMP
WHERE ENAME LIKE ‘%A%’ OR ENAME LIKE ‘%S%’
GROUP BY DEPTNO
HAVING COUNT(*) > = 3;
```

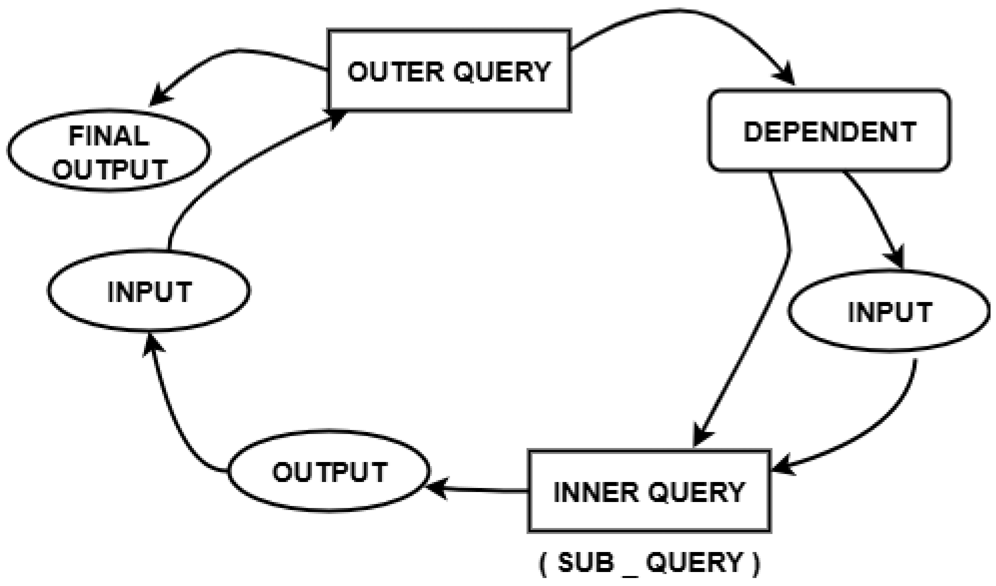
10. WAQTD MIN AND MAX SALARY OF EACH JOB IF MIN SHARE IS MORE THAN 1000 AND MAX SHARE IS LESS THAN 5000

```
SELECT MIN(SAL) , MAX(SAL)
FROM EMP
GROUP BY JOB
HAVING MIN(SAL) > 1000 AND MAX(SAL) < 5000;
```

SUB-QUERY

- A query written inside another query is known as a sub-query.

WORKING PRINCIPLE OF SUBQUERY



- Let us consider two queries , They are:
    1. Outer query
    2. Inner query / sub-query
- Inner query starts execution and gives output.
- The output of the inner query will be fed as input to the outer query.
- Outer query starts the execution and gives the final output.
- Therefore, outer query is dependent on inner query.

WHEN OR WHY DO WE USE A SUBQUERY?

- **Case 1** : Whenever we have unknowns present in the question, we use subquery case 1 to find that unknown.

1. **WAQTD DETAILS OF THE EMPLOYEES EARNING MORE THAN 800 (DIRECT QUESTION).**

```
SELECT *
FROM EMP
WHERE SAL > 800 ;
```

2. **WAQTD DETAILS OF THE EMPLOYEES EARNING MORE THAN SMITH (INDIRECT-QUESTIONS) .**

```
SELECT *
FROM EMP
WHERE SAL > ( SELECT SAL
              FROM EMP
              WHERE ENAME = 'SMITH' );
```

QUESTIONS ON SUB-QUERY CASE-1

1. WAQTD DETAILS OF EMPS WORK IN SAME DEPT AS ALLEN.

```
SELECT *
FROM EMP
WHERE DEPTNO = ( SELECT DEPTNO
                 FROM EMP
                 WHERE ENAME = ‘ALLEN’ );
```

2. WAQTD DETAILS OF THE EMPS EARNING LESS THAN KING.

```
SELECT *
FROM EMP
WHERE SAL < ( SELECT SAL
              FROM EMP
              WHERE ENAME = ‘KING’ );
```

3. WAQTD DETAILS OF THE EMPS WORKING IN THE SAME DESIGNATION AS JONES.

```
SELECT *
FROM EMP
WHERE JOB = ( SELECT JOB
              FROM EMP
              WHERE ENAME = ‘JONES’ );
```

4. WAQTD DETAILS OF THE EMPS EARN SAME SAL AS SCOTT.

```
SELECT *
FROM EMP
WHERE SAL = ( SELECT SAL
              FROM EMP
              WHERE ENAME = ‘SCOTT’ );
```

5. **WAQTD DETAILS OF THE EMPLOYEES WORKING IN THE SAME DEPARTMENT AS JAMES AND EMPLOYEE SHOULD BE WORKING AS SALESMAN .**

```
SELECT *
FROM EMP
WHERE JOB = 'SALESMAN' AND DEPTNO =
                    ( SELECT DEPTNO
                     FROM EMP
                     WHERE ENAME = 'JAMES');
```

6. **WAQTD DETAILS OF THE EMPLOYEES WORKING IN THE SAME DEPARTMENT AS JONES AND EMPLOYEES SHOULD BE WORKING AS CLERK.**

```
SELECT *
FROM EMP
WHERE JOB = 'CLERK' AND DEPTNO =
                    ( SELECT DEPTNO
                     FROM EMP
                     WHERE ENAME = 'JONES');
```

7. **WAQTD DETAILS OF THE EMPLOYEES EARNING MORE THAN SMITH AND LESS THAN KING.**

```
SELECT *
FROM EMP
WHERE SAL > (SELECT SAL
                     FROM EMP
                     WHERE ENAME = 'SMITH') AND
           SAL < (SELECT SAL
                     FROM EMP WHERE ENAME = 'KING');
```

8. **WAQTD DETAILS OF THE EMPLOYEES HIRED AFTER ALLEN AND BEFORE MILLER.**

```
SELECT *
FROM EMP
WHERE HIREDATE > ( SELECT HIREDATE
                   FROM EMP
                   WHERE ENAME = ‘ALLEN’) AND
      HIREDATE < ( SELECT HIREDATE
                   FROM EMP
                   WHERE ENAME = ‘MILLER’);
```

9. **WAQTD NAME OF THE EMPLOYEES EARNING MORE THAN ADAMS.**

```
SELECT ENAME
FROM EMP
WHERE SAL > ( SELECT SAL
              FROM EMP
              WHERE ENAME = ‘ADAMS’  );
```

10. **WAQTD NAME AND SALARY OF THE EMPLOYEES EARNING LESS THAN KING.**

```
SELECT ENAME , SAL
FROM EMP
WHERE SAL < ( SELECT SAL
              FROM EMP
              WHERE ENAME = ‘KING’  );
```

11. WAQTD NAME AND DEPARTMENT NUMBER OF EMPLOYEES WORKING IN THE SAME DEPARTMENT AS JONES.

```
SELECT ENAME , DEPTNO
FROM EMP
WHERE DEPTNO = ( SELECT DEPTNO
                 FROM EMP
                 WHERE ENAME = 'JONES'  );
```

12. WAQTD NAME AND JOB OF ALL THE EMPLOYEES WORKING IN THE SAME DESIGNATION AS JAMES.

```
SELECT ENAME , JOB
FROM EMP
WHERE JOB = ( SELECT JOB
              FROM EMP
              WHERE ENAME = 'JAMES'  );
```

13. WAQTD EMPLOYEE NUMBER AND E-NAME ALONG WITH ANNUAL SALARY OF THE EMPLOYEES IF THEIR ANNUAL SALARY IS GREATER THAN WARD ANNUAL SALARY.

```
SELECT EMPNO , ENAME , SAL*12 AS ANNUAL_SALARY
FROM EMP
WHERE SAL * 12 > ( SELECT SAL*12
                   FROM EMP
                   WHERE ENAME = 'WARD'  );
```

14. WAQTD NAME AND HIRE DATE OF THE EMPLOYEES IF THEY ARE HIRED BEFORE SCOTT.

```
SELECT ENAME , HIREDATE
FROM EMP
WHERE HIREDATE < ( SELECT HIREDATE
                   FROM EMP
                   WHERE ENAME = 'SCOTT'  );
```

15. WAQTD NAME AND HIRE DATE OF THE EMPLOYEES IF THEY ARE HIRED AFTER THE PRESIDENT.

```
SELECT ENAME , HIREDATE
FROM EMP
WHERE HIREDATE > ( SELECT HIREDATE
                   FROM EMP
                   WHERE JOB = 'PRESIDENT'  );
```

16. WAQTD NAME AND SALARY OF THE EMPLOYEES EARNING LESS THAN THE EMPLOYEES WHOSE EMPLOYEE NUMBER IS 7839.

```
SELECT ENAME , SAL
FROM EMP
WHERE SAL < ( SELECT SAL
              FROM EMP
              WHERE EMPNO = 7839  );
```

17. WAQTD ALL THE DETAILS OF THE EMPLOYEES IF THEY ARE HIRED BEFORE MILLER.

```
SELECT *
FROM EMP
WHERE HIREDATE < ( SELECT HIREDATE
                   FROM EMP
                   WHERE ENAME = ‘MILLER’  );
```

18. WAQTD NAME AND EMPLOYEE NUMBER OF THE EMPLOYEES WHO ARE EARNING MORE THAN ALLEN.

```
SELECT ENAME , EMPNO
FROM EMP
WHERE SAL > ( SELECT SAL
                   FROM EMP
                   WHERE ENAME = ‘ALLEN’  );
```

SUB-QUERY CASE 2

- Whenever the data to be selected and the condition to be executed are present in different tables, we use Subquery Case 2.

EXAMPLE :

CASE 1 EXAMPLE

WAQTD DETAILS OF THE EMPLOYEES EARNING LESS THAN KING.

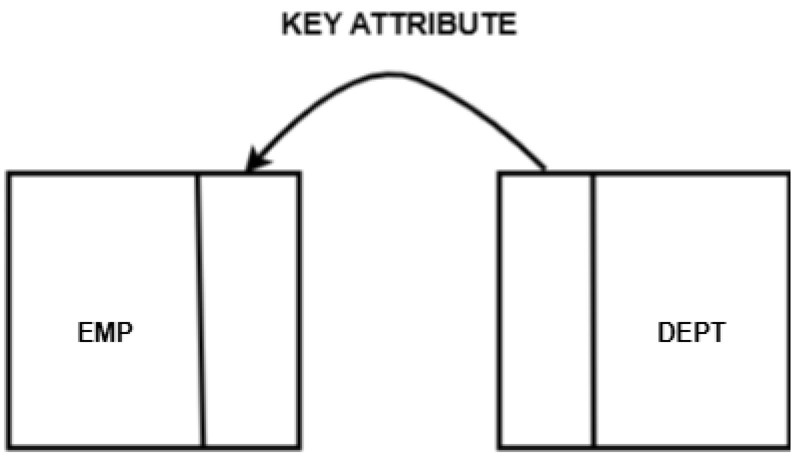
```
SELECT *
FROM EMP
WHERE SAL < ( SELECT SAL
              FROM EMP
              WHERE ENAME = ‘KING ’  );
```

CASE 2 EXAMPLE

WAQTD THE DNAME OF EMPLOYEE SMITH.

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO = ( SELECT DEPTNO
                 FROM EMP
                 WHERE ENAME = ‘SMITH ’  );
```

NOTE :



- The common column between tables is “ DEPTNO “ .

NOTE :

- In the inner or subquery, we cannot SELECT more than one column.
- The corresponding columns name need not be the same, but the data types of those have to be the same.

QUESTIONS ON SUBQUERY CASE 2

1. WAQTD DNAME OF THE EMPLOYEES WHOSE NAME IS ALLEN.

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO = ( SELECT DEPTNO
                 FROM EMP
                 WHERE ENAME = ‘ALLEN’  );
```

2. WAQTD DNAME AND LOC OF THE EMPLOYEES WHOSE ENAME IS KING.

```
SELECT DNAME , LOC
FROM DEPT
WHERE DEPTNO = ( SELECT DEPTNO
                 FROM EMP
                 WHERE ENAME = ‘KING’  );
```

3. WAQTD LOC OF THE EMPLOYEE WHOSE EMPLOYEE NUMBER IS 7902.

```
SELECT LOC
FROM DEPT
WHERE DEPTNO = ( SELECT DEPTNO
                 FROM EMP
                 WHERE EMPNO = 7902  );
```

4. WAQTD DNAME AND LOC ALONG WITH DEPARTMENT NUMBER OF THE EMPLOYEE WHOSE NAME ENDS WITH ‘R’.

```
SELECT DNAME , LOC , DEPTNO
FROM DEPT
WHERE DEPTNO = ANY( SELECT DEPTNO
                    FROM EMP
                    WHERE ENAME LIKE ‘%R’  );
```

5. WAQTD DNAME OF THE EMPS WHOSE JOB IS PRESIDENT.

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO = ( SELECT DEPTNO
                 FROM EMP
                 WHERE JOB = ‘PRESIDENT’  );
```

6. WAQTD NAMES OF THE EMPLOYEES WORKING IN THE ACCOUNTING DEPARTMENT.

```
SELECT ENAME
FROM  EMP
WHERE DEPTNO = ( SELECT DEPTNO
                 FROM DEPT
                 WHERE DNAME = ‘ACCOUNTING’  );
```

7. WAQTD ENAME AND SALARIES OF THE EMPLOYEES WHO ARE WORKING IN THE LOCATION CHICAGO.

```
SELECT ENAME , SAL
FROM  EMP
WHERE DEPTNO = ( SELECT DEPTNO
                 FROM DEPT
                 WHERE LOC = ‘CHICAGO’  );
```

8. WAQTD DETAILS OF THE EMPLOYEES WORKING IN SALES.

```
SELECT *
FROM  EMP
WHERE DEPTNO = ( SELECT DEPTNO
                              FROM DEPT
                              WHERE DNAME = 'SALES'
);
```

9. WAQTD DETAILS OF THE EMPLOYEES ALONG WITH ANNUAL SALARY IF EMPLOYEES ARE WORKING IN NEW YORK.

```
SELECT EMP.* , SAL*12 AS ANNUAL_SALARY
FROM  EMP
WHERE DEPTNO IN ( SELECT DEPTNO
                              FROM DEPT
                              WHERE LOC = 'NEW YORK'  );
```

10. WAQTD NAMES OF THE EMPLOYEES WORKING IN THE OPERATIONS DEPARTMENT.

```
SELECT ENAME
FROM  EMP
WHERE DEPTNO IN ( SELECT DEPTNO
                              FROM DEPT
                              WHERE DNAME = 'OPERATIONS'  );
```

TYPES OF SUBQUERY

- We can classify sub queries into two types :
  - SINGLE ROW – SUB QUERY
  - MULTI ROW – SUB QUERY

SINGLE ROW SUB-QUERY

- Inner query or subquery which returns SINGLE outputs is known as a SINGLE row subquery.
- For single row subquery, we can pass both normal operators (=) and special operators (IN).

EXAMPLE

WQATD DNAME OF THE EMPLOYEE JAMES.

OUTPUT: SALES

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO = ( SELECT DEPTNO
                 FROM EMP
                 WHERE ENAME = 'JAMES'  );
```

MULTI-ROW SUBQUERY

- Inner query or subquery which returns multiple outputs is known as a multi-row subquery.
- For multi-row subquery, we should pass only special operators (IN). If we pass normal operators, it will end up in error.

EXAMPLE :

WAQTD DNAME OF THE EMPLOYEES SMITH, ALLEN.

OUTPUT: RESEARCH AND SALES

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO IN ( SELECT DEPTNO
                  FROM EMP
                  WHERE ENAME IN ( 'SMITH', 'ALLEN'  );
```

NOTE:

- It is difficult to identify whether a query belongs to a single-row or multi-row subquery.
- So, it is always recommended to use special operators (IN) to compare the values.

OPERATORS OF SUBQUERY

- In subquery, we have two types of operators:

1. ALL operator
2. ANY operator

ALL OPERATOR

- ALL operator is a subquery operator. It should be used along with the relational operator to compare both the queries.
- ALL operator considers all the values to generate the final output.

ALL OPERATOR EXAMPLE :

WAQTD DETAILS OF THE EMPLOYEE EARNING MORE THAN SALESMAN.

OR

WAQTD DETAILS OF THE EMPLOYEE EARNING MORE THAN ALL SALESMAN.

```
SELECT *
FROM EMP
WHERE SAL > ALL (SELECT SAL
                  FROM EMP
                  WHERE JOB = ‘SALESMAN’ );
```

ANY OPERATOR

- ANY operator is a subquery operator. It should be used along with the relational operator to compare both the queries.
- ANY operator considers any one of the values to generate the output.

ANY OPERATOR EXAMPLE :

WAQTD DETAILS OF THE EMPLOYEE EARNING MORE THAN ATLEAST A SALESMAN.

OR

WAQTD DETAILS OF ALL EMPLOYEES EARNING MORE THAN ANY ONE OF THE SALESMEN.

```
SELECT *
FROM EMP
WHERE SAL > ANY (SELECT SAL
                    FROM EMP
                    WHERE JOB = ‘SALESMAN’ );
```

NESTED SUB-QUERY

- A sub-query returned inside another sub-query is known as a nested sub-query.
- We can nest up to 255 sub-queries.

NOTE: IT IS RECOMMENDED TO WRITE NESTED QUERY FROM THE INNER TO OUTER IN THE INTIAL STAGE FOR AVOIDING CONFUSION.

NOTE :

MAX() WE USE <

MIN() WE USE >

MAX()

1. **WAQTD 3RD MAX OF SALARY.**

```
SELECT MAX(SAL)
FROM EMP
WHERE SAL < (SELECT MAX(SAL)
             FROM EMP
             WHERE SAL < (SELECT MAX(SAL)
                          FROM EMP));
```

MIN()

1. WAQTD 3RD MIN OF SALARY.

```
SELECT MIN(SAL)
FROM EMP
WHERE SAL > (SELECT MIN(SAL)
             FROM EMP
             WHERE SAL > (SELECT MIN(SAL)
                          FROM EMP));
```

2. WAQTD DNAME OF THE EMPLOYEES EARNING 3RD MIN OF SALARY.

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO = (
  SELECT DEPTNO
  FROM EMP
  WHERE SAL = (
    SELECT MIN(SAL)
    FROM EMP
    WHERE SAL > (
      SELECT MIN(SAL)
      FROM EMP
      WHERE SAL > (
        SELECT MIN(SAL)
        FROM EMP
      )
    )
  )
);
```

QUESTIONS ON NESTED SUB-QUERY

1. WAQTD 2ND MINIMUM SALARY.

```
SELECT MIN(SAL)
FROM EMP
WHERE SAL > (SELECT MIN(SAL)
             FROM EMP);
```

2. WAQTD 5TH MAXIMUM SALARY.

```
SELECT MAX(SAL)
FROM EMP
WHERE SAL < (
   SELECT MAX(SAL)
   FROM EMP
   WHERE SAL < (
      SELECT MAX(SAL)
      FROM EMP
      WHERE SAL < (
         SELECT MAX(SAL)
         FROM EMP
         WHERE SAL < (
            SELECT MAX(SAL)
            FROM EMP
         )
      )
   )
);
```

3. **WAQTD NAME OF THE EMPLOYEE EARNING 3RD MAXIMUM SALARY.**

```
SELECT ENAME
FROM EMP
WHERE SAL = (SELECT MAX(SAL)
             FROM EMP
             WHERE SAL < (SELECT MAX(SAL)
                          FROM EMP
                         WHERE SAL < (SELECT MAX(SAL)
                                      FROM EMP)));
```

4. **WAQTD EMPLOYEE NUMBER OF THE EMPLOYEE EARNING 4TH MAXIMUM SALARY.**

```
SELECT EMPNO
FROM EMP
WHERE SAL = (
   SELECT MAX(SAL)
   FROM EMP
   WHERE SAL < (
      SELECT MAX(SAL)
      FROM EMP
      WHERE SAL < (
         SELECT MAX(SAL)
         FROM EMP
         WHERE SAL < (
            SELECT MAX(SAL)
            FROM EMP
         )
      )
   )
);
```

5. **WAQTD DEPARTMENT NAME OF AN EMPLOYEE GETTING 5TH MAX SALARY.**

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO = (
   SELECT DEPTNO
   FROM EMP
   WHERE SAL = (
      SELECT MAX(SAL)
      FROM EMP
      WHERE SAL < (
         SELECT MAX(SAL)
         FROM EMP
         WHERE SAL < (
            SELECT MAX(SAL)
            FROM EMP
            WHERE SAL < (
               SELECT MAX(SAL)
               FROM EMP
               WHERE SAL < (
                  SELECT MAX(SAL)
                  FROM EMP
               )
            )
         )
      )
   )
);
```

6. WAQTD DETAILS OF THE EMPLOYEE WHO WAS HIRED 2ND.

```
SELECT *
FROM EMP
WHERE HIREDATE = (
   SELECT MIN(HIREDATE)
   FROM EMP
   WHERE HIREDATE > (
      SELECT MIN(HIREDATE)
      FROM EMP
   )
);
```

7. WAQTD NAME OF THE EMP HIRED BEFORE THE LAST EMP.

```
SELECT ENAME
FROM EMP
WHERE HIREDATE = (
   SELECT MAX(HIREDATE)
   FROM EMP
   WHERE HIREDATE < (
      SELECT MAX(HIREDATE)
      FROM EMP));
```

8. WAQTD LOC OF THE EMPLOYEE WHO WAS HIRED FIRST.

```
SELECT LOC
FROM DEPT
WHERE DEPTNO = (
   SELECT DEPTNO
   FROM EMP
   WHERE HIREDATE = (
      SELECT MIN(HIREDATE)
      FROM EMP ));
```

9. **WAQTD DETAILS OF THE EMPLOYEE EARNING 7TH MINIMUM SALARY.**

```
SELECT *
FROM EMP
WHERE SAL = (
   SELECT MIN(SAL)
   FROM EMP
   WHERE SAL > (
      SELECT MIN(SAL)
      FROM EMP
      WHERE SAL > (
         SELECT MIN(SAL)
         FROM EMP
         WHERE SAL > (
            SELECT MIN(SAL)
            FROM EMP
            WHERE SAL > (
               SELECT MIN(SAL)
               FROM EMP
               WHERE SAL > (
                  SELECT MIN(SAL)
                  FROM EMP
                  WHERE SAL > (
                     SELECT MIN(SAL)
                     FROM EMP
                  )
               )
            )
         )
      )
   )
);
```

10. WAQTD DNAME OF EMP GETTING 2ND MAX SAL.

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO IN (
   SELECT DEPTNO
   FROM EMP
   WHERE SAL IN (SELECT MAX(SAL)
                  FROM EMP
                  WHERE SAL < (SELECT MAX(SAL)
                               FROM EMP)));
```

QUESTIONS ON SUB-QUERIES

1. WAQTD ENAME AND SALARY OF ALL THE EMPLOYEES WHO ARE EARNING MORE THAN MILLER BUT LESS THAN ALLEN.

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL > (SELECT SAL FROM EMP WHERE ENAME =
'MILLER')
AND   SAL < (SELECT SAL FROM EMP WHERE ENAME =
'ALLEN');
```

2. WAQTD ALL THE DETAILS OF THE EMPLOYEES WORKING IN THE DEPARTMENT AND WORKING IN THE SAME DESIGNATION AS SMITH

```
SELECT *
FROM EMP
WHERE DEPTNO = (SELECT DEPTNO FROM EMP WHERE
ENAME = 'SMITH')
AND   JOB    = (SELECT JOB FROM EMP WHERE ENAME =
'SMITH');
```

3. **WAQTD ALL THE DETAILS OF THE EMPLOYEES WORKING AS MANAGER IN THE SAME DEPARTMENT AS TURNER**

```
SELECT *
FROM EMP
WHERE DEPTNO = (SELECT DEPTNO FROM EMP WHERE
ENAME = 'TURNER')
AND   JOB = 'MANAGER';
```

4. **WAQTD NAME AND HIRE DATE OF THE EMPLOYEES HIRED AFTER 1980 AND BEFORE KING**

```
SELECT ENAME, HIREDATE
FROM EMP
WHERE HIREDATE > TO_DATE('01-JAN-1981','DD-MON-
YYYY')
AND   HIREDATE < (SELECT HIREDATE FROM EMP WHERE
ENAME = 'KING');
```

5. **WAQTD NAME AND SAL ALONG WITH ANNUAL SAL FOR ALL EMPLOYEES WHOSE SALARY IS LESS THAN BLAKE AND MORE THAN 3500**

```
SELECT ENAME, SAL, SAL * 12 AS ANNUAL_SAL
FROM EMP
WHERE SAL < (SELECT SAL FROM EMP WHERE ENAME =
'BLAKE')
AND   SAL > 3500;
```

6. **WAQTD ALL THE DETAILS OF THE EMPLOYEES WHO EARN MORE THAN SCOTT BUT LESS THAN KING**

```
SELECT *
FROM EMP
WHERE SAL > (SELECT SAL FROM EMP WHERE ENAME =
'SCOTT')
AND   SAL < (SELECT SAL FROM EMP WHERE ENAME =
'KING');
```

7. **WAQTD NAME OF THE EMPLOYEES WHOSE NAME STARTS WITH A AND WORKS IN THE SAME DEPARTMENT AS BLAKE**

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE 'A%'
AND   DEPTNO = (SELECT DEPTNO FROM EMP WHERE
ENAME = 'BLAKE');
```

8. **WAQTD NAME AND COMMISSION IF EMPLOYEES EARN COMMISSION AND WORK IN THE SAME DESIGNATION AS SMITH**

```
SELECT ENAME, COMM
FROM EMP
WHERE COMM IS NOT NULL
AND   JOB = (SELECT JOB FROM EMP WHERE ENAME =
'SMITH');
```

9. **WAQTD DETAILS OF ALL THE EMPLOYEES WORKING AS CLERK IN THE SAME DEPARTMENT AS TURNER**

```
SELECT *
FROM EMP
WHERE JOB = 'CLERK'
AND   DEPTNO = (SELECT DEPTNO FROM EMP WHERE
ENAME = 'TURNER');
```

10. **WAQTD ENAME, SAL, AND JOB OF THE EMPLOYEES WHOSE ANNUAL SALARY IS MORE THAN SMITH AND LESS THAN KING**

```
SELECT ENAME, SAL, JOB
FROM EMP
WHERE SAL * 12 > (SELECT SAL FROM EMP WHERE ENAME
= 'SMITH') * 12
AND   SAL * 12 < (SELECT SAL FROM EMP WHERE ENAME =
'KING') * 12;
```

11. **WAQTD NAME OF THE EMPLOYEES EARNING MORE THAN SCOTT IN ACCOUNTING DEPARTMENT**

```
SELECT ENAME
FROM EMP
WHERE SAL > (SELECT SAL FROM EMP WHERE ENAME =
'SCOTT')
AND   DEPTNO = (SELECT DEPTNO FROM DEPT WHERE
DNAME = 'ACCOUNTING');
```

12. WAQTD DETAILS OF THE EMPLOYEES WORKING AS MANAGER IN THE LOCATION CHICAGO

```
SELECT *
FROM EMP
WHERE JOB = 'MANAGER'
AND   DEPTNO = (SELECT DEPTNO FROM DEPT WHERE LOC
= 'CHICAGO');
```

13. WAQTD NAME AND SAL OF THE EMPLOYEES EARNING MORE THAN KING IN THE DEPARTMENT ACCOUNTING

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL > (SELECT SAL FROM EMP WHERE ENAME =
'KING')
AND   DEPTNO = (SELECT DEPTNO FROM DEPT WHERE
DNAME = 'ACCOUNTING');
```

14. WAQTD DETAILS OF THE EMPLOYEES WORKING AS SALESMAN IN THE DEPARTMENT SALES

```
SELECT *
FROM EMP
WHERE JOB = 'SALESMAN'
AND   DEPTNO = (SELECT DEPTNO FROM DEPT WHERE
DNAME = 'SALES');
```

15. WAQTD NAME, SAL, JOB, HIRE DATE OF THE EMPLOYEES WORKING IN OPERATIONS DEPARTMENT AND HIRED BEFORE KING

```
SELECT ENAME, SAL, JOB, HIREDATE
FROM EMP
WHERE DEPTNO = (SELECT DEPTNO FROM DEPT WHERE
DNAME = 'OPERATIONS')
AND   HIREDATE < (SELECT HIREDATE FROM EMP WHERE
ENAME = 'KING');
```

16. WAQTD LIST ALL THE EMPLOYEES WHOSE DEPARTMENT NAME ENDS WITH 'S'.

```
SELECT *
FROM EMP
WHERE DEPTNO IN (SELECT DEPTNO FROM DEPT WHERE
DNAME LIKE '%S');
```

17. WAQTD DISPLAY THE DEPARTMENT NAME OF THE EMPLOYEES WHOSE NAMES HAVE THE CHARACTER 'A' IN IT.

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO IN (SELECT DEPTNO FROM EMP WHERE
ENAME LIKE '%A%');
```

18. WAQTD DISPLAY THE DEPARTMENT NAME AND LOCATION OF THE EMPLOYEES WHOSE SALARY IS 800.

```
SELECT DNAME, LOC
FROM DEPT
WHERE DEPTNO = (SELECT DEPTNO FROM EMP WHERE
SAL = 800);
```

19. WAQTD SHOW THE DEPARTMENT NAME OF THE EMPLOYEES WHO EARN COMMISSION.

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO IN (SELECT DEPTNO FROM EMP WHERE
COMM IS NOT NULL);
```

20. WAQTD DISPLAY THE LOCATION OF THE EMPLOYEES IF THEY EARN COMMISSION AND BELONG TO DEPARTMENT 40.

```
SELECT LOC
FROM DEPT
WHERE DEPTNO = 40
AND   DEPTNO IN (SELECT DEPTNO FROM EMP WHERE
COMM IS NOT NULL);
```

21. WAQTD SHOW THE NAME OF THE EMPLOYEES EARNING SALARY MORE THAN ALL MANAGERS.

```
SELECT ENAME
FROM EMP
WHERE SAL > ALL (SELECT SAL FROM EMP WHERE JOB =
'MANAGER');
```

22. WAQTD DISPLAY THE DETAILS OF THE EMPLOYEES HIRED AFTER ALL THE CLERKS.

```
SELECT *
FROM EMP
WHERE HIREDATE > ALL (SELECT HIREDATE FROM EMP
WHERE JOB = 'CLERK');
```

23. WAQTD SHOW THE NAME AND SALARY OF ALL THE EMPLOYEES IF THEY ARE EARNING LESS THAN AT LEAST ONE MANAGER.

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL < ANY (SELECT SAL FROM EMP WHERE JOB =
'MANAGER');
```

24. WAQTD SHOW THE NAME AND HIRE DATE OF THE EMPLOYEES HIRED BEFORE ALL THE MANAGERS.

```
SELECT ENAME, HIREDATE
FROM EMP
WHERE HIREDATE < ALL (SELECT HIREDATE FROM EMP
WHERE JOB = 'MANAGER');
```

25. WAQTD DISPLAY THE NAME OF THE EMPLOYEES HIRED AFTER ANY MANAGER AND EARNING SALARY MORE THAN ALL THE CLERKS.

```
SELECT ENAME
FROM EMP
WHERE HIREDATE > ANY (SELECT HIREDATE FROM EMP
WHERE JOB = 'MANAGER')
AND   SAL > ALL (SELECT SAL FROM EMP WHERE JOB =
'CLERK');
```

26. WAQTD DISPLAY THE DETAILS OF THE EMPLOYEES WORKING AS CLERK AND HIRED BEFORE AT LEAST ONE SALESMAN.

```
SELECT *
FROM EMP
WHERE JOB = 'CLERK'
AND   HIREDATE < ANY (SELECT HIREDATE FROM EMP
WHERE JOB = 'SALESMAN');
```

27. WAQTD SHOW THE DETAILS OF EMPLOYEES WORKING IN ACCOUNTING OR SALES DEPARTMENT.

```
SELECT *
FROM EMP
WHERE DEPTNO IN (
   SELECT DEPTNO
   FROM DEPT
   WHERE DNAME IN ('ACCOUNTING', 'SALES')
);
```

28. WAQTD SHOW THE DEPARTMENT NAME OF THE EMPLOYEES SMITH, KING, AND MILLER.

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO IN (
   SELECT DEPTNO
   FROM EMP
   WHERE ENAME IN ('SMITH', 'KING', 'MILLER')
);
```

29. WAQTD SHOW THE DETAILS OF EMPLOYEES WORKING IN NEW YORK OR CHICAGO.

```
SELECT *
FROM EMP
WHERE DEPTNO IN (
   SELECT DEPTNO
   FROM DEPT
   WHERE LOC IN ('NEW YORK', 'CHICAGO')
);
```

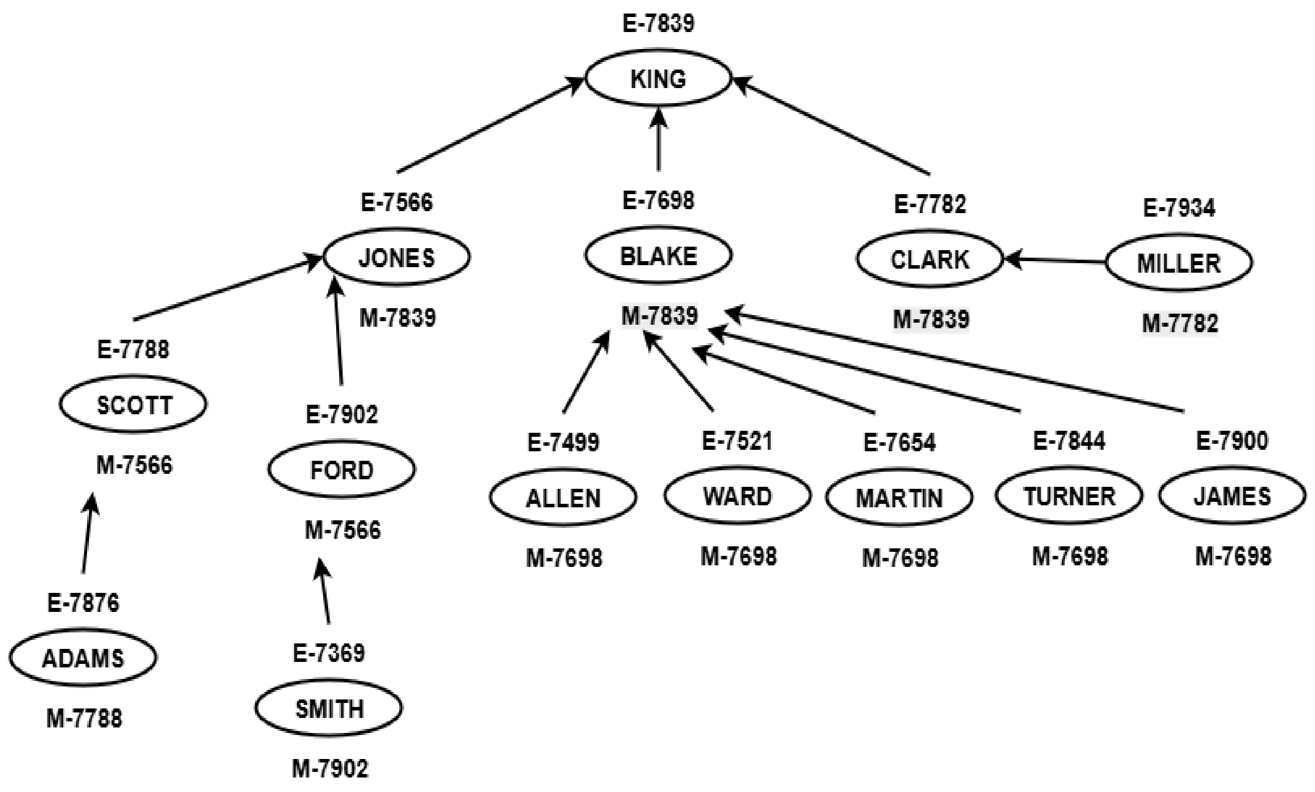
30. WAQTD DISPLAY THE EMPLOYEE NAME IF THEY ARE HIRED AFTER ALL THE EMPLOYEES OF DEPARTMENT 10.

```
SELECT ENAME
FROM EMP
WHERE HIREDATE > ALL (
   SELECT HIREDATE
   FROM EMP
   WHERE DEPTNO = 10
);
```

EMPLOYEE REPORTING MANAGER (MGR RELATIONSHIP)

```
    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
```

RELATIONSHIP DIAGRAM FOR MGR



EMPLOYEE REPORTING MANAGER (MGR RELATIONSHIP)

NOTE:

EMPNO = Employee number

MGR = Reporting Manager's Employee Number

CASE 1 – TO FIND REPORTING MANAGER DETAIL

1. **WAQTD SMITH'S REPORTING MANAGER DETAIL**

```
SELECT *
FROM EMP
WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'SMITH');
```

2. **WAQTD JAMES' REPORTING MANAGER DETAILS**

```
SELECT *
FROM EMP
WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'JAMES');
```

3. **WAQTD JONES' REPORTING MANAGER DETAILS**

```
SELECT *
FROM EMP
WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'JONES');
```

4. **WAQTD ADAMS’ REPORTING MANAGER DETAILS**

```
SELECT *
FROM EMP
WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'ADAMS');
```

5. **WAQTD MILLER’S REPORTING MANAGER DETAILS**

```
SELECT *
FROM EMP
WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'MILLER');
```

6. **WAQTD SMITH’S REPORTING MANAGER DETAILS**

```
SELECT *
FROM EMP
WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'SMITH');
```

7. **WAQTD ALLEN’S REPORTING MANAGER DETAILS**

```
SELECT *
FROM EMP
WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'ALLEN');
```

CASE 2 – TO FIND EMPLOYEES REPORTING TO A PARTICULAR REPORTING MANAGER

1. WAQTD DETAILS OF EMPLOYEES REPORTING TO KING

```
SELECT *
FROM EMP
WHERE MGR = (SELECT EMPNO FROM EMP WHERE
ENAME = 'KING');
```

2. WAQTD DETAILS OF EMPLOYEES REPORTING TO JONES

```
SELECT *
FROM EMP
WHERE MGR = (SELECT EMPNO FROM EMP WHERE
ENAME = 'JONES');
```

3. WAQTD DETAILS OF EMPLOYEES REPORTING TO CLARK

```
SELECT *
FROM EMP
WHERE MGR = (SELECT EMPNO FROM EMP WHERE
ENAME = 'CLARK');
```

4. WAQTD DETAILS OF EMPLOYEES REPORTING TO BLAKE

```
SELECT *
FROM EMP
WHERE MGR = (SELECT EMPNO FROM EMP WHERE
ENAME = 'BLAKE');
```

QUESTIONS ON EMP-MGR RELATIONSHIP

1. WAQTD ALLEN'S REPORTING MANAGER'S NAME

```
SELECT ENAME
FROM EMP
WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'ALLEN');
```

2. WAQTD MILLER'S REPORTING MANAGER'S NAME

```
SELECT ENAME
FROM EMP
WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'MILLER');
```

3. WAQTD DNAME OF JAMES MGR

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO = (
   SELECT DEPTNO
   FROM EMP
   WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'JAMES'));
```

4. WAQTD ADAMS MGR'S MGR NAME

```
SELECT ENAME
FROM EMP
WHERE EMPNO = (
   SELECT MGR
   FROM EMP
   WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'ADAMS'));
```

5. **WAQTD DNAME OF JONES MGR**

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO = (
   SELECT DEPTNO
   FROM EMP
   WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'JONES')
);
```

6. **WAQTD ADAMS REPORTING MANAGER'S NAME**

```
SELECT ENAME
FROM EMP
WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'ADAMS');
```

7. **WAQTD NUMBER OF EMPLOYEES REPORTING TO CLARK'S MGR**

```
SELECT COUNT(*)
FROM EMP
WHERE MGR = (SELECT MGR FROM EMP WHERE ENAME =
'CLARK');
```

8. **WAQTD MILLER'S MGR SALARY**

```
SELECT SAL
FROM EMP
WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'MILLER');
```

9. **WAQTD LOC OF SMITH'S MGR'S MANAGER**

```
SELECT LOC
FROM DEPT
WHERE DEPTNO = (
   SELECT DEPTNO
   FROM EMP
   WHERE EMPNO = (
      SELECT MGR
      FROM EMP
      WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'SMITH')
   )
);
```

10. **WAQTD KING'S REPORTING MANAGER'S NAME**

```
SELECT ENAME
FROM EMP
WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'KING');
```

11. **WAQTD DNAME OF EMPLOYEES REPORTING TO TURNER'S MGR**

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO IN (
   SELECT DEPTNO
   FROM EMP
   WHERE MGR = (SELECT MGR FROM EMP WHERE ENAME
= 'TURNER'));
```

12. WAQTD DNAME OF EMPLOYEES REPORTING TO BLAKE

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO IN (
   SELECT DEPTNO
   FROM EMP
   WHERE MGR = (SELECT EMPNO FROM EMP WHERE
ENAME = 'BLAKE'));
```

13. WAQTD DNAME OF JONES MGR

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO = (
   SELECT DEPTNO
   FROM EMP
   WHERE EMPNO = (SELECT MGR FROM EMP WHERE
ENAME = 'JONES'));
```

14. WAQTD NUMBER OF EMPLOYEES REPORTING TO KING

```
SELECT COUNT(*)
FROM EMP
WHERE MGR = (SELECT EMPNO FROM EMP WHERE
ENAME = 'KING');
```

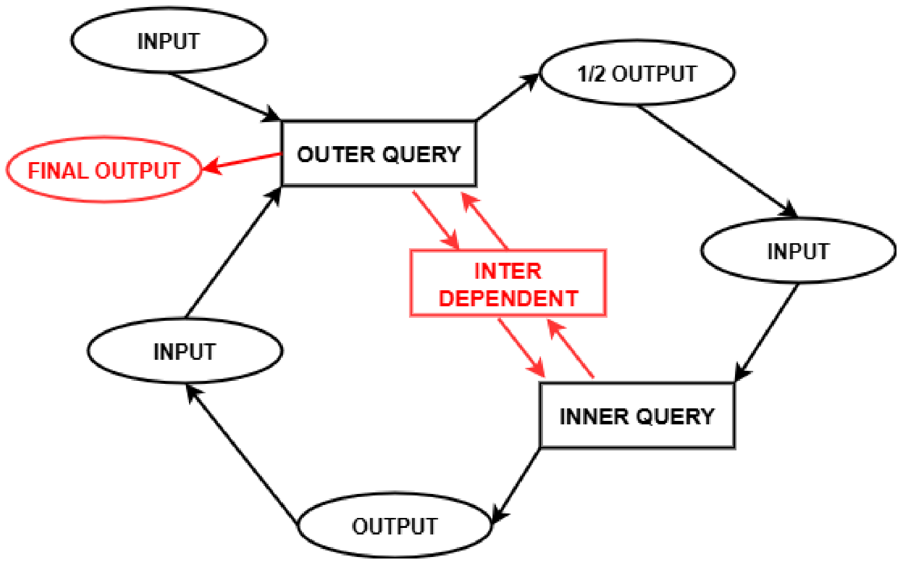
15. WAQTD DETAILS OF EMPLOYEES REPORTING TO JONES

```
SELECT *
FROM EMP
WHERE MGR = (SELECT EMPNO FROM EMP WHERE
ENAME = 'JONES');
```

CO-RELATED SUBQUERY

- A query written inside another query where both the queries are interdependent is known as a Co-related Subquery.

WORKING PRINCIPLE OF CORELATED SUBQUERY



- Let us consider two queries , They are:

1. Outer Query
2. Inner Query

- Outer Query starts the execution and gives partial output.
- The partial output will be fed as input to Inner Query.
- Inner Query starts the execution and gives output.
- The output of Inner Query will be fed as input to Outer Query.
- Outer Query will start the execution a second time to generate final output.
- Hence, both the queries are inter-dependent. So, it is called as Corelated Subquery.

JOINS

- ➢ The process of retrieval of data from multiple tables simultaneously is known as joins.

WHY / WHEN WE USE JOINS

- ➢ Whenever the attribute is to be selected from both the tables, we use joins.

TYPES OF JOINS

- ➢ We have 5 types of joins.

1. Cartesian join / Cross join
2. Inner join / Equi join
3. Outer join
    1. Left outer join
    2. Right outer join
    3. Full outer join
4. Natural join
5. Self join

CARTESIAN JOIN / CROSS JOIN

- In Cartesian join, a whole table 1 will be merged with the first record of table 2 and it continues.

RESULT TABLE DETAILS

- Number of columns in the result table :

NUMBER OF COLUMN = NUMBER OF COLUMN IN TABLE 1 + NUMBER OF COLUMN TABLE 2 WILL BE EQUIVALENT TO THE SUMMATION OF THE COLUMNS PRESENT IN BOTH THE TABLES
= 8 + 3 = 11 COLUMNS

- Number of row in the result table :

NUMBER OF ROWS = NUMBER OF ROWS IN TABLE 1 X NUMBER OF ROWS IN TABLE 2 WILL BE EQUIVALENT TO THE PRODUCT OF THE NUMBER OF ROWS PRESENT IN BOTH THE TABLES,
= 14 X 4 = 56 ROWS.

NOTE : ANSI (AMERICAN NATIONAL STANDARD INSTITUTE).

SYNTAX IN ANSI

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 CROSS JOIN TABLE_NAME2
```

EXAMPLE :

```
SELECT *
FROM EMP CROSS JOIN DEPT ;
```

SYNTAX IN ORACLE

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 , TABLE_NAME2 ;
```

EXAMPLE :

```
SELECT *
FROM EMP , DEPT ;
```

INNER JOIN OR EQUI-JOIN

- Inner join is used to obtain only matching records or a record which has a pair.

JOIN CONDITION

- It is a condition on which two tables are merged.

```
TABLE_NAME1 . COLUMN_NAME = TABLE_NAME2 .
COLUMN_NAME
```

EXAMPLE :

```
EMP . DEPTNO = DEPT . DEPTNO
```

SYNTAX IN ANSI

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 INNER JOIN TABLE_NAME2
ON < JOIN_CONDITION > ;
```

EXAMPLE :

```
SELECT *
FROM EMP INNER JOIN DEPT
ON EMP . DEPTNO = DEPT . DEPTNO ;
```

SYNTAX IN ORACLE

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 , TABLE_NAME2
WHERE < JOIN_CONDITION > ;
```

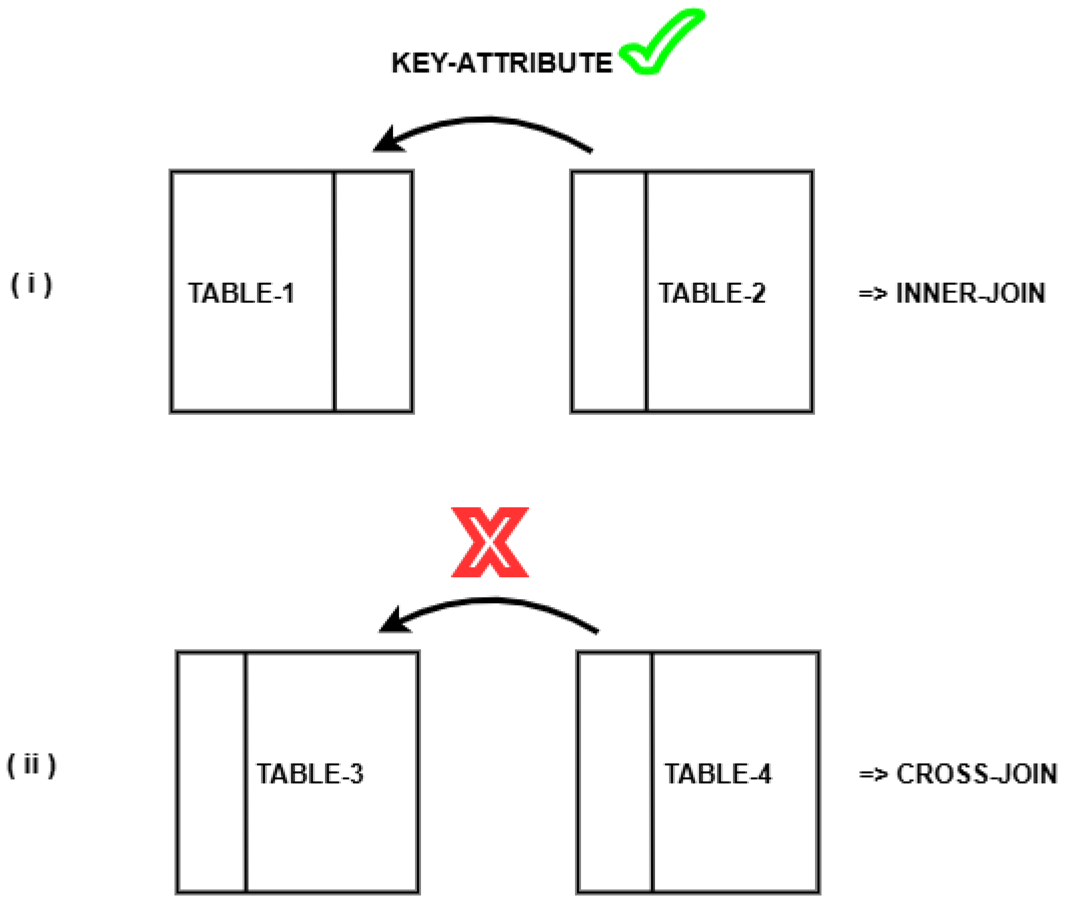
EXAMPLE :

```
SELECT *
FROM EMP E , DEPT  D
WHERE E . DEPTNO = D . DEPTNO ;
```

NOTE : WE MUST KNOW AN IMPORTANT CONCEPT . SOME TABLE WILL HAVE LARGE TABLE NAMES SO IT WILL BE DIFFICULT TO WRITE THE NAME EVERYTIME . SO , WE CAN WRITE THIS INSTEAD EMP E AND DEPT D . WHERE 'E' WILL BE EQUAL TO EMP AND 'D' WILL BE EQUAL TO DEPT .

NATURAL JOIN

- It behaves as inner join if there is a relation between a given table, else it behaves as cross join.



SYNTAX IN ANSI

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 NATURAL JOIN TABLE_NAME2 ;
```

EXAMPLE :

BEHAVES AS INNER JOIN

```
SELECT *
FROM EMP NATURAL JOIN DEPT ;
```

BEHAVES AS CROSS JOIN

```
SELECT *
FROM EMP NATURAL JOIN SAL_GRADE ;
```

EXAMPLE QUESTIONS ON INNER JOINS

NOTE : I HAVE WRITTEN IN BOTH ANSI FORMAT AS WELL AS ORACLE FORMAT . FIRST ONE IS ANSI AND SECOND ONE IS ORACLE IN EACH QUESTIONS

1. WAQTD ENAME AND DEPARTMENT NAME FOR ALL THE EMPLOYEES?

```
SELECT E.ENAME, D.DNAME
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO;
```

```
SELECT E.ENAME, D.DNAME
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO;
```

2. WAQTD ENAME, LOC FOR ALL THE EMPLOYEES WORKING AS MANAGER?

```
SELECT E.ENAME, D.LOC
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE E.JOB = 'MANAGER';
```

```
SELECT E.ENAME, D.LOC
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND   E.JOB = 'MANAGER';
```

3. WAQTD ENAME, SAL AND DNAME OF THE EMPLOYEES WORKING AS CLERK IN DEPARTMENT 20 WITH SALARY LESS THAN 1800?

```
SELECT E.ENAME, E.SAL, D.DNAME
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE E.JOB = 'CLERK'
AND   E.DEPTNO = 20
AND   E.SAL < 1800;
```

```
SELECT E.ENAME, E.SAL, D.DNAME
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND   E.JOB = 'CLERK'
AND   E.DEPTNO = 20
AND   E.SAL < 1800;
```

4. **WAQTD E-NAME, D-NAME AND LOC FOR THE EMPLOYEES EARNING MORE THAN 2000 IN NEW YORK?**

```
SELECT E.ENAME, D.DNAME, D.LOC
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE E.SAL > 2000
AND   D.LOC = 'NEW YORK';
```

```
SELECT E.ENAME, D.DNAME, D.LOC
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND   E.SAL > 2000
AND   D.LOC = 'NEW YORK';
```

5. **WAQTD E-NAME, JOB, DNAME AND DEPARTMENT NUMBER FOR THE EMPLOYEES EARNING LESS THAN 5000?**

```
SELECT E.ENAME, E.JOB, D.DNAME, E.DEPTNO
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE E.SAL < 5000;
```

```
SELECT E.ENAME, E.JOB, D.DNAME, E.DEPTNO
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND   E.SAL < 5000;
```

6. WAQTD NAME OF THE EMPLOYEES AND LOCATION.

```
SELECT E.ENAME, D.LOC
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO;
```

```
SELECT E.ENAME, D.LOC
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO;
```

7. WAQTD DNAME AND SALARY FOR ALL THE EMPLOYEES WORKING IN ACCOUNTING.

```
SELECT D.DNAME, E.SAL
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE D.DNAME = 'ACCOUNTING';
```

```
SELECT D.DNAME, E.SAL
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND   D.DNAME = 'ACCOUNTING';
```

8. WAQTD DNAME AND ANNUAL SAL FOR ALL EMPLOYEES WHOSE SAL IS MORE THAN 2340.

```
SELECT D.DNAME, E.SAL * 12 AS ANNUAL_SAL
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE E.SAL > 2340;
```

```
SELECT D.DNAME, E.SAL * 12 AS ANNUAL_SAL
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND   E.SAL > 2340;
```

9. WAQTD E-NAME AND D-NAME FOR EMPLOYEES HAVING CHARACTER ‘A’ IN THEIR D-NAME .

```
SELECT E.ENAME, D.DNAME
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE D.DNAME LIKE '%A%';
```

```
SELECT E.ENAME, D.DNAME
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND   D.DNAME LIKE '%A%';
```

10. WAQTD ENAME AND DNAME FOR ALL THE EMPLOYEES WORKING AS SALESMAN .

```
SELECT E.ENAME, D.DNAME
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE E.JOB = 'SALESMAN';
```

```
SELECT E.ENAME, D.DNAME
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND   E.JOB = 'SALESMAN';
```

11. WAQTD D-NAME AND JOB FOR ALL THE EMPLOYEES WHOSE JOB AND D-NAME STARTS WITH CHARACTER ‘S’ .

```
SELECT D.DNAME, E.JOB
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE E.JOB LIKE 'S%'
AND   D.DNAME LIKE 'S%';
```

```
SELECT D.DNAME, E.JOB
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND   E.JOB LIKE 'S%'
AND   D.DNAME LIKE 'S%';
```

12. WAQTD D-NAME AND MGR NO. FOR EMPLOYEES REPORTING TO 7839.

```
SELECT D.DNAME, E.MGR
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE E.MGR = 7839;
```

```
SELECT D.DNAME, E.MGR
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND   E.MGR = 7839;
```

13. WAQTD D-NAME AND HIRED DATE FOR EMPS HIRED AFTER 83 INTO ACCOUNTING OR RESEARCH DEPARTMENT.

```
SELECT D.DNAME, E.HIREDATE
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE E.HIREDATE > TO_DATE('01-JAN-1984', 'DD-MON-
YYYY')
AND   D.DNAME IN ('ACCOUNTING', 'RESEARCH');
```

```
SELECT D.DNAME, E.HIREDATE
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND   E.HIREDATE > TO_DATE('01-JAN-1984', 'DD-MON-
YYYY')
AND   D.DNAME IN ('ACCOUNTING', 'RESEARCH');
```

14. WAQTD E-NAME, DEPT NO. AND D-NAME OF THE EMPS WHO ARE GETTING COMMISSION IN DEPT 10 OR 30.

```
SELECT E.ENAME, E.DEPTNO, D.DNAME
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE E.COMM IS NOT NULL
AND   E.DEPTNO IN (10, 30);
```

```
SELECT E.ENAME, E.DEPTNO, D.DNAME
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND   E.COMM IS NOT NULL
AND   E.DEPTNO IN (10, 30);
```

15. WAQTD DNAME AND DEPARTMENT NO. FOR ALL THE EMPLOYEES WHOSE EMPLOYEE NO. ARE (7839, 7902) AND ARE WORKING IN LOC NEW YORK.

```
SELECT D.DNAME, D.DEPTNO
FROM EMP E
INNER JOIN DEPT D ON E.DEPTNO = D.DEPTNO
WHERE E.EMPNO IN (7839, 7902)
AND   D.LOC = 'NEW YORK';
```

```
SELECT D.DNAME, D.DEPTNO
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND   E.EMPNO IN (7839, 7902)
AND   D.LOC = 'NEW YORK';
```

OUTER JOIN

- It is used to obtain the unmatched records from the particular table along with the matched records.
- In outer join, we have three types they are :

1. Left outer join
2. Right outer join
3. Full outer join

LEFT OUTER JOIN

- It is used to obtain the unmatched records from the left table along with the matched records.

SYNTAX IN ANSI

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 LEFT OUTER JOIN TABLE_NAME2
ON < JOIN_CONDITION > ;
```

EXAMPLE :

```
SELECT *
FROM EMP E LEFT OUTER JOIN DEPT D
ON E . DEPTNO = D . DEPTNO ;
```

SYNTAX IN ORACLE

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 , TABLE_NAME2
WHERE TABLE_NAME1 . COLUMN_NAME = TABLE_NAME2 .
COLUMN_NAME [ + ] ;
```

EXAMPLE :

```
SELECT *
FROM EMP E , DEPT  D
WHERE E . DEPTNO = D . DEPTNO ( + ) ;
```

RIGHT OUTER JOIN

- It is used to obtain the unmatched records from the right table along with the matched records.

SYNTAX IN ANSI

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 RIGHT OUTER JOIN TABLE_NAME2
ON < JOIN_CONDITION > ;
```

EXAMPLE :

```
SELECT *
FROM EMP E RIGHT OUTER JOIN DEPT D
ON E . DEPTNO = D . DEPTNO ;
```

SYNTAX IN ORACLE

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 , TABLE_NAME2
WHERE TABLE_NAME1 . COLUMN_NAME [ + ]  = TABLE_NAME2
. COLUMN_NAME ;
```

EXAMPLE :

```
SELECT *
FROM EMP E , DEPT  D
WHERE E . DEPTNO ( + ) = D . DEPTNO ;
```

FULL OUTER JOIN

- It is used to obtain the unmatched records from both the tables along with the matched records.

SYNTAX IN ANSI

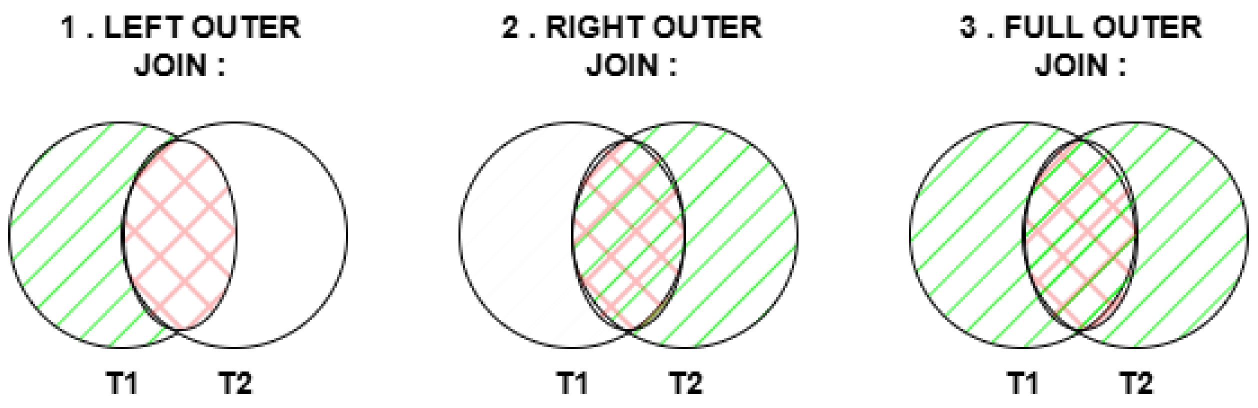
```
SELECT COLUMN_NAME
FROM TABLE_NAME1 FULL OUTER JOIN TABLE_NAME2
ON < JOIN_CONDITION > ;
```

EXAMPLE :

```
SELECT *
FROM EMP E FULL OUTER JOIN DEPT D
ON E . DEPTNO = D . DEPTNO ;
```

OUTER JOIN OPERATOR

EXAMPLE :



NOTE :

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 , TABLE_NAME2 ;
```

TABLE_NAME1 = LHS (LEFT TABLE)
TABLE_NAME2 = RHS (RIGHT TABLE)

WHERE TO USE OUTER JOIN OPERATOR (+) : (ORACLE)

1. LEFT OUTER JOIN : EMP.DEPTNO = DEPT.DEPTNO (+)
2. RIGHT OUTER JOIN : EMP.DEPTNO (+) = DEPT.DEPTNO

SELF JOIN

- Joining a table by itself is known as self-join.

WHY / WHEN WE USE SELF JOIN

- Whenever the data to be selected is present in the same table, but present in different records, we use self-join.

SYNTAX IN ANSI

```
SELECT COLUMN_NAME
FROM TABLE_NAME T1 JOIN TABLE_NAME T2
ON < JOIN_CONDITION > ;
```

EXAMPLE :

```
SELECT *
FROM EMP E1 JOIN EMP E2
ON E1 . MGR = E2 . EMPNO ;
```

NOTE :

```
EMP E1 = EMPS DETAILS
EMP E2 = MGR DETAILS
JOIN CONDITION = E1 . MGR = E2 . EMPNO ;
```

SYNTAX IN ORACLE

```
SELECT COLUMN_NAME
FROM TABLE_NAME T1 , TABLE_NAME T2
WHERE  < JOIN_CONDITION > ;
```

EXAMPLE :

```
SELECT *
FROM EMP E1 , EMP E2
WHERE E1 . MGR = E2 . EMPNO ;
```

EXAMPLE QUESTIONS ON SELF JOIN

NOTE : I HAVE WRITTEN IN ANSI FORMAT AS WELL AS ORACLE FORMAT . FIRST ONE IS ANSI AND SECOND ONE IS ORACLE

1. WAQTD EMPLOYEE NAME ALONG WITH THEIR REPORTING MANAGER'S NAME FROM THE EMPLOYEE TABLE.

```
SELECT E.ENAME AS EMPLOYEE_NAME, M.ENAME AS
MANAGER_NAME
FROM EMP E
JOIN EMP M ON E.MGR = M.EMPNO;
```

```
SELECT E.ENAME AS EMPLOYEE_NAME, M.ENAME AS
MANAGER_NAME
FROM EMP E, EMP M
WHERE E.MGR = M.EMPNO;
```

2. **WAQTD EMPLOYEE'S NAME, EMPLOYEE'S SALARY ALONG WITH THEIR REPORTING MANAGER'S NAME AND SALARY.**

```
SELECT E.ENAME AS EMPLOYEE_NAME, E.SAL AS
EMP_SAL,
      M.ENAME AS MANAGER_NAME, M.SAL AS
MANAGER_SAL
FROM EMP E
JOIN EMP M ON E.MGR = M.EMPNO;
```

```
SELECT E.ENAME AS EMPLOYEE_NAME, E.SAL AS
EMP_SAL,
      M.ENAME AS MANAGER_NAME, M.SAL AS
MANAGER_SAL
FROM EMP E, EMP M
WHERE E.MGR = M.EMPNO;
```

3. **WAQTD EMPLOYEE NAME AND EMPLOYEE'S DESIGNATION, REPORTING MANAGER'S NAME, HIS DESIGNATION, AND EMPLOYEES SHOULD BE WORKING AS CLERK, REPORTING MANAGER SHOULD BE WORKING AS ACTUAL MANAGER.**

```
SELECT E.ENAME AS EMPLOYEE_NAME, E.JOB AS
EMP_JOB,
       M.ENAME AS MANAGER_NAME, M.JOB AS
MANAGER_JOB
FROM EMP E
JOIN EMP M ON E.MGR = M.EMPNO
WHERE E.JOB = 'CLERK'
AND   M.JOB = 'MANAGER';
```

```
SELECT E.ENAME AS EMPLOYEE_NAME, E.JOB AS
EMP_JOB,
       M.ENAME AS MANAGER_NAME, M.JOB AS
MANAGER_JOB
FROM EMP E, EMP M
WHERE E.MGR = M.EMPNO
AND   E.JOB = 'CLERK'
AND   M.JOB = 'MANAGER';
```

4. WAQTD EMPLOYEE NAME AND EMPLOYEE'S DESIGNATION, REPORTING MANAGER'S DEPARTMENT NUMBER, AND BOTH THE EMPLOYEES SHOULD BE WORKING IN SAME DEPARTMENT.

```
SELECT E.ENAME AS EMPLOYEE_NAME, E.JOB AS
EMP_JOB,
      M.DEPTNO AS MANAGER_DEPTNO
FROM EMP E
JOIN EMP M ON E.MGR = M.EMPNO
WHERE E.DEPTNO = M.DEPTNO;
```

```
SELECT E.ENAME AS EMPLOYEE_NAME, E.JOB AS
EMP_JOB,
      M.DEPTNO AS MANAGER_DEPTNO
FROM EMP E, EMP M
WHERE E.MGR = M.EMPNO
AND   E.DEPTNO = M.DEPTNO;
```

5. WAQTD REPORTING MANAGER'S SALARY, EMPLOYEE'S SALARY, AND REPORTING MANAGER SHOULD BE EARNED MORE THAN EMPLOYEE.

```
SELECT M.SAL AS MANAGER_SAL, E.SAL AS EMPLOYEE_SAL
FROM EMP E
JOIN EMP M ON E.MGR = M.EMPNO
WHERE M.SAL > E.SAL;
```

```
SELECT M.SAL AS MANAGER_SAL, E.SAL AS EMPLOYEE_SAL
FROM EMP E, EMP M
WHERE E.MGR = M.EMPNO
AND   M.SAL > E.SAL;
```

6. **WAQTD EMPLOYEE NAME, HIRE DATE, REPORTING MANAGER'S NAME, HIRE DATE, AND REPORTING MANAGER SHOULD BE HIRED BEFORE THE EMPLOYEES.**

```
SELECT E.ENAME AS EMPLOYEE_NAME, E.HIREDATE AS
EMP_HIREDATE,
      M.ENAME AS MANAGER_NAME, M.HIREDATE AS
MGR_HIREDATE
FROM EMP E
JOIN EMP M ON E.MGR = M.EMPNO
WHERE M.HIREDATE < E.HIREDATE;
```

```
SELECT E.ENAME AS EMPLOYEE_NAME, E.HIREDATE AS
EMP_HIREDATE,
      M.ENAME AS MANAGER_NAME, M.HIREDATE AS
MGR_HIREDATE
FROM EMP E, EMP M
WHERE E.MGR = M.EMPNO
AND   M.HIREDATE < E.HIREDATE;
```

STATEMENTS

1. Data Definition Language
2. Data Manipulation Language
3. Transaction Control Language
4. Data Control Language
5. Data Query Language

DATA DEFINITION LANGUAGE

- Data Definition Language is used to maintain and manage the structure of the table.
- In DDL, we have 5 statements. They are:

1. **CREATE**
2. **RENAME**
3. **ALTER**
4. **TRUNCATE**
5. **DROP**

CREATE

- This statement is used to create a new table in the database.

HOW TO CREATE A TABLE?

1. Tables cannot have the same names.
2. Assign a name to the table.
3. Assign the number of columns.
4. Assign names to the columns.
5. Assign data types for the columns – mandatory.
6. Assign constraints – not mandatory.

EXAMPLE:
TABLE NAME = PRODUCT
NO OF COLUMN = 4

COLUMN_NAMES	PID	PNAME	PRICE	DIST
DATATYPE	NUMBER(02)	VARCHAR	NUMBER(8,2) NOT NULL	NUMBER(8,2)
CONSTRAINTS	UNIQUE , NOT NULL	NOT NULL		
	PRIMARY KEY			

SYNTAX

```
CREATE TABLE TABLE_NAME (
COL_NAME1 DATATYPE CONSTRAINTS ,
COL_NAME2 DATATYPE CONSTRAINTS ,
COL_NAME3 DATATYPE CONSTRAINTS ,
,
,
COL_NAME_N DATATYPE CONSTRAINTS ,);
```

EXAMPLE :

```
CREATE TABLE PRODUCT (
PID NUMBER(02) NOT NULL ,
PNAME VARCHAR(20) NOT NULL ,
PRICE NUMBER(8,2) NOT NULL ,
DIST NUMBER(8,2)
);
```

RENAME

- This statement is used to change the table name to a new name.

SYNTAX

```
RENAME CURRENT_TABLE_NAME TO NEW_TABLE_NAME ;
```

EXAMPLE :

```
RENAME PRODUCT TO PRO ;
```

TO CREATE A COPY OF A TABLE

SYNTAX

```
CREATE TABLE TABLE_NAME
AS
SQL STATEMENT ;
```

EXAMPLES:

```
CREATE TABLE CLERK
AS
SELECT *
FROM EMP
WHERE JOB = ‘CLERK’ ;
```

```
CREATE TABLE SALESMAN
AS
SELECT EMPNO , ENAME , JOB , SAL , COMM , DEPTNO
FROM EMP
WHERE JOB IN ‘SALESMAN’
ORDER BY COMM DESC;
```

```
CREATE TABLE MANAGER
AS
SELECT E.EMPNO , E.ENAME , D.DNAME , E.SAL , E.DEPTNO ,
D.LOC , E.HIREDATE
FROM EMP E , DEPT D
WHERE E.DEPTNO = D.DEPTNO AND JOB IN ‘MANAGER’ ;
```

ALTER

- This statement is used to modify the structure of the table.
- In Alter, we have 5 sub-statements.

1. TO ADD A COLUMN:

SYNTAX

```
ALTER TABLE TABLE_NAME
ADD COLUMN_NAME DATATYPE CONSTRAINTS ;
```

EXAMPLE :

```
ALTER TABLE PRO
ADD MODEL_NO VARCHAR(10) NOT NULL ;
```

2. TO DROP A COLUMN :

SYNTAX

```
ALTER TABLE TABLE_NAME
DROP COLUMN COLUMN_NAME ;
```

EXAMPLE :

```
ALTER TABLE PRO
DROP COLUMN  MODEL_NO ;
```

3. TO RENAME THE COLUMN:

SYNTAX

```
ALTER TABLE TABLE_NAME
RENAME COLUMN COLUMN_NAMETO NEW_COLUMN_NAME ;
```

EXAMPLE :

```
ALTER TABLE PRO
RENAME COLUMN PRICE TO MRP ;
```

4. TO MODIFY / CHANGE THE DATATYPE :

SYNTAX

```
ALTER TABLE TABLE_NAME
MODIFY COLUMN_NAME NEW_DATATYPE ;
```

EXAMPLE :

```
ALTER TABLE PRO
MODIFY PID CHAR(04) ;
```

5. TO MODIFY THE CONSTRAINTS :

SYNTAX

```
ALTER TABLE TABLE_NAME
MODIFY COLUMN_NAME DATATYPE NEW_CONSTRAINTS
[ NULL / NOT NULL ] ;
```

EXAMPLE :

```
ALTER TABLE PRO
MODIFY DIST NUMBER(8,2) NOT NULL ;
```

TRUNCATE

- This statement is used to empty the table without disturbing the structure of the table.

SYNTAX

```
TRUNCATE TABLE TABLE_NAME ;
```

EXAMPLE :

```
TRUNCATE TABLE SALESMAN ;
```

TABLE1 :

TRUNCATE
------------>
EMPTY

COLUMN1	COLUMN2
-------------	-------------
-------------	-------------
-------------	-------------
-------------	-------------

-----> TRASH

DROP

- This statement is used to drop the table from the database to the Recycle Bin.

SYNTAX

```
DROP TABLE TABLE_NAME ;
```

1. FLASHBACK

- This statement is used to restore the table from the Recycle Bin back to the database.

SYNTAX

```
FLASHBACK TABLE TABLE_NAME
TO BEFORE DROP ;
```

2. PURGE

- This statement is used to delete the table from the Recycle Bin.

SYNTAX

```
PURGE TABLE TABLE_NAME ;
```

SCENARIOS :

1. **TO DELETE A TABLE PERMANENTLY :**

   - **DROP**
   - **PURGE**

2. **TO GET BACK THE TABLE FROM RECYCLE BIN :**

   - **DROP**
   - **FLASH BACK**

DATA MANIPULATION LANGUAGE

- This language is used to maintain and manage the records of a particular table.
- In DML, we have three statements:
  1. **INSERT**
  2. **UPDATE**
  3. **DELETE**

1. INSERT

- This statement is used to add records to an existing table.

SYNTAX 1

INSERT INTO TABLE _NAME VALUES (V_1 , V_2 , V_3 , … , V_N) ;

EXAMPLE :

INSERT INTO PRO VALUES (01 , ‘M1’ , 16999 , 499);

SYNTAX 2

INSERT INTO TABLE_NAME VALUES (&COL_1 , &COL_2 , …. , &COL_N) ;

EXAMPLE:

```
INSERT INTO PRO VALUES ( &PID , &PNAME , &PRICE , &DIST );
```

2. UPDATE

- This statement is used to modify a record.

SYNTAX

```
UPDATE TABLE_NAME
SET COL1 =V1
[WHERE < FILTER_CONDITION > ]
```

EXAMPLE :

```
UPDATE PRO
SET DIST = 999
WHERE PID = 2 ;
```

3. DELETE

- This statement is used to delete a specific value.
- The Delete statement can also perform the truncate operation.

SYNTAX

```
DELETE
FROM TABLE_NAME
[ WHERE < FILTER_CONDITION > ] ;
```

EXAMPLE :

```
DELETE
FROM PRO
WHERE PID = 2 ;
```

NOTE:

REDUNDANCY

- The unwanted repetition of data in the table is known as redundancy.

ANOMALY

- The side effect that occurs during the DML operations (Insert, Update, Delete) is known as an anomaly.

TRANSACTION CONTROL LANGUAGE

- This language is used to control transactions between the user interface and the database.
- In TCL, we have 3 statements. They are:
    1. **COMMIT**
    2. **SAVEPOINT**
    3. **ROLLBACK**

COMMIT

- This statement is used to save SQL operations permanently to the database.

SYNTAX

```
COMMIT ;
```

EXAMPLE:

```
COMMIT ;
```

SAVEPOINT

- This statement is used to create savepoints or checkpoints while performing SQL operations.

SYNTAX

```
SAVEPOINT SAVEPOINT_NAME ;
```

EXAMPLE :

```
SAVEPOINT S1;
```

ROLLBACK

- This statement is used to perform the undo process on a particular savepoint.
- This statement is completely dependent on savepoints, and without savepoints, this statement will not work.

SYNTAX

```
ROLLBACK ;
```

ROLLBACK TO SAVEPOINT

SYNTAX

```
ROLLBACK TO SAVEPOINT_NAME ;
```

EXAMPLE :

```
ROLLBACK TO S1 ;
```

DATA CONTROL LANGUAGE

- DCL is used to control the dataflow between the users.
- In DCL we have 2 statements , they are :

  1. **GRANT**
  2. **REVOKE**

GRANT

- This statement is used to give permission to a user on a table by using SQL statements.

SYNTAX

```
GRANT SQL_STATEMENT
ON TABLE_NAME
TO USER_NAME ;
```

EXAMPLE :

```
GRANT SELECT
ON EMP
TO HR ;
```

REVOKE

- This statement is used to take back the given permission from the user.

SYNTAX

```
REVOKE SQL_STATEMENT
ON TABLE_NAME
FROM USER_NAME;
```

EXAMPLE :

```
REVOKE SELECT
ON EMP
FROM HR ;
```

NOTE :

SHOW USER	**EX: SHOW USER**
USER IS “USER NAME “	USER IS “ SCOTT “
CONNECT USER NAME	**EX: CONNECT SCOTT**
ENTER PASSWORD : ***** CONNECTED	ENTER PASSWORD : ***** CONNECTED

TO FETCH DATA IN OTHER USER ACCOUNT

```
SELECT *
FROM USER_NAME. TABLE_NAME ;
```

EXAMPLE :

```
SELECT *
FROM SCOTT. EMP ;
```

ATTRIBUTES

- Attributes are nothing but columns.
- In attributes we have 7 types , they are :

1. **KEY ATTRIBUTE / CANDIDATE KEY**
2. **NON KEY ATTRIBUTE**
3. **PRIME KEY ATTRIBUTE**
4. **NON PRIME KEY ATTRIBUTE**
5. **COMPOSITE KEY ATTRIBUTE**
6. **SUPER KEY ATTRIBUTE**
7. **FOREIGN KEY ATTRIBUTE**

1. **KEY ATTRIBUTE / CANDIDATE KEY**
   - An attribute which is used to identify a record uniquely from a table is known as a key attribute.
2. **NON-KEY ATTRIBUTE**
   - All the attributes except the key attribute are known as non-key attributes.
3. **PRIME KEY ATTRIBUTES**
   - Among the key attributes, the attribute that is chosen to be the main attribute to identify the records uniquely from the table is known as the prime key attribute.
4. **NON-PRIME KEY ATTRIBUTE**
   - All the key attributes except the prime key attributes are known as non-prime key attributes.
5. **COMPOSITE KEY ATTRIBUTE**
   - It is a combination of two or more non-key attributes which is used to identify a record uniquely from the table.
6. **SUPER KEY ATTRIBUTE :** It is a set of all the key attributes
7. **FOREIGN KEY ATTRIBUTES**
   - It behaves like an attribute of another entity which is used to represent the relationship.

FUNCTIONAL DEPENDENCIES

- Let us consider a employee table with two columns such as
  1. EMPNO (Employee number)
  2. ENAME (Employee name)

[OR]

- Let us consider a relation with two attributes ‘x’ and ‘y’ respectively . X detrmines y or we can also say that y is dependent on x
  - R -> { X , Y }
  - X -> Y , Therefore , Y is dependent on X
- Types of functional dependencies :
  1. TOTAL F.D (T .F.D)
  2. PARTIAL F.D (P.F.D)
  3. TRANSITIVE F.D (TR.F.D)

TOTAL FUNCTIONAL DEPENDENCY

- If all the attributes in a relation are determined by a single attribute which is a key attribute and then there exists TOTAL F.D .
- In TOTAL F.D there are no redundancy and anomaly

REDUNDANCY : THE UNWANTED REPETITION OF DATA IN THE TABLE IS KNOWN AS REDUNDANCY .

ANOMALY : THE SIDE EFFECTS OCCURRED DURING THE DML OPERATIONS (INSERT , UPDATE , DELETE) IS KNOWN AS ANOMALY .

EXAMPLE:

LET US CONSIDER A RELATION OF 4 ATTRIBUTES A,B,C,D…… WHERE 'A' IS THE KEY ATTRIBUTE

R -> { A , B , C , D }

A -> B, A -> C , A -> D …….. A -> { B , C , D }

PARTIAL FUNCTIONAL DEPENDENCY

- For a PARTIAL.F.D to exist there must be a composite key relation.
- One of the attribute in the composite key relation determines another attribute separately and there exist PARTIAL.F.D .
- In P.F.D we have redundancy and anomaly .
- Let us consider a relation with 4 attributes a,b,c,d in which 'a' and 'b' are composite key attributes .

R→ { A, B ,C , D }

(A , B) -> D
B -> C

TRANSITIVE FUNCTIONAL DEPENDENCY

- An attribute is determined by a non key attribute which is internely determined by a key attribute then their exist TRANSITIVE F.D .
- In TR.F.D we have redundancy and anomaly .
- Let us consider a relation with four attributes a,b,c,d … where 'a' is the key attribute .

  R -> { A , B , C , D }

  A -> B , A -> D , D -> C … , A -> C

SYNTAX SHEET

SYNTAX FOR PROJECTION

```
SELECT */ [ DISTINCT ] COLUMN_NAME / EXPRESSION [ ALIAS ]
FROM TABLE_NAME;
```

FORMULA TO CALCULATE SALARY PERCENTAGE

- SAL + SAL * A / 100
- SAL - SAL * A / 100

FORMULAS FOR PERCENTAGE SALARY HIKE

- NORMAL SALARY PERCENTAGE HIKE
  SAL + SAL * A/100
- ANNUAL-SALARY PERCENTAGE HIKE
  SAL * 12 + SAL * 12 * A/100
- HALF-TERM SALARY PERCENTAGE HIKE
  SAL * 6+ SAL *6 * A/100
- QUARTER-TERM SALARY PERCENTAGE HIKE
  SAL * 3 + SAL *3 * A/100

TO DISPLAY A COLUMN ALONG WITH WHOLE TABLE

SYNTAX:

```
TABLE_NAME .* , COLUMN_NAME / EXPRESSION
```

SYNTAX FOR SELECTION

```
SELECT */[DISTINCT] COLUMN_NAME / EXPRESSION [ALIAS]
FROM TABLE_NAME
WHERE <FILTER_CONDITION>;
```

SYNTAX FOR ESCAPE CHARACTER

```
COLUMN_NAME / EXPRESSION LIKE / NOT LIKE '
PATTERN_TO_MATCH ' ESCAPE ' SPECIAL_CHARACTER' ;
```

SYNTAX FOR ORDER BY CLAUSE

```
SELECT GROUP_BY_EXPRESSION / GROUP_FUNCTION
FROM TABLE_NAME
WHERE <FILTER_CONDITION>
GROUP BY COLUMN_NAME / EXPRESSION ->( IF USED )
HAVING [ < GROUP_FILTER_CONDITIONS> ] -> ( IF USED )
ORDER BY COLUMN_NAME [ASC / DESC] ; -> (IF USED )
```

TYPES OF SINGLE ROW FUNCTIONS

- In Single Row Function, we have 14 types:

1. **LENGTH()**
2. **UPPER()**
3. **LOWER()**
4. **INITCAP()**
5. **REVERSE()**
6. **SUBSTR()**
7. **INSTR()**
8. **REPLACE()**
9. **MOD()**
10. **ROUND()**
11. **TRUNC()**
12. **ADD_MONTHS()**
13. **TO_CHAR()**
14. **NVL() — (NULL VALUE LOGIC)**

TYPES OF MULTI-ROW FUNCTION

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

SYNTAX FOR GROUP BY CLAUSE

```
SELECT GROUP_BY_EXPRESSION / GROUP_FUNCTION
FROM TABLE_NAME
[ WHERE < FILTER_CONDITION > ]
GROUP BY COLUMN_NAME / EXPRESSION ;
```

SYNTAX FOR HAVING CLAUSE

```
SELECT GROUP_BY_EXPRESSION / GROUP_FUNCTION
FROM TABLE_NAME
[ WHERE < FILTER_CONDITION > ]
GROUP BY COLUMN_NAME / EXPRESSION
HAVING < GROUP_FILTER_CONDITION > ;
```

SYNTAX FOR CROSS-JOIN IN ANSI

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 CROSS JOIN TABLE_NAME2
```

EXAMPLE :

```
SELECT *
FROM EMP CROSS JOIN DEPT ;
```

SYNTAX FOR CROSS-JOIN IN ORACLE

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 , TABLE_NAME2 ;
```

EXAMPLE :

```
SELECT *
FROM EMP , DEPT ;
```

SYNTAX FOR INNER-JOIN IN ANSI

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 INNER JOIN TABLE_NAME2
ON < JOIN_CONDITION > ;
```

EXAMPLE :

```
SELECT *
FROM EMP INNER JOIN DEPT
ON EMP . DEPTNO = DEPT . DEPTNO ;
```

SYNTAX FOR INNER-JOIN IN ORACLE

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 , TABLE_NAME2
WHERE < JOIN_CONDITION > ;
```

EXAMPLE :

```
SELECT *
FROM EMP E , DEPT  D
WHERE E . DEPTNO = D . DEPTNO ;
```

SYNTAX FOR NATURAL-JOIN IN ANSI

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 NATURAL JOIN TABLE_NAME2 ;
```

SYNTAX FOR LEFT-OUTER IN ANSI

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 LEFT OUTER JOIN TABLE_NAME2
ON < JOIN_CONDITION > ;
```

EXAMPLE :

```
SELECT *
FROM EMP E LEFT OUTER JOIN DEPT D
ON E . DEPTNO = D . DEPTNO ;
```

SYNTAX FOR LEFT-OUTER IN ORACLE

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 , TABLE_NAME2
WHERE TABLE_NAME1 . COLUMN_NAME = TABLE_NAME2 .
COLUMN_NAME [ + ] ;
```

EXAMPLE :

```
SELECT *
FROM EMP E , DEPT  D
WHERE E . DEPTNO = D . DEPTNO ( + ) ;
```

SYNTAX FOR RIGHT-OUTER JOIN IN ANSI

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 RIGHT OUTER JOIN TABLE_NAME2
ON < JOIN_CONDITION > ;
```

EXAMPLE :

```
SELECT *
FROM EMP E RIGHT OUTER JOIN DEPT D
ON E . DEPTNO = D . DEPTNO ;
```

SYNTAX FOR RIGHT-OUTER JOIN IN ORACLE

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 , TABLE_NAME2
WHERE TABLE_NAME1 . COLUMN_NAME [ + ]  = TABLE_NAME2
. COLUMN_NAME ;
```

EXAMPLE :

```
SELECT *
FROM EMP E , DEPT  D
WHERE E . DEPTNO ( + ) = D . DEPTNO ;
```

SYNTAX FOR FULL-OUTER JOIN IN ANSI

```
SELECT COLUMN_NAME
FROM TABLE_NAME1 FULL OUTER JOIN TABLE_NAME2
ON < JOIN_CONDITION > ;
```

EXAMPLE :

```
SELECT *
FROM EMP E FULL OUTER JOIN DEPT D
ON E . DEPTNO = D . DEPTNO ;
```

SYNTAX SELF-JOIN IN ANSI

```
SELECT COLUMN_NAME
FROM TABLE_NAME T1 JOIN TABLE_NAME T2
ON < JOIN_CONDITION > ;
```

EXAMPLE :

```
SELECT *
FROM EMP E1 JOIN EMP E2
ON E1 . MGR = E2 . EMPNO ;
```

NOTE :

```
EMP E1 = EMPS DETAILS
EMP E2 = MGR DETAILS
JOIN CONDITION = E1 . MGR = E2 . EMPNO ;
```

SYNTAX FOR SELF-JOIN IN ORACLE

```
SELECT COLUMN_NAME
FROM TABLE_NAME T1 , TABLE_NAME T2
WHERE  < JOIN_CONDITION > ;
```

EXAMPLE :

```
SELECT *
FROM EMP E1 , EMP E2
WHERE E1 . MGR = E2 . EMPNO ;
```

SYNTAX TO CREATE A TABLE

```
CREATE TABLE TABLE_NAME (
COL_NAME1 DATATYPE CONSTRAINTS ,
COL_NAME2 DATATYPE CONSTRAINTS ,
COL_NAME3 DATATYPE CONSTRAINTS ,
,
,
COL_NAME_N DATATYPE CONSTRAINTS ,);
```

SYNTAX TO CREATE A COPY OF A TABLE

```
CREATE TABLE TABLE_NAME
AS
SQL STATEMENT ;
```

SYNTAX FOR RENAME TABLE

```
RENAME CURRENT_TABLE_NAME TO NEW_TABLE_NAME ;
```

SYNTAX FOR ALTER

SYNTAX TO ADD A COLUMN

```
ALTER TABLE TABLE_NAME
ADD COLUMN_NAME DATATYPE CONSTRAINTS ;
```

SYNTAX TO DROP A COLUMN

```
ALTER TABLE TABLE_NAME
DROP COLUMN COLUMN_NAME ;
```

SYNTAX TO RENAME A COLUMN

```
ALTER TABLE TABLE_NAME
RENAME COLUMN COL_NAME TO NEW_COL_NAME ;
```

SYNTAX TO MODIFY DATATYPE OF A COLUMN

```
ALTER TABLE TABLE_NAME
MODIFY COLUMN_NAME NEW_DATATYPE ;
```

SYNTAX TO MODIFY CONSTRAINTS OF A COLUMN

```
ALTER TABLE TABLE_NAME
MODIFY COL_NAME DATATYPE NEW_CONSTRAINT
[NULL / NOT NULL] ;
```

SYNTAX FOR TRUNCATE A TABLE

```
TRUNCATE TABLE TABLE_NAME ;
```

SYNTAX FOR DROP A TABLE

```
DROP TABLE TABLE_NAME ;
```

SYNTAX FOR FLASHBACK A TABLE

```
FLASHBACK TABLE TABLE_NAME
TO BEFORE DROP ;
```

SYNTAX FOR PURGE A TABLE

```
PURGE TABLE TABLE_NAME ;
```

SYNTAX TO INSERT VALUE IN A TABLE

SYNTAX 1

```
INSERT INTO TABLE _NAME VALUES ( V1  , V2  , V3 , … , VN ) ;
```

SYNTAX 2

```
INSERT INTO TABLE_NAME VALUES ( &COL1 , &COL2 , …. ,
&COLN ) ;
```

SYNTAX TO UPDATE A TABLE

```
UPDATE TABLE_NAME
SET COL1 =V1
[WHERE < FILTER_CONDITION > ]
```

SYNTAX TO DELETE A TABLE

```
DELETE
FROM TABLE_NAME
[ WHERE < FILTER_CONDITION > ] ;
```

SYNTAX TO COMMIT IN TCL

```
COMMIT;
```

SYNTAX TO SAVEPOINT IN TCL

```
SAVEPOINT SAVEPOINT_NAME ;
```

SYNTAX TO ROLLBACK IN TCL

```
ROLLBACK;
```

SYNTAX TO GRANT IN DCL

```
GRANT SQL_STATEMENT
ON TABLE_NAME
TO USER_NAME ;
```

SYNTAX TO REVOKE IN DCL

```
REVOKE SQL_STATEMENT
ON TABLE_NAME
FROM USER_NAME;
```