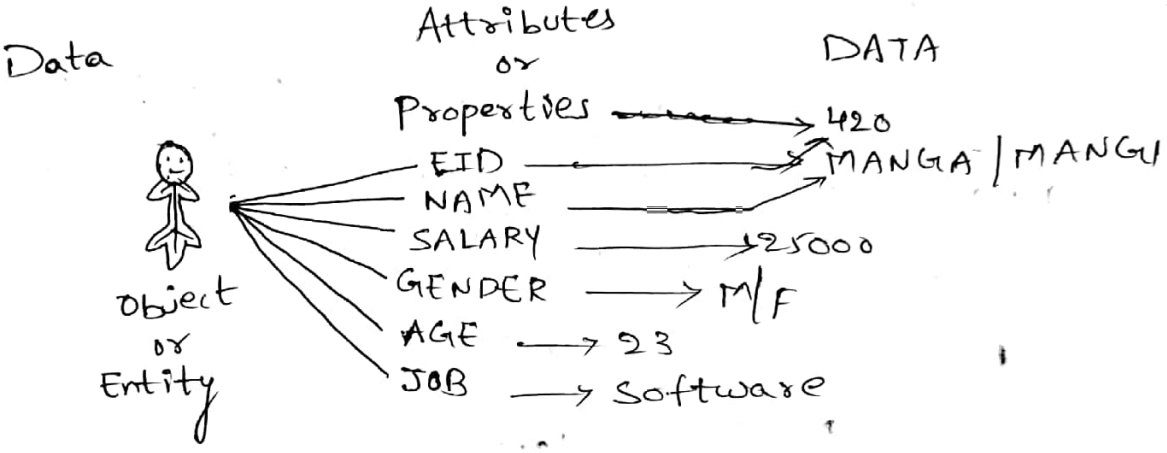
Data:

Data is a raw fact which describes the attributes of an entity.

Database:-

Database is a place or medium to store the data in a systematic and organised manner.

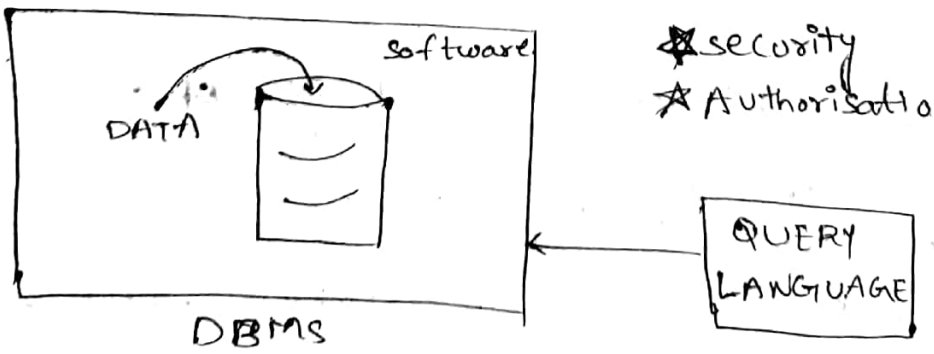
Example for data :-



Example for database:

DBMS (Database management system) :-

DBMS is a software which is used to maintain and manage the database.



In database we can perform some operations called "CRUD" operations.

C - CREATE / INSERT

R - READ / RETRIEVE

U - UPDATE / MODIFY

D - DELETE / DRop

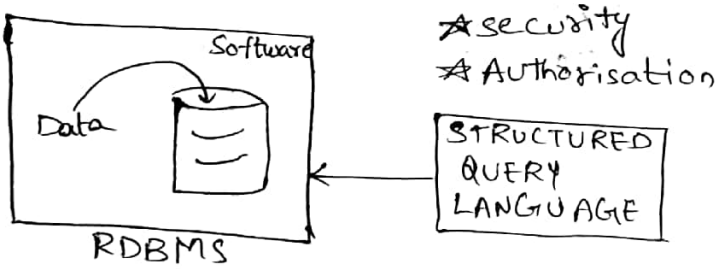
→ In DBMS, we use QUERY language to interact. or communicate with the database.

→ DBMS provides two features : security and authorization.

→ In DBMS we will store data in the form of files.

RDBMS (Relational database management system):-

-RDBMS is a type of DBMS software which is used to store the data in the form of tables.

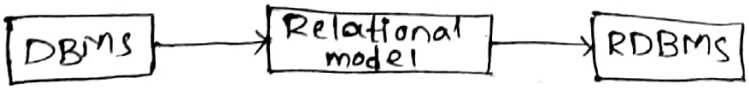


In RDBMS to interact with the database we use structured query language.

In RDBMS we will store everything in the form of tables.

Relational model:-

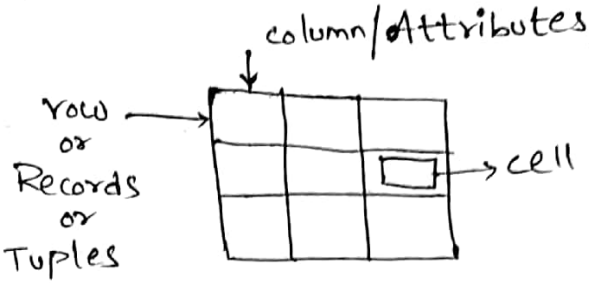
Relational model was designed by : EF CUDD.



Any DBMS which follows a relational model will become RDBMS.

Table :-

Table is a logical organization of data consists of rows and columns.



Ex:

Employee

EID	ENAME	SAL
1	ALIEN	2000
2	SMITHA	3000
3	JAMER	5000

Cell :-

Cell is a single unit present in the table.

Rules of E.F CODD :-

1. A data entered into a cell must be a single valued data.

✗ (invalid)

EID	ENAME	PH.NO	
1	A	123,456	✗
2	B	376,457	✗
3	C,D	477	

⟹

✓ (valid)

EID	ENAME	L.NAME	PH NO.1	PH NO.2
1	A	NULL	123	456
2	B	NULL	376	451
3	C	D	427	NULL

2. According to E.F CODD we will store the data in multiple tables. if needed we can establish a connection between multiple tables with the help of key attributes.

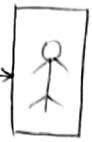
3. we will store everything in the form of tables including meta data.

Metadata:-

The details about the data is known as Metadata.

EMP

EID	ENAME	PIC
1	A	
2	B	
3	C	
4	D	



Properties → DATA

IMAGE_NAME → IMG_1

FORMAT → JPEG

SIZE → 150KB

Metadata is stored in metatable and it is auto created.

META table:-

IMAGE_NAME	FORMAT	SIZE
IMG_1	JPEG	150KB

4. To validate the data we have 2 steps:

(i) By assigning the data types.

(ii) By assigning constraints.

Data types are mandatory and constraints are optional.

Data types:-

Datatype is used to determine the type of data to be stored in the particular memory location.

1. CHAR
2. VARCHAR/VARCHAR2
3. DATE
4. NUMBER
5. LARGE OBJECTS
   - CHARACTER LARGE OBJECT
   - BINARY LARGE OBJECT

1. CHAR datatype:-

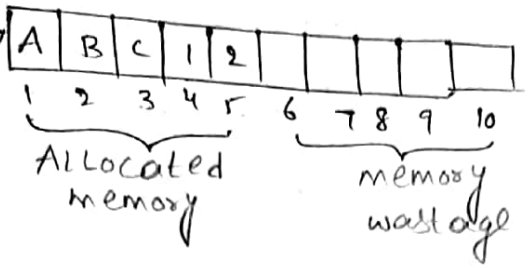
CHAR is a data type used to store characters consisting of alphabets ('A-z' and 'a-z') or numbers ('0-9') or special symbols ('#', '$', '_', ...)etc..

Syntax :-

CHAR (SIZE)

Eg:

CHAR(10)

'ABC12' → 

→ Whenever we are using the CHAR datatype we must mention the size.

→ By using CHAR data type we can store 2000 characters

→ CHAR data type follows an fixed length memory allocation.

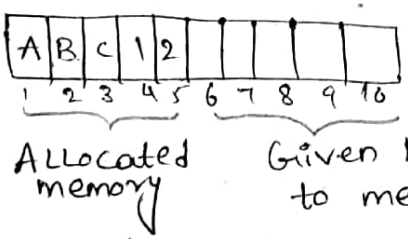
2. VARCHAR :-

VARCHAR data type is also used to store the characters consists of alphabets ('A-Z' and 'a-z') or numbers ('0-9') or special symbols ('#', '$', '_' ..)etc.,

Syntax :- VARCHAR (SIZE)

Eg:

VARCHAR (10)

'ABC12' 

→ we must mention the size.

→ The values should be enclosed with a pair of single coders

→ In VARCHAR, also we can store 2000 characters.

→ VARCHAR data type follows an variable length memory allocation.

VARCHAR2 :-

VARCHAR2 is similar to VARCHAR. here, we can store upto 4000 characters.

3. DATE :-

DATE is a data type used to store date.

Syntax: DATE

In ORACLE SQL, we have two formats.

'DD-MON-YY' (OR) 'DD-MON-YYYY'

Ex: '11-MAR-21' Ex: '11-MAR-2021'

4. NUMBER :-

NUMBER is a data type used to store numeric values.

Syntax: NUMBER (PRECISION, [SCALE])

Precision :-

Precision determines the number of integer values that we want to store.

Scale :-

scale determines the number of decimal points within the given precision.

Eg: i) NUMBER (2) (ii) NUMBER (4,2) (iii) NUMBER (2,4)

±99 ±99.99 ±0.0099

* The default value for scale is '0'.
* The range for precision is '1-38'.

5. LARGE OBJECTS :-

i) CHARACTER LARGE OBJECT (CLOB) :-

CHARACTER LARGE OBJECT is used to store the characters, files, documents upto 4 GB.

ii) Binary Large object (BLOB) :-

BLOB is used to store the binary values of images, MP3, MP4, videos upto 4 GB.

Constraints:-

Constraints is a rule given to a column.

1. UNIQUE
2. NOT NULL
3. CHECK
4. PRIMARY KEY
5. FORIEGN KEY

1. UNIQUE CONSTRAINT:-

UNIQUE Constraint is assigned to avoid the duplicate values into the column or table.

2. NOT NULL Constraint:-

NOTNULL is used to avoid NULL ~~Values~~ in the table

3. CHECK Constraint:-

CHECK Constraint is a extra validation given to a column with Condition, if the condition is satisfied, value will be accepted else ~~Value~~ is rejected.

4. PRIMARY KEY:-

PRIMARY KEY is a Constraint used to identify a record uniquely from the table.

Characteristics of primary key:-

1) Primary key is a combination of unique and not null Constraint.
2) There must be a only one primary key in the table.
3) Primary key cannot accept duplicates or null. in the table. or into the column.

5. FOREIGN KEY:-

Foreign key is a Constraint used to establish a Connection between the tables or multiple tables.

Characteristics of foreign keys:-

1) To refer it as a foreign key first it should be a Primary key.
2) Foreign key can accept null and duplicates.
3) We can have more than one foreign key in the table.

Primary & foreign key relationship:-

PK-Primary key
FK- Foreign key

EMPLOYEE

PK EID	ENAME	SAL	DNO FK
1	PENGA	10000	10
2	DINGA	25000	10
3	MANGA	30000	NULL
4	KONGA	35000	20

CHILD TABLE

DEPT

PK DNO	DNAME	LOC
10	D1	L1
20	D2	L2
30	D3	L3

PARENT TABLE

The primary key is converted as a foreign key it will follows foreign key rules.

Statements:-

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Transaction Control Language (TCL)
4. Data Control Language (DCL)
5. Data Query Language (DQL)

DATA QUERY LANGUAGE (DQL):-

DQL is used to retrieve the data from the table.

DQL has four statements they are: 1) SELECT
2) PROJECTION
3) SELECTION
4) JOINS

SELECT:-

SELECT CLAUSE is used to select the columns and display it.

PROJECTION:-

It is a process of retrieving the data by selecting only columns, by default the values present in the column will be displayed.

Syntax:-

```
SELECT */[DISTINCT]Column_Name/Expression [ALIAS]
FROM Table_Name;
```

Order of execution:-

1. FROM
2. SELECT

1) Write a Query to display employee name and salary.
2) write a Query to display salary and commision of employes.
3) write a Query to display department names present in department table.
4) Write a Query to display department name, Location and department number.
5) Write a query to display employee name and ~~higher~~ hire date.

1)
```
SELECT ENAME, SAL
FROM EMP;
```

2)
```
SELECT SAL, COMM
FROM EMP;
```

3)
```
SELECT DNAME
FROM DEPT;
```

4)
```
SELECT DNAME, LOC, DEPT NO
FROM DEPT;
```
(or)
```
SELECT *
FROM DEPT;
```

5)
```
SELECT ENAME, HIREDATE
FROM EMP;
```

Employee table :-

ROWID	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	1980-12-17	800.00	null	20
2	7369	SMITH	CLERK	7902	1980-12-17	800.00	null	20
3	7499	ALLEN	SALESMAN	7698	1981-02-20	1600.00	300.00	30
4	7521	WARD	SALESMAN	7698	1981-02-22	1250.00	500.00	30
5	7566	JONES	MANAGER	7839	1981-04-02	2975.00	null	20
6	7654	MARTIN	SALESMAN	7698	1981-09-28	1250.00	1400.00	30
7	7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	null	30
8	7782	CLARK	MANAGER	7839	1981-06-09	2450.00 3000.00	null	10
9	7788	SCOTT	ANALYST	7566	1982-12-09	3000.00 ~~5000.00~~	null	20
10	7839	KING	PRESIDENT	null	1981-~~11-17~~	17 5000.00 ~~1500.00~~	null	10
11	7839	TURNER	SALESMAN	7698	1981-09-08	1500.00 1100.00	0.00	30
12	7876	ADAMS	CLERK	7788	1983-01-12	1100.00 950.00	null	20
13	7900	JAMES	CLERK	7698	1981-12-03	950.00 3000.00	null	30
14	7902	FORD	ANALYST	7566	1981-12-03	3000.00	null	20
15	7934	MILLER	CLERK	7782	1982-01-23	1300.00	null	10

Department table :-

ROWID	DEPTNO	DNAME	LOC
1	10	ACCONTING	NEW YORK
2	20	REASEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON

1) Write a Query to display employee name and salary.

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

ROWID	ENAME	SAL
1	SMITH	800.00
2	SMITH	800.00
3	ALLEN	1600.00
4	WARD	1250.00
5	JONES	2975.00
6	MARTIN	1250.00
7	BLAKE	2850.00
8	CLARK	2450.00
9	SCOTT	3000.00
10	KING	5000.00
11	TURNER	1500.00
12	ADAMS	1100.00
13	JAMES	950.00
14	FORD	3000.00
15	MILLER	1300.00

2) Write a Query to display salary and commission of employee.

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

ROWID	SAL	COMM
1	800.00	null
2	800.00	null
3	1600.00	300.00
4	1250.00	500.00
5	2975.00	null
6	1250.00	1400.00
7	2850.00	null
8	2450.00	null
9	3000.00	null
10	5000.00	null
11	1500.00	0.00
12	1100.00	null
13	950.00	null
14	3000.00	null
15	1300.00	null

3) Write a Query to display department name present in department table.

```
SELECT DNAME
FROM DEPT;
```

ROWID	DNAME
1	ACCOUNTING
2	RESEARCH
3	SALES
4	OPERATIONS

4) Write a Query to display department name, location and department number.

```
SELECT DNAME, LOC, DEPT NO
FROM DEPT;
```

(or)

```
SELECT *
FROM DEPT;
```

ROWID	DEPTNO	DNAME	LOC
1	10	ACCONTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATONS	BOSTON

5) write a query to display employee name and hire date.

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

ROWID	ENAME	HIREDATE	ROWID	ENAME	HIREDATE
1	SMITH	1980-12-17	11	TURNER	1981-09-08
2	SMITH	1980-12-17	12	ADAMS	1983-01-12
3	ALLEN	1981-02-20	13	JAMES	1981-12-03
4	WARD	1981-02-22	14	FORD	1981-12-03
5	JONES	1981-04-02	15	MILLER	1982-01-23
6	MARTIN	1981-09-28			
7	BLAKE	1981-05-01			
8	CLARK	1981-06-09			
9	SCOTT	1982-12-09			
10	KING	1981-11-17			

SELECTION :-

SELECTION is a process of retrieving the data by selecting columns as well as rows.

Syntax :-

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

Order of execution :-

1. FROM
2. WHERE
3. SELECT

Query using - SELECTION :-

```
SELECT ENAME
FROM EMP
WHERE EID = 2;
```

EMP

EID	ENAME	SAL
1	DINGA	2000
2	MANGA	3000
3	MANGI	3500
4	DINGI	2000

Result of FROM CLAUSE

1 DINGA 2000	1 = 2 F
2 MANGA 3000	2 = 2 T
3 MANGI 3500	3 = 2 F
4 DINGI 2000	4 = 2 F

Result of WHERE CLAUSE

2	MANGA	3000

Result of SELECTION CLAUSE

ENAME
MANGA

1) FROM CLAUSE starts execution.
2) The job of FROM CLAUSE is go to the database and search for the table and put that table under execution.
3) After the execution of FROM CLAUSE, WHERE CLAUSE will be executed.
4) WHERE CLAUSE executes row by row.

5) WHERE CLAUSE returns boolean output i.e, true or false.

6) WHERE CLAUSE is used to filter the records.

7) After the execution of WHERE CLAUSE, SELECT CLAUSE will be executed.

8) SELECT CLAUSE is responsible to prepare or generate the result table.

1) write a Query to display the employee details where EMPNO is 7839.

```
SELECT *
FROM EMP
WHERE EMPNO = 7839
```

ROWID	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7839	KING	PRESIDENT	null	1981-11-17	500000	null	10

2) write a Query to display ENAME who is working in DEPT NO. 10

```
SELECT ENAME
FROM EMP
WHERE DEPTNO=10;
```

ROWID	ENAME
1	CLARK
2	KING
3	MILLER

PROJECTION:-

WAQTD names of all the employees.

```
SELECT ENAME
FROM EMP;
```

WAQTD name and Commision given to all the employees.

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

WAQTD EID, Dept NO of all the employees in EMP table.

```
SELECT EMPNO, DEPT NO
FROM EMP;
```

WAQTD NAME and JOB of all the employees.

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

WAQTD NAME, JOB. and SALARY given all the employees.

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

SELECTION :-

1) write a query to display ANNUAL SALARY of the employee who's name is SMITH.

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

2) write a query to display name of the employees who are working CLERK.

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

3) write a query to display salary of the employees who are working as SALESMAN.

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

4) write a query to display details of the EMP who earns more than 2000.

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

5) write a query to display details of the EMP whose name is JONES.

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

6) write a query to display details of the EMP who was hired after 01-JAN-81.

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

7) write a query to display name and salary along with his ANNUAL SALARY. if the ANNUAL SALARY is more than 12000.

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

8) write a query to display EMPNO of the employees who are working in DEPT 30.

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

9) write a query to display ENAME and HIREDATE if they are HIRED before 1981.

```
SELECT ENAME, HIREDATE
FROM EMP
WHERE HIREDATE < '31-DEC-81';
```

10) WAQTD details of the EMP working as MANAGER.

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

11) WAQTD NAME & SAL given to an employee if employee earns a commission of rupees 1400.

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

12) WAQTD details of EMP 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 < '31-DEC-87';
```

14) WAQTD details of EMP working as an ANALYST.

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

15) WAQTD details of EMP earning more than 2000 rupees per month.

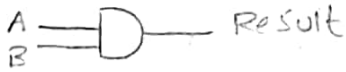
```
SELECT *
FROM EMP
WHERE SAL > 2000;
```

Logical operators:-

AND operator:-

AND is a binary operator which will take two inputs and gives a boolean output.

Symbol:



Truth table:

A	B	Result
T	T	T
T	F	F
F	T	F
F	F	F

OR operator:-

OR is a binary operator which will take two inputs and gives a boolean output.

Symbol :

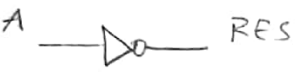
Truth table :

A	B	RES
T	T	T
T	F	T
F	T	T
F	F	F

NOT operator :-

NOT is a uniary operator which will take one input and gives a one output.

Symbol :



Truth table :

A	RES
T	F
F	T

DISTINCT CLAUSE :-

DISTINCT CLAUSE is used to remove the duplicates from the column.

To the SELECT CLAUSE, DISTINCT CLAUSE is the first argument.

we can pass any number of columns after the DISTINCT CLAUSE.

Eg: SELECT DISTINCT ENAME, SAL
FROM EMP ;

EID	ENAME	SAL
1	A	200
2	A	300
3	B	400
4	C	400
5	B	400

⇒ ① Removed

Result

ENAME	SAL
A	200
A	300
B	400
C	400

WAQTD the different salaries of employees.

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

Eg: 2 :

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

JOB	DEPTNO
MANAGER	20
PRESIDENT	10
CLERK	10
SALESMAN	30
ANALYST	20
MANAGER	30
MANAGER	10
CLERK	30
CLERK	20

EXPRESSION :-

Any statement generates a result ~~table~~ we will Call it as EXPRESSION.

Ex: SAL * 12 → operator

OPERANDS

↳ COLUMN_NAME

↳ LITERALS

LITERALS :-

LITERALS are the values passed by the user.

we have 3 types of LITERALS :

1) CHARACTER LITERAL } Case sensitive
2) DATE LITERAL
3) NUMBER LITERAL

ALIAS :-

ALIAS is a alternative name given to a column or expression.

→ we will use AS keyword to write the ALIAS name but it is not mandatory.

→ we can write the ALIAS name with the help of underscore ('_') or double codes (" ").

Ex:

```
SELECT SAL*12 (AS) ANNUAL_SAL OR "ANNUAL SAL"
FROM EMP;
```

(AS → NOT mandatory)

write a query to display the annual salaries of the employees with ALIAS.

```
SELECT SAL * 12 AS ANNUAL-SAL
FROM EMP;
```

WAQTD all the employee details along with the annual Salary.

(table name → EMP, access operation → *)

```
SELECT EMP.* , SAL*12 "ANNUAL SAL"
FROM EMP;
```

Operators in SQL :-

1. ARITHEMATIC OPERATORS :- (+, -, *, /)
2. CONCATINATION OPERATOR :- (||)
3. COMPARISON OPERATORS :- (=, != or <>)
4. RELATIONAL OPERATOR :- (>, <, >=, <=)
5. SPECIAL OPERATOR :- (1. IN
   2. NOT IN
   3. BETWEEN
   4. NOT BETWEEN
   5. IS
   6. IS NOT
   7. LIKE
   8. NOT LIKE)

6. SUBQUERY OPERATORS :- (1. ALL
2. ANY
3. EXISTS
4. NOT EXISTS)

CONCATINATION OPERATOR :-

CONCATINATION OPERATOR is used to join the strings.

Eg:

```
SELECT 'HI' || ENAME
FROM EMP;
```

SPECIAL OPERATORS:-

IN OPERATOR:-

IN is a multi valued operator which will take one value in L.H.S and multiple values in R.H.S.

Syntax :

```
column_name/expression IN (V1, V2, .... Vn);
```

Ex:

```
SELECT *
FROM EMP
WHERE DEPTNO IN(10, 20, 30);
```

DEPTNO
10
20
10
30

1st Iteration

10 IN (10,20,30)

10=10 T
10=20 F
10=30 F
} T

2nd Iteration

20 IN (10,20,30)

20=10 F
20=20 T
20=30 F
} T

3rd Iteration

10 IN (10,20,30)

10=10 T
10=20 F
10=30 F
} T

4th Iteration

30 IN (10,20,30)

30=10 F
30=20 F
30=30 T
} T

WAQTD the employee details who are working in DEPTNO 10, 20, 30, 40.

```
SELECT *
FROM EMP
WHERE DEPTNO IN (10, 20, 30, 40);
```

WAQTD the employee name and SAL who are working as MANAGER or SALESMAN in DEPTNO 10 OR 30.

```
SELECT ENAME, SAL
FROM EMP
WHERE JOB IN('MANAGER', 'SALESMAN') AND DEPTNO IN(10,30);
```

WAQTD employee name and Hiredate earning SAL more than 3000 and working as ANALYST OR CLERK.

```
SELECT ENAME, HIREDATE
FROM EMP
WHERE JOB IN('ANALYST', 'CLERK') AND SAL>3000;
```

NOT IN operator:-

NOT IN is similar to IN operator instead of selecting the records it will reject the records.

Syntax: column_name/expression NOT IN (V1, V2, ... Vn);

WAQTD the employee details except the employees working in dept no 10 or 20.

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

WAQTD the employee name, job excluding MANAGER.

```
SELECT ENAME, JOB
FROM EMP
WHERE JOB NOT IN ('MANAGER');
```

WAQTD HIREDATE and SAL with ANNUAL SAL of employees excluding dept no 30.

```
SELECT HIREDATE, SAL, SAL*12 AS ANNUAL_SAL
FROM EMP
WHERE DEPTNO NOT IN (30);
```

BETWEEN operator:-

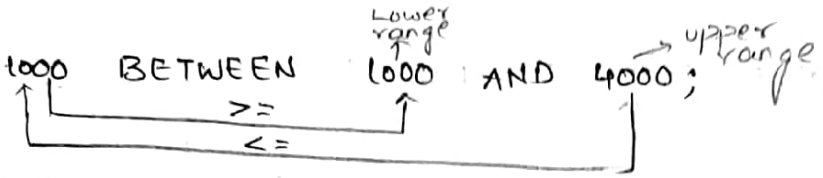
BETWEEN operator is used whenever we know the range (lower range and upper range)

Syntax: Column_name/expression BETWEEN lower_range AND higher_range;

* BETWEEN operator is not mandatory
* while using BETWEEN operator we should not to interchange the range.

Ex:
```
SELECT *
FROM EMP
WHERE SAL BETWEEN 1000 AND 4000;
```

SAL
1000
1500
2000



WAQTD the employee name and sal ranges BETWEEN 4000- to 5000.

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL BETWEEN 4000 AND 5000;
```

WAQTD all the details of employees hired during the year 81.

```
SELECT *
FROM EMP
WHERE HIREDATE BETWEEN '01-JAN-81' AND '31-DEC-81';
```

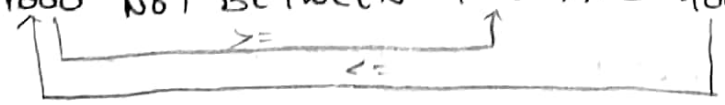
NOT BETWEEN operator:-

NOT BETWEEN is similar to BETWEEN operator instead of selecting the records it will reject the records.

Syntax: Column_name/expression NOT BETWEEN lower_range AND higher_range;

Ex:
```
SELECT *
FROM EMP
WHERE SAL NOT BETWEEN 1000 AND 4000;
```

SAL
1000
1500
2000
5000



WAQTD the employee details excluding the salary ranges between 1000 and 3000.

```
SELECT *
FROM EMP
WHERE SAL NOT BETWEEN 1000 AND 3000;
```

18/03/21

5. IS operator:-

IS is the operator which is used to compare with the null.

Syntax: Column_name/expression IS NULL;

Ex:

1) WAQTD the employee details who are not getting salary.

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

EMP

ENAME	SAL
DINGA	2000
MANGA	5000
KONGA	NULL
KONGI	4000

Comparing SAL with NULL

2000 IS NULL F
5000 IS NULL F
NULL IS NULL T
4000 IS NULL F

Result

EName	SAL
KONGA	

6. IS NOT operator:-

IS NOT is similar to the IS operator instead of selecting the records it will reject the records.

Syntax: column_name/expression IS NOT NULL;

Ex: WAQTD the employee details who are earning commission.

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

7. LIKE operator:-

LIKE operator is used to perform pattern matching.

Syntax: column_name/expression LIKE 'Pattern_to_match';

To acheive pattern matching we have two special characters:- '%', '_'.

'%' : Percentail(%) character will take any number of characters it might be anything.

'_' : Underscore(_) will take exactly one character it might be any character.

Eg: SELECT *
FROM EMP
WHERE ENAME LIKE '%s%' ;

For '%':
'%s%' ; → (names start with 's')
'%s%' ; → (names having 's')
'%s' ; → (names end with 's')
'%ss%' ; → (names have consecutive 's') (eg: success)
'%s%s%' ; → (names have two nonconsecutive 's'. (eg: salesman)

For '_':
'_s%' ; → (name having exact 2nd character is 's')
'_____' ; (name having only 5 characters)
'%s_' ; → (name having 2nd letter from last is 's' Eg: INCREASE

8) NOT LIKE:-
NOT LIKE is similar to LIKE operator instead of selecting the records it will reject the records.

Syntax: column_name/expression NOT LIKE 'Pattern_to_match';

Eg:
```
SELECT *
FROM EMP
WHERE ENAME NOT LIKE 'S%';
```

Questions on operators:-

1) List all the employees whose commission is NULL.
```
SELECT *
FROM EMP
WHERE COMM IS NULL;
```

2) List all the employees who don't have a REPORTING MANAGER. (MGR).
```
SELECT *
FROM EMP
WHERE MGR IS NULL;
```

3) List all the SALESMAN in DEPT 30.

```
SELECT *
FROM EMP
WHERE JOB = 'SALESMAN' AND DEPTNO IN (30);
```

(or JOB IN ('SALESMAN'))

4) List all the SALESMAN IN DEPTNO 30 AND HAVING salary greater than 1500.

```
SELECT *
FROM EMP
WHERE JOB = 'SALESMAN' AND DEPTNO IN (30) AND SAL > 1500;
```

(or JOB IN ('SALESMAN'))

5) List all the employees whose Name starts with 'S' or 'A';

```
SELECT *
FROM EMP
WHERE ENAME LIKE 'S%' OR ENAME LIKE 'A%';
```

6) List all the employees except those who are working in Dept 10 & 20.

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

7) List the employees whose name does not start with 's'.

```
SELECT *
FROM EMP
WHERE ENAME NOT LIKE 'S%';
```

8) List all the employees who are having REPORTING MANAGERS IN DEPT 10.

```
SELECT *
FROM EMP
WHERE MGR IS NOT NULL AND DEPTNO IN (10);
```

9) List all the employees whose commission is NULL and working as CLERK.

```
SELECT *
FROM EMP
WHERE JOB = 'CLERK' AND COMM IS NULL;
```

10) List all the employees who don't have a REPORTING MANAGER IN DEPTNO 10 OR 30.

```
SELECT *
FROM EMP
WHERE MGR IS NULL AND DEPTNO IN (10,30);
```

Assignment:

11) List all the salesman in dept 30 with salary more than 2450.

```
SELECT *
FROM EMP
WHERE JOB IN ('SALESMAN') AND DEPTNO IN (30) AND SAL>2450;
```

12) List all the analyst in deptno 20 and having salary greater than 2500.

```
SELECT *
FROM EMP
WHERE JOB IN ('ANALYST') AND DEPTNO IN (20) AND SAL>2500;
```

13) List all the employees whose name start with 'M' or 'J'?

```
SELECT *
FROM EMP
WHERE ENAME LIKE 'M%' OR ENAME LIKE 'J%';
```

14) List all the employees with ANNUAL Salary except those who are working in DEPT 30.

```
SELECT EMP.*, SAL*12 AS ANNUAL_SAL
FROM EMP
WHERE DEPTNO NOT IN (30);
```

15) List the employees whose name does not end with 'ES' OR 'R'.

```
SELECT *
FROM EMP
WHERE ENAME NOT LIKE '%ES' AND ENAME NOT LIKE '%R';
```

16) List all the employees who are having REPORTING MANAGER IN DEPT 10 along with 10% HIKE in Salary.

```
SELECT EMP.*, SAL+SAL*0.1 AS H_SAL
FROM EMP
WHERE MGR IS NOT NULL AND DEPTNO IN (10);
```

17) Display all the employee who are 'SALESMAN's Having 'E' as the last but one character in ENAME but salary having exactly 4 character.

```
SELECT *
FROM EMP
WHERE JOB IN ('SALESMAN') AND ENAME LIKE '%E' AND SAL LIKE '____';
```

18) Display all the employee who are joined after year 81.

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

19) Display all the employee who are joined in FEB.

```
SELECT *
FROM EMP
WHERE HIREDATE LIKE '%FEB%';
```

~~WHERE HIREDATE BETWEEN '01-JAN-81' AND '31-DEC-81';~~

(or) `HIREDATE LIKE '__FEB___';`

20) List the employees who are not working as MANAGER and CLERKS in DEPT 10 and 20 with a salary in the range of 1000 to 3000.

```
SELECT *
FROM EMP
WHERE JOB NOT IN ('MANAGER' OR 'CLERK') AND DEPTNO IN(10,20)
      AND SAL BETWEEN 1000 AND 3000;
```

21) List the employees whose salary NOT IN the range of 1000 to 2000 AND working in DEPT 10, 20 OR 30 except all SALESMAN.

```
SELECT *
FROM EMP
WHERE SALARY NOT BETWEEN 1000 AND 2000 AND DEPTNO IN (10,20,30)
      AND JOB NOT IN 'SALESMAN';
```

22) List the department names which are having letter 'o' in their locations as well as their department names.

```
SELECT *DNAME
FROM DEPT
WHERE LOC LIKE '%O%' AND DNAME LIKE '%O%';
```

23) Display all the employees whose job has string 'MAN' in IT.

```
SELECT *
FROM EMP
WHERE JOB LIKE '%MAN%';
```

24) List the employees who are hired after 82 and before 87.

```
SELECT *
FROM EMP
WHERE HIREDATE BETWEEN '01-JAN-83' AND '31-DEC-86';
```

(or) `HIREDATE > '31-DEC-1982' AND HIREDATE < '01-JAN-1987';`

WAQTD all the details of employees hired in November and december.

```
SELECT *
FROM EMP
WHERE HIREDATE LIKE '%NOV%' OR HIREDATE LIKE '%DEC%';
```

26) List all the employee names and commision for those employees who earn commission more than their salary.

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

27) WAQTD name and designation for all the employees having reporting manager and also their names starting with 's'.

```
SELECT ENAME, JOB
FROM EMP
WHERE MGR IS NOT NULL AND ENAME LIKE 'S%';
```

28) WAQTD name and salary of all the employees if their ANNUAL salary ends with '0'.

```
SELECT ENAME, SAL, SAL*12 As Annual_SAL
FROM EMP
WHERE SAL*12 LIKE ('%0');
```

29) WAQTD name of the employee having atleast 2L's in his name.

```
SELECT *
FROM EMP
WHERE ENAME LIKE ('%L%L%');
```

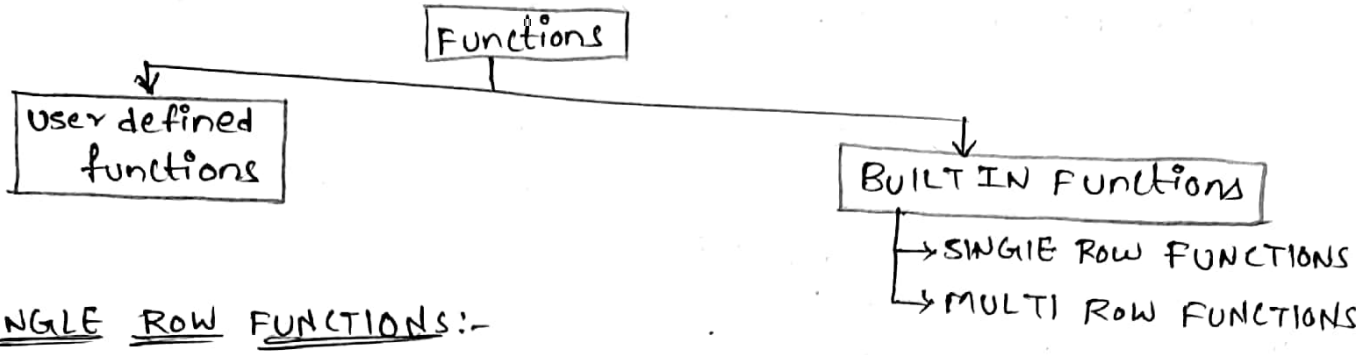
30) WAQTD name of the employees whos name start with a 'VOWEL'.

```
SELECT *
FROM EMP
WHERE ENAME LIKE 'A%' OR ENAME LIKE 'E%' OR
ENAME LIKE 'I%' OR ENAME LIKE 'O%' OR ENAME LIKE 'U%';
```

19/03/21
Friday

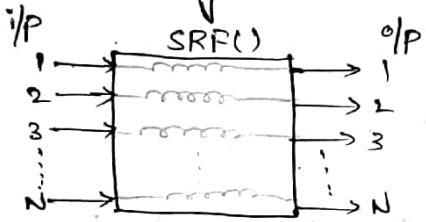
Functions:-

Function is a block of code or set of programs to be Processed and executed.



SINGLE ROW FUNCTIONS:-

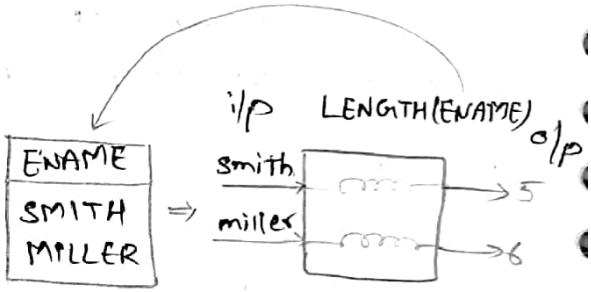
SINGLE ROW FUNCTION will take 'N' number of Inputs and processed it and gives 'N' number of outputs.



Ex:

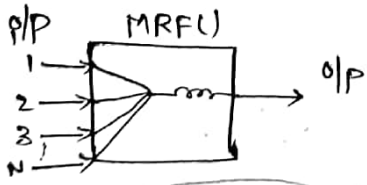
LENGTH()

```
SELECT ENAME, LENGTH(ENAME)
 FROM EMP;
```



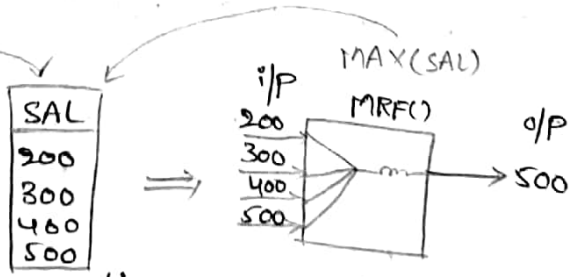
MULTI ROW FUNCTION:-

MULTI ROW FUNCTION will take 'N' number of Inputs Combined it in one shot and processed it and gives a single output.



Ex:

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



There are 5 MULTIROW Functions: 1. MAX()
2. MIN()
3. SUM()
4. AVG()
5. COUNT()

MAX() MULTI ROW FUNCTION is used to find the maximum value present in the column.

MIN() :-

MIN() MULTI ROW FUNCTION is used to find the minimum value present in the column.

SUM() :-

SUM() MULTI ROW FUNCTION is used to find the total sum of values present in the column.

AVG() :-

AVG() MULTI ROW FUNCTION is used to find the average of all the values present in the column.

COUNT() :-

COUNT() MULTI ROW FUNCTION is used to find the number of values present in the column.

Rules to use MULTI ROW FUNCTIONS :-

1) whenever we are using MRF's in SELECT CLAUSE we should not use column name along with the MRF.
2) we must and should pass an arguement to the MRF i.e., only one argument.
3) we cannot use MRF in WHERE CLAUSE.
4) MRF ignores NULL.
5) COUNT() is the only MRF which will take Aestric (*) as an argument.

20/03/24

WAQTD the avg salary of employees working as 'SALESMAN'.

```
SELECT AVG(SAL)
  FROM EMP
 WHERE JOB='SALESMAN'
```

WAQTD the MAX and MIN salary of employees who are working as CLERK in DEPT20.

```
SELECT MAX(SAL), MIN(SAL)
FROM EMP
WHERE JOB='CLERK' AND DEPTNO IN(20);
```

WAQTD the SUM of salary of employees who hired during the year 2021.

```
SELECT SUM(SAL)
FROM EMP
WHERE HIREDATE BETWEEN '01-JAN-2021' AND '31-DEC-2021';
```

WAQTD the no. of employees working as ANALYST in deptno. 20 OR 30.

```
SELECT COUNT(*)
FROM EMP
WHERE JOB='ANALYST' AND DEPTNO IN(20,30);
```

WAQTD the no. of employees and max salary of employees whose salary ranges from 4000 to 5000.

```
SELECT COUNT(*), MAX(SAL)
FROM EMP
WHERE SAL BETWEEN 4000 AND 5000;
```

WAQTD the no. of employees in each department.

```
SELECT COUNT(*), deptno
FROM EMP
GROUP BY DEPTNO;
```

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;
```

ORder of execution:-

1- FROM

2- WHERE (if used) [ROW-BY-ROW]

3- GROUP BY [ROW-BY-ROW]

4- SELECT [GROUP-BY-GROUP].

→ GROUP BY CLAUSE executes ROW by ROW. execution of

→ Any CLAUSE which executes after the group by clause will execute the group by group.

→ After the ~~WHERE~~ execution of WHERE CLAUSE, GROUP BY CLAUSE will be executed.

→ In GROUP BY CLAUSE, we will pass columns as an arguement.

→ The columns which we are passing in GROUP BY CLAUSE we can write that columns in SELECT CLAUSE.

Ex:
Write a query to display no: of employees working in each department.

```
SELECT COUNT(*), DEPTNO
FROM EMP
GROUP BY DEPTNO;
```

Result of FROM CLAUSE

EMP

EMPNO	ENAME	SAL	DEPTNO
1	ALLEN	2000	10
2	SMITH	2500	20
3	MILLER	3000	30
4	JONES	3500	10
5	KING	5000	20
6	ADAM	4500	10

Result of GROUP BY CLAUSE

10

1	ALLEN	2000	10
4	JONES	3500	10
6	ADAM	4500	10

20

2	SMITH	2500	20
5	KING	5000	20

30

3	MILLER	3000	30

Result of SELECT CLAUSE

COUNT(*)	DEPTNO
3	10
2	20
1	30

Queries on GROUP BY CLAUSEE:-

1) WAQTD number of employees working in each department except PRESIDENT.

```
SELECT COUNT(*)
FROM EMP
WHERE JOB NOT IN ('PRESIDENT');
```

2) WAQTD total salary needed to pay all the employees in each job.

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

3) WAQTD number of employees working as manager in each department.

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

4) WAQTD AVG salary needed to pay all the employees in each department excluding the employees of Deptno 20.

```
SELECT AVG(SAL), DEPTNO
FROM EMP
WHERE DEPTNO NOT IN (20)
GROUP BY DEPTNO;
```

5) WAQTD Number of employees having character 'A' in their names in each job.

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

6) WAQTD number of employees and avg salary needed to pay the employees who salary in greater than 2000 in each dept.

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

7) WAQTD total salary needed to pay and number of salesmans in each dept.

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

9) WAQTD MAX salaries given to an employee working in each dept.

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

10) WAQTD NO. of times the salaries have been repeated in employee table.

```
SELECT COUNT(*)-1, SAL   (or) SELECT COUNT(*), SAL
FROM EMP
GROUP BY SAL;
```

11) WAQTD number of employees HIRED on the same day into the same department.

```
SELECT COUNT(*), HIREDATE, DEPTNO
FROM EMP
GROUP BY HIREDATE, DEPTNO;
```

12) WAQTD number of employees getting the same salary in the same department.

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

13) WAQTD number of employees having the same name in the same department.

```
SELECT COUNT(*), ENAME, DEPTNO
FROM EMP
GROUP BY ENAME, DEPTNO;
```

14) WAQTD number of employees getting the same salary in the same department and having the same name.

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

15) WAQTD number of employees getting the same salary in the same department and hired on the same day.

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

22/03/21

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 BY col_name/exp [ASC]/DESC;
```

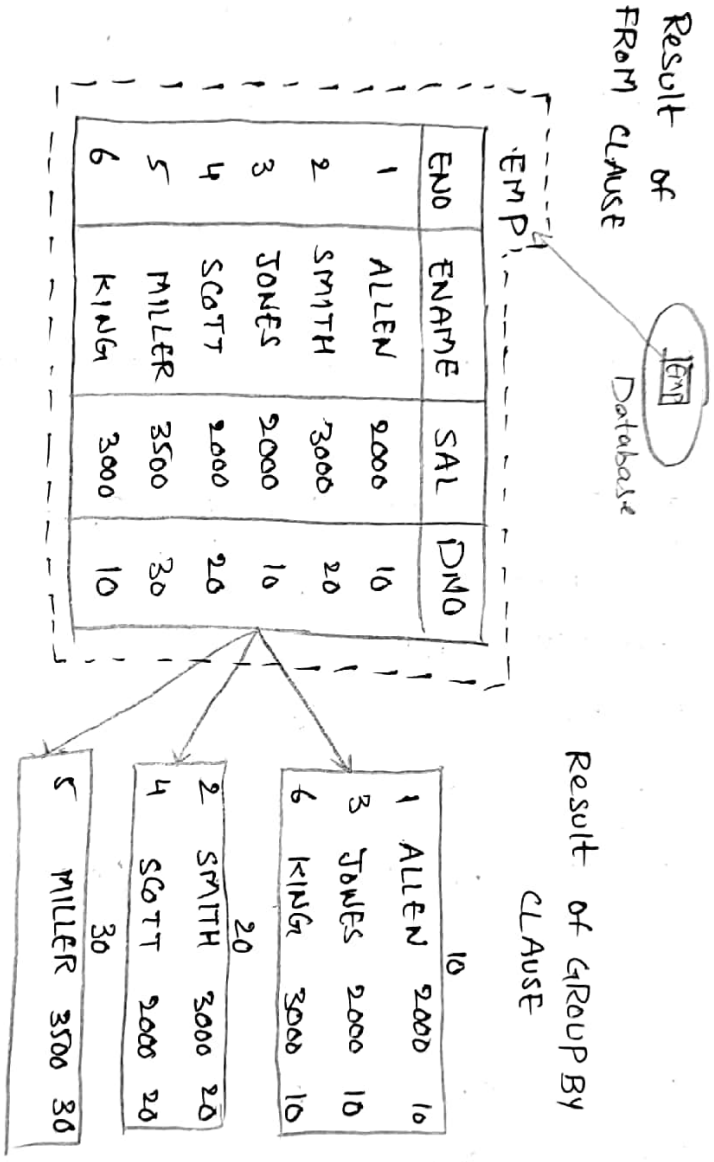
ORder of execution :-

1 - FROM
2 - WHERE (if used) [ROW-BY-ROW]
3 - GROUP BY (if used) [ROW-BY-ROW]
4 - HAVING (if used) [GROUP-BY-GROUP]
5 - SELECT [GROUP-BY-GROUP]
6 - ORDER BY [ROW-BY-ROW]

→ Without GROUP BY CLAUSE we cannot use HAVING CLAUSE.
→ HAVING CLAUSE executes GROUP BY GROUP.
→ Any CLAUSE which executes after GROUP BY CLAUSE will execute GROUP BY GROUP.

Ex: WAQTD Number of employees in each department atleast there should be 2 employees in each department.

```
SELECT COUNT(*), DNO
FROM EMP
GROUP BY DNO
HAVING COUNT(*)>=2;
```



WAQTD the number of employees working in each department with maximum salary 6000.

```
SELECT COUNT(*), DEPTNO
FROM EMP
GROUP BY DEPTNO
HAVING MAX(SAL) = 6000;
```

WAQTD the number of employees and minimum salary getting in each job atleast there should be 2 SALESMAN.

```
SELECT COUNT(*), JOB , MIN(SAL)
FROM EMP
WHERE JOB = 'SALESMAN'
GROUP BY JOB
HAVING COUNT(*) >= 2;
```

WAQTD DNO and number of EMP working in each dept if there are atleast 2 CLERKS in each Dept.

```
SELECT COUNT(*), DNO
FROM EMP
WHERE JOB IN('CLERK')
GROUP BY DEPTNO
HAVING COUNT(*) >= 2;
```

WAQTD DNO and total salary needed to pay all EMP in each Dept if there are atleast 4 EMP in each Dept.

```
SELECT SUM(SAL), DEPTNO, COUNT(*)
FROM EMP
GROUP BY DEPTNO
HAVING COUNT(*) >= 4;
```

WAQTD number of EMP earning SAL more than 1200 in each job and the total sal needed to pay EMP of each job must exceeds 3800.

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

WAQTD DEPTNO and number of EMP working only if there are 2 EMP working in each dept as MANAGER.

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

WAQTD JOB and max SAL of EMP in each job if the max sal exceeds 2600.

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

WAQTD the salaries which are repeated in EMP table.

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

WAQTD the HIREDATE which are duplicated in EMP table.

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

WAQTD AVG salary of each dept if AVG SAL is less than 3000.

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

WAQTD DEPTNO if there are atleast 3 EMP in each dept who's name has CHAR 'A' OR 'S'.

```
SELECT COUNT(*), DEPTNO
 FROM EMP
WHERE ENAME LIKE '%A%' OR ENAME LIKE '%S%'
   GROUP BY DEPTNO
```

```
HAVING COUNT(*) >= 3;
```

WAQTD MIN AND MAX salaries of each job if min sal is more than 1000 and max sal is less than 5000.

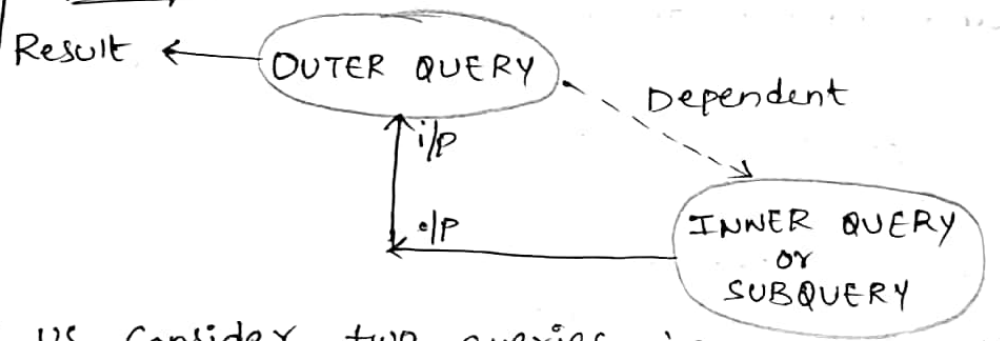
```
SELECT MIN(SAL), MAX(SAL), JOB
  FROM EMP
  GROUP BY JOB
  HAVING MIN(SAL) > 1000 AND MAX(SAL) < 5000;
```

23/03/21

SUBQUERY:-

A QUERY written inside another QUERY we will call it as SUBQUERY.

Working principle:-



→ Let us Consider two queries i.e., outer query and inner query.

→ Inner query or subquery starts execution first

→ Inner query or subquery generates a result or output that result will be fetched as a input to the outer query.

→ By taking the input from sub query outer query will execute Completely and gives a final result.

→ Hence, we can state that outer query is dependent on inner query or sub query.

Why, when:-

Case 1: Whenever we have unknowns or indirect data given in the Question at that time we will go for subqueries.

Ex: WAQTD ENAME earning SAL more than SMITH.

EID	ENAME	SAL	DNO
1	ALLEN	2000	10
2	SMITH	3500	20
3	MILLER	2000	30
4	JONES	3500	10
5	KING	5000	20

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

1) WAQTD Name of the employees earning more than ADAMS

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

2) 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');
```

3) WAQTD name and deptno of the employees if they are working in the same dept as JONES.

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

4) 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');
```

5) WAQTD EMPNO and ENAME along with ANNUAL salary of all the employees if their ANNUAL salary is greater than WARDS

ANNUAL salary.

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

6) WAQTD name and HIREDATE of the employees if they are hired before scott.

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

7) WAQTD name and hiredate of the employees if they are hired after the president.

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

8) WAQTD name and sal of the employee if they are earning SAL less than the employee who's EMPNO is 7839.

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

9) WAQTD all the details of the employees if the employees are hired before miller.

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

10) WAQTD ENAME and EMPNO of the employees if employees are earning more than ALLEN.

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

Assignment:

11) 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');
```

12) WAQTD all the details of the employees working in DEPT 20 and working in the same designation as SMITH.

```
SELECT *
FROM EMP
WHERE DEPTNO = 20
AND JOB = (SELECT JOB
           FROM EMP
           WHERE ENAME = 'SMITH');
```

13) WAQTD all the details of the employees working as MANAGER in the same DEPT as TURNER.

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

14) WAQTD NAME and HIREDATE of the employees HIRED after 1980 and before KING.

```
SELECT ENAME, HIREDATE
FROM EMP
WHERE HIREDATE > '31-DEC-1980' AND
      HIREDATE < (SELECT HIREDATE
                  FROM EMP
                  WHERE ENAME = 'KING');
```

15) WAQTD name and SAL along with ANNUAL SAL for all employees who's SAL is less than BLAKE and more than 3500.

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

16) WAQTD all the details of 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');
```

17) WAQTD NAME of the employees who's name starts with 'A' AND works in the same DEPT as BLAKE.

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

18) WAQTD NAME and COMM if employees earn COMISSION 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');
```

19) WAQTD Details of all the employees working as CLERK in the same dept as TURNER.

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

20) WAQTD ENAME, SAL AND DESIGNATION of the employees who's ANNUAL SALARY is more than SMITH And less than KING.

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

26/03/21

Note :.

→ In the inner Query / sub Query we cannot select more than one Column.

→ The corresponding columns need not be same, but the datatypes of those has to be same.

Case 2 :-

Whenever the data to be selected and the Condition to be executed are present in different tables we use sub Query.

Eg:

EMP

EID	ENAME	SAL	DEPTNO
1	ALLEN	1000	20
2	BLAKE	2000	10
3	CLARK	3000	30
4	MILLER	1500	10
5	ADAM's	2500	20

Foreign Key

DEPT

DEPTNO	DNAME	LOC
10	D_1	L1
20	D_2	L_2
30	D_3	L_3

1. WAQTD LOC of the ADAMS.

```
SELECT LOC
FROM DEPT
WHERE DEPTNO = (SELECT DEPTNO
                FROM EMP
                WHERE ENAME = 'ADAMS');
```

2. WAQTD names of the employees working in Location L_2.

```
SELECT ENAME
FROM EMP
WHERE DEPTNO = (SELECT DEPTNO
                FROM DEPT
                WHERE LOC = 'L2');
```

3. WAQTD number of employees working in dept D_3.

```
SELECT COUNT(*)
FROM EMP
WHERE DEPTNO = (SELECT DEPTNO
                FROM DEPT
                WHERE DNAME = D3);
```

4) WAQTD ENAME, sal of all the employee earning more than scott and working in dept 20.

```
SELECT ENAME, SAL
FROM EMP
WHERE DEPTNO = 20
      AND SAL > (SELECT SAL
                 FROM EMP
                 WHERE ENAME = 'SCOTT');
```

5. WAQTD all the details of the employee working as a manager in the dept Accounting.

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

6. WAQTD all the details of the employees working in the same designation as MILLER and works in Location new york.

```
SELECT *
FROM EMP
WHERE JOB = (SELECT JOB
             FROM EMP
             WHERE ENAME = 'MILLER') AND
      DEPTNO = (SELECT DEPTNO
                FROM DEPT
                WHERE LOC = 'NEW YORK');
```

7) WAQTD number of employees working as a clerk in the same deptno as SMITH and earning more than KING hired after MARTIN in the Location BOSTON.

```
SELECT COUNT(*)
FROM EMP
WHERE JOB = 'CLERK' AND DEPTNO = (SELECT DEPTNO
                                  FROM EMP
                                  WHERE ENAME = 'SMITH')
      AND SAL > (SELECT SAL
                 FROM EMP
                 WHERE ENAME = 'KING')
      AND HIREDATE > (SELECT HIREDATE
                      FROM EMP
                      WHERE ENAME = 'MARTIN')
```

```
AND DEPTNO = (SELECT DEPTNO
              FROM DEPT
              WHERE LOC = 'BOSTON');
```

8) WAQTD maximum salary given to a person working in DALLAS.

```
SELECT MAX(SAL)
FROM EMP
WHERE DEPTNO = (SELECT DEPTNO
                FROM DEPT
                WHERE LOC = 'DALLAS');
```

27/03/21

Types of SUB-QUERY :-

1. SINGLE ROW SUB-QUERY
2. MULTI ROW SUB-QUERY

1. **SINGLE ROW SUB-QUERY :-**

If the sub query returns exactly 1 record/value we call it as single row sub query.

If it returns only 1 value then we can use the normal operators or the special operators to compare the values.

Eg:
WAQTD dname of ALLEN.

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

10

Emp

EID	ENAME	SAL	DEPTNO
1	ALLEN	1000	20
2	BLAKE	2000	10
3	CLARK	3000	30
4	MILLER	1500	10
5	SMITH	2500	10

DEPT

DEPTNO	DNAME	LOC
10	D1	L1
20	D2	L2
30	D3	L3

Note:
In single row sub query we can use normal operators or special operators.

2. MULTI ROW SUB QUERY:-

If the sub query returns more than 1 record/value we call it as multi Row sub query.

If it returns more than 1 value then we cannot use the normal operators we have to use only special operators to compare the values.

Note:- It is difficult to identify whether a query belongs single or multi row so, it is always recommended to use special operators to compare the values.

Ex:
WAQTD dnames of ALLEN and SMITH.

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

20
10

DEPTNO
20
10
30
10
10

Here, since the sub query returns 2 records we cannot use '=' Op. we've to use IN op.

WAQTD ename and salary of the employees earning more than employees of dept 10.

EID	ENAME	SAL	DEPTNO
1	ALLEN	1000	20
2	BLAKE	2000	10
3	CLARK	3000	30
4	MILLER	1500	10
5	SMITH	2500	10

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL > (SELECT SAL
             FROM EMP
             WHERE DEPTNO = 10);
```

2000
1500
2500

Here we cannot use > symbol to compare multiple values. we can't use IN or NOT IN as well because it is used for = and != symbols.

Therefore we have to use <u>sub query operators</u> for Comparing Relational operators such as (>, <, >=, <=).

Sub Query operators :-

1. ALL operator :-

It is special operator used along with a relational operator (>, <, >=, <=) to compare the values present at the RHS.

ALL operator returns true if all the values at the RHS have satisfied the condition.

ex: WAQTD Employee name and SAL of employees earning more than all the employees of deptno 10.

CLARK, 2000

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

2000
1500
2500

SAL
1000
2000
3000
1500
2500

1st iteration: 1000 > ALL (2000, 1500, 2500)

1000 > 2000 F
1000 > 1500 F
1000 > 2500 F

} F

2nd iteration:

2000 > ALL (2000, 1500, 2500)

2000 > 2000 F
2000 > 1500 T } F
2000 > 2500 F

3rd iteration:

3000 > ALL (2000, 1500, 2500)

3000 > 2000 T
3000 > 1500 T } T
3000 > 2500 T

4th iteration:

1500 > ALL (2000, 1500, 2500)

1500 > 2000 F
1500 > 1500 F } F
1500 > 2500 F

5th iteration:

2500 > ALL (2000, 1500, 2500)

2500 > 2000 T
2500 > 1500 T } F
2500 > 2500 F

3) ANY operator :-

It is special operator used along with a relational operator (>, <, >=, <=) to compare the values present at the RHS.

Any operator returns true if one of the values at the RHS have satisfied the condition.

Ex:
WAQTD employee name and salary of the employees earning more than any of the salaries in Dept 10.

2000
1500
2500

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL > ANY (SELECT SAL
                 FROM EMP
                 WHERE DEPTNO = 10);
```

SAL
1000
2000
3000
1500
2500

1st iteration: 1000 > ANY (2000, 1500, 2500)

1000 > 2000 F
1000 > 1500 F } F
1000 > 2500 F

2nd iteration: 2000 > ANY (2000, 1500, 2500)

2000 > 2000 F
2000 > 1500 T } T
2000 > 2500 F

3rd iteration: 3000 > ANY (2000, 1500, 2500)

3000 > 2000 T
3000 > 1500 T } T
3000 > 2500 T

4th iteration: 1500 > ANY (2000, 1500, 2500)

1500 > 2000 F
1500 > 1500 F } F
1500 > 2500 F

5th iteration: 2500 > ANY (2000, 1500, 2500)

2500 > 2000 T
2500 > 1500 T } T
2500 > 2500 F

MAX & MIN :-

EID	ENAME	SAL	DEPTNO
1	ALLEN	1000	20
2	BLAKE	2000	10
3	CLARK	3000	30
4	MILLER	1500	10
5	ADAMS	2500	20

Ex:

1) WAQTD maximum salary of an employee.

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

2. WAQTD name of the employee getting maximum salary.

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

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

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

3) WAQTD name and salary earned by the employee getting minimum salary.

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL = (SELECT MIN(SAL)
             FROM EMP);
```

30/8/21

NESTED SUB-QUERY :-

A sub query written inside a sub query is known as Nested sub query.

We can nest about 255 sub queries.

Ex:

1) WAQTD maximum salary given to an employee.

```
SELECT MAX(SAL)        -- 5000
FROM EMP;
```

SAL
1000
2000
4000
3000
5000

2) WAQTD second maximum salary given to an employee.

```
SELECT MAX(SAL)        -- 4000
FROM EMP
WHERE SAL < (SELECT MAX(SAL)   -- 5000
             FROM EMP);
```

3) WAQTD 3rd maximum salary.

```
SELECT MAX(SAL)                                   3000
FROM EMP
WHERE SAL < (SELECT MAX(SAL)                      4000
             FROM EMP
             WHERE SAL < (SELECT MAX(SAL)         5000
                          FROM EMP));
```

4) WAQTD 4th maximum salary.

```
SELECT MAX(SAL)                                   2000
FROM EMP
WHERE SAL < (SELECT MAX(SAL)                      3000
             FROM EMP
             WHERE SAL < (SELECT MAX(SAL)         4000
                          FROM EMP
                          WHERE SAL < (SELECT MAX(SAL)   5000
                                       FROM EMP)));
```

5) WAQTD 3rd minimum salary.

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

6) WAQTD Dept name of the employee getting 2nd minimum 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)));
```

Note :-

MAXIMUM	MAX()	<
MINIMUM	MIN()	>

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

4) WAQTD EMPNO of the employee earning 2nd maximum Salary.

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

5) WAQTD department name of an employee getting 4th maximum salary.

```
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
                                           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 IN (SELECT MIN(HIREDATE)
                   FROM EMP
                   WHERE HIREDATE > (SELECT MIN(HIREDATE)
                                     FROM EMP));
```

7) WAQTD name of the employee hired before the last employee.

```
SELECT ENAME
FROM EMP
WHERE HIREDATE IN (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 IN(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))))));
```

b) WAQTD DNAME of employee getting 2nd maximum salary.

```
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)));
```

31/03/21

EMPLOYEE AND MANAGER RELATION :-

EID	ENAME	MGR
1	ALLEN	3
2	SMITH	1
3	JAMES	2
4	KING	3

Case1: To find the reporting manager or manager.

1) WAQTD name of Allen's manager.

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

2) WAQTD name of SMITH's manager.

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

3) WAQTD name of SMITH's manager's manager.

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

EID	ENAME	MGR
1	ALLEN	3
2	SMITH	1
3	JAMES	2
4	KING	3

4) WAQTD dname of king's manager.

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO = (SELECT DEPTNO
                FROM EMP
                WHERE EID = (SELECT MGR
                             FROM EMP
                             WHERE ENAME = 'KING'));
```

5) WAQTD Location of Adam's manager's manager.

```
SELECT LOC
FROM DEPT
WHERE DEPTNO = (SELECT DEPTNO
                FROM EMP
                WHERE EID = (SELECT MGR
                             FROM EMP
                             WHERE EID = (SELECT MGR
                                          FROM EMP
                                          WHERE ENAME
                                          = 'ADAMS')));
```

Case 2:

To find the employees.

1) WAQTD names of the employees reporting to KING.

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

2) WAQTD name and salary given to the employees reporting to James.

```
SELECT ENAME, SAL
FROM EMP
WHERE MGR = (SELECT EID
             FROM EMP
             WHERE ENAME = 'JAMES');
```

Note:

To find Manager	select MGR in sub Query
To find Employees	select EID in subQuery

3) WAQTD dname of the employee reporting to President.

```
SELECT DNAME
FROM DEPT
WHERE DEPTNO = (SELECT DEPTNO
                FROM EMP
                WHERE MGR = (SELECT EID
                             FROM EMP
                             WHERE JOB = 'PRESIDENT'));
```

4) WAQTD department details of the employees who are reporting to MILLER.

```
SELECT *
FROM DEPT
WHERE DEPTNO = (SELECT DEPTNO
                FROM EMP
                WHERE MGR IN (SELECT EID
                              FROM EMP
                              WHERE ENAME = 'MILLER'));
```

SUB QUERY :-

→ what is sub Query?

→ Explain ? (draw)

→ why ? when ?

→ Types of sub Query ?

- single row sub query
- Multi row sub query

→ sub Query operators

- ALL
- ANY

→ Nested sub Query ?

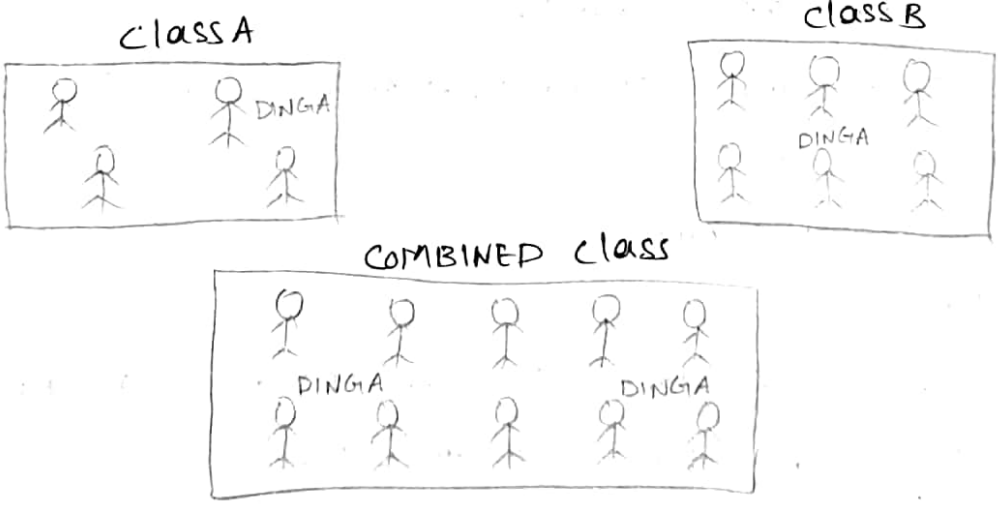
01/04/21

JOINS

The process of retrieval of data from multiple tables simultaneously is known as JOINS.

Why ? when ?

whenever the attributes 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

i) LEFT OUTER JOIN

ii) RIGHT OUTER JOIN

iii) FULL OUTER JOIN

4) SELF JOIN

5) NATURAL JOIN

1. CARTESIAN JOIN / CROSS JOIN :-

In cartesian join a record from table 1 will be merged with all the records of table 2.

EMP

ENAME	DEPTNO
A	20
B	30
C	10

Table 1 (T_1)

DEPT

DNAME	DEPTO
D_1	10
D_2	20
D_3	30

Table 2 (T_2)

Number of columns in the Result table :-

Will be equivalent to the summations of Columns present in both the tables.

Number of Col = Number of Col T_1 + Number of Col T_2

= 2 + 2

= 4 columns.

Number of Rows in the Result table :-

will be equivalent to the product of number of rows present in the both the tables.

Number of Rows = Number of Rows T_1 * Number of Rows T_2

= 3 x 3

= 9 rows.

Result table :-

ENAME	DEPTNO	DNAME	DEPTNO
A	20	D_1	10
A	20	D_2	20
A	20	D_3	30
B	30	D_1	10
B	30	D_2	20
B	30	D_3	30
C	10	D_1	10
C	10	D_2	20
C	10	D_3	30

Syntax :-

1. ANSI (American National standard institute)

```
SELECT Column_Name
FROM Table_Name 1 CROSS JOIN Table_Name 2;
```

2. Oracle :

```
SELECT Column_Name
FROM Table_Name 1, Table_Name 2;
```

1. WAQTD ename and dept name for all the employees.

```
SELECT ENAME, DNAME
FROM EMP, DEPT;
```
(ORACLE)

```
SELECT ENAME, DNAME
FROM EMP CROSS JOIN DEPT;
```
(ANSI)

02/04/21

2. INNER JOIN :.

It is used to obtain only matching records or A records which has a pair.

EMP

ENAME	DEPTNO
A	20
B	30
C	10

DEPT

DNAME	DEPTNO
D_1	10
D_2	20
D_3	30

Join Condition :-

It is a condition on which the two tables are merged.

Syntax: Table_Name1.Col_Name = Table_Name2.Col_Name

Join Condition : EMP.DEPTNO = DEPT.DEPTNO

20 = 10 F ✗ | 30 = 10 F ✗ | 10 = 10 T ✓

20 = 20 T ✓ | 30 = 20 F ✗ | 10 = 20 F ✗

20 = 30 F ✗ | 30 = 30 T ✓ | 10 = 30 F ✗

Result table :-

ENAME	EMP.DEPTNO	DNAME	DEPT.DEPTNO
A	20	D_2	20
B	30	D_3	30
C	10	D_1	10

Syntax :

1. ANSI (American National standard Institute)

```
SELECT Column_Name
FROM Table_Name 1 INNER JOIN Table_Name2
ON < JOIN_CONDITION>;
```

Eg:

```
SELECT *
FROM EMP INNER JON DEPT
ON EMP.DEPTNO = DEPT.DEPTNO;
```

2. Oracle :

```
SELECT Column_Name
FROM Table_Name1 , Table_Name2
WHERE <JOIN CONDITION>;
```

Eg.

```
SELECT *
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO;
```

1. WAQTD ename and deptname for all the employees

```
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO;
```

2. WAQTD ename and Loc for all the employees working as manager.

```
SELECT ENAME, LOC
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
    AND JOB = 'MANAGER';
```

3. WAQTD ename, deptno and dname, loc of the employee earning with a salary of more than 2000 in New york.

```
SELECT ENAME, EMP.DEPTNO, DNAME , LOC
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO AND SAL>2000
AND LOC = 'NEW YORK';
```

4. WAQTD ename, sal, and dname of the employee working as clerk in dept20 with a salary of more than 1800.

```
SELECT ENAME, SAL, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO AND
EMP.DEPTNO = 20 AND JOB = 'CLERK' AND SAL > 1800;
```

Assignment:-

1) Name of the employee and his location of all the employees.

```
SELECT ENAME, LOC
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO;
```

2) WAQTD DNAME and salary for all the employee working in Accounting.

```
SELECT DNAME, SAL
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
    AND DNAME = 'ACCOUNTING';
```

3) WAQTD DNAME and Annual salary for all employees who's salary is more than 2340.

```
SELECT DNAME, SAL*12 AS "ANNUAL SAL"
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
    AND SAL > 2340.
```

4) WAQTD ename and dname for employees having character 'A' in their Dname.

```
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
    AND DNAME LIKE '%A%';
```

5) WAQTD Ename and dname for all the employees working as salesman.

```
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
    AND JOB = 'SALESMAN';
```

6) WAQTD DName and job for all the employees who's job and Dname starts with character 's'.

```
SELECT DNAME, JOB
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO AND JOB LIKE 'S.%'
    AND DNAME LIKE 'S.%';
```

7) WAQTD DName and MGR NO. for employees reporting to 7839.

```
SELECT DNAME, MGR
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
        AND MGR = 7839.
```

8) WAQTD Dname and hiredate for employees hired after 83 into accounting or research dept.

```
SELECT DNAME, HIREDATE
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
AND HIREDATE > '31-DEC-83' AND
        DNAME IN ('ACCOUNTING', 'RESEARCH');
```

9) WAQTD Ename and dname of the employees who are getting comm in Dept 10 or 30.

```
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
AND COMM IS NOT NULL AND EMP.DEPTNO IN (10, 30);
```

10) WAQTD DNAME and EMPNO for all the employees who's EMPNO are (7839, 7902) and are working in Loc New York.

```
SELECT DNAME, EMPNO
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
```

AND EMPNO IN (7839, 7902) AND LOC = 'NEW YORK';

11) WAQTD LOC and average salary given for each location by excluding all the employees who's second char is 'A' in their name.

```
SELECT LOC, AVG(SAL)
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO AND
      ENAME NOT LIKE '_A%'
GROUP BY LOC;
```

12) WAQTD name of the emp and his LOC if employee is working as manager and working under the employee who's empno is 7839.

```
SELECT ENAME, LOC
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO AND JOB='MANAGER'
      AND MGR = 7839;
```

13) WAQTD Dname and employee ID's of all the employees who are clerks and having reporting managers.

```
SELECT DNAME, EMPNO
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO AND JOB = 'CLERK'
  AND MGR IS NOT NULL;
```

14) WAQTD Dname and total salary given to that dept if there are atleast 4 employees working for each dept.

```
SELECT DNAME, SUM(SAL)
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
GROUP BY DNAME
HAVING COUNT(*) >= 4;
```

15) WAQTD Dname and number of employees working in each dept only if there are manager or clerks.

```
SELECT DNAME, COUNT(*)
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
  AND JOB IN ('MANAGER', 'CLERK')
```

GROUP BY DNAME ;

3. Outer JOIN

It is used to obtain un-matched Records. along with the matched records.

i) Left outer join:-

It is used to obtain un-matched records of left table along with matching records.

Ex:

EMP

ENAME	DEPTNO
A	20
B	NULL
C	10
D	NULL

Left

DEPT

DNAME	DEPTNO
D_1	10
D_2	20
D_3	30
D_4	40

Right

Result table:-

ENAME	EMP.DEPTNO	DNAME	DEPT.DEPTNO
A	20	D2	20
C	10	D1	10
B	NULL	NULL	NULL
D	NULL	NULL	NULL

Syntax :

1. ANSI [American National standard Institute]

```
SELECT Column_Name
FROM Table_Name1 LEFT [OUTER] JOIN Table_Name2
ON <JOIN_CONDITION> ;
```

Ex:

```
SELECT *
FROM EMP LEFT JOIN DEPT
ON EMP.DEPTNO = DEPT.DEPTNO ;
```

2. Oracle

```
SELECT Column_Name
FROM Table_Name1, Table_Name2
WHERE Table1.col_Name = Table2.col_Name(+);
```

(+) ← Outer join operator

Ex:

```
SELECT *
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO(+);
```

Ex: WAQTD names and dnames of all the employees even Though the employees don't work in any dept.

```
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO(+);
```

ENAME	DNAME
A	D2
C	D1
B	NULL
D	NULL

WAQTD names, deptno and LOC, DNames of all the employees even though the employees don't work in any dept.

```
SELECT ENAME, EMP.DEPTNO, LOC, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO(+);
```

(ii) Right outer join :-

It is used to obtain un-matched records of right table along with matching records.

Ex:

EMP

ENAME	DEPTNO
A	20
B	NULL
C	10
D	NULL

Left

DEPT

DNAME	DEPTNO
D1	10
D2	20
D3	30
D4	40

Right

Result table:

ENAME	EMP.DEPTNO	DNAME	DEPT.DEPTNO
A	20	D2	20
C	10	D1	10
NULL	NULL	D3	30
NULL	NULL	D4	40

} matching records

} unmatched records

Syntax:

1. ANSI [American National Standard Institute]

```
SELECT column_name
FROM Table_Name1 RIGHT [OUTER] JOIN Table_Name2
ON <JOIN_CONDITION>;
```

ex:

```
SELECT *
FROM EMP RIGHT JOIN DEPT
ON EMP.DEPTNO = DEPT.DEPTNO;
```

2. Oracle

```
SELECT Column_Name
FROM Table_Name1, Table_Name2
WHERE Table.Col_Name(+) = Table2.Col_Name;
```

ex:

```
SELECT *
FROM EMP,DEPT
WHERE EMP.DEPTNO(+) = DEPT.DEPTNO;
```

Ex: WAQTD names and dnames of all the employees even though there are no employees in a dept.

```
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO(+) = DEPT.DEPTNO;
```

ENAME	DNAME
A	D2
C	D1
NULL	D3
NULL	D4

iii) **Full outer join :-**

It is used to obtain un-matched records of both left & right table along with matching records.

Ex:

EMP

ENAME	DEPTNO
A	20
B	NULL
C	10
D	NULL

Left

DEPT

DNAME	DEPTNO
D1	10
D2	20
D3	30
D4	40

Right

Result table :-

ENAME	EMP.DEPTNO	DNAME	DEPT.DEPTNO
A	20	D2	20
C	10	D1	10
B	NULL	NULL	NULL
D	NULL	NULL	NULL
NULL	NULL	D3	30
NULL	NULL	D4	40

unmatched records of left table

unmatched records of right table

Syntax :-

ANSI [American National standard Institute]

```
SELECT Column_Name
FROM Table_Name1 FULL [OUTER] JOIN Table_Name2
ON < JOIN_CONDITION >;
```

Ex:

```
SELECT *
FROM EMP FULL JOIN DEPT
ON EMP.DEPTNO = DEPT.DEPTNO;
```

WAQTD names and dnames of all the employees and depts even though the employees don't work in any dept and a dept having no employees.

```
SELECT ENAME, DNAME
FROM EMP FULL OUTER JOIN DEPT
ON EMP.DEPTNO = DEPT.DEPTNO;
```

ENAME	DNAME
A	D2
C	D1
B	NULL
D	NULL
NULL	D3
NULL	D4

4) SELF JOIN :-

"Joining a table by itself is known as self join."

Why? When?

"Whenever the data to select is in the same table but present in different records we use self-join."

Eg:

EMP E_1

EID	ENAME	E_1.MGR
1	ALLEN	3
2	SMITH	1
3	MILLER	2

EMP E_2

E_2.EID	ENAME	MGR
1	ALLEN	3
2	SMITH	1
3	MILLER	2

Join Condition : $E_1.MGR = E_2.EID$

Result table :-

E_1.EID	E_1.ENAME	E_1.MGR	E_2.EID	E_2.ENAME	E_2.MGR
1	ALLEN	3	3	MILLER	2
2	SMITH	1	1	ALLEN	3
3	MILLER	2	2	SMITH	1

Employee details - E_1 | Managers details - E_2

Syntax:

1. ANSI

```
SELECT column_name
FROM Table_name 1 JOIN Table_name 2
ON <JOIN_CONDITION>;
```

Eg:

```
SELECT *
FROM EMP E1 JOIN EMP E2
ON E1.MGR = E2.EID
```

2. Oracle

```
SELECT Column_Name
FROM Table_Name1 T1, Table_Name2 T2
WHERE <JOIN_CONDITION>;
```

Eg:

```
SELECT *
FROM EMP E1, EMP E2
WHERE E1.MGR = E2.EID;
```

1. WAQTD Ename and manager's name for all the employees.

```
SELECT E1.ENAME, E2.ENAME
FROM EMP E1, EMP E2
WHERE E1.MGR = E2.EMPNO;
```

2. WAQTD Ename, sal along with manager's name and manager's salary for all the employees.

```
SELECT E1.ENAME, E1.SAL, E2.ENAME, E2.SAL
FROM EMP E1, EMP E2
WHERE E1.MGR = E2.EMPNO;
```

3. WAQTD ename, manager's name along with their deptno if employee is working as clerk.

```
SELECT E1.ENAME, E2.ENAME, E1.DEPTNO, E2.DEPTNO
FROM EMP E1, EMP E2
WHERE E1.MGR = E2.EMPNO AND E1.JOB = 'CLERK';
```

4. WAQTD ename, manager's job if manager works as Analyst.

```
SELECT E1.ENAME, E2.JOB
FROM EMP E1, EMP E2
WHERE E1.MGR = E2.EMPNO AND E2.JOB = 'ANALYST';
```

5. WAQTD ename and manager's name along with their job if emp and manager are working for same designation.

```
SELECT E1.ENAME, E2.ENAME, E1.JOB, E2.JOB
FROM EMP E1, EMP E2
WHERE E1.MGR = E2.EMPNO AND E1.JOB = E2.JOB;
```

6) WAQTD ename, emp salary manager's name, manger's salary if manager earns more than employee.

```
SELECT E1.ENAME, E1.SAL, E2.ENAME, E2.SAL
FROM EMP E1, EMP E2
WHERE E1.MGR = E2.EMPNO AND E2.SAL > E1.SAL;
```

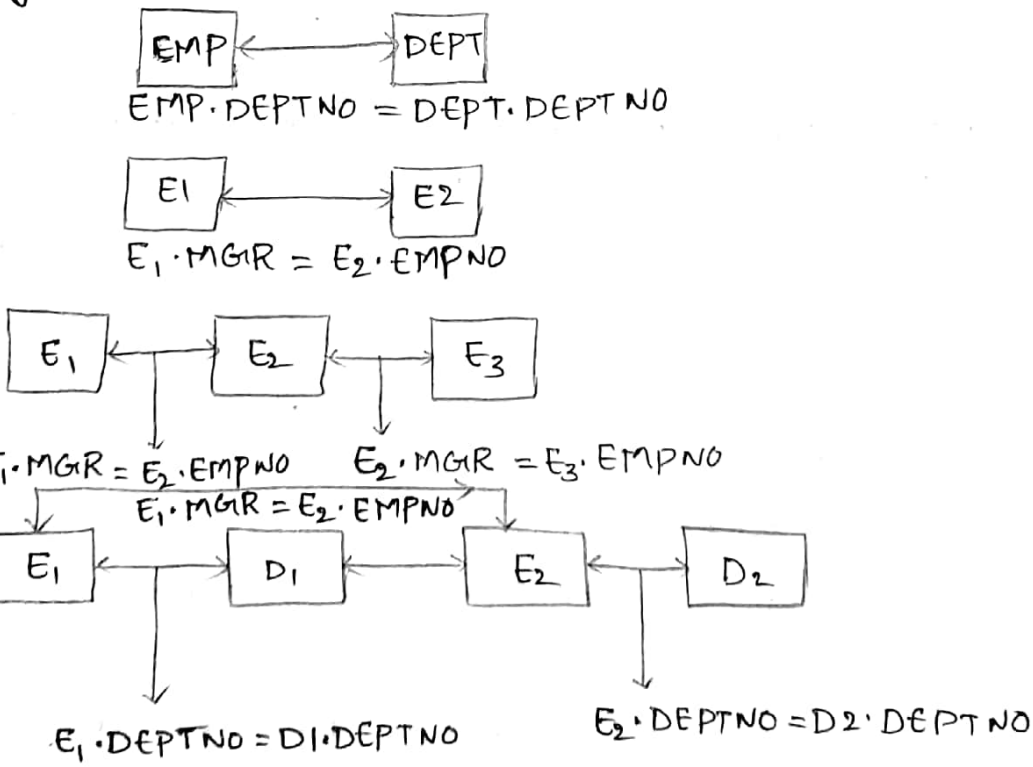
7) WAQTD ename and manager's name along with manager's commission if manager earns commission.

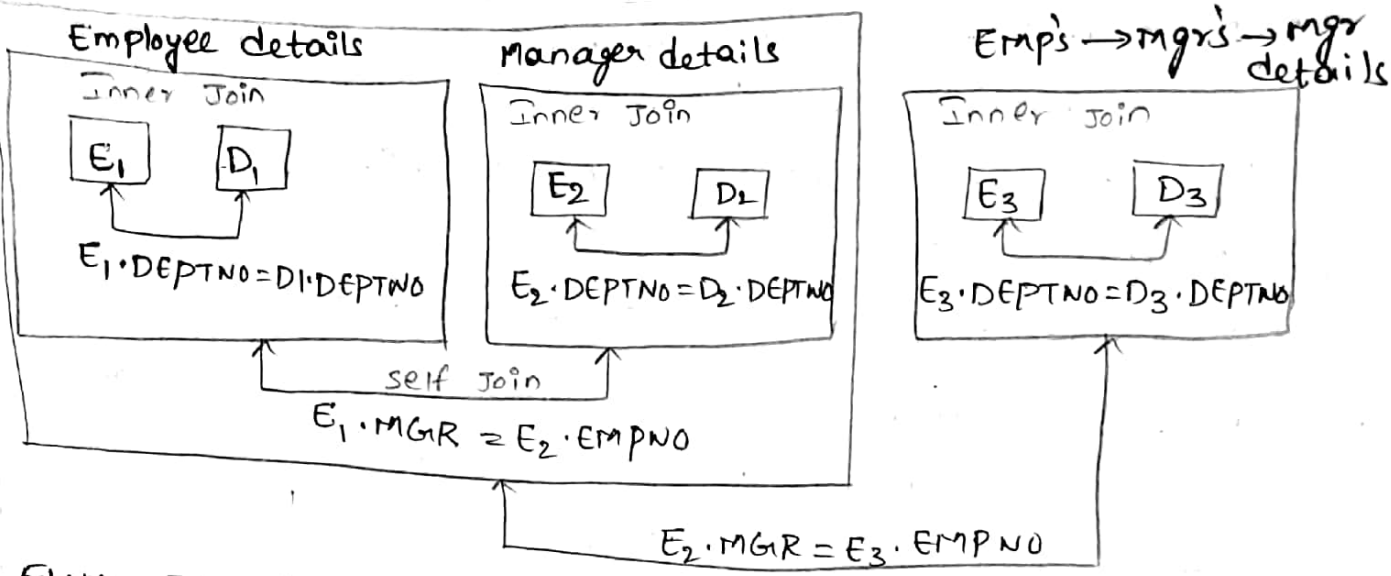
```
SELECT E1.ENAME, E2.ENAME, E2.COMM
FROM EMP E1, EMP E2
WHERE E1.MGR = E2.EMPNO AND E2.COMM IS NOT NULL;
```

Note:-
To join 'N' number of tables we need to write 'N-1' number of join conditions.

06/04/21

Joining more than one table :-



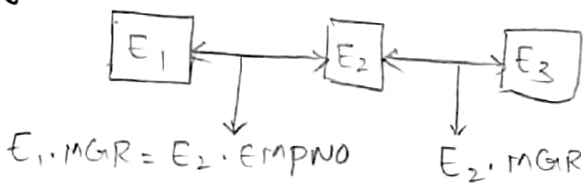


~~6) NATURAL JOIN~~

1) WAQTD employee name, emp's department name and his manager name and his department name.

```
SELECT E1.ENAME, D1.DNAME, E2.ENAME, D2.DNAME
 FROM EMP E1, DEPT D1, EMP E2, DEPT D2
WHERE E1.DEPTO = D1.DEPTO AND E1.MGR = E2.EMPNO AND
                              E2.DEPTNO = D2.DEPTNO;
```

2) WAQTD employee name his manager name and manager's manager name.




```
SELECT E1.ENAME, E2.ENAME, E3.ENAME
 FROM EMP E1, EMP E2, EMP E3
WHERE E1.MGR = E2.EMPNO AND E2.MGR = E3.EMPNO;
```

(or)

```
SELECT E1.ENAME EMP_NAME, E2.ENAME MGR_NAME,
 E3.ENAME MGR_MGR_NAME
 FROM EMP E1, EMP E2, EMP E3
WHERE E1.MGR = E2.EMPNO AND E2.MGR = E3.EMPNO;
```

5. Natural Join :-

It behaves as INNER JOIN if there is a relation between the given two tables, else it behaves as CROSS JOIN.

Syntax :

ANSI

```
SELECT Col_Name
FROM Table_Name 1 NATURAL JOIN Table_Name 2;
```

Ex:

EMP

ENAME	DEPTNO
A	20
B	30
C	10

DEPT

DNAME	DEPTNO
D1	10
D2	20
D3	30

Result table has a relation (Inner Join)

DEPTNO	ENAME	DNAME
20	A	D2
30	B	D3
10	C	D1

Ex:2

EMP

ENAME	DEPTNO
A	20
B	30
C	10

CUSTOMER

CNAME	CID
X	101
Y	102
Z	103

Result table: has no relation (cross join)

ENAME	DEPTNO	CNAME	CID
A	20	X	101
A	20	Y	102
A	20	Z	103
B	30	X	101
B	30	Y	102
B	30	Z	103
C	10	X	101
C	10	Y	102
C	10	Z	103

3/04/24

ORDER BY CLAUSE:-

ORDER BY CLAUSE is used to rearrange the records in the result table either in ascending or descending order.

ORDER BY SYNTAX:-

```
SELECT group_by_expression /group_function
FROM table_name
[WHERE <filter_condition>]
[GROUP BY column_name/expression]
[HAVING <group_filter_condition>]
 ORDER BY col_name/expression [ASC]/DESC ;
```

ORDER OF EXECUTION:-

1- FROM
2- WHERE (if used) [ROW-BY-ROW]
3- GROUP BY (if used) [ROW-BY-ROW]
4- HAVING (if used) [GROUP-BY-GROUP]
5- SELECT [GROUP-BY-GROUP]
6- ORDER BY [ROW-BY-ROW]

WAQTD Salary of employees in ascending order.

```
SELECT SAL
FROM EMP
ORDER BY SAL ASC;
```

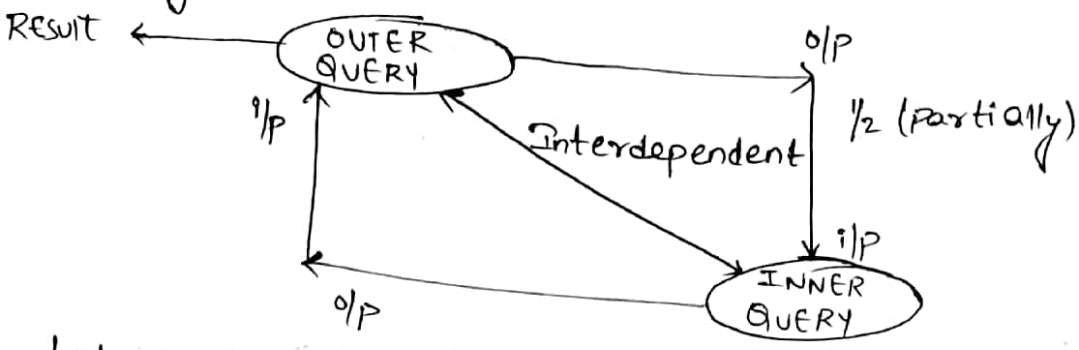
WAQTD the salary of employees in descending order.

```
SELECT SAL
FROM EMP
ORDER BY SAL DESC;
```

CO-RELATED SUB-QUERY

A query written inside another query such that the outer query and the inner query are dependent on each other, this is known as Co-Related sub-Query.

Working principle:-



Let us Consider two queries inner and outer query respectively,

1) Outer query executes first but partially.
2) The partially executed output is given as an input to the inner query.
3) The inner query executes completely and generates an output.
4) The output of inner query is fed as an input to the outer query and outer query produces the result.
5) Therefore, we can state that the outer query and the inner query both are INTERDEPENDENT (dependent on each other).

Note:-

1) In Co-related sub query a Join Condition is a must, And must be written Only in the inner query.
2) Co-Related sub query works with the principles of both SUB QUERY & JOINS.

Difference between sub query and co-related sub query:-

SUB QUERY	Co-Related sub query
1) Inner query executes first	1) Outer query executes first
2) Outer query is dependent on inner query.	2) Both are interdependent

3) Join condition not mandatory.	3) Join condition is mandatory and must be written in inner query.
4) Outer query executes once.	4) Outer query executes twice.

Ex: WAQTD department names of all employees who are working.

Dept

DNAME	DNO
D_1	10
D_2	20
D_3	30
D_4	40

Emp

ENAME	DNO
A	20
B	10
C	20
D	30

D_1 D_2 D_3 → final result

```
SELECT DNAME
FROM DEPT D
WHERE D.DNO IN (SELECT E.DNO
                FROM EMP E
                WHERE D.DNO = E.DNO);
```

D.DNO:

D.DNO
10
20
30
40

E.DNO:

E.DNO
20
10
20
30

10 / (20,20) / 30

D.DNO = E.DNO

1st iteration:
- 10 = 20 F
- 10 = 10 T
- 10 = 20 F
- 10 = 30 F

2nd iteration:-
- 20 = 20 T
- 20 = 10 F
- 20 = 20 T
- 20 = 30 F

3rd iteration:
- 30 = 20 F
- 30 = 10 F
- 30 = 20 F
- 30 = 30 T

4th iteration:
- 40 = 20 F
- 40 = 10 F
- 40 = 20 F
- 40 = 30 F

Outer query final execution:-

1st iteration:
10 IN (10, (20,20), 30)
- 10 = 10 T
- 10 IN (20,20)
- 10 = 20 F
- 10 = 20 F
- 10 = 30 F

2nd iteration
20 IN (10, (20,20), 30)
- 20 = 10 F
- 20 IN (20,20)
- 20 = 20 T
- 20 = 20 T
- 20 = 30 F

3rd iteration
30 IN (10, (20,20), 30)
- 30 = 10 F
- 30 IN (20,20)
- 30 = 20 F
- 30 = 20 F
- 30 = 30 T

4th iteration
40 IN (10, (20,20), 30)
- 40 = 10 F
- 40 IN (20,20)
- 40 = 20 F
- 40 = 20 F
- 40 = 30 F

final result:-

D_1
D_2
D_3

WAQTD the Loc of employees who are working.

```
SELECT LOC
FROM DEPT D
WHERE D.DNO IN (SELECT E.DNO
                FROM EMP E
                WHERE D.DNO = E.DNO);
```

WAQTD dname in which there are no employees working.

```
SELECT DNAME
FROM DEPT D
WHERE D.DEPTNO NOT IN (SELECT E.DEPTNO
                       FROM EMP E
                       WHERE D.DEPTNO = E.DEPTNO);
```

EXISTS & NOT EXISTS OPERATORS :-

1. EXISTS OPERATOR :-

"EXISTS operator is a unary operator (one operand) which can accept one operand towards RHS and that operand has to be a co-related sub query."

→ Exists operator returns true if the sub query returns any value other than NULL.

Ex: WAQTD department names of all employees who are working.

```
SELECT DNAME
FROM DEPT D
WHERE EXISTS (SELECT E.DNO
              FROM EMP E
              WHERE D.DNO = E.DNO);
```

2. NOT EXISTS :-

"NOT EXISTS operator is a unary operator (one operand) which can accept one operand towards RHS and that operand has to be a co-related sub query."

→ NOT EXISTS operator returns true if the sub query returns NULL.

Ex: WAQTD Dname, DEPTNO in which they are no employees working.

```
SELECT DNAME, DEPTNO
FROM DEPT D
WHERE NOT EXISTS (SELECT E.DEPTNO
                  FROM EMP E
                  WHERE D.DEPTNO = E.DEPTNO)
```

09/04/21

To find MAX & MIN salary:-

To find maximum salary:-

Syntax:

```
SELECT SAL
FROM EMP E1
WHERE (SELECT COUNT(DISTINCT SAL)
       FROM EMP E2
       WHERE E1.SAL < E2.SAL) = N-1;
```

Eg: write a query to display 3rd MAX SAL.

```
SELECT SAL
FROM EMP E1
WHERE (SELECT COUNT(DISTINCT SAL)
       FROM EMP E2
       WHERE E1.SAL < E2.SAL) = 2;
```

E1.SAL	E2.SAL
1000	1000
3000	3000
2000	2000
3000	3000
2000	2000
4000	4000
5000	5000

1st iteration:

E1.SAL		E2.SAL	
1000	<	1000	F
1000	<	3000	T ✓
1000	<	2000	T ✓
1000	<	3000	T ✓
1000	<	2000	T ✓
1000	<	4000	T ✓
1000	<	5000	T ✓

2nd & 4th iteration

3000	<	1000	F
3000	<	3000	F
3000	<	2000	F
3000	<	3000	F
3000	<	2000	F
3000	<	4000	T ✓
3000	<	5000	T ✓

3rd & 5th iteration

2000	<	1000	F ✗
2000	<	3000	T ✓
2000	<	2000	F ✗
2000	<	3000	T ✗
2000	<	2000	F ✗
2000	<	4000	T ✓
2000	<	5000	T ✓

WAQTD 2nd, 4th, 5th, 7th MAX salary.

```
SELECT SAL
FROM EMP E1
WHERE (SELECT COUNT(DISTINCT SAL)
       FROM EMP E2
       WHERE E1.SAL < E2.SAL) in (1,3,4,6);
```

WAQTD 10th maximum Salary.

```
SELECT SAL
FROM EMP E1
WHERE (SELECT COUNT(DISTINCT SAL)
       FROM EMP E2
       WHERE E1.SAL < E2.SAL) = 9;
```

2) WAQTD 25th min SAL

```
SELECT SAL
FROM EMP E1
WHERE (SELECT COUNT(DISTINCT SAL)
       FROM EMP E2
       WHERE E1.SAL > E2.SAL) = 25;
```

3) WAQTD 7th max sal.

```
SELECT SAL
FROM EMP E1
WHERE (SELECT COUNT(DISTINCT SAL)
       FROM EMP E2
       WHERE E1.SAL < E2.SAL) = 6;
```

4) WAQTD Top 3 earners.

```
SELECT SAL
FROM EMP E1
WHERE (SELECT COUNT(DISTINCT SAL)
       FROM EMP E2
       WHERE E1.SAL < E2.SAL) = 2
       ORDER BY SAL DESC;
```

5) WAQTD least 3 earners.

```
SELECT SAL
FROM EMP E1
WHERE (SELECT COUNT(DISTINCT SAL)
       FROM EMP E2
       WHERE E1.SAL > E2.SAL) = 3
       ORDER BY SAL ASC;
```

6) WAQTD 1, 3, 5 max salaries.

```
SELECT SAL
FROM EMP E1
WHERE (SELECT COUNT(DISTNCT SAL)
       FROM EMP E2
       WHERE E1.SAL < E2.SAL) = (0, 2, 4));
```

7) WAQTD 2,4,6 min salaries.

```
SELECT SAL
FROM EMP E1
WHERE (SELECT COUNT(DISTINCT SAL)
       FROM EMP E2
       WHERE E1.SAL > E2.SAL = (1,3,5));
```

17/04/21

SINGLE ROW FUNCTIONS :-

1. LENGTH()
2. CONCAT()
3. UPPER()
4. LOWER()
5. INITCAP()
6. REVERSE()
7. SUBSTR() (SUBSTR())
8. INSTR()
9. REPLACE()
10. MOD()
11. TRUNC()
12. ROUND()
13. MONTHS_BETWEEN()
14. LAST_DAY()
15. TO_CHAR()
16. NVL()

1. LENGTH :-

"It is used to count the number of characters present in the given string."

Syntax : LENGTH('string')

Ex:
WAQTD COUNT number of characters present in 'SMITH'.

```
SELECT LENGTH(ENAME)
FROM EMP
WHERE ENAME = 'SMITH';
```

LENGTH(ENAME)
5

(OR)

```
SELECT LENGTH('SMITH')
FROM DUAL;
```

NOTE: DUAL TABLE :-

It is a DUMMY table which has 1 col and 1 row. which is used to output the result.

→ DESC DUAL ;-

→ SELECT *
FROM DUAL;

Ex SELECT LENGTH('HELLO WORLD')
FROM DUAL; → Output : 11

2. CONCAT() :-

"It is used to join the given two strings".

Syntax : CONCAT ('string 1', 'string 2')

Ex:

input : Smith

Output : Mr. Smith

```
SELECT CONCAT ('Mr.', ENAME)
FROM EMP
WHERE ENAME = 'SMITH';
```

3. UPPER() :-

"It is used to Convert a given string to upper Case".

Eg: UPPER('dinga') → DINGA Syntax : UPPER ('string')

4. LOWER() :-

"It is used to Convert a given string to Lower case".

Syntax : LOWER ('string')

Eg: LOWER('DINGA') → dinga

5. INITCAP() :-

"It is used to Convert a given string to initial capital letter case".

Syntax : INITCAP('string')

6 Ex: INITCAP('dinGA') ⇒ Result : Dinga
FROM DUAL;

6) REVERSE() :-

"It is used to reverse a given string."

Syntax: REVERSE('string')

Ex:

SELECT REVERSE('SMITH')
FROM DUAL; ⇒ HTIMS

7) SUBSTR() :-

"It is used to extract a part of string from the given Original string".

Syntax: SUBSTR('original_string', Position, [length])

Note:
Length is not mandatory, if length is not mentioned then consider the complete string.

Ex:

-ve	-7	-6	-5	-4	-3	-2	-1
	Q	S	P	I	D	E	R
+ve	1	2	3	4	5	6	7

SUBSTR('QSPIDER', 2, 3) → SPI

SUBSTR('QSPIDER', 3, 3) → PID

SUBSTR('QSPIDER', 2) → SPIDER

SUBSTR('QSPIDER', 1, 6) → QSPIDE

SUBSTR('QSPIDER', 4, 1) → I

SUBSTR('QSPIDER', 1, 1) → Q

SUBSTR('QSPIDER', 7, 1) → R

SUBSTR('QSPIDER', 6) → ER

SUBSTR('QSPIDER', 0, 3) → QSP

SUBSTR('QSPIDER', 6, 6) → ER

SUBSTR('QSPIDER', -2, 1) → E

SUBSTR('QSPIDER', -5, 3) → PID

SUBSTR('QSPIDER', -7, 2) → QS

SUBSTR('QSPIDER', -1) → R

WAQTD extrait first 3 characters of the emp names.

```
SELECT SUBSTR(ENAME, 1, 3)
FROM EMP;
```

WAQT extract last 3 characters of the employee names.

```
SELECT SUBSTR(ENAME, -3)
FROM EMP;
```

WAQTD first half of the employee names

```
SELECT SUBSTR(ENAME, 1, LENGTH(ENAME)/2)
FROM EMP;
```

WAQTD second half of the employee names.

```
SELECT SUBSTR(ENAME, LENGTH(ENAME)/2 + 1)
FROM EMP;
```

ex for first half of the employee names.

SMITH:

SUBSTR(ENAME, 1, LENGTH(ENAME)/2)

SUBSTR('SMITH', 1, LENGTH('SMITH')/2)

SUBSTR('SMITH', 1, 5/2)

SUBSTR('SMITH', 1, 2)

Result : SM

WARD: SUBSTR(ENAME, 1, LENGTH(ENAME)/2)

SUBSTR('WARD', 1, LENGTH('WARD')/2)

SUBSTR('WARD', 1, 4/2)

SUBSTR('WARD', 1, 2) ⟹ Result : WA

Eg: for second half of employee names.

SMITH: SUBSTR(ENAME, LENGTH(ENAME)/2+1)

SUBSTR('SMITH', LENGTH('SMITH')/2+1)

SUBSTR('SMITH', 5/2+1)

SUBSTR('SMITH', 3)

Result: ITH

WARD: SUBSTR(ENAME, LENGTH(ENAME)/2+1)

SUBSTR('WARD', LENGTH('WARD')/2+1)

SUBSTR('WARD', 4/2+1)

SUBSTR('WARD', 3)

Result: RD

8. REPLACE():

"It is used to replace a string with another string in the original string."

Syntax: REPLACE('original_string', 'string', ['new_string']

Ex:

REPLACE('BANANA', 'A', 'C') → BCNCNC

REPLACE('BANANA', 'N', 'ABC') → BAABC AABCA

REPLACE('DPPO', 'O', 'J') → JPPJ

REPLACE('BANANA', 'A') → BNN

REPLACE('ENGINEERING', E) → NGINRING

REPLACE('BANANA', 'N', 'INDIA') → BAINDIAAINDIAA

NOTE:

If the third argument is not mentioned the default value of it is NULL.

9. INSTR():

"It is used to obtain the position in which the string is present in the original string". it is used to

search for a string in the original string if present it returns the POSITION else it returns "0".

Syntax: INSTR('original_string', 'string', 'Position', [occurence])

Note:
If occurrence is not mentioned then, the default value of occurence is 1.

Ex:

B	A	N	A	N	A
1	2	3	4	5	6

INSTR('BANANA', 'A', 1, 1) → POS: 2

INSTR('BANANA', 'A', 2, 1) → POS: 2

INSTR('BANANA', 'A', 1, 2) → POS: 4

INSTR('BANANA', 'A', 1, 3) → POS: 6

INSTR('BANANA', 'A', 1, 4) → POS: 0

INSTR('BANANA', 'A', 4, 2) → POS: 6

INSTR('BANANA', 'A', 2) → POS: 2

INSTR('BANANA', 'N', 2, 1) → POS: 3

INSTR('BANANA', 'O', 1, 1) → POS: 0

INSTR('BANANA', 'NA', 2, 2) → POS: 5

INSTR('BANANA', 'A', 3, 3) → POS: 0

INSTR('BANANA', 'AN', 1, 2) → POS: 4

INSTR('BANANA', 'ANA', 1, 2) → POS: 4

→ SELECT INSTR('DINGA', 'G', 3) ABD
FROM DUAL;
output: ABD: 4

→ SELECT INSTR('DINGA', 'A', 1, 3)
FROM DUAL;
output: 0

search for a string in the original string if present it returns the POSITION else it returns 0.

Syntax: INSTR('original_string', 'string', 'Position', [occurence])

Note:
If occurrence is not mentioned then, the default value of occurence is 1.

Ex:

B	A	N	A	N	A
1	2	3	4	5	6

INSTR('BANANA', 'A', 1, 1) → POS: 2

INSTR('BANANA', 'A', 2, 1) → POS: 2

INSTR('BANANA', 'A', 1, 2) → POS: 4

INSTR('BANANA', 'A', 1, 3) → POS: 6

INSTR('BANANA', 'A', 1, 4) → POS: 0

INSTR('BANANA', 'A', 4, 2) → POS: 6

INSTR('BANANA', 'A', 2) → POS: 2

INSTR('BANANA', 'N', 2, 1) → POS: 3

INSTR('BANANA', 'O', 1, 1) → POS: 0

INSTR('BANANA', 'NA', 2, 2) → POS: 5

INSTR('BANANA', 'A', 3, 3) → POS: 0

INSTR('BANANA', 'AN', 1, 2) → POS: 4

INSTR('BANANA', 'ANA', 1, 2) → POS: 4

→ SELECT INSTR('DINGA', 'G', 3) ABD
FROM DUAL;
output: ABD: 4

→ SELECT INSTR('DINGA', 'A', 1, 3)
FROM DUAL;
output: 0

10) MOD() :-

"It is used to obtain modulus/remainder of the given number."

Syntax: MOD(m,n)

Ex: SELECT MOD(5,2)
FROM DUAL; → 1

1. WAQTD ENAMES of the employees who earn salary in multiples of 3.

```
SELECT ENAME
FROM EMP
WHERE MOD(SAL, 3) = 0;
```

2. WAQTD details of the employee who have odd EID.

```
SELECT *
FROM EMP
WHERE MOD(EID, 2) = 1;
```

11. ROUND() :-

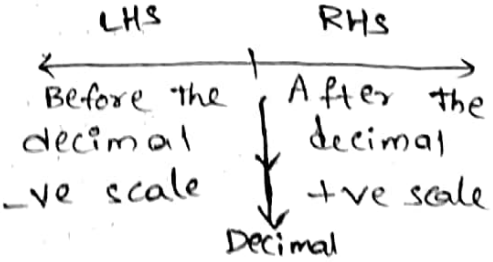
"It is used to Round-off the given number based on the scale value".

Syntax: ROUND(Number, [scale])

Note :- The default value of scale is 0.

Ex:

ROUND(5.6) → 6
ROUND(5.5) → 6
ROUND(5.4) → 5
ROUND(9.9) → 10
ROUND(9.4) → 9
ROUND(8.6,0) → 9



Note :-

When the scale is -ve it indicated the digits before the decimal and the digit count begins from 1.

Ex:

ROUND (8421.12, -1) → 8420

ROUND (8426.12, -1) → 8430

ROUND (154264.12, -2) → 154300

ROUND (338222, -4) → 340000

ROUND (2514, -3) → 3000

Note:-

5.6 ≥ 6 (Rounding off to units place)

Note:-

When the scale is +ve it indicated the digits after the decimal and the digit starts count from 0.

Eg:

ROUND (124.23541, 0) → 124

ROUND (124.23541, 1) → 124.2

ROUND (124.23541, 2) → 124.24

ROUND (124.2354391, 5) → 124.23544

12) TRUNC ():-

"It is similar to ROUND() but it always rounds-off the given number to the lower value."

Syntax: TRUNC (Number, [scale])

Ex:

TRUNC (5.6) → 5

TRUNC (5.5) → 5

TRUNC (5.4) → 5

TRUNC (9.9) → 9

TRUNC (9.4) → 9

TRUNC (8.6,0) → 8

TRUNC (451258.32541, -5) → 400000

NOTE:

DATE COMMANDS:-

1. SYSDATE: "SY is used to obtain today's date".
2. CURRENT_DATE: "It is also used to obtain today's date".
3. SYSTIMESTAMP: "It is used to obtain date, time and time zone".

Eg:

```
SELECT SYSDATE
FROM DUAL;
```

→

```
SYSDATE
17-MAY-20
```

```
SELECT CURRENT_DATE
FROM DUAL;
```

→

```
CURRENT_DATE
17-MAY-20
```

```
SELECT SYSTIMESTAMP
FROM DUAL;
```

→

```
SYSTIMESTAMP
17-MAY-20 05.05.52.356000
PM +05.30
```

3) MONTHS_BETWEEN():-

"It is used to obtain the number of months present between the given two dates."

syntax: MONTHS_BETWEEN(DATE1, DATE2)

Eg:

```
SELECT TRUNC(MONTHS_BETWEEN(SYSDATE, HIREDATE)||"Months"
FROM EMP;
```

```
TRUNC(MONTHS_BETWEEN(SYSDATE, HIREDATE)||"MONTH'
473 months
470 months
```

4) LAST_DAY():

"It is used to obtain the last day in the particular of the given date".

syntax: LAST_DAY(DATE);

Eg:

```
SELECT LAST_DAY(SYSDATE)
FROM DUAL;
```

SYSDATE: 08-JUL-2020

```
AST_DAY
31-JUL-20
```

15. TO_CHAR() :-

"It is used to convert the given date into string format based on the model given."

Syntax: TO_CHAR (DATE, 'Format_Models')

Format models :-

1) YEAR : TWENTY TWENTY
2) YYYY : 2020
3) YY : 20
4) MONTH : JULY
5) MON : JUL
6) MM : 07
7) DAY : WEDNESDAY
8) DY : WED
9) DD : 08
10) D : 4 (day of the week)
11) HH24 : 17 hours
12) HH12 : 5 hours
13) MI : 22 minutes
14) SS : 53 seconds
15) 'HH12 : MI : SS' : 5 : 22 : 53
16) 'DD - MM - YY' : 17 - 05 - 20
17) 'MM - DD - YYYY' : 05 - 17 - 2020.

Eg:

```
SELECT TO_CHAR (SYSDATE , 'YEAR')
FROM DUAL ;
```

TO_CHAR(SYSDATE,'YEAR')

TWENTY TWENTY ONE

* SELECT TO_CHAR(SYSDATE, 'DD')
FROM DUAL;

TO_CHAR(SYSDATE, 'DD')
→ 17

* SELECT TO_CHAR(SYSDATE, 'DAY')
FROM DUAL;

TO_CHAR(SYSDATE, 'DAY')
→ SATURDAY

* SELECT TO_CHAR(SYSDATE, 'D')
FROM DUAL;

TO_CHAR(SYSDATE, 'D')
→ 7

1. WAQTD details of the employee who was hired on a SUNDAY.

```
SELECT *
FROM EMP
WHERE TO_CHAR(HIREDATE,'DAY') = 'SUNDAY';
```

2. WAQTD details of an employee hired on MONDAY AT 10AM.

```
SELECT *
FROM EMP
WHERE TO_CHAR(HIREDATE, 'D') = 2 AND
      TO_CHAR(HIREDATE, 'HH24') = 10;
```

16) NVL()[NULL VALUE LOGIC]() :-

"It is used to eliminate the side effects of using null in arithmetic operations."

Eg: WAQTD Name and total salary of all the employees.

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

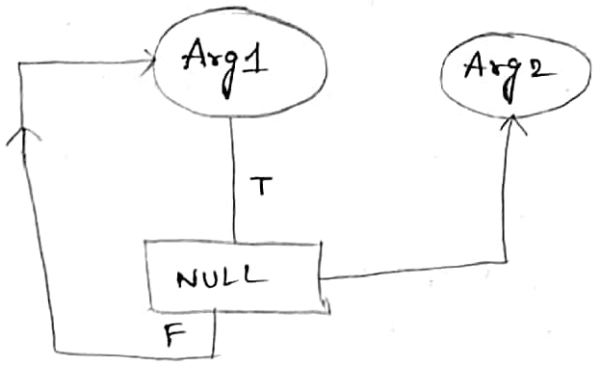
ENAME	SAL	COMM
A	500	100
B	1000	NULL
C	2000	200
D	2000	NULL

ENAME	SAL+COMM
A	600
B	NULL
C	2200
D	NULL

Syntax: NVL (Argument 1, Argument 2)

Argument 1:

Here write any column/exp which can result in null.



Argument 2:

Here we write a numeric value which will be substituted if argument 1 result in NULL.

If argument 1 is not null, then the same value will be considered.

Eg:

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

A	500 + NVL (100, 0)	500+100	600
B	1000 + NVL (null, 0)	1000+0	1000
C	2000 + NVL (200, 0)	2000+200	2200
D	2000 + NVL (null, 0)	2000+0	2000

After using NVL

ENAME	SAL + nvl (COMM, 0)
A	600
B	1000
C	2200
D	2000

Data definition Language (DDL) :-

DDL is used to construct an object in the database and deals with the structure of the object.

It has 5 statements.

1. CREATE
2. RENAME
3. ALTER
4. TRUNCATE
5. DROP

1) CREATE:

It is used to build [construct an object

object/Entity can be a table or a view (virtual table)

How to create a table:-

→ Name of the table

⇒ Tables cannot have same names

→ Number of columns

→ Names of columns

→ Assign datatypes for the columns.

→ Assign Constraints [NOT MANDATORY].

Example:

Table_Name : CUSTOMER

Number of columns : 4

customer

Column_Name	CID	CNAME	CNO	ADDRESS
Datatypes	Number(2)	Varchar(10)	Number(10)	Varchar(15)
NULL/Not NULL	Not NULL	Not NULL	Not NULL	NULL
Unique	Unique		Unique	
check			check (length(CNO) = 10)	
Primary key	Primary key			
Foreign key				
Not mandatory				

Syntax to create a table:-

```
CREATE TABLE Table_Name (
    Column_Name1 datatype constraint_type,
    Column_Name2 datatype constraint_type,
    Column_Name3 datatype constraint_type,
    :
    Column_NameN datatype constraint_type.
);
```

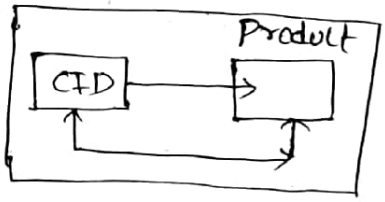
Eg:

```
CREATE TABLE CUSTOMER
(
    CID Number(2) primary key
    CNAME varchar(10),
    CNO Number(10) not null check(length(CNO)=10),
    ADDRESS varchar(15)
);
```

Note:-

To describe the table,

Syntax: DESC Table_Name;



Eg:2:-

Table_Name : PRODUCT

Number of Columns: 4

Product

Column_Name	PID	PNAME	PRICE	CID
DatatypEs	Number(2)	Varchar(10)	Number(7,2)	Number(2)
NULL/Not NULL	Not NULL	Not NULL	Not NULL	NULL
Unique	Unique			
check			check(price>0)	
Foreign key				Foreign key

Syntax to create a table:-

```
CREATE TABLE Table_Name(
    Column_Name1  datatype  Constraint_type,
    Column_Name2  datatype  Constraint_type,
    Column_Name3  datatype  Constraint_type,
    :
    Column_Name N  datatype  Constraint_type
    Constraint  Foreign key references parent_table_Name(
                                        Column_Name)
);
```

Eg:

```
CREATE TABLE STUDENT
(
    SID NUMBER(2),
    SNAME VARCHAR(20),
    BRANCH VARCHAR(20),
    PERCENTAGE NUMBER(3,1),
)
```

op: Table created

2) RENAME:-

It is used to change the name of the object.

syntax:

```
RENAME Table_Name TO New_Name;
```

Eg:

```
RENAME customer to cust;
RENAME PRODUCT TO PRO;
```

3) ALTER:

It is used to modify the structure of the table.

TO ADD A COLUMN:-

syntax:

```
ALTER TABLE Table_Name
    ADD column_Name Datatype Constraint_type;
```

Eg:

```
ALTER TABLE cust
ADD MAIL_ID Varchar(15);
```

TO DROP A COLUMN:

Syntax: ALTER TABLE Table_Name
DROP COLUMN Column_Name;

Eg:
ALTER TABLE cust
DROP COLUMN Mail_ID;

TO RENAME A COLUMN :-

syntax: ALTER TABLE Table_Name
RENAME COLUMN Column_Name to new_column_Name;

Eg: ALTER TABLE cust
RENAME COLUMN CNO TO PHONE_NO;

TO MODIFY THE DATATYPE :-

Syntax: ALTER TABLE Table_Name
MODIFY COLUMN_NAME New_Data type;

Eg: ALTER TABLE cust
MODIFY CNAME CHAR(16);

TO MODIFY NOT NULL CONSTRINTS :-

syntax: ALTER TABLE Table_Name
MODIFY COLUMN_NAME Existing_datatype [NULL] / NOT NULL;

Eg: ALTER TABLE cust
MODIFY ADDRESS varchar(15) Not Null;

4) TRUNCATE:-
It is used to remove all the records from the table permenently.

syntax: TRUNCATE TABLE Table_Name;

cust

CID	cname	Phone no	Address
1	A	1234567890	BANGLORE
2	B	1234567899	MYSORE
3	C	1234567880	MANGALORE

Eg: TRUNCATE TABLE Cust;

Cust

Cid	cname	phone no	Address

5) DROp :-

It is used to remove the table from the database

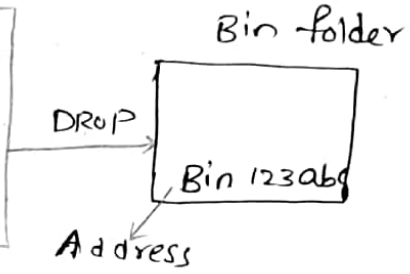
Syntax: DROP TABLE Table_Name;

Eg:

DATABASE

Cust

CID	CName	phone no	Address
1	A	1234567890	BANGLORE
2	B	1234567899	MYSORE
3	C	1234567880	MANGLORE

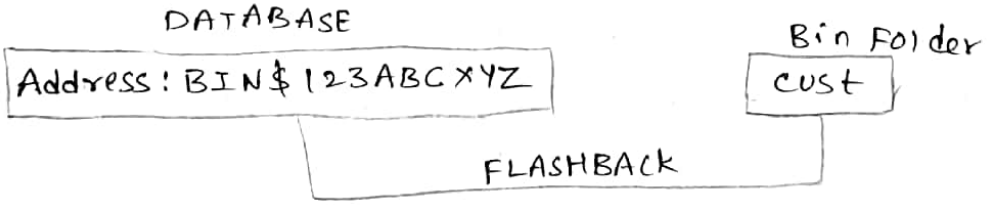


To Recover the table:-

syntax:-

FLASHBACK TABLE Table_Name
TO BEFORE DROP;

Eg:

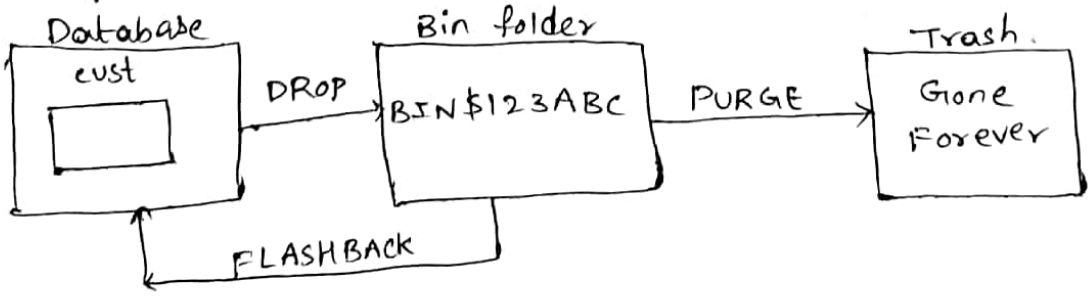


FLASHBACK TABLE Cust
TO BEFORE DROp;

To delete the table from Bin folder:-

Syntax: PURGE TABLE Table_Name;

Eg: PURGE TABLE cust;



Note:-
DDL statements are Auto-Commit statements.

Eg:
⇒ drop table corona;
o/p: Table dropped

⇒ flashback table corona
to before drop
o/p: flashback complete

⇒ Purge table corona
o/p: Table purged

Auto commit:
Whatever the operations to be performed on database it will automatically stored in database.

2) Data Manipulation Language (DML):-
It is used to manipulate the object by performing insertion, updating and deletion.

1) INSERT
2) UPDATE
3) DELETE

1) INSERT :-
It is used to insert | create records in the table

Syntax: INSERT INTO TABLE_NAME VALUES ($V_1, V_2, V_3 \cdots$)

CUSTOMER

CID	CNAME	CNO	ADDRESS
NUMBER(2)	VARCHAR(10)	NUMBER(10)	VARCHAR(20)

INSERT INTO CUSTOMER VALUES (1, 'DINGA', 9876543210, 'BANGALORE');

CID	CNAME	CNO	ADDRESS
NUMBER(2)	VARCHAR(10)	NUMBER(10)	VARCHAR(20)
1	DINGA	9876543210	BANGALORE

INSERT INTO CUSTOMER VALUES (2, 'DINGI', 9876543211, 'MANGALORE');

CID	CNAME	CNO	ADDRESS
NUMBER(2)	VARCHAR(10)	NUMBER(10)	VARCHAR(20)
1	DINGA	9876543210	BANGALORE
2	DINGI	9876543211	MANGALORE

PRODUCT

PID	PNAME	PRICE	CID
NUMBER(2)	VARCHAR(10)	NUMBER(6,2)	NUMBER(3)

INSERT INTO PRODUCT VALUES (11, 'iphone', 1000, 2);

PID	PNAME	PRICE	CID
NUMBER(2)	VARCHAR(10)	NUMBER(6,2)	NUMBER(3)
11	'iphone'	10000	2

INSERT INTO PRODUCT VALUES (22, 'mac Book', 20000, 1);

PID	PNAME	PRICE	CID
NUMBER(2)	VARCHAR(10)	NUMBER(6,2)	NUMBER(3)
11	'iphone'	10000	2
22	mac Book	20000	1

2. UPDATE :-
It is used to modify an existing value.

Syntax :-
```
UPDATE Table_Name
SET colName . value , colName = value....
[WHERE stmt];
```

CID	CNAME	CNO	ADDRESS
NUMBER(2)	VARCHAR(10)	NUMBER(10)	VARCHAR(20)
1	ABHI	1234567890	BANGLORE
2	ABDUL	9876543210	MANGALORE

WAQT update the phone number of Abdul to 77888994
```
UPDATE CUSTOMER
SET CNO = 77888994
WHERE CNAME = 'ABDUL';
```

CID	CNAME	CNO	ADDRESS
NUMBER(2)	VARCHAR(10)	NUMBER(10)	VARCHAR(20)
1	ABHI	1234567890	BANGLORE
2	ABDUL	77888994	MANGALORE

WAQT change the address of the customer to Mysore whose CID is 1.

```
UPDATE CUSTOMER
 SET ADDRESS = 'MYSORE'
 WHERE CID = 1;
```

CID	CNAME	CNO	ADDRESS
NUMBER(2)	VARCHAR(10)	NUMBER(10)	VARCHAR(20)
1	ABHI	1234567890	MYSORE
2	ABDUL	77888994	MANGALORE

3) DELETE :-

It is used to remove a particular record from the table.

Syntax :

```
DELETE FROM table_Name
 [WHERE stmt];
```

CID	CNAME	CNO	ADDRESS
NUMBER(2)	VARCHAR(10)	NUMBER(10)	VARCHAR(20)
1	ABHI	1234567890	BANGALORE
2	ABDUL	1234567891	MANGALORE

WAQT Remove abdul from the list of customers.

```
DELETE FROM CUSTOMER
WHERE CNAME = 'ABDUL';
```

CID	CNAME	CNO	ADDRESS
NUMBER(2)	VARCHAR(10)	NUMBER(10)	VARCHAR(20)
1	ABHI	1234567890	BANGALORE

3) TRANSACTION CONTROL LANGUAGE (TCL):-

"It is used to control the transactions done on the database". the DML operations performed on the database are known as transactions such as insertion, updating and deletion.

we have 3 statements.

1) COMMIT
2) ROLL BACK
3) SAVEPOINT

1) COMMIT:

This statement is used to save the transaction into the database.

Syntax: COMMIT;

Eg: Query

```
CREATE T1
```

WORK PLACE

NAME	SAL

→ DATA BASE

NAME	SAL

```
INSERT INTO T1
VALUES ('A', 100);
COMMIT;
```

NAME	SAL
A	100

→

NAME	SAL
A	100

```
INSERT INTO T1
VALUES ('B', 200);
INSERT INTO T1
VALUES ('C', 300);
COMMIT;
```

NAME	SAL
A	100
B	200
C	300

→

Name	sal
A	100
B	200
C	300

```
UPDATE T1
SET SAL=1000
WHERE NAME='A'.
```

2) ROLLBACK :-

This statement is used to obtain only the saved data from the DB. it will bring you to the point where you have committed for the last time.

Syntax : ROLLBACK;

3) SAVEPOINT :-

This statement is used to mark the positions or reservation points (noting related to DB).

Syntax: SAVEPOINT save point_name;

Eg :-

Query

→ INSERT INTO T_1
VALUES ('A', 100);

→ SAVEPOINT S_1;

→ INSERT INTO T_1
VALUES ('B', 200);

→ INSERT INTO T_1
VALUES ('c', 300);

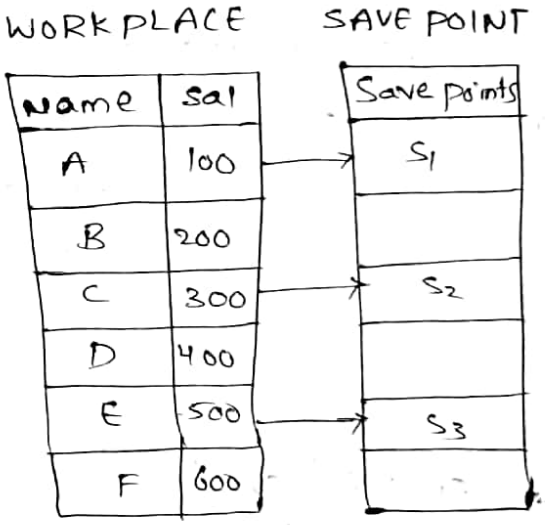
→ SAVEPOINT S_2;

→ INSERT INTO T_1
VALUES ('D', 400);

→ INSERT INTO T_1
VALUES ('E', 500);

→ SAVEPOINT S_3;

→ INSERT INTO T_1
VALUES ('F', 600);



4) DATA CONTROL LANGUAGE (DCL):-

This statement is used to control the flow of data between the users.

we have 2 statements:

1) GRANT
2) REVOKE

1) GRANT:

This statement is used to give permission to a user.

Syntax: Grant sql_statement
on Table_Name
To user_Name;

2) REVOKE:-

This statement is used to take back the permission from the user.

Syntax: REVOKE SQL_STATEMENT
ON TABLE_NAME

eg:

User 1 : SCOTT

EMP

ENAME	SAL
A	100
B	200

USER2 : HR

```
select *
From SCOTT.EMP;
```

HR cannot access SCOTT Emp table by getting error.

GRANT SELECT ON EMP TO HR;

EMP

ENAME	SAL
A	100
B	200

```
select *
from SCOTT.EMP;
```

HR can access SCOTT Emp table by granting Permission to HR.

REVOKE SELECT ON EMP FROM HR;

ENAME	SAL
A	100
B	200

```
select *
from SCOTT.Emp;
```

Take back permission by SCOTT from HR.

SQL > SHOW USER;

USER is "SCOTT"

SQL > CONNECT

Enter User_name : HR

Enter password : ***** (tiger)

Connected

SQL > SHOW USER

USER is "HR".

List of ATTRIBUTES:-

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:

It is an attribute which is used to identify the record uniquely from the table. we can have any number of key attributes in the table.

2. NON-KEY ATTRIBUTE:

All the attribute except key attribute is known as Non-key attribute.

3. PRIME KEY ATTRIBUTE:

Among the key attributes an attribute is choosen to be the main attribute for identify the record uniquely from the table. this is known as prime key attribute. we can have only one prime key attribute in the table.

4. NON PRIME KEY ATTRIBUTE:

All the key attribute except prime key attribute is known as non prime key attribute.

5. COMPOSITE KEY ATTRIBUTE:

It is a combination of two or more non-key attributes by which we can identify a record uniquely from the table is known as Composite key attribute.

6. SUPER KEY ATTRIBUTE:

It is a set which consist of all the key attributes in it.

7. FOREIGN KEY ATTRIBUTE:

It is an attribute which behaves as an attribute of another table to represent the relationship.

FUNCTIONAL DEPENDENCIES:-

A relation exist such that an attribute determines another attribute uniquely in the relation is known as functional dependency.

Eg: Let us consider relation R with two attributes x and y.

R: {x, y}

The functional dependency can be expressed as follows.

FD: x ---> y

x determines y/y is dependent on x.

Eg: emp: {eid, ename, sal}

eid → ename

eid → sal

FD: EID ---> {ENAME, SAL}

TYPES:

1. TOTAL FUNCTIONAL DEPENDENCY
2. PARTIAL FUNCTIONAL DEPENDENCY
3. TRANSITIVE FUNCTIONAL DEPENDENCY.

1. TOTAL FUNCTIONAL DEPENDENCY:

If all the attributes in a relation are dependent on single attribute we call it as total functional dependency.

(OR)

If an attribute in the relation determines all the other attributes we call it as total functional dependency.

Eg:

Let us consider a Relation R with four attributes A, B, C and D in which A is key attribute.

The total functional dependency can be expressed as follows.

$R \dashrightarrow \{A, B, C, D\}$

$A \rightarrow B$

$A \rightarrow C$

$A \rightarrow D$

Therefore $A \rightarrow (B, C, D)$

Therefore total functional dependency satisfied and NO redundany and anomalies.

2. PARTIAL FUNCTIONAL DEPENDENCY:-

Partial functional dependency occurs only if there is a composite key attribute.

Their exists a dependency such that an attribute is determined by another attribute is determined by another attribute which is part of Composite key attribute.

Eg: Let us Consider Relation $R \dashrightarrow \{A, B, C, D\}$ in which A and B are composite key attributes.

partial functional dependency expressed as follows.

$R \dashrightarrow \{A, B, C, D\}$

A and B $\dashrightarrow$ composite key A.

$(A, B) \dashrightarrow D$

$B \dashrightarrow C$

Therefore partial functional dependency.

3) TRANSITIVE FUNCTIONAL DEPENDENCY:-

A Relation exists such that an attribute is determined by a non key attribute which is inturn determined by the key attribute. this is known as Transitive functional dependency.

Eg:

Let us Consider relation R with four attributes A, B, C, D in which A is the key attribute.

Transitive functional dependency can be expressed as

R ----> {A, B, C, D}

A ----> B

A ----> D

D ----> C

A ----> C

Therefore transitive functional dependency.

Note:

Redundancy:-

The repeatation of unwanted data is known as Redundancy.

Anomaly:

The side effects that occur due to DML operations are referred as Anomaly.

Total functional dependency	partial functional dependency	Transitive functional dependency
NO Redundency	Redundency are present	Redundency are present
NO anomalies	Anomalies occur	Anomalies occur

What is Normalization?

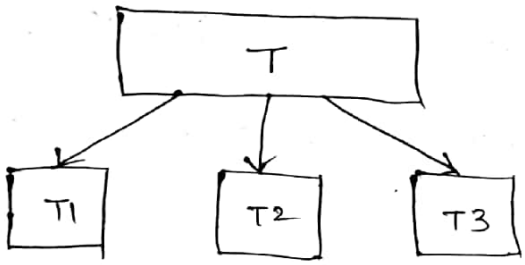
"It is the process of reducing a large table into smaller tables in order to remove redundancies (duplicate data) and anomalies (side effects for insertion, deletion) by identifying their functional dependencies is known as Normalization.

(or)

"The process of decomposing a large table into smaller table is known as Normalization.

(or)

"Reducing a table to its Normal form is known as Normalization."



What is Normal Form?

A table without redundancies and anomalies are said to be in Normal form.

Levels of Normal form:

1. First Normal Form (1NF)
2. Second Normal Form (2NF)
3. Third Normal Form (3NF)
4. Boyee-Codd Normal Form (BCNF)

Note:

If any table/entity is reduced to 3NF, then the table is said to be normalized.

1. First Normal Form (1NF):-

→ No duplicates records.

→ multivalued data should not be present.

QSPIDERS

OID	NAME	COURSE
1	A	JAVA
2	B	JAVA, SQL
3	C	MT, SQL
4	A	MT

→

OID	NAME	C1	C2	C3
1	A	JAVA	.	MT
2	B	JAVA	SQL	
3	C	.	SQL	MT

2. Second Normal Form (2NF):-

→ Table should be in 1NF.

→ Table should not have partial Functional Dependency.

EMPLOYEE - (EID, ENAME, SAL, DEPTNO, DNAME, LOC)

EID	ENAME	sal	DEPTNO	dname	Loc
1	A	100	10	D1	L1
2	B	120	20	D2	L2
3	C	320	10	D1	L1
4	D	251	10	D1	L1

Eid - ename, sal

Deptno - dname, Loc.

:- (Eid, deptno) → (Ename, sal, Dname, Loc) Composite key attribute results in PFD.

R_1 - (EID, ENAME, SAL)

R_2 - (DEPTNO, DNAME, Loc)

Eid	ename	Sal
1	A	100
2	B	120
3	C	320
4	D	251

Deptno	dname	Loc
10	D_1	L_1
20	D_2	L_2

3) Third Normal Form (3NF):-

→ Table should be in 2NF.

→ Table should not have Transitive Functional dependency.

Employee - (EID, Ename, sal, comm, pin code, state, Country).

EID → ENAME
SAL
COMM
PINCODE → STATE
COUNTRY

:- TRANSITIVE FUNCTIONAL DEPENDENCY.

R_1 - (eid, ename, comm)

R_2 - (pincode, state, country).

4) Boyee - Codd Normal form (BCNF):-

stronger 3NF and table should be in 3NF.

First Normal Form (1NF)	1NF	second Normal Form (2NF)	2NF	Third Normal Form (3NF)
• Single atomic value in each column		• Satisfy all 1NF conditions.		• satisfy all Conditions of 2NF.
• Each row have unique identifier.		• partial dependencies must be removed from the table.		• Transitive dependency of non-key attributes on key column must be removed.