SQL

- ER Diagram
- Constraints
- Data types
- > DQL
- selection and projection
- Different types of SQL clause
- sql function
  - single Row function
  - Multi Row function
- Subquery
  - co-related Subquery
  - Group function
  - Joins :- (1) Equi
    (2) self
    (3) Non-equi
    (4) Outer
- DDL :-
  - create
  - Alter
  - Rename
  - Drop
  - Truncate
- DML :-
  - Insert
  - update
  - Delete
- DTL :-
  - commit
  - Rollback
  - savepoint
- DCL :-
  - Grant
  - Revoke

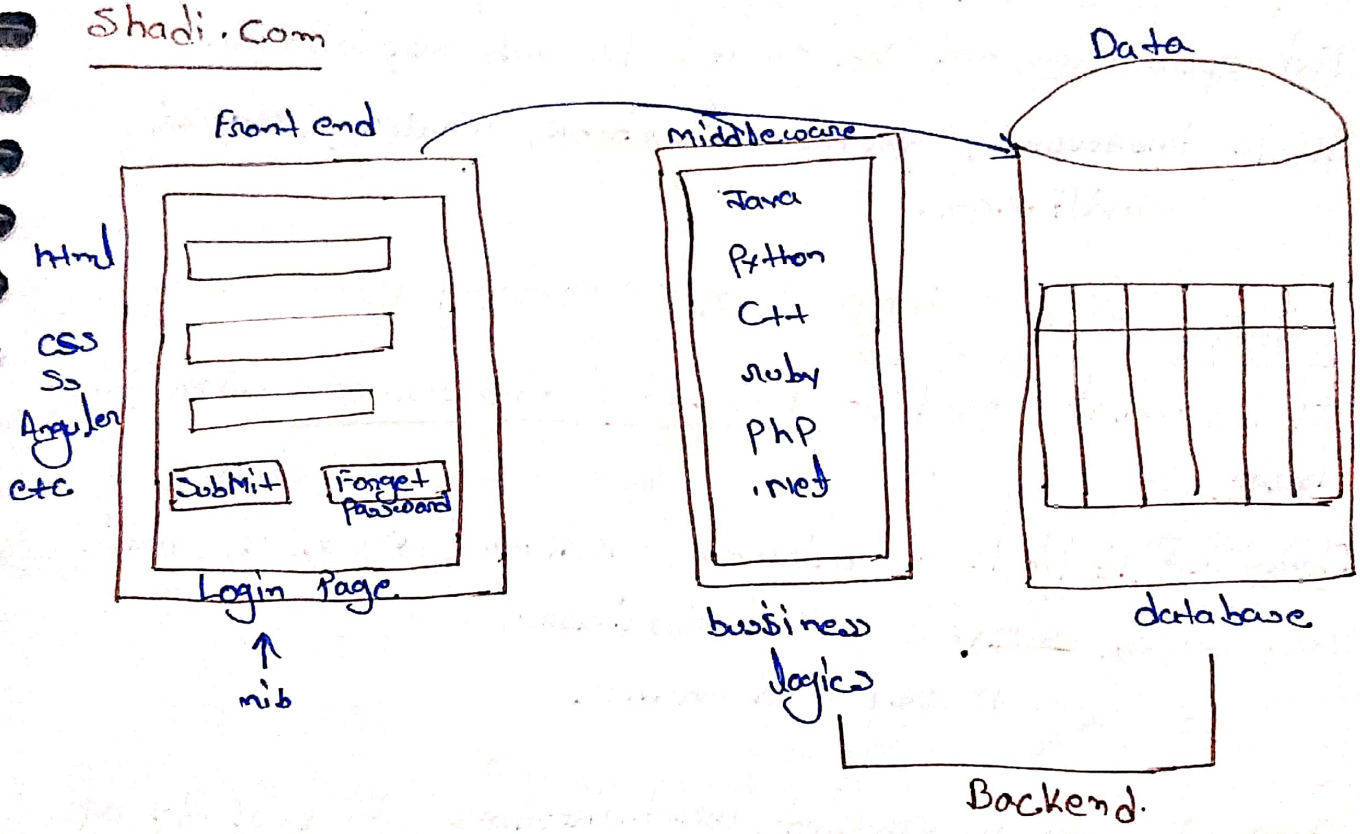
- Normalization
- Basic PL / SQL

SQL

⇒ Three types of Application :⇒

① Android or Mobile Application. / client Server
② DesKtop Application. / Stand alone
③ Web Application.. / ~~client server~~

Shadi.com



⇒ What is Application.
⇒ It is the collection of programs.

Types of Application :-

① Mobile Application :- Stand alone application.

The application that we are Installing in our Mobile.

like :- calculater, calender, notes, camera, Gallery, File manager, Phone, message etc.

② Desktop Application :=> The application that we Install in our laptop, or desktop is desktop application.

Like :- Notepad, GITAM CT, Pubg PC, Vscode, workbench, Eclipse.

③ Web Application :=>

That application we use on web is web application.

like :- Instagram, youtube, facebook, whatsApp, Naukri, shaadi.com.

SQL (STRUCTURED QUERY LANGUAGE)

- It is mainly used to Interact / communicate with data base.

Database :=> It is a container / medium / space in which we store data is called database.
in organised manner

Data :=> Data is raw fact, which discribes an property of an entity.

entity :- object

property :- Attribute

Raw fact :- the information that cannot be changed.

(1a) Car - object

property	value
Brand	Tata
Model	Harrios
price	28 lac
color	Black
Mileage	17
Power	150
Fuel type	Diesel

manufact

(Q.) write three objects and its property.

1) Book - object

Title - The notebook

Author - Nicholas sparks

Genre - Romance

No. of Pages - 272

Publisher - Grand Central Publishing

(3) student - object

Name - Johny

Age - 21

DOB - 20/06/2003

Education - B.Sc

skill - computer

Contactno - 1234567890

2) Phone - object

Brand - Motorola

Color - , Blue

Memory - Ram-4GB, ROM-64GB

Processor - snapdragon

Camera - 64 MP

Display - Gorilla Glass

Battery - 5000 mAh

CRUD operations

C - create / Insert

R - Read / Retrive / Access

U - update

D - Delete

DBMS

- DBMS mainly stands for Database Management System.
- It is a software which is mainly used to manage the database.
- To main features provide by DBMS

① Security — Username, Password

② Authorization → eg:- OTP

- To communicate with DBMS we use Query language.
- Data is store in form of files in DBMS

RDBMS (Relational Database Management System)

(1) In RDBMS we store in table form.

② SQL is used to communicate with RDBMS

③ Two features provide by RDBMS

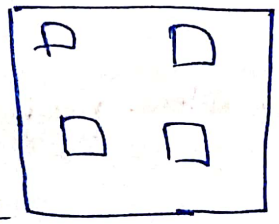
① Security

(ii) Authorization

Difference b/w DBMS and RDMS

DBMS

1. stands for Database management system.
2. stores Data in files.
3. uses Query language to communicate.
4. Provide two features
5. Eg:- 10 students



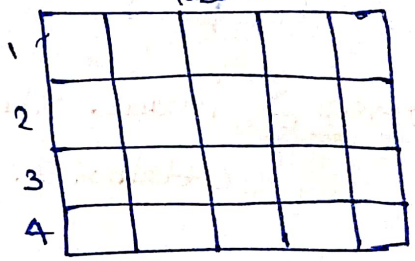
CRUD
- Creat
- Insert
- Read
- update
- Delete

- It's time consuming process

RDMS

1. stands for Relational Data Base management system.
2. stores Data in tables.
3. uses SQL to communicate.
4. Provide two features.
5. Eg:- 10 students

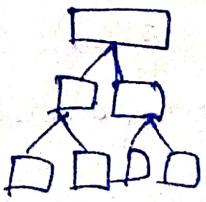
Tables



CRUD is easy to use. In RDBMS it is easy to use.

Types of DBMS

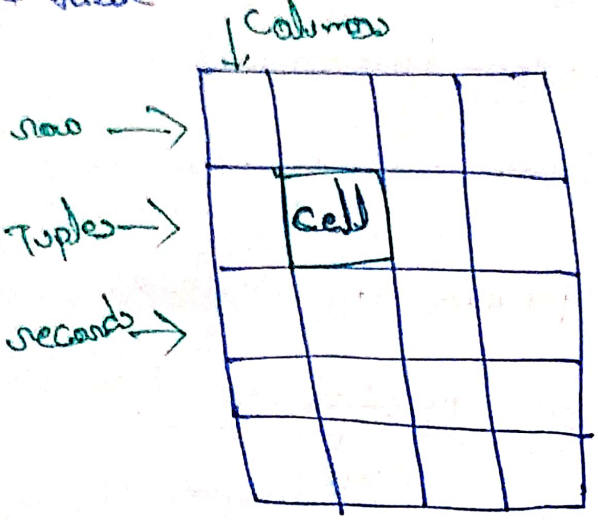
1. RDBMS
2. object Oriented DBMS → store data is ~~form~~ form of object.
3. Hierarchical Database (DBMS) →



4. Cloud → virtual

Relational Model ⇒ In 1980's Dr. Edgar. Frank proposed E.F Codd (Data scientist)

It states that the data stored in RDBMS in the form of table.



① **Columns** ⇒ Inside the columns we store properties or attributes.

② **Rows** ⇒ other name of rows are tuples and records, we store data in rows.

③ **Cells** ⇒ Intersection of row and columns cell is the smallest unit of a table.

Assignment → **History of SQL**

* SQL stands for structured Query Language which is a computer languages for storing, manipulating and retrieving data stored in a relational database.
* SQL was developed in the 1970 by IBM computer scientists.
* in 1974 Address as structured Query language.

Donald D Chamberlin and Raymond F. Boyce developed the Initially named.

⇒ EF Codd Rules :⇒ total Rules is (13).

DBMS + relational model → RDBMS

− + (4) + ——

Rule.

(1) First Rule ⇒

The data enter into the table must be single valued data.

Eg :⇒ student

sid	sname	Ph-no
1	Jaksh	123 ---
2	Katrina	324 ---, 234 ---
3	Mukesh	450 - --

(Not more than single valued data)

↓ Make single valued data

New column

Sid	Sname	Ph-no	Alternate Ph-no
1	Jaksh	123 ---	Null
2	Katrina	324 ---	234 ———
3	Mukesh	450 ---	Null
2	Katrina	234 ---	

New Row

→ Creating new column or Row we can make single valued data.

(2) 2nd Rule :⇒ In RDBMS we can Enter / Insert Everything including meta data

(string, Num, char, dates, images, document, binary data, audio, video)

(data about data)

Eg => Employee

id	ename	images
1	Raj	
2	Akash	
3	Prakash	

Property !

* audio
* video
* document

Metadata

Name	iname	Size	Type
Size	img001	20KB	JPG
Doc	img002	34MB	PNG
Type	img003	42KB	JPEG

compiler

=> 3rd Rule !=> In RDBMS we can insert the data into the multiple tables if necessary we have to create a relation between multiple tables. with help of key attributes.

key attributes

Student — it is mainly used to establish a relationship between two or more tables

Foreign key

Primary key

A primary key is mainly used to identified the record uniquely in the table

Trainer

sid	Sname	add	Tid
1	Aditya	xyz	1
2	Jonny	yzx	1
3	Nitish	zxy	1
1	Aditya	xyz	2
2	Jonny	yzx	4
3	Nitish	zxy	5

Tid	Tname	Sub
1	HARSHAL	SQL
2	Debasis	Java
3	Hemant	webTech
4	Nabin	Python
5	Anvit	Adv. Java
6	Chandrakanta	M.T

=> 4th Rule :=> The data enter into the table must be validated by two steps.

- ① by the data types
- ② by the Constraints

=> Data types :=> The types of data to be get inserted into a particular memory location.

① CHAR
② VARCHAR / VARCHAR 2
③ Date
④ Number
⑤ Larger object { CLOB – Character Large object.
BLOB – Binary Large object.

① CHAR :=>

(i) 'A' to 'Z', 'a' to 'z', '0' to '9'

special character :- '@', '#', '$', '_', '!', -

alpha numeric values :- password, email id, username, Bathcode, PanCard, voterCard

(ii) Syntax :=> it way of writing query in sql.

```
ColName CHAR(size)
```

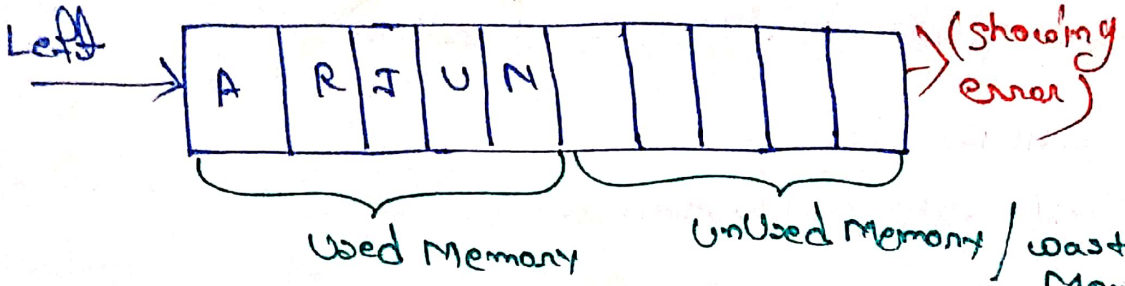
(iii) size is total no of character insert into particular column.

(iv) We have to pass character or string within a single quotes.

(v) We can enter Max 2000 character.

(vi) Fixed length Memory allocation system.

Eg :=> Name char(10) ⟶ Arjun



Eg => Phone-NO, gender, bloodgroup, Pincode, Adharcard no Pancard, ATM No.

here we can used fixed length Memory allocation.

(2) VARCHAR (variable Character) =>

(i) ↳ 'A' to Z, 'a' to 'z', '0' to '9'.
special characters :- '@', '#', '$', '_', '!', ...
alpha numeric values - 'password', emailid, username, batchcode, Pancard, voterid. ---

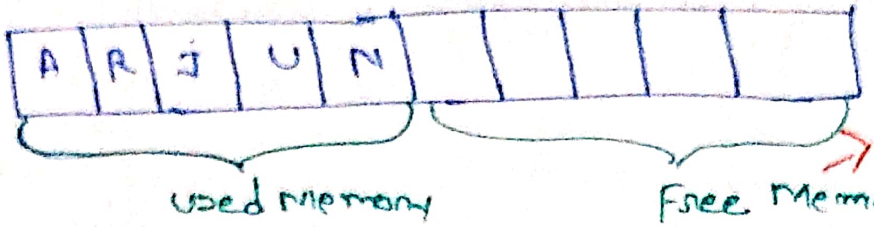
(ii) Syntax :=> ColName VARCHAR(SIZE)
↓
variable

(iii) within a single quotes.

(vi) Max 2000 characters.

(iv) Variable length memory allocation system.

Eg:- Name VARCHAR(10) → ARJUN



Eg:- Name, add, emailid, bankName ....

here we can use variable length memory allocation system.

Data type

=> Date :- It is mainly used to insert a date into a column in a specified formate. (oracle specific formate).

① 'DD - MON - YY' → Current Century formate
14 - FEB - 24

② 'DD - MON - YYYY' → All Century Formate
14 - FEB - 2024

Eg:- DOB,
DO Joining,
expiry date,
Manufacturing Date.
Date of Relase.

Number :=> It is mainly used to accept the numeric values.

Syntax :- `ColName Number(Precision [, scale])`

are used for optional / not mendatory.

(i) **Precision :-** It is mainly used to accept the **Integers** type of value.

- **Range :=>** 0 to 38

∵ Precision is mendatory to pass.

(ii) **Scale :=>** It is mainly used to accept the Decimal value.

- **Range :=>** -84 to 127

∴ It is optional to pass.

Syntax :=> `ColName Number(Precision [, scale])`

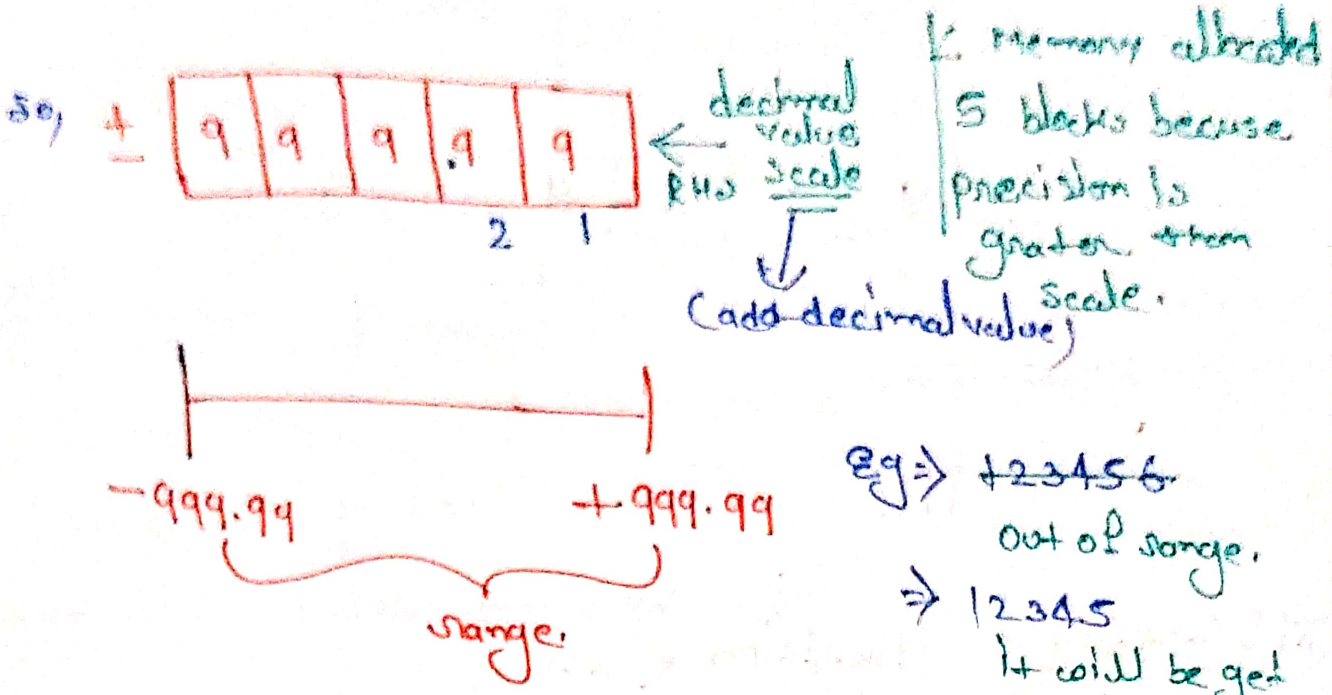
① **Case(1) :=>** **Precision > scale**

Eg => Number (5,2)

∵ 5 > 2

So, precision > Scale

→ always Preference greater value to create memory blocks.

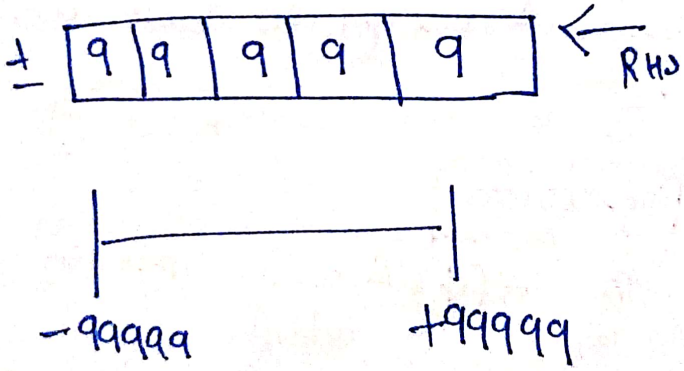


eg ⇒ ~~+23456~~
out of range.
⇒ 12345
It will be get Invert.
O/P ⇒ 123.45

⇒ if not Passing scale

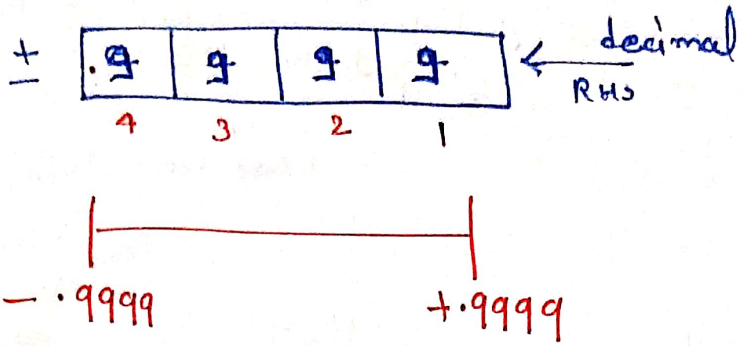
⇒ 100
O/P ⇒ 001.00
or 1.00

eg(ii) ⇒ Number (5) ~~5⇒0~~
5>0 ← scale



(ii) Case(2) ⇒ **Precision = Scale**

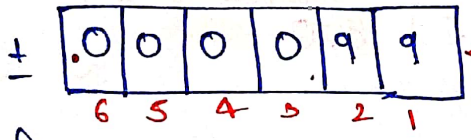
Eg. ⇒ Number (4,4) ← scale



(iii) Case(3) ⇒ **Precision < scale**

Eq ⇒ Number (2, 6)

∴ 2<6 ← always prefer greater value to create Memory blocks.



(a) Given preference or According to scale we have allocated Memory.

(b) **Scale − Precision**

6 − 2 = (4) According this No Zero added

(c) Zero will be added from LHS. Eg.

(d) ~~Accordi~~ According to scale decimal point added.

VARCHAR 2 =>

(i) → 'A' to Z, 'a' to 'z', '0' to '9'

special character :- '@', '#', '$', '_', '!', ...

alpha numeric values :- 'password', email id, 'username', batch code, pancard, voter id, ...

(ii) Syntax :- ColName VARCHAR2 (SIZE)

(iii) write in a single quotes.

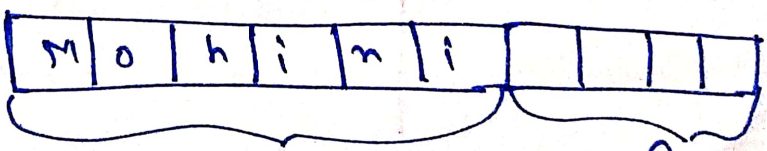
(iv) Max 4000 characters.

(v) VARCHAR internally consider as VARCHAR2.

Eg => varchar (1000) ⌐
varchar (2000) ⌐ } Pass
varchar (3000) ⌐
↓
varchar2 (3000)

(vi) variable length memory allocation system.

Eg => Name VARCHAR2(10) → Mohini



used Memory | free Memory.

Resend back to the Memory for the further use.

Eg :- Name
add
Email-id
bank Name
etc.
} here we can use variable length Memory allocations.

Assig :> Write a diff. b/w , CHAR and VARCHAR and VARCHAR 2.

CHAR

① Syntax :-
ColName CHAR(size)

② Insert Size Max 2000

③ Fixed length Memory allocation

④ In char if Memory is not used then it called unused Memory or waste Memory.

⑤ In char unused Memory showing Error and also its wastage of memory.

VARCHAR

① Syntax :-
ColName VARCHAR(size)

① Size Max 2000

② variable length Memory allocation

④ In varchar Memory is not used then. it is called free Memory.

⑤ In varchar free Memory going back the Memory for the further Use.

VARCHAR2

① Syntax
ColName VARCHAR2 (size)

② size Max 4000

③ variable length Memory allocation

④ here also Memory is not used is called free Memory.

⑤ here also free Memory going back to the Memory for the further use. and varchar is Internally consider as Varchar 2.

=> Large object !=>

Type => (1) character large object
(2) Binary large object.

=> It is mainly used to insert large amount of values.

(1) character large object !=> It is mainly used to accept the large amount of characters.

(i) Syntax !=> ColName CLOB

(ii) It can store character upto 4GB.

Eg !=> Subtitle,
Book,
script,
etc.

(2) Binary large object !=> It is mainly used to store audio, video, documents, images in form of binary values.

(ii) Syntax !=> ColName BLOB

(iii) Size 4GB.

⇒ If we only using data type.

Eg ⇒ Employee

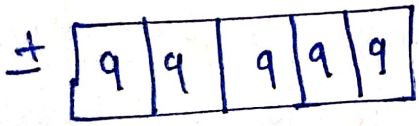
Empid	Name	SAL	Phone-no
Number(5)	varchar(20)	number(5)	number(10)
66284	Ashok	10000	9999000099
66285	Akash	11000	1234567899
420	Binod	3000	42512345
-12345	Rali	-2000	-1234567899

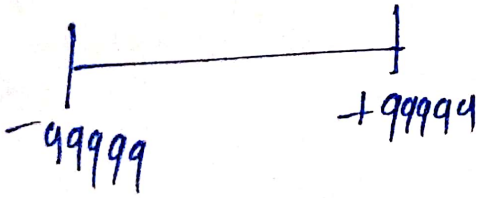
∴ ⇒ here two drawbacks ⇒ (i) Less then specified size

(ii) Negative value also entered.

this drawback we can solve with help of constrains.

number (5)





NOTE

* datatype mendonty to pass where as constraint
* Is ~~optional~~ optional.

Constraints :=> Constraints are the Rules and the regulation which are necessary in order to Insert a record into a column.

=> Types of constraints :=>

① Unique Constraints => Unique is a constraint which is used to insert a unique records in the column.

Syntax :=> UNIQUE

Eg=> Create table EMP (emid number (5) UNIQUE)
(datatype → , constrain →)

Eg=> id, phon_no, adhar_no, pan_no, VoterId, Account_no.

② Not Null Constraints :=> If you are passing a not Null Constraints to a column then you cannot keep a cell as Empty or blank.

NULL
□ -> empty/blank

Eg=> salary, phone_no, id, adhar_no, name, Date of Joining, etc.

Syntax :=> NOT NULL

=> Create table EMP (emid, number (5) NOTNULLE)

NOTE :=>
SQL is non case sensitive.
but data is case sensitive.

(3) Check constraints :=> It is constraint used mainly along with a condition. if the data to be entered is satisfying that condition then only that data will be get inserted into the column. Otherwise the data is rejected.

Syntax :=> Check (condition)

phone_no number(10) check (phone>0) Check (length (phon_no) = 10)

- number(10) → datatype
- check → constraints
- phone>0 → condition given by you.
- Check (length (phon_no) = 10) → condition given by you.

eg=>

table

phone_no
1234567890

→ this data entered because it's satisfied all condition.

— 1234567890 this data can not enter table because it's negative value, it's this is less than 0

1234678 this data length is not equal 10.

④ Primary Key Constraint :=> Primary key constraint mainly used to identify the records uniquely in the table.

Ex:=> Employee

{eid, name, phon-no, salary, JOD, department, address, DOI, DOB, Manager, email, adhar, pan, ~~three~~ Acc.no, voter_id, id.... }

∠ we make them primary key. but we always make eid as primary key. (PK)

=> Characteristics of Primary Key :=>

1. Primary key will accept unique value.
2. Primary key will follow unique constraints and we cannot keep Null value.
3. primary key will not accept any Null value.
4. Primary key follows NOT NULL constraints.
5. We have only one primery key in a table.

⑤ Foreign Key Constraint :=> It is constraint mainly used to establish a relation between two or more tables.

∠ consider that we create two tables:-

1st Table :=> Student

id (Primery Key)	name	phone-no	email
1	Ankit	1234567890	ankit@gmail.com
2	bharat	2222222222	bharat@gmail.com
3	Ram	1111111111	Ram@gmail.com
4	Dhoni	7777777777	dhol@gmail.com

2nd table :- Trainer

Primary Key

Tid	Tname	Course
T1	harshal	SQL
T2	Anvit	Ad. Java
T3	Debasis	Java
T4	Nabim	Python
T5	Chandakant	M.T
T6	Hemant	web

∴ Let's make relationship ~~between~~ between two columns with help of key attributes (PK, FK).

⇒ Student

id (Primary Key)	name	phone-no	email	Tid (Foreign Key)
1	Ankit	1234567890	ankit@gmail.co	T1
2	bharat	2222222222	bharat@ " "	T6
3	Ram	1111111111	Ram@ " "	T4
4	Dhoni	7777777777	dhal@ " "	T5
1	Ankit	1234567890	ankit@ " "	T8

New Row added Because EF Codd 1st Rule.

** Common column between two tables is relation of the table.

(e) Let's make relationship between three table. Student, Trainer, Devices.

⇒ Devices

Primary Key

Did	Dname	Price
D1	MAC	100000
D2	HP	50000

⇒ student

PK

id	Name	Phone-no	@mail id @gmail	Tid (FK)	Did (FK)
1	Ankit	1234567890	ankit@gmail.com	T1	D2
2	bharat	2222222222	bharat@gmail.com	T6	D1
3	Ram	11 111111 11	Ram@gmail.com	T4	NULL
4	Dhoni	7777777777	dhoni@gmail.com	T5	D1
1	AnKH	12345677890	amkit@gmail.com	T3	D2

⇒ Characteristics of Foreign Key :-

① Foreign key will accept duplicate values.
② The Foreign key will not follow unique constraints.
③ Foreign key will accept NULL values.
④ Foreign will not follow NULL constraints.
⑤ can A can have Multiple Foreign Key in a table.
⑥ Foreign Key is also called as Referential ~~Integrity~~ Integrity

Integrity Constraints.

⑦ To become Foreign key it should be a primary key in it's own table.

Statements :=> A statment is any command or instruction given to a relational database management

① Data Quary Language (DQL) system using SQL.

② Data Defination Language (DDL)

- ① Creat :- it is used to create a new table in the database.
- ② Rename :=> used rename the table.
- ③ alter :=> mainly used to alter the structure of the database. this change could by either to modify the characteristics of an existing attribute or probably add new attributes.
- ④ Truncate :=> it is used to delete all the rows from the table and free space containing the table.
- ⑤ Drop :=> it is used to delete both the structure and record stored in table.

③ Data Manipulation Language (DML) => Transaction

- ① Insert statment is SQL query used to insert data into the row of a table
- ② update :=> This command used to update or modify the value of column in the table.
- ③ delete :=> used to remove one or more row from table.

=> This are not automated to save. to save them we used TCL.

④ Transaction control Language (TCL)

- ① Commit :=> To save the Records or save all the transaction to the database;
- ② Roll Back :=> command is used to undo transaction that have not already been saved to the database.
- ③ Savepoint :=> it is used to roll the transaction back to a certain point without rolling back the entire transaction;

⑤ Data control Language (DCL)

- ① Grant :=> it is used to give user access privileges to a database
- ② Revoke :=> it is used to take back permissions from the user

Data Query Language :=>

① DQL mainly stand for Data Query Language.

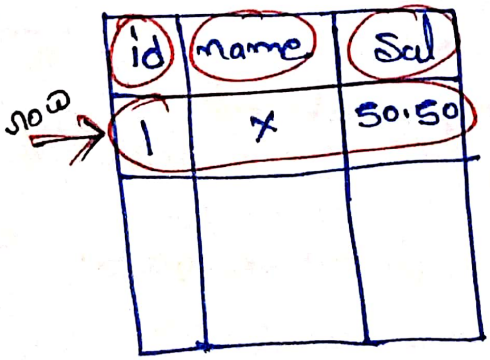
=> Data Query Language is mainly to retrive or fatch the data from the tables.

(i) select statment :=> select statment is mainly used to obtain the particuler records and to display it on the output screen or consol.

(ii) Projection :=> projection is mainly used to retrive the data by selecting ~~out~~ only the columns.

Eg :=>

(iii) selection :=> selection is mainly is used retrive the data by selecting rows as well as columns.



selecting row automatically selecting rows.

(iv) Joins :- It is mainly used retrive the data from the multiple tables.

=> Projection :- Projection is mainly used to retrive the data by selecting only the columns.

Syntax :-

optional / not mandatory

```
select */[DISTINCT] CollumnName/Expression [Alias]
From Tablename ;
```

all records

Eg :- Emp

empno	ename	sal
1	A	10
2	B	20
3	C	30
4	D	40

Write a query to display all the records of employee.

all the data, all the records

```
select *
From emp ;
```

the way of writing query

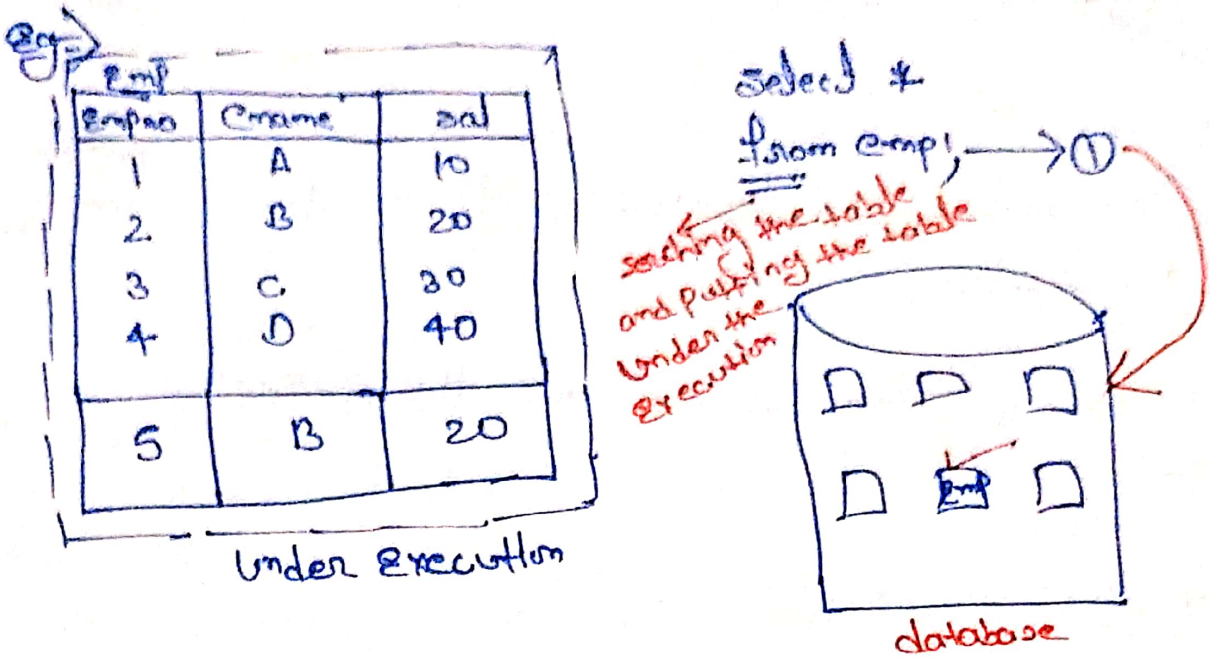
top to bottom

top to bottom

way of execution query is different

display all the data, all the records.

This clause searching the table and putting the table under the execution.



Execution Process !=>

```
Select *
From emp;
```

① → this clause execute 1st.

② From clause will go into the ~~data~~ database searching for same table name and put the table under Execution. so we can pass table.

③ After the From clause select clause will execute.

④ Inside a select clause we can pass three arguments

(i) ~~Asterisk~~ *

(ii) Column name

(iii) Expression

⑤ Job of the select clause is to go to the table under execution and display ColumnName

on the Output screen.

* :=>
- * is mainly used to display all the details from the table.

; :=> this means end of the query.

query :=>

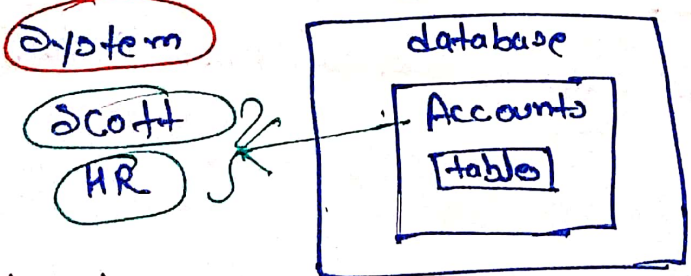
① query for clear screen :=>

SQL> (i) Clear screen
(ii) cl scr. } two types of query we write to clear the screen.

② query for showing user :=>

SQL> Show User; → this query showing user : table.

NOTE :=> he is Admin, gives permission to any accounts.



③ Changing password query :=>

=> Alter user UserName identified by TIGER

Account UNLOCK

:=> Alter user UserName Account UNLOCK identified by TIGER// (Password)

Query 1 → showing user query

SQL> Show user.

USER IS "SYSTEM"

SQL> CREATE USER MOHINI IDENTIFIED BY TIGER;

User created. ← Creating new user query.

SQL> CONN MOHINI → connect to new user query

Enter Password : ~~tiger~~ TIGER

ERROR!

ORA-01045: User MOHINI lacks CREATE SESSION privilege;
logon denied

Warning: You are no longer connect to ORACLE.

SQL> CONN SYSTEM

Enter Password : TIGER

Connected.

SQL> GRANT CREATE SESSION → give the permission to create session by the system to the new user which are created.

2 TO MOHINI;

Grant succeeded.

SQL> CONN MOHINI

Enter password : ***** /*(TIGER)*/

Connected

SQL> CREATE TABLE DEMO → Create table query.

2 (

3 NAME VARCHAR (20)

4);

Error :→ insufficient Privileges.

SQL> EDIT → using this to handle the text file.

wrote file afiedt.buf

using this open one text file we are not give the ; on this text file.

```
1 CREATE TABLE DEMO
2 (
3 NAME VARCHAR(20)
4 * )
SQL> /
```

Error :⇒ Insufficient privileges

```
SQL> CONN SYSTEM
Enter password : TIGER
connected.
SQL> CONN MOHINI
Enter Password : TIGER
```

SQL> GRANT CREATE TABLE → give the permission to create table by the system to new user which are we created.

```
  2 TO MOHINI
  3 ;
Grant Succeeded.
SQL> CONN MOHINI
Enter Password : TIGER
connected.
SQL> CONN MOHINI
Enter Password : *****
connected.
```

```
SQL> CREATE TABLE DEMO
  2 (
  3 NAME VARCHAR (20) oooo
  4 );
```

Error: Missing right ~~par~~ parenthesis.

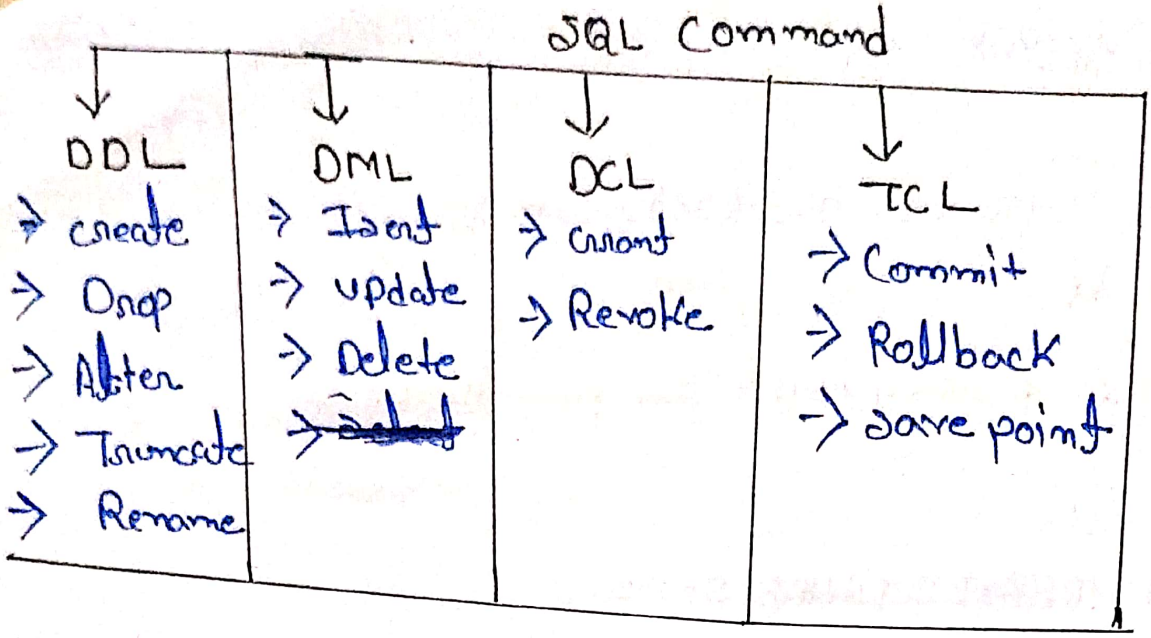
```
SQL> ED
  1 CREATE TABLE DEMO
  2 (
  3 NAME VARCAAR (20) OOOO
  4 );
SQL> /
```

Error! Missing right parenthesis

```
SQL> CONN SYSTEM
Enter password : Tiger
SQL> GRANT ALL PRIVILGES TO MOHINI;
Grant Succeeded.
SQL> CONN MOHINI
Enter Password : Tiger
SQL> CREATE TABLE DEMO
  2 (
  3 NAME VARCHAR(20)
  4 );
```

query for give all privilges to user by the system.

Table created.

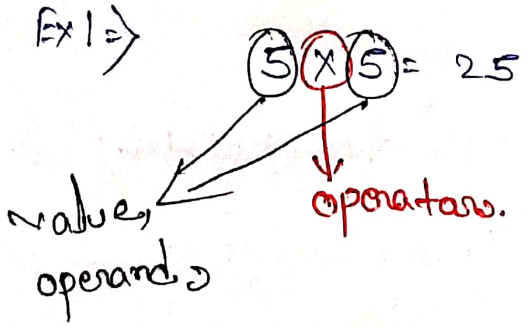


Expression :=> Expression are nothing but the statment that gives you some result.

operators :=> operators are nothing but the Mathmetical symbols.

operands :=> operands are nothing but the values.

Ex :=>



Syntax :=>

select */[DISTINCT] columnName / Expression [Alias];

Syntax :=> select Expression
From Tablename;

Distinct :=> Distinct is mainly used to avoid repeated values from the result table.

Characteristics of Distinct :=>

① Distinct is always pass as a 1st argument inside the select clause.

② If we have multiple columns in Distinct clause then it will remove combination of values from the ~~Coloumn~~ column in which records are repeating.

Syntax :=> select Distinct Branch from student;

@@ TO the unique Branch of students.

↳ tablename

Ex :=>

Student

sid	sname	branch	per.
1	Akash	ME	78
2	devi	CSE	75
3	Lakhman	ECE	82
4	Binod	civil	60
5	Katrina	ME	85
6	Abdul	CSE	75

this will get 1st preference.

database

Student

under execution

o/p

Branch
ME
CSE
ECE
civil

∴ because b. of distinct repeated data not be in output table. but in database original table does not change.

∵ In repeated value, 1st value get preference get to enter in output table.

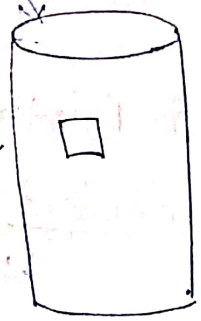
(Q2) => WAQTD Distinct of Branch and Percentage of a Student.

=> Select Distinct Branch, Percentage
From (Student);
Tablename

NOTE :=> (1) Preference get 1st column always.
(2) Both Column value is matched then we call it repeated value, otherwise not.

EX=> Student

Sid	Sname	Branch	Per
1	A	ME	78
2	B	CSE	75
3	C	ECE	82
4	D	Civil	60
5	E	ME	85
6	F	CSE	75



Under the Execution.

=> O/P

Branch	Per
ME	78
CSE	75
ECE	82
Civil	60
ME	85

(Q3) :=> WAQTD details of employee along with annual salary of all the Employee.

=> Syntax :=>

① select *, sal*12 ✗
from emp;

This syntax is wrong because if you use * (astrik) you not allow to use any column. name or you can't pass column with *.

=> Select emp.* (sal*12)
from Emp;

→ using . (dot) operator you can pass the column.

=> Alias :=>

① Alias is nothing but an alternative name given to a column temporarily.

② we can give an alias with or without a keyword 'AS'.

1st way :=>

using As keyword & double quotes (" "):-

Select SAL*12 AS "Anual Salay"
From Emp;

→ this space also count as one character.

2nd way :=>

Pass alias without " " :-

select SAL*12 As Annual_Salary/or AnnualSalary
From Emp;

3rd way :=> without using As Keyword and " " (double quotes):-

```
Select SAL * 12 Annual_Salary
From emp;
```

4th way :=> without using AS Keyword but by using double quotes (" ");

```
Select SAL * 12 " Annual Salary"
From emp;
```

Query :=> (Assi-1)

(Q1) :=> WRITE a query to display all the details from the emp table.

=>
```
Select *
From emp;
```

(Q2) :=> WAQTD Name and salary Given to all the Employees

=>
```
Select ename , SAL
From emp;
```

(Q3) :=> WAQTD Name of all the employees.

=>
```
Select ename
From emp;
```

(Q4) :=> WAQTD Name and comission given to all the Employees.

=>
```
Select ename, Comm
From emp;
```

(Q5) => WAQTD employee ID and department number of all the employee in the emp table.

=> select empno, Deptno
From Emp;

(Q6) => WAQTD ename and Hiredate of all the employees.

=> select ename, hiredate
From emp;

(Q7) => WAQTD name and designation of All the Employees.

=> Select ename, d.JOB
From emp;

(Q8) => WAQTD name, Job and salary Given to all the Employees.

=> Select ename, JOB, SAL
From emp;

(Q9) => WAQTD Dname present in department table.

=> Select Dname
From DEPT;

(Q10) => WAQTD Dname and Location present in DEPT table.

=> select Dename, LOC
Forom DEPT;

Queries (Assi-2)

(Q1):- WQTD name of the employee along with their Annual salary.

=> Select ename, sal * 12
From Emp;

20/08/2022

=> Selection :- It is used retrive the data by selecting rows as well as columns.

Syntax :=>

```
Select ColName / Expression   → 3
From TableName                → 1
Where <filter condition>;     → 2
```

Order of Execution.

Order of Execution :=>

① 1st From Clause will execute, it will put the table under execution and Execute Row by Row.

② Then Where Clause will execute Row by Row.

③ Then Select clause will execute Row by Row and display selecting Row on output screen.

Ex (Q.1) => WQTD details of an Employee who are working in department 20.

=>
```
Select *          → 3
From Emp          → 1
where deptno = 20; → 2
```

Note

∴ Select clause Execute at last.

=> EMP

Empno	ename	Job	Sal	Deptno
1	Rohit	Clark	3500	10
2	Debi	MANAGER	5000	20
3	Abhishek	SD	2000	30
4	Nitin	WD	3000	20
5	Sourav	WD	4000	10
6	SONU	MANAGER	1000	30
7	Akash	Clark	3500	20

∴ for From clause, this table going to under Execution.
: after that where checked the condition
where deptno = 20
10 = 20 ✗
20 = 20 ✓
30 = 20 ✗
20 = 20 ✓
10 = 20 ✗
30 = 20 ✗
20 = 20 ✓

this three are match the condition so,

O/P => EMP

EmpNo	Ename	Job	Sal	Deptno
2	Devi	MANAGER	5000	20
4	Nitin	WB	3000	20
7	Akash	Clark	3500	20

(Q2) => WQTD the details of an employee who are working as a manager.

O/P =>

=> Select *
From Emp
where Job = 'MANAGER';

Empno	ename	Job	sal	De
2	Debi	MANAGER		
5	Sourav	MANAGER		

Empno	Ename	Job	Sal
2	Debi	MANAGER	5000
5	Sourav	MANAGER	4000

Note :=>
Only Alias we use " ".

(Q3) => WAQTD the details of Sonu.

=> Select *
From Emp
where ename = 'SONU';

(Q4) => WAQTD salary of an employee who are earning more then 2000;

=> Select SAL.
From Emp
where sal > 2000;

(Q5) => WAQTD hiredate of an employee who are hired before 83.

=> Select hiredate
From Emp
where hiredate < '01-JAN-83';

21/08/2024 => **OPERATORS**

=> Operators :=> It is nothing but Mathmetical Symbol.

1. Airthmetic operator (+, -, *, /)
2. Concatination operator (||)
3. Comparison operator (=, !=)
4. Relational operator (>, <, >=, <=)
5. Logical operator (AND, OR, NOT)

(6) Special operators

(i) → IN operator
(ii) → NOT IN operator
(iii) → IS operator
(iv) → IS NOT operator
(v) → Between
(vi) → NOT Between
(vii) → LIKE
(viii) → NOT LIKE

(7) Subquery operators

(i) → ALL
(ii) → EXISTS
(iii) → ANY
(iv) → NOT EXISTS

=> Concatination operator (||) :=>

(1) WAQTD the string 'harshal' 'Rule' on otp screen?

=> Select 'harshal' || 'Rule'
From DUAL;

o/p => harshalRule

:. Dummy Table called DUAL; is have too 1 row, 1 column

Concatination handle string ColumnName value.

(2) WAQTD string like 'Mr smith' for each Employee.

=> Select 'Mr' || ENAME
From Emp;

(3) WAQTD string like 'Hi I am Scott and my salary is 1000' for each employee.

=> Select 'Hi I am' || ENAME, 'AND My salary is' ||
From Emp;

(4) WAQTD string like 'Hi my name is SMITH I am Earning 5000 salary I am a Manager and my Employee id is 420'.

⇒ Select 'Hi My Name is' || ENAME, 'I am Earning' || SAL,
'I am a' || Job, 'AND My Employee ID is' || Empno
From EMP;

⇒ Logical operator :⇒ In AND operator, if all condition are satisfied then only will get an output.

(i) AND :-

i/p	i/p	o/p
1	1	1
0	1	0
1	0	0
0	0	0

(ii) OR operator :⇒ if any one condition is satisfied then we will get output.

i/p	i/p	o/p
1	1	1
0	1	1
1	0	1
0	0	0

(iii) NOT operator :-

i/p	o/p
1	0
0	1

True → False

False → True.

Query Using 'AND' & 'OR' operator :=>

(Q.1) => WAQTD details of an Employee if an Employee is working in department 20 and his salary is > 2500.

=> Select *
From Emp
where deptno = 20 AND SAL > 2500;

(Q2) => WAQTD Employee name, salary, deptno, of an Employee who are hierd after 83 and salary is < 4000;

=> Select Ename, SAL, DEPTNO
From Emp
where Hiredate > '31-DEC-83' AND SAL < 4000;

(Q3) => WAQTD Name and deptno of an Employee who are working in dept 10, 20.

=> Select Ename, Deptno
From Emp
where DEPTNO = 10 OR DEPTNO = 20;

=> dep = 10,20 ✗
we can compain one value at a time, so that's way not possible.

NOTE :=>
* One Column then use OR operator.
* Multiple Column then use AND operator.

(Q4) => WAQTD salary of an Employee who are Earning salary > 2000 and salary < 3000.

=> Select SAL
From Emp
where SAL > 2000 AND SAL < 3000;

Relation operator are there that way we use AND Logical operator.

⇒ Special operators !⇒

① **IN operator !⇒** In operator is mainly used to handle the multiple values on the Right hand side (RHS).

:⇒ If any one values is satisfying the condition then In operator will give you that records.

Syntax !⇒ ColumnName IN (V1, V2, V3, - - Vn)

* If possible multiple value then it must to Passing ( ).

* Not need ( ) if you passing single value.

(Q) ⇒ WAQTD Name and Job of employee who are working as Manager, Analyst and Salesmen.

⇒ Select Ename, Job
From Emp
where Job IN ('Manager', 'Analyst', 'Salesmen');

∴ working like =, OR operator

② **NOT IN operator !⇒** NOT IN operator is mainly used to handle the multiple values on the RHS. ~~Instead~~ Instead of selecting the values it will Reject those values. Syntax !⇒ ColumnName NOT IN (V1, V2, V3 - -

(d) => WAQTD the details of the Employee who are NOT working in deptno 10, 20.

=> Select *
From Emp
where deptno NOT IN (10,20);

(e) WAQTD name and Job of employee expect the Manager.

=> Select Ename, Job
From Emp
where JOB NOT IN ('MANAGER');

(3) Is operator :=> Is operator is mainly use to compair the NULL values.

Syntax :=> Column Name Is NULL;

(a) => WAQTD details of the Employee who are NOT Earning Comm.

=> select *
From Emp
where COMM Is NULL;

(b) => WAQTD details of the Employee who dont have any manager.

=> Select *
From EMP
where MGR Is NULL;

④ **IS NOT operator :=>** Is Not operator is mainly is used to compain not Null value.

Syntax :=> ColName IS NOT NULL;

Q1 => WAQTD Name of an Employee who are earning any commision.

=> Select Ename
From Emp
where COMM is Not Null;

⑤ **Between operator :=>** Between operater is mainly used to obtain the value between the range of lower value and higher value.

- It will accept the ranges of value as well.

Eg :=> 2000 ——— 3000
accept 2000 and 3000 also.

Syntax :=> Colname Between LowerValue AND Higher value;

Case ① =>

Q WAQTD the employee name and salary of an Employee, is the employee is earning in the range of 2000 to 3000;

=> Select Ename, Sal
From Emp
where Sal Between 2000 and 3000;

Case (2) :=>

(Q) WAQTD Name and salary of an Employee if an Employee is earning greater than 2000 less then 3000.

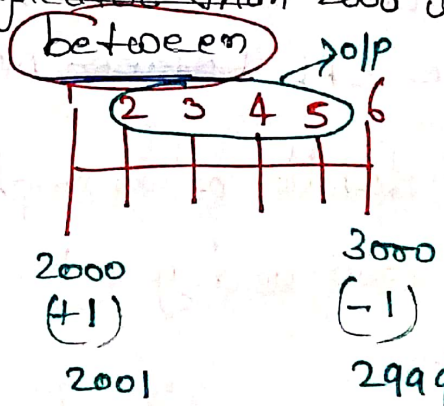
=> select ename, SAL
from Emp
where SAL Between 2001 AND 2999;

Ref
2000 — 3000
2001 2999

Case (3) :=>

(Q) WAQTD Name and salary of an Employee if an Employee is earning ~~greater than~~ 2000 ~~less than~~ And 3000.

=> select enam, sal
from Emp
where sal Between 2001
AND 2999;



(Q2) => WAQTD details of an employee who are hired in 87 with the help of Between operation.

=> select *
from Emp
where Hiredate between '01-JAN-87' AND
'31-DEC-87';

(6) NOT Between operator :=>

(i) It is also similar to the between operator.

(ii) Instead of selecting the value. It will reject the values.

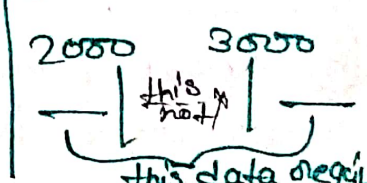
Syntax :=>

ColName NOT BETWEEN Lower_value AND Higher_value;

(Q1)=> WAQTD details of an employee who are not earning in the range of 2000 and 3000.

=> Select *
From EMP
where SAL ~~Between~~ NOT Between 2000 AND 3000;

Ref



(Q2)=> WAQTD details of an employee who are earning less then 2000 and greater then 3000.

=> select *
From EMP
where SAL NOT Between 2000 AND 3000;

(Q3)=> WAQTD details of an employee who are hired ~~after 85;~~ before 82 and hired after 85;

=> select *
From EMP
where Hiredate NOT Between '01-JAN-82' AND '31-DEC-85';

LIKE operator :=> It is mainly used to match the patterns.

(i) → % (Percentile) operator :=> This operator will accept Multiple value or it will accept Zero (0) values.

(ii) → _ (Underscore) operator :=> It will accept only one value.

Syntax => colName LIKE 'Pattern';

(Q1) => WAQTD names of an Employee whose name start with 'A'. => 'A%' follow this Pattern

=> select ename
From EMP
where ENAME LIKE 'A%';

(Q2) => WAQTD names of an Employee whose name End with 'T'; ('%T')

=> select ename
From EMP
where Ename LIKE '%T';

(Q3) WAQTD details of an Employee whose Name consist of 'AA'. → '%A%A%'

=> Select Ename
From EMP
Where Ename LIKE '%A%A%';

Ex1=>

'%A%A%'

ADAMS

Zero Value

HARSHAL

% A % A %

(8) => NOT LIKE Operator :=> It is similar to like operator. Instead of selecting it will reject the Pattern.

syntax :=> ColName NOT LIKE 'Pattern';

(Q1) => WAQTD Details of an Employee if Employee name is not starting with 'S';

=> Select *
From EMP
where Ename NOT LIKE 'S%';

=> Query using LIKE operator :=>

(Q1) => WAQTD details of an Employee whose name consist of ~~Consec~~ consecutive 'A'/'L'.

=> select Ename
From EMP
where Ename LIKE '%AA%';

(Q2) WAQTD Name and Job of an Employee whose Job consist of 'MAN' as three characters.

=> select ename, JOB
From EMP
where JOB LIKE 'MAN%';

Patterns

(Q3) ⇒ WAQTD Name and JOB of an Employee whose Job consist of 'MAN' as last three characters.

⇒ Select Ename, JOB
From EMP
where JOB LIKE '%MAN';

(Q4) ⇒ WAQTD Name of Employee who are hired in Feb.

⇒ Select Ename
From EMP
where Hiredate LIKE '%FEB%';

(Q5) ⇒ WAQTD details of an Employee whose name consist of 'A' as second character.

⇒ Select *
From EMP
where EName LIKE '_A%';

(Q6) ⇒ WAQTD ename of Employee whose name consist of 'E' as 2nd last character.

⇒ Select Ename
From EMP
where Ename LIKE '%A_';

(Q7) ⇒ WAQTD details of an Employee if an Employee is earning 4 digits of salary.

⇒ Select *
From EMP
where SAL LIKE '____';

FUNCTION

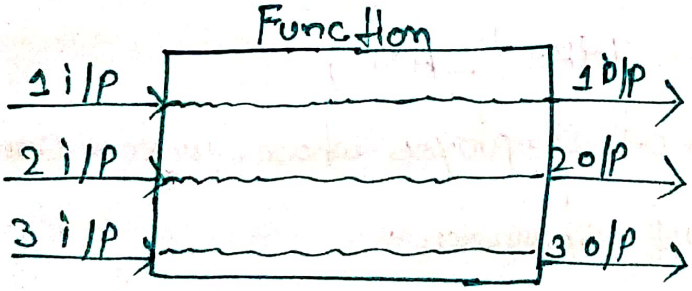
Function :=> Function is a sets of ~~get~~ instruction as blocks of code which is used to performe a specific task is called as Function.

=> Function are Tooo Types :>

1. Predefine
2. User define :- (Storing Procedure) PLSQL.

=> **Predefine :-** Predefine is also tooo types :-

1. **Single Row Function :=>** Single Row function is a function which accept Multiple inputs and gives us multiple respective output.



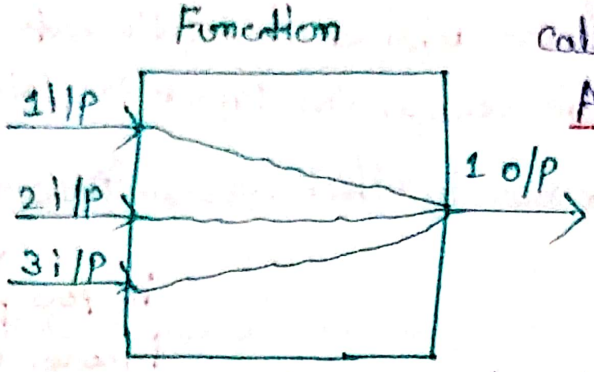
eg :=> Length (ename)

WARD — (4)

MILLER — (6)

Multi Row Function :=> Multi Row function is a function which accept multiple inputs in a single shot and gives us only one output is called Multi Row Function.

Multi Row Function also called as Group Function / Aggregate Function.



① Max( ) :=> It is mainly used to obtain max value from Column.

② Min( ) :=> It is mainly used to obtain min value from column.

③ Sum( ) :=> It is mainly used to obtain addition of value from Column.

④ Avg( ) :=> It is mainly used to obtain Avg value from Column.

⑤ Count( ) :=> Count is mainly used to obtain the total Number of records present inside a Column

=> Characteristics of Multi Row Function :=>

① Multi Row function will accept only one argument.
Ex :=> Max (ColName / Expression)

② If we are passing multirow function along with that we are not supposed to pass any column name inside the select clause.

Eg :=> select Max( ), ENAME
- Max( ) → It will give one o/p.
- ENAME → not pass colName like this.

③ Multi row function will ignore the NULL values.

④ We can not pass a multirow function inside the where clause.

[because where clause will execute the data Row by Row and multirow value accept the Input in single shot.]

⑤ Count is only function in which we can pass * (asterisk).

↓

if you passing not ignore NULL values.

query 1=>

① WAQTD an MAX salary of an Employee.

~~Select (MAX)~~

```
Select MAX (SAL)
From EMP;
```

② WAQTD MAX salary given to a Manager.

```
Select MAX (SAL)
From EMP
where JOB IN 'Manager';
```

(OR)

```
where JOB = 'Manager';
```

③ WAQTD min salary of employee who are hierd after 81 and before 87.

```
Select Min (SAL) AS Min_Salary
From EMP
where Hiredate between '01-JAN-82' AND
                       '31-DEC-86';
```

④ WAQTD total salary Earn by Emp who are working in dept 10;

⇒ Select SUM (SAL) AS Total_Salary
From EMP
Where DEPTNO IN (10);
(or)
where DEPTNO = '10';

⑤ WAQTD total No of Emp earning more than 1500 In deptno 10.

⇒ select count (*) AS Total_employess
From Emp
where SAL > 1500 AND DEPTNO = 10;

⑥ WAQTD Avg salary of all the emp whose name started with 'S';

⇒ Select Avg (SAL)
From EMP
where Ename Like 'S%';

⇒ Group By / ⇒ MultiRow Function / aggregate Function !⇒

Group By is mainly used to group the similer type of records.

Syntax !⇒ Select Group by colName / Group by Expression
From TableName ——→ 1st Execute ↓ last/4th Execute
[where <filter Condition>] ——→ 3rd Execution
Group by colName / Expression; ——→ 2nd Execute

Oder of Execution :=>

1. From
2. where → Row by Row
3. Group by → Row by Row
4. Select → Group by Group.

NOTE :=> after Executing Group by clause any clause Executing then it will Execute Group by Groupby.

=> Characteristics of Group By clause :=>

1. Group by clause will execute Row by Row
2. After Execution of group by clause we will get groups of similer type of records.
3. after group by clause any clause will execute group by groupby.
4. The colName pass inside the group by clause can also pass inside select clause along with multi row function. It know as group by Expression.
5. Group by colName is also group by function.

(10) => WAQTO MAX salary Earn by an Employee in Each Job.

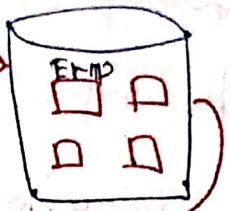
```
=> select Max(SAL), JOB
   From EMP
   Group by JOB;
```

Q2 => WAQTD avg sal earn by Manager in each department and having salary between 1500 and 5000.

```
=> select Avg(SAL)
   From EMP
   where JOB = 'Manager' and SAL between 1501 AND
                                         4999
   Group by DEPTNO;
```

Q3 => WAQTD total no of emp works in each department?

```
=> select count(*), deptno  -> (3)
   from emp  -> (1)
   Group by deptno;  -> (2)
```



EMP Table

empno	ename	JOB	deptno
1	Ankit	salesman	10
2	Madnav	Manager	20
3	Rajesh	Analyst	30
4	Jethalal	salesman	20
5	Babita	Analyst	10

10 -> (2) record

1	Ankit	sal	10
5	Babita	Ana	10

10 -> 2 record

2	Madnav	Mang	20
4	Jethala	sale	20

30 -> (1) record

3	Rajesh	Analyst	30

O/P =>

Count(*)	deptno
2	20
2	10
1	30

=> Having :=> having clause is mainly use to filter the groups.

- Inside having we pass multirow functions.

Syntax :=>

⑤ ← Select group-function / group by Expression → ①
From TableName

optional → [Where <filter-Condition>] → ②

[Group by colName / EXP → ③

having <group filter condition>; → ④]

Oder of Execution :=>

① From
② where [row by row]
③ Group by [row by row]
④ having [group by Group]
⑤ Select [group by Group]

(Q1) => WAQTD total no of Employee working in each department in which minimum two Employee are working. =>

```
Select Count(*)
From EMP
Group by deptno
having Count(*) >= 2;
```

o/p =>

Count(*)
2
2

(Q2) => WAQTD JOB of Employee. In which minimum 3 Employees are working.

=> Select count (*)
From EMP
Group by JOB
having count (*) >= 3;

(Q3) => WAQTD salary of an Employee which is repeated.

=> Select SAL, count (*)
From EMP
Group by SAL
having count (*) > 1;

(Q4) => WAQTD no of Employee working in each department having at least two employees Character 'A' or 'S' in their Name.

=> Select count (*), deptno
From EMP
where ENAME LIKE '%A%' ~~AND~~ OR ENAME LIKE '%S%'
Group by deptno
having count (*) >= 2;

⇒ **Order by** :⇒ order by clause is mainly used to arrange the records ~~in asc~~ in ascending or descending order.

Syntax :⇒

```
Select colName / EXP                    (5)
From TableName                          (1)
[Where <filter condition>]              (2)
[Group by colName]                      (3)
[having <group filter condition>]       (4)
Order by colName [ASC] / DESC;          (6)
```

Order of Execution

optional

↓ by default it is taking in ascending order.

⇒ **Characteristics of Order by** :⇒

1. Order by clause will always Execute at last.
2. default order in order by clause is ascending.
3. ASC Key word is optional to pass.
4. To arrange the records in descending order we will use DESC Key words.
5. We can pass multiple columns in order by clause.

(Q1) :> WAQTD Salary of an Employee in ascending order.

=> Select SAL
From EMP
Order by SAL;

(Q2) => WAQTD salary of an Eployee in descending order.

=> Select SAL
From EMP
Order by SAL DESC;

(Q3) :> WAQTD Employee name, Job, SAL of an Employee arrange it according to JOB and salary.

=> ~~Sele Select descending~~
Select SAL
From EMP
Order by JOB, SAL DESC;

o/p

JOB	SAL
Manager	2000
Manager	1000

if o/p

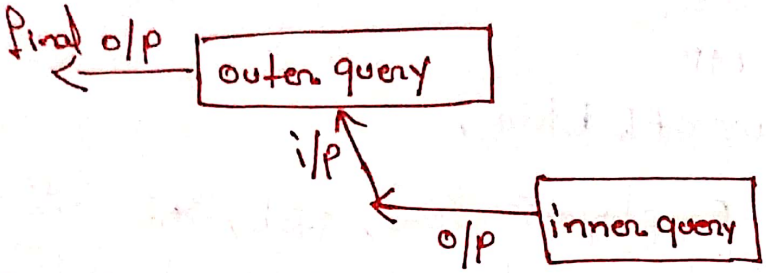
JOB	SAL
Manager	2000
Manger	2000

→ oder by follow insertion

Subquery Operations

=> Subquery :- A query written inside the another query is called as Subquery.

Execution Process :=>



① 1st Innerquery will execute and gives us some o/p.

② This o/p of Inner query is given us a input to outer query.

③ Outer query will execute and gives us a final result.

Case I :=> Whenever unknow is present in the question then we will use Subquery CaseI.

Q1 => WAQTD names of emp who earning more than 2000.

=>
```
Select ename
From Emp
where SAL > 2000 ;
```

Q2 => WAQTD names of Employee who Earning more than 'ALLEN';

=>
```
Select Ename
From EMP
where SAL > (select SAL
             From EMP
             where Ename = 'ALLEN');
```

(Q3) ⇒ WAQTD details of an EMP who are earning less than 'Simith'.

⇒ Select *
From EMP
where SAL < (Select SAL
From EMP
where Ename = 'Simith');

(Q4) ⇒ WAQTD details of an Employee who is working in same department as ~~was~~ 'WARD'.

⇒ Select *
From EMP
where DEPTNO = (Select DEPTNO
From EMP
where Ename = 'WARD');

(Q5) ⇒ WAQTD Name, JOB, Hiredate of an Employee if that Employee is working in same JOB as 'Scott'.

⇒ Select Ename, JOB, Hiredate
From EMP
where JOB = (Select Job
From EMP
where Ename = 'SCOTT');

(Q6) ⇒ WAQTD details of an Employee who are hired after 'BLAKE';

⇒ Select *
From EMP
where Hiredate > (Select Hiredate
From EMP
where Ename = 'BLAKE');

(Q7) ⇒ WAQTD details of an Employee who are Earning more than 1500 working in same DEPTNO as 'KING'.

⇒ Select *
From EMP
where SAL > 1500 AND DEPTNO = (Select DEPTNO
From EMP
where Ename
= 'KING'
);

(Q8) ⇒ WAQTD details of an Employee who are Earning more than Smith and Earning less than 'KING';

⇒ Select *
From EMP
where SAL > (Select SAL
From EMP
where Ename = 'Smith') AND
SAL < (Select SAL
From EMP
where Ename = 'KING');

(Q9) ⇒ WAQTD Name, salary and DEPTNO of an Employee if an Employee is Earning comission in deptNO in 20 and earning salary more than 'SCOTT';

⇒ Select ename, SAL, DEPTNO
From EMP
where COMM IS NOT NULL
AND DEPTNO = 20
AND SAL > (Select SAL
From EMP
where ename = 'SCOTT');

who name ends with 'S' and hiredate after 'Jam

⇒ select ename, hiredate
From EMP
where ename LIKE '%S'
AND ~~hire~~ hiredate > (select hiredate
From EMP
where ename = 'Jems')

(11) ⇒ WAQTD name of an Employee working in same deptno as Jems and earning salary more than 'Adm and working as same JOB as 'Miller'. and hire after 'Martin'.

⇒ Select ename
From Emp
where DEPTNO = (select DEPTNO
From EMP
where ename = 'Jems')
AND SAL > (select SAL
From EMP
where ename = 'Adms')
AND JOB = (select JOB
From EMP
where ename = 'Miller'),
AND Hiredate > (select hiredate
From EMP
where ename = 'Hiredate');

(Q10) => WAQTD name and hiredate of an Employee who name ends with 'S' and hiredate after 'Jems'.

=> select ename, hiredate
From EMP
where ename Like '%S'
AND ~~hire~~ hiredate > (select hiredate
From EMP
where ename = 'Jems');

(Q11) => WAQTD name of an Employee working in same deptno as Jems and earning salary more than 'Adms' and working as same Job as 'Miller' and hire after 'Martin'.

=> Select ename
From Emp
where DEPTNO = (select DEPTNO
From EMP
where ename = 'Jems')
AND SAL > (select SAL
From EMP
where ename = 'Adms')
AND JOB = (select JOB
From EMP
where ename = 'Miller')
AND Hiredate > (select hiredate
From EMP
where ename = 'Hiredate');

Case II :=> Whenever data to be selected is present in one table and condition to be applied is present in another table. then we can use Subquery Case II

(Q1) => WAQTD department name of Employee Adam.
(dept) (emp)

=> select Dname
from Dept
where deptno = (select deptno
From Emp
where ename = 'ADAM');

(Q2) => WAQTD Location of 'BLAKE'.

=> Select LOC
From DEPT
where DEPTNO = (select DEPTNO
From EMP
where ename = 'BLAKE');

(Q3) => WAQTD ename, Salary, hiredate, Deptno of an Employee who are working in 'neco york'.

=> select ename, sal, hiredate, Deptno
From Emp
where Deptno = (select Deptno
From DEPT
Where LOC = 'NecoYork');

(Q4) ⇒ WAQTD all the department details of an employee if the Employee is working as Manager and Earning more than 3000, and hired after 81 and before 83.

⇒ Select *
From DEPT
where DEPTNO = (Select DEPTNO
From EMP
where JOB = 'Manager'
AND SAL > 3000
AND Hiredate between '01-JAN-81'
AND '31-DEC-86');

(Q5) ⇒ WAQTD all the details of an Employee working as Manager in department Accounting.

⇒ Select *
From EMP
where JOB = 'Manager'
AND DEPTNO = (Select DEPTNO
From DEPT
where Dname = 'Accounting');

(Q6) ⇒ WAQTD no of employees working as Clerk in the same department as Smith and earning more than King and hiredafter Martin and LOC is BOOSTEN.

⇒ select count(*)
From EMP
where JOB = 'Clerk'
AND DEPTNO = (select DEPTNO
from EMP
where ename = 'SMITH')
AND SAL > (select SAL

(Q4) ⇒ WAQTD all the department details of an employee if the Employee is working as Manager and Earning more than 3000, and hired after 81 and before 87.

⇒ Select *
From DEPT
where DEPTNO = (Select DEPTNO
From EMP
where JOB = 'Manager'
AND SAL > 3000
AND Hiredate between '01-JAN-82'
AND '31-DEC-86');

(Q5) ⇒ WAQTD all the details of an Employee working as Manager in department Accounting.

⇒ Select *
From EMP
where JOB = 'Manager'
AND DEPTNO = (Select DEPTNO
From DEPT
where Dname = 'Accounting');

(Q6) ⇒ WAQTD no of employees working as Clark in the same department as Smith and earning more than King and Hiredafter Martin and LOC is BOOSTEN.

⇒ select count(*)
From EMP
where JOB = 'Clark'
AND DEPTNO = (select DEPTNO
from EMP
where ename = 'SMITH')
AND SAL > (Select SAL
From EMP where ename = 'martin')

AND DEPTNO = (select DEPTNO
From DEPT
where LOC = 'BOOSTEN');

(Q4) WAQTD Max salary of an Employee.

⇒ Select Max(SAL)
From EMP;

(Q5) ⇒ WAQTD name, Sal of an Emp who are earning Maximum SAL.

⇒ select ename, SAL
From EMP
where SAL = (Select Max (SAL)
From EMP);

(Q6) ⇒ WAQTD name of an Employee who are earning Minimum salary.

⇒ select ename
From EMP
where SAL = (select Min (SAL)
From EMP);

Types of subquery :=>

1. Single row subquery
2. Multi Row subquery

1. Single Row subquery =>

(i) A subquery returning only one value / output is called as single Row subquery.

(ii) This output of single row subquery is given as input to the outer query.

(iii) Both `= , IN` operator can be use to handle single output of single Row Sub query.

Ex:=> WAQTD Dname of 'ALLEN';

```
=> select Dname
   from DEPT
   where DEPTNO = (select DEPTNO
                   from EMP
                   where Ename = 'ALLEN');
```

2. Multi Row subquery :=>

(i) The subquery returning multiple values of output is know as MultiRow subquery.

(ii) We can use only `IN` operator in multi Row subquery.

NOTE :=>

Subquery operators are use with Relational operator.

- ALL → working like AND operator
- ANY → working like OR operator

Ex:- WAQTD Dname of miller & ward;

A) Select Dname
From DEPT
Where DEPTNO IN (select DEPTNO
From EMP
where ename IN
('miller', 'ward'));

10, 30

Subquery Operator :-

(1) ALL operator :- (i) ALL operator will return true if all the value on the RHS are satisfying the condition.

(ii) ALL operator is always use along with Relational operator.

Example :- > ALL
< ALL
>= ALL
<= ALL

NOTE :-
ALL operator is work like a 'AND' operator.

(Q) :- WAQTD name & salary of an employee & an employee is earning more than salary of emp ~~and~~ of department 10.

=> select ename, SAL
From EMP
where SAL > ALL (select SAL
① From emp
② where deptno = 10);

800, 3000
5000

EMP

empno	ename	SAL	deptno
1	Scott	800	10
2	Allen	2000	20
3	Miller	5000	30
4	Turner	500	20
5	King	3000	10
6	Adam	1500	30

✗ 800 > 800 AND 800 > 3000
✗ 2000 > 800 AND 2000 > 3000
✗ 5000 > 800 AND 5000 > 3000
✗ 500 > 800 AND 500 > 3000
✗ 3000 > 800 AND 500 > 3000
✗ 1500 > 800 AND 1500 > 3000

O/P =>

empno	ename	SAL	deptno
3	Miller	5000	30

(2) ANY Operator :=> (i) Any operator will return true if any of the values on the RHS is satisfying the condition.
(ii) This ANY operator is also use along with relational operator.

Example :=> > ANY
< ANY
>= ANY
<= ANY

NOTE :-
Any operator work like OR operator.

Q :=> WAQTD name and salary of an Employee if any Employee is earning more than any one Employee of deptNO is 10.

=> select ename, emp
from EMP
where SAL > ANY (select SAL
① ← from EMP
② ← where deptNO = 10);

Emp

empno	ename	SAL	deptno
1	Scott	800	10
2	ALLEN	2000	20
3	Miller	5000	30
4	Turner	500	20
5	King	3000	10
6	Adam	1500	30

✗ 800 > 800 or 800 > 3000

✓ 2000 > 800 or 2000 > 3000

✓ 5000 > 800 or 5000 > 3000

✗ 500 > 800 or 500 > 3000

✓ 3000 > 800 or 3000 > 3000

✓ 1500 > 800 or 1500 > 3000

O/P

empno	ename	SAL	deptno
2	ALLEN	2000	20
3	MILLER	5000	30
5	KING	3000	10
6	Adam	1500	30

(Q1) => WAQTD name of an Employee if the Employee Earn less than the Employee working as Salesman.

```
Select ename
From EMP
where SAL < ALL ( select SAL
                  From EMP
                  where ename = 'SALESMAN');
```

(Q1) => WAQTD name of Employee if the employee earn less than the atleast one working as salesman.

=> Select ename
From EMP
where SAL < ANY (Select SAL
From EMP
where ename = 'salesman');

=> Nested Subquery :=>

(1) A subquery written inside another subquery is called as Nested Subquery.

(2) We can write maximum 255 nested Sub query.

Ex :=> WAQTD 2nd Max salary :=>

=> Select Max (SAL) => 3000
From EMP
where SAL < (select Max (SAL)
From EMP);

o/p => 3000 2nd Max salary.

2000 < 5000 ✓
1000 < 5000 ✓
1500 < 5000 ✓
2500 < 5000 ✓
5000 < 5000 ✗ this is reject.
3000 < 5000 ✓

⇒ Employee, Manager relationship :-

① WAQTD name of SMITH's Manager.

⇒
```
Select ename
From EMP
where empno = (select MGR
               From EMP
               where ename = 'SMITH');
```

∴ NOTE
Use = or IN
Operator any one you can Use it

② ⇒ WAQTD Manager Details of WARD.

⇒
```
Select *
From EMP
where EMPNO = (Select MGR
               From EMP
               where ename = 'WARD');
```

③ ⇒ WAQTD Dname and Loc of Manager of CLARK.

⇒
```
Select Dname, LOC
From DEPT
where DEPTNO IN(select DEPTNO
                From EMP
                where EMPNO IN (select MGR
                                From EMP
                                where ename
                                = 'CLARK');
```

(Q) WAQTD details of SMITH Manager's, manager.

⇒
```
select *
From EMP
where EMPNO IN (select MGR
                From EMP
                where EMPNO IN (select MGR
                                From EMP
                                where ename
                                = 'SMITH'));
```

(Q) WAQTD department details of Scott's Mangers, Manager.

⇒
```
select *
From DEPT
where DEPTNO = (select DEPTNO
                From EMP
                where EMPNO = (select MGR
                               From EMP
                               where EMPNO =
                               (select MGR
                                From EMP
                                where Ename
                                = 'SCOTT'));
```

(6) WAQTD details of an Employees working Under BLAKE.

```
Select *
From EMP
where MGR IN (select EMPNO
              From EMP
              where ENAME = 'BLAKE');
```

(7) WAQTD details of an Employee who are working under King.

```
select *
From EMP
where MGR IN (select EMPNO
              From EMP
              where Ename = 'King');
```

8) WAQTD department details of an emp who are reporting to Jones.

```
select *
From DEPT
where DEPTNO IN (select DEPTNO
                 From EMP
                 where MGR IN (select EMPNO
                               From EMP
                               where Ename
                                 = 'JONES'));
```

① Arithmetic operator :=> Arithmetic operators are used to perform arithmetic operation such as addition, subtraction, division and multiplication. This operators usually accept numeric operands.

Ex :=> WAQTD details of employees along with annual salary of the employee.

=>
```
Select Emp.*, (sal*12)
From EMP;
```
→ using dot operator you can pass colname.

② Comparison operator :=> Comparison operators in SQL are used to check the equality of two Expression. It checks whether one expression is identical to another. types of comparison operator are (=, !=).

Ex1 :=> WAQTD ename and deptno of an Employee who are working in dept 10, 20.

=>
```
Select ename, deptno
From EMP
where deptno = 10 OR deptno = 20;
```
Equal operator.

③ Logical operators :=> Logical operators are those operators that take two Expression as operands and return TRUE or FALSE.

Types of Logical operator are :=>
AND, OR, NOT.

Ex :=> WAQTD salary of an Employee who are Earning salary > 2000 and salary < 3000.

=> select SAL
From EMP
where SAL > 2000 AND SAL < 3000;

④ Concatination operator (||) :=> This operator is mainly used to concatenate two string or one string one ~~table~~ colname.

Ex :=> ① ~~select~~ WAQTD the string 'Mohini' 'Shrivastav' on output screen.

=> select 'Mohini' || 'Shrivastav'
From DUAL; two string concatenate.
→ dummy table

② WAQTD string like 'Mr Smith' for Each Employee.
→ colname
=> select 'Mr' || Ename from EMP Table.
From EMP; this is string.

(5) Relational operator :=> Relational operators are used to compare the value and establish relationship between data stored in tables.

Type :=> [>, <, >=, <=]

Ex :=> WRITE Salary of an Employee who are Earning more than 2000;

=> Select SAL
From EMP
where SAL > 2000;

(6) Special operator :=> Special operators are used in SQL queries to perform specific operations like comparing values, checking for existence, and filtering data based on certain condition.

(7) Subquery operator :=> A query written inside the another query is called as subquery.

types :=> ALL, ANY, EXISTS, NOT EXISTS.

Drawback of Subquery :=> ① Data from multiple tables can not be display simultaneously on the output screen from multiple table.

② As condition are increase the length of sub query is also increased.

This two drawbacks can be solve with help of **Joins**.

Joins :=> Joins are mainly used to retrive the data from the multiple tables.

Types of Joins :-

① Cross Join / Cartession Join
② Inner Join
③ Natural Join
④ outer Join → Left outer Join, Right outer Join, Full outer Join.
⑤ self Join

Cross Join :=> Cross join is mainly use to merge all the records of table 1 with all the records of table 2.

Syntax :=>

ANSI :=> select ColName / Expression
From TableName 1 Cross Join TableName 2;

ORACLE :=> select ColName / Expression
From TableName1 , TableName2;

Emp

Empno	ename	deptno
1	Scott	20
2	BLAKE	30
3	Miller	10

Dept

deptno	Dname	LOC
10	Accounting	M
20	sales	G
30	research	B

```
select *
from emp, dept;
```

Total column :- C1 + C2
total records :- R1 × R2

Output Table 1 =>

Empno	ename	deptno	deptno	Dname	LOC
1	Scott	20	10	Acc ---	M
1	Scott	20	20	sales	G
1	Scott	20	30	research	B
2	BLAKE	30	10	Acc ---	M
2	BLAKE	30	20	Sales	G
2	BLAKE	30	30	research	B
3	Miller	10	10	Acc ---	M
3	Miller	10	20	sales ---	G
3	Miller	10	30	research	B

Drawback 1 => Number of unmatched records are more then the matching records.

② INNER Joins :=> Inner Join is mainly used to obtain matching records from two or more tables.

Syntax :=>

ANSI :=>
```
select colName | Expression
From Tablename1 Inner Join TableName2
on <Join Condition>;
```

Oracle :=>
```
select colName | Expression
From TableName1 , TableName2
where <Join condition>;
```

Join Condition

TableName1 . CommonColName = TableName2 . CommonColN2

Ex :=>
```
Emp . deptno = Dept . Deptno;
```

=>
```
select *
From Emp , dept
where Emp . Deptno = Dept . Deptno;
```

Emp

Empno	Ename	deptno	deptno	Dname	Loc
1	Scott	20	10	Accounting	M
2	Blake	30	20	Sales	G
3	Miller	10	30	Research	B

20 = 10 ×
20 = 20 ✓
20 = 30 ×

30 = 10 ×
30 = 20 ×
30 = 30 ✓

10 = 10 ✓
10 = 20 ×
10 = 30 ×

Output Table !=>

Empno	ename	deptno	deptno	Dname	LOC
1	Scott	20	20	Sales	G
2	blake	30	30	research	B
3	Miller	10	10	ACC	M

(a) => WAQTO Ename and Dname of all the employee.

=> Select Ename, Dname
From Emp, Dept
Where Emp.Deptno = Dept.Deptno;

(b) => WAQTO ename and Loc for all the employees working as Analyst.

=> Select ename, Loc
From Emp, Dept
Where Emp.Deptno = Dept.Deptno
AND Job = 'Analyst';

(6) WAQTD deptno and Ename and Loc of all the Employees.

=> select Ename, Emp.Deptno, Loc
From Emp, Dept
where Emp.Deptno = Dept.Deptno;

Error !=> column 'ab' ambiguously defined.

Natural Join :⇒ Natural join will behave like an Inner join if there is relation between two or more table else it behaves like a cross join.

Syntax :⇒

Ans :⇒ Select colName / Expression
From TableName1 Natural Join TableName2;

Ex :⇒ Select *
From Emp Natural Join Dept;

By Default it makes relation if there is relation between to tables.

Emp

empno	ename	deptno
1	Scott	20
2	Smith	30
3	Allen	10

dept

deptno	dname	LOC
10	Acc..	M
20	Sales	G
30	research	B

output Table :⇒

Deptno	empno	ename	dname	LOC
20	1	Scott	Sales	G
30	2	Smith	research	B
10	3	Allen	Acc..	M

∴ here Natural Join work like Inner Join.

Ex(2) :=> Select *
from Emp Natural Join Device;

Emp

Empno	ename	deptno
1	Scott	20
2	Smith	30
3	Allen	10

Device

Did	DName	Price
D1	HP	13000
D2	Dell	50000

Output :=>

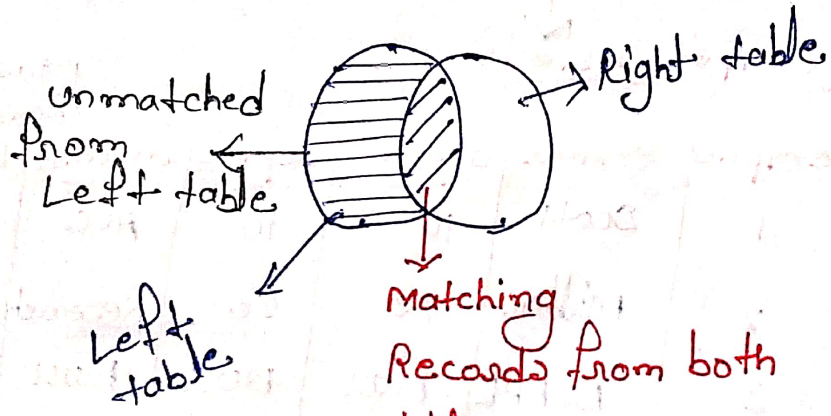
Empno	ename	deptno	Did	DName	Price
1	Scott	20	D1	HP	13000
1	Scott	20	D2	Dell	50000
2	Smith	30	D1	HP	13000
2	Smith	30	D2	Dell	50000
3	Allen	10	D1	HP	13000
3	Allen	10	D2	Dell	50000

here Natural Join work like Cross Join.

=> Outer Join => Left outer Join
=> Right Outer Join
=> Full outer Join

=> ④ OuterJoin :=> Outer Join mainly used to obtain unmatched record along with that you will get matching records.

① Left Outer Join :=> Left Outer Join is used obtain unmatched record from left table and matching records from the both table.



Syntax :=>

① ANSI :=> select ColName / Expression
From TableName1 Left [outer] Join TableName2
ON <Join Condition>;

② ORACLE :=>
select ColName / Expression
From tableName1 , TableName2
where <Join condition>;

Emp.deptNo = dept.deptNo(+)

Ex1=> select *
From Emp , dept
where Emp.deptno = dept.deptno(+);

Emp

Empno	ename	deptno
1	Scott	10
2	Miller	20
3	BLAKE	
4	King	

dept

deptno	dname	LOC
10	ACC	M
20	Research	D
30	Sales	B
40	operation	P

10 = 10 ✓
10 = 20 ✗
10 = 30 ✗
10 = 40 ✗
20 = 10 ✗
20 = 20 ✓
20 = 30 ✗
20 = 40 ✗

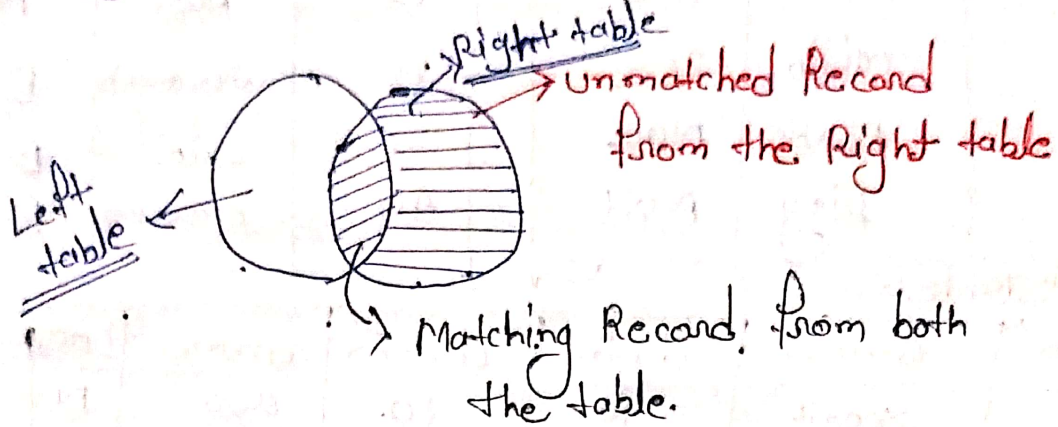
output Table 1=>

Empno	ename	deptno	deptno	dname	LOC
1	Scott	10	10	ACC	M
2	Miller	20	20	Research	D
3	BLAKE	NULL	NULL	NULL	NULL
4	King	NULL	NULL	NULL	NULL

Matching Records

unmatched Records.

(2) Right Outer Join :=> Right outer join is used obtain unmatched record from Right table and matching Records from the both table.



Syntax :=>

ANSI :=> select colName / Expression
From tableName1 Right [outer] Join TableName2
ON < Join Condition >;

ORACLE :=>
select colName / Expression
From TableName1, TableName2
where < Join condition >;
↳ Emp.Deptno(+) = dept.deptno;

Ex :=>
select *
From Emp, Dept
where Emp.DeptNo(+) = Dept.Deptno;

Emp Table Number 1.

Empno	Ename	deptno
1	Scott	10
2	Miller	20
3	Blake	NULL
4	King	NULL

dept Table Number 2

deptno	Dname	Loc
10	Acc..	M
20	research	D
30	Sales	B
40	operation	P

Out Put Table 1⇒

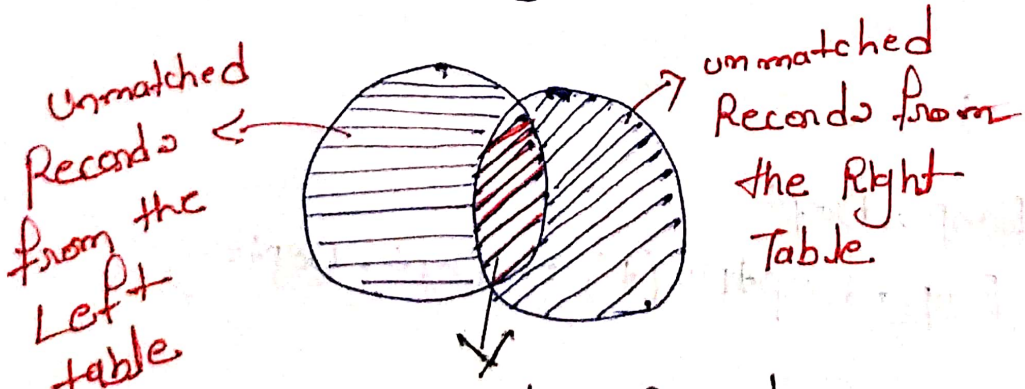
Empno	Ename	deptno	deptno	dname	Loc
1	Scott	10	10	acc	M
2	Miller	20	20	research	D
NULL	NULL	NULL	30	Sales	B
NULL	NULL	NULL	40	operation	P

Matching Records from both the table

unmatched Records from the Right table

③ Full Outer Join ⇒

Full outer Join is used to obtain unmatched Records from the Right table and from the Left table, also obtain matching Records from the both table.



Syntax :>

ANSI :> Select colName / Expression
From TableName1 Full [outer] Join TableName2
ON <Join condition>;

Ex1:> Select *
From Emp Full Join Dept
ON Emp.Deptno = Dept.Deptno;

TableName1
Emp

Empno	ename	deptno
1	Scott	10
2	Miller	20
3	Blake	NULL
4	King	NULL

TableName2
Dept

deptno	Dname	LOC
10	Accounting	M
20	Research	D
30	sales	B
40	operation	P

output Table :=>

Empno	ename	deptno	deptno	dname	LOC
1	Scott	10	10	acc..	M
2	Miller	20	20	Research	D
NULL	NULL	NULL	30	sales	B
NULL	NULL	NULL	40	operation	P
3	Blake	NULL	NULL	NULL	NULL
4	King	NULL	NULL	NULL	NULL

Matching Records from Both the table

Unmatched from Right table

Unmatched from Left table

⑤ **Self Join :=>** Self Join is mainly use to Join the table with itself.

* if you wanted Employee name and also their Manager Name you go for Self Join.

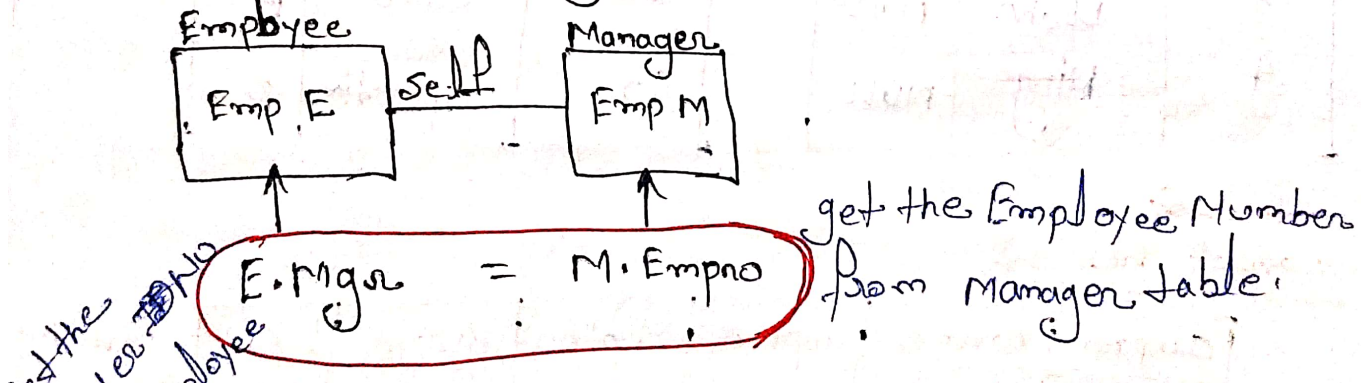
Syntax :=>

ANSI :=>
```
Select colName / Expression
From TableName 1 [self] Join TableName 1
ON <Join Condition>;
```

ORACLE :=>
```
Select colName / Expression
From TableName 1, TableName 1
where <Join Condition>;
```

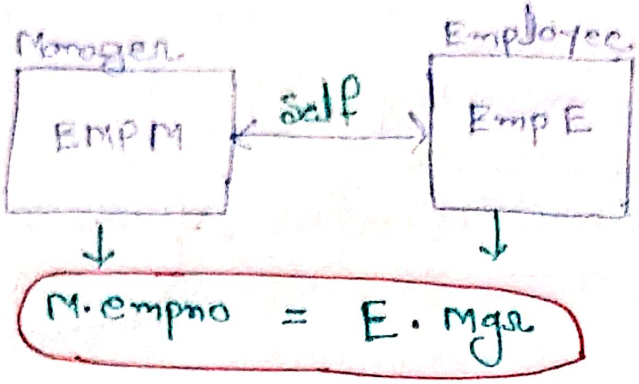
Logic :=>

Q=> Find out Manager :=>

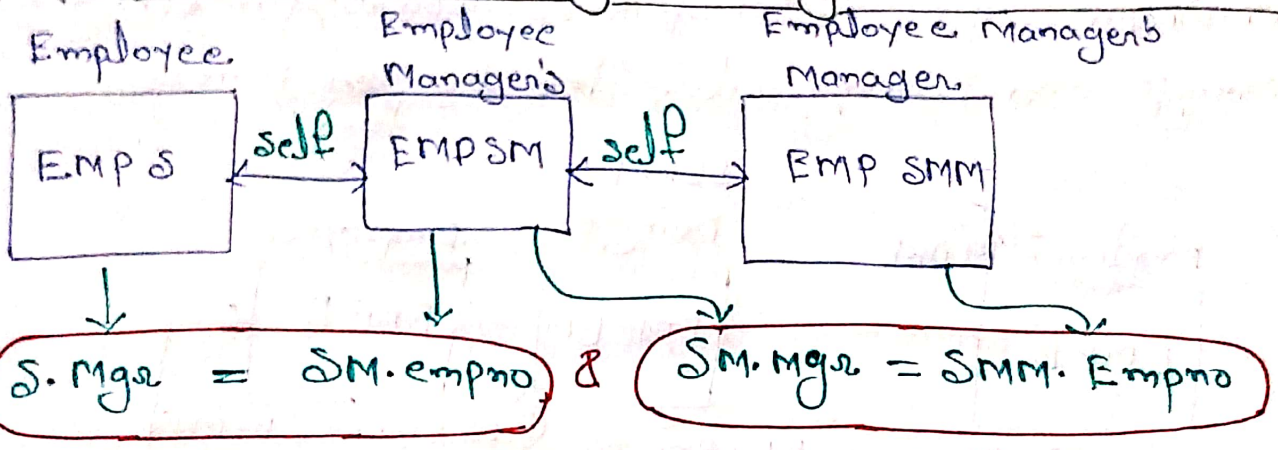



```
select E.Ename, M.Ename
from Emp E, Emp M
where E.mgr = M.empno;
```

⇒ ② Find out Employee Under Manager :⇒

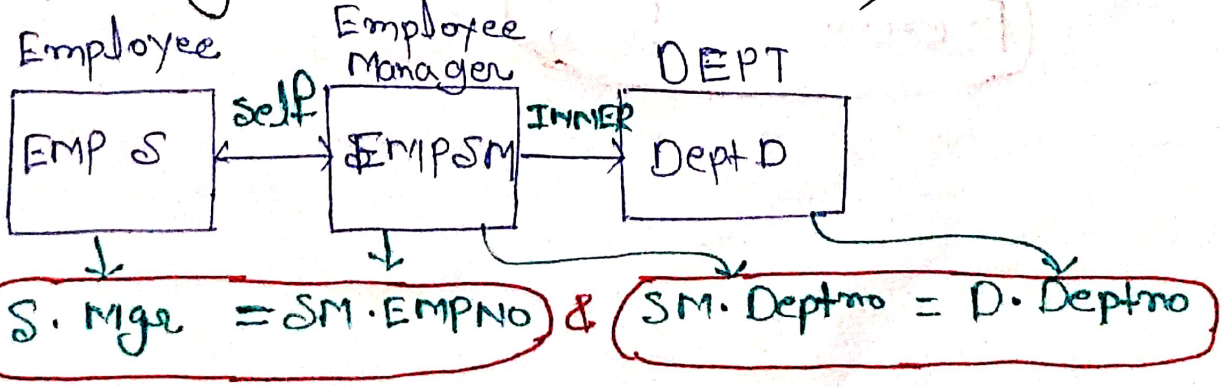


⇒ ③ Find out Employee Manager's Manager use Two self Join

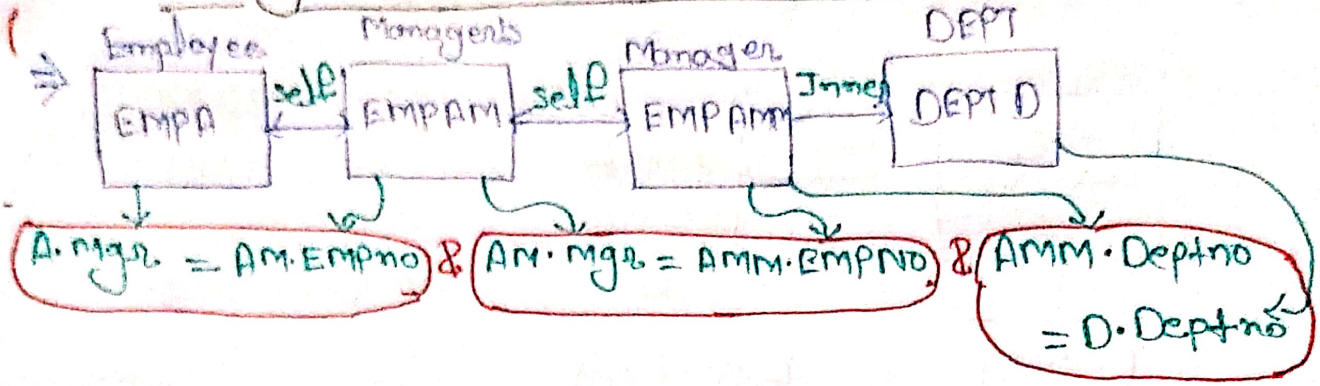


⇒ Select S.Ename, SM.Ename, SMM.ename
From Emp S, EMPSM, EMP SMM
where S.Mgr = SM.Empno
AND SM.Mgr = SMM.EMPNO
AND S.Ename = 'SMITH';

⇒ ④ Using Self and Inner Join :⇒



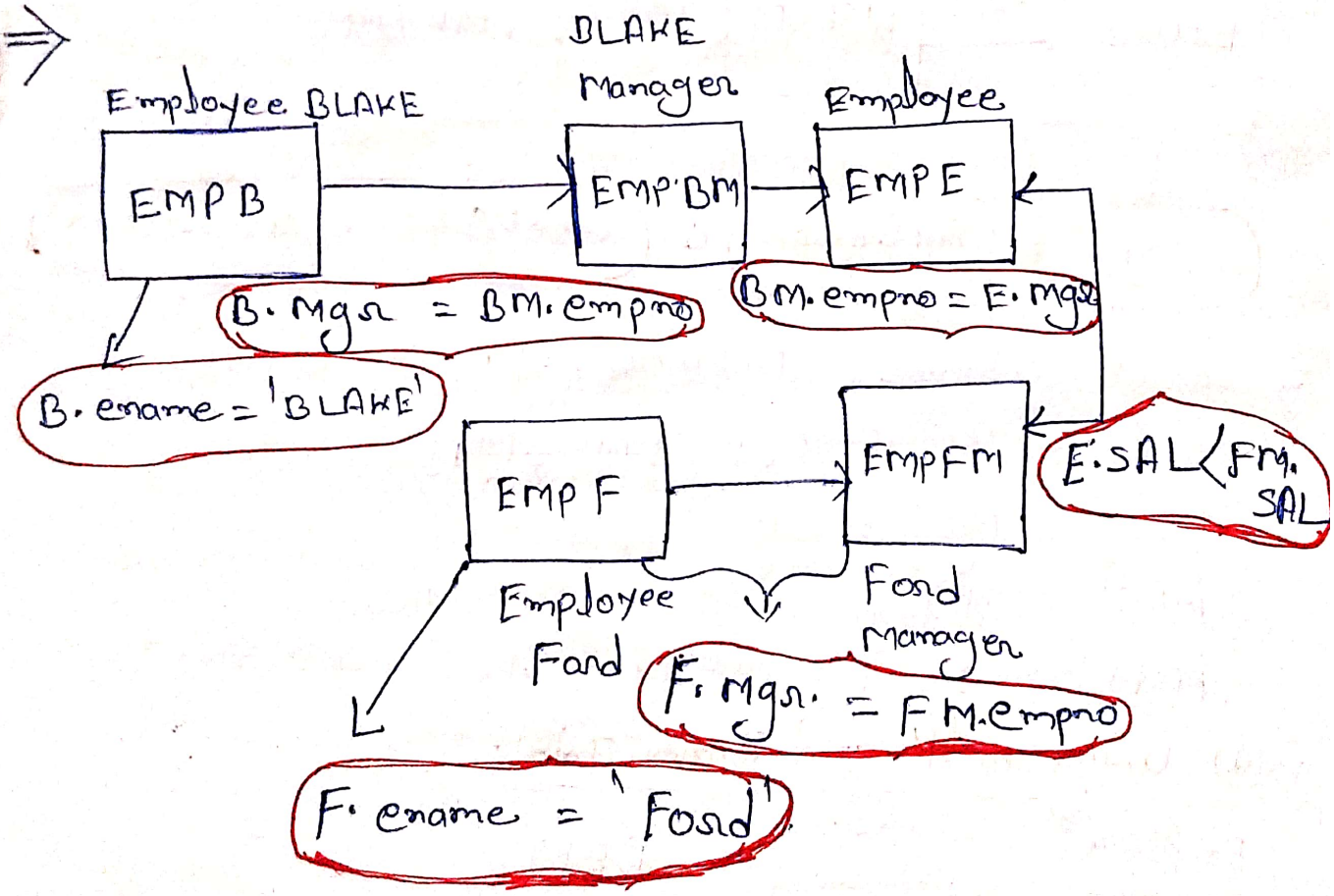
⑤ Using Self, Self and Inner Joins :-



Query ⇒

① WAQTO number of Employees reporting to BLAKE's Manager and earning salary less the ~~Ford~~ Ford Manager.

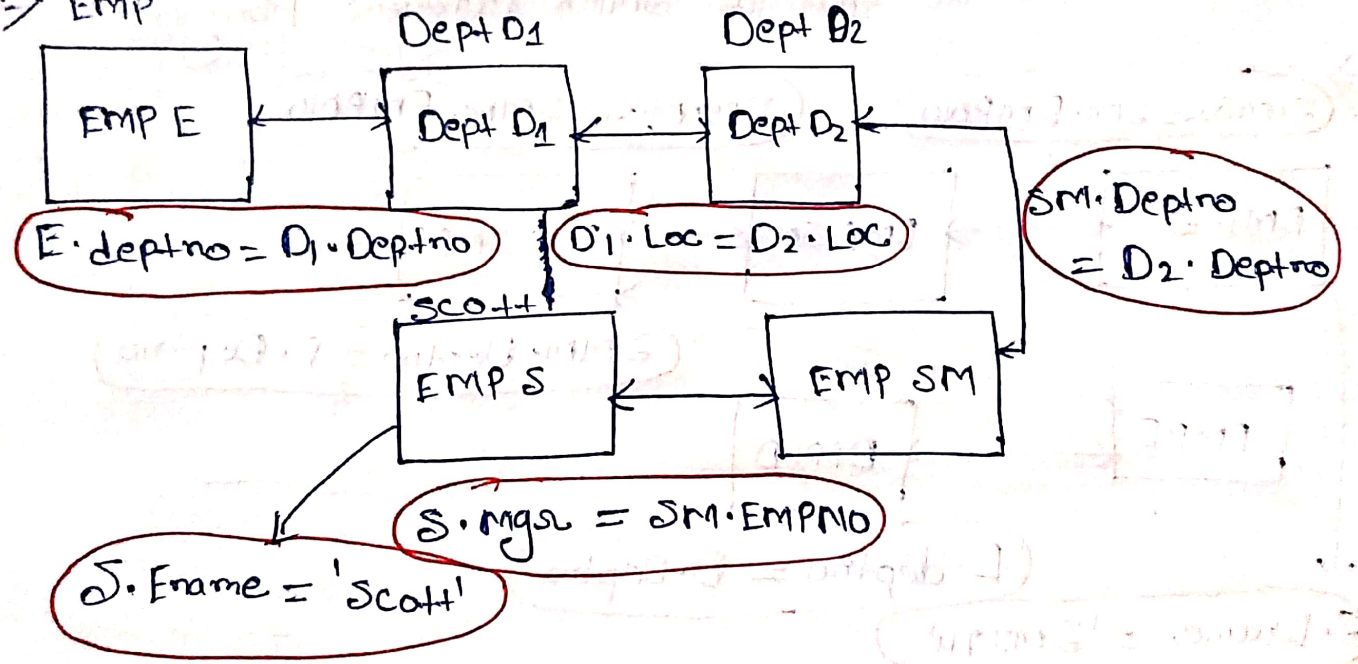
⇒



```
select count(*)
From EMP B, EMP BM, EMPE, EMP F, EMPFM
where B.MGR = BM.EMPNO
AND B.ENAME = IN 'BLAKE'
AND BM.EMPNO = E.MGR
AND F.MGR = FM.EMPNO
AND F.ENAME IN 'FORD'
AND E.SAL < FM.SAL;
```

② WAQTD Total salary required to pay all the Employee working in same location as scott's Manager.

⇒ EMP

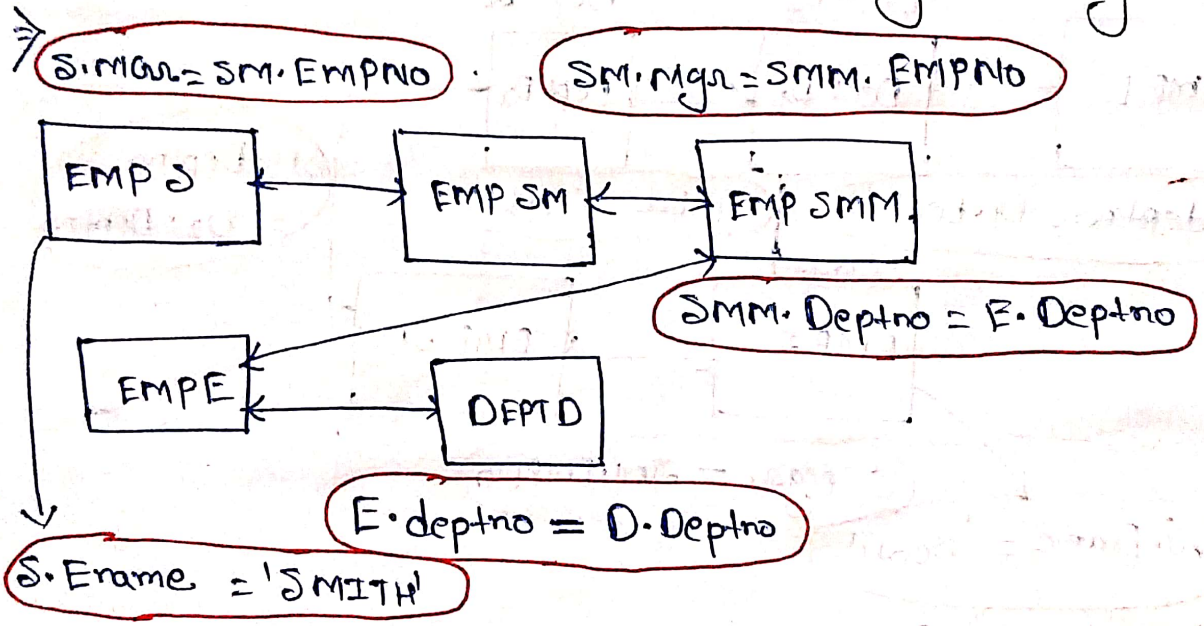


⇒ Select Sum (E.SAL)
From EMP S, EMPSM, DEPT D1, EMP E, DEPT D2
where S.MGR = SM.EMPNO
AND SM.DEPTNO = D2.DEPTNO
AND E.DEPTNO = D2.D1.Deptno

⇒ select SUM (E.SAL)
from EMP S, EMP SM, DEPT D1, EMP E, DEPT D2
where S.MGR = SM.EMPNO
AND SM.DEPTNO = D2.DEPTNO
AND E.DEPTNO = D2.DEPTNO
AND D1.LOC = D2.LOC
AND S.ENAME IN 'SCOTT';

(3) WAQTD Name, JOB, Department Name, and Location of all the Employees working in same department as that of SMith Managers Manager.

⇒

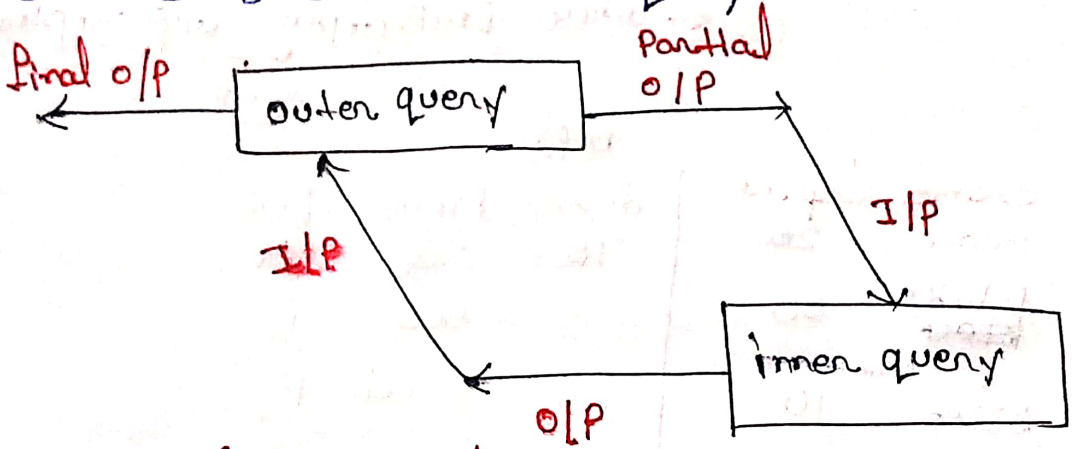


(4) WAQTD Ename and his dname with Manager name, Managers dname if manager is earning greater then Employee and commision of Employee should be NULL And Employee should be hierd after the Manager.

=> SELECT E.ename, D.DNAME, E.MGR
from EMP M, EMP E, DEPT D
where M.MGR = E.EMPNO
AND E.DEPTNO = D.DEPTNO
AND M.SAL > E.SAL
AND E.COMM IS NULL
AND E.HIREDATE < M.HIREDATE;

=> Co-related Subquery :=>

Co-related Subquery is a Subquery in which both Outer query and Inner query are interdependent on each other. is called as Co-related subquery.



Execution of co-related Subquery :=>
process

1. 1st outer query will execute partially and gives us partial output.
2. This output of outer query is given as Input to the Inner query.
3. Inner query execute gives us output.

④ This o/p of Inner query is given as a input to the outer query.

⑤ At last outer query will fully execute and gives us final Result or output.

Example 1 => WAQTO Dname of an Emp if emp is working any department.

=> Select Dname.
① ← from dept (20, 30, 10, 20, 10)
② where deptno IN (select deptno
③ from Emp ← ①
② ← where EMP.deptno = dept.deptno);
② ②

EMP

EMPNO	ename	deptno
1	Scott	20
2	Blake	30
3	ALLen	10
4	King	20
5	Miller	10

Dept

deptno	Dname	LOC
10	Sales	M
20	Acc	D
30	research	P
40	operation	K

STEP ①

20 = 10 ✗
20 = 20 ✓
20 = 30 ✗
20 = 40 ✗

Step ②

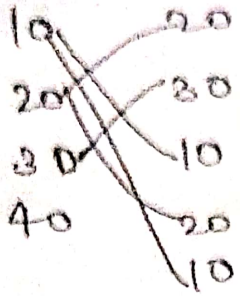
30 = 10 ✗
30 = 20 ✗
30 = 30 ✓
30 = 40 ✗

10 = 10 ✓
10 = 20 ✗
10 = 30 ✗
10 = 40 ✗

20 = 10 ✗
20 = 20 ✓
10 = 10
10 = 10

Step (3)

∴ output ⇒ o/p



sales..
Acc..
Research

Exists ⇒ Exists operator is a unary operator.

Exists operator returns true if subquery is returning any value other than NULL.

This is also called optimize Execution.

Query ⇒ WAQTD dname of an Emp if emp is working in any department using Exists operator.

⇒
```
Select Dname
from Dept
where Exists (select deptno
              from Emp
              where emp.deptno = dept.deptno
```

NOTE ⇒

If we get any output from the inner query then only Exists allow to Execute select Clause.

EMP

Empno	Ename	deptno
1	Scott	20
2	Blake	30
3	Allen	10
4	King	20
5	Miller	10

dept

deptno	Dname	Loc
10	Sales	M
20	Acc..	D
30	Research	P
40	operation	K

Step(1)

10
20
30
40

Step(2)

20 = 10 ✗
20 = 20 ✓
20 = 30 ✗
20 = 40 ✗

emp.deptno = dept.deptno

30 = 10 ✗
30 = 20 ✗
30 = 30 ✓
30 = 40 ✗

10 = 10 ✓
10 = 20 ✗
10 = 30 ✗
10 = 40 ✗

20 = 10 ✗
20 = 20 ✓

10 = 10 ✓

O/P => output Table

Sales
Acc..
Research

(Q1) :- WAQTO 4th Max Salary.

⇒ select SAL

From EMP E1

where (N-1) IN (select count (Distinct (SAL))

(3) From EMP E2

where E1.SAL < E2.SAL);

EMP E1

SAL
2000
1500
1000
5000
3000
850
900

EMP E2

SAL
2000
1500
1000
5000
3000
850
1000
900

∴ N = 4

∵ N - 1 = 4 - 1 = (3)

E1.SAL < E2.SAL

2000 < 2000 ✗
2000 < 1500 ✗
2000 < 1000 ✗
2000 < 3000 ✓
2000 < 5000 ✓
2000 < 850 ✗
2000 < 1000 ✗
2000 < 900 ✗

1500 < 2000 ✓
1500 < 1500 ✗
1500 < 1000 ✗
1500 < 5000 ✓
1500 < 3000 ✓
15000 < 850 ✗
1500 < 1000 ✗
1500 < 900 ✗

2 ≠ 3 ✗

∴ 3 = 3

⇒ output ⇒

SAL
1500

(Q2) ⇒ WAQTD 7th Max Salary.

⇒ Select SAL
From EMP E1
where (N-1) IN (Select count (Distinct (SAL))
From EMP E2
where E1.SAL < E2.SAL);

(Q3) WAQTD 7th minimum Salary.

⇒ Select SAL
From EMP E1
where (N-1) IN (select count (Distinct (SAL))
From EMPE2
where E1.SAL > E2.SAL);

(Q4) ⇒ WAQTD 3rd Maximum, 4th, 7th and 8th maximum Salary.

⇒ Select SAL
From EMP E1
where (select count (Distinct (SAL))
from EMPE2
where E1.SAL < E2.SAL) IN (2,3,6,7,8);

(25) WAQTD Ename, SAL, 3rd, 4th, 5th, 6th, 8th maximum Salary.

⇒ Select E1.SAL, ENAME
From EMP E1
where (select count (Distinct (E2.SAL))
From EMP E2
where E1.SAL < E2.SAL) IN (2,3,6,7,8);

Single Row Function ⇒

Single Row Function is a function which accept Multiple inputs and gives us multiple respective output.

Types of single Row Function ⇒

(1) length()
(2) Concat()
(3) Lower()
(4) Upper()
(5) INITCAP()
(6) REPLACE()
(7) REVERSE()
(8) SUBSTR()
(9) INSTR()
(10) MOD()
(11) ROUND()
(12) TRUNC()
(13) Months_BETWEEN()
(14) Last_DAY()
(15) TO_Char()
(16) NVL()

① Length ( ) :=> Length ( ) is mainly used to Count the total Number of characters Present inside a particular string.

Syntax :=> Length ('Str')

(Q1) WAQTD The total Number of character present inside an Employee name.

=> Select Length (EName)
From EMP;

(Q2) WAQTD Name of the Employee which Consist of total 6 characters.

=> Select Ename
From EMP
where Length (Ename) = 6;

(Q3) WAQTD salary of an Employee who are Earning total 4 digits of salary.

=> Select SAL
From EMP
where Length (SAL) = 4;

② Concat( ) ⇒ It is mainly used to concat or merge the two string.

Syntax ⇒

Concat ('str1', 'str2')

(Q) WAQTD the string in a format by concatinating 'Mr' & 'HARSHAL'?

⇒ Select concat ('Mr', 'HARSHAL')
From DUAL;

(Q2) WAQTD the string like 'Mr scott' for each Employee

⇒ Select concat ('Mr', Ename)
From EMP;

∴ 'Ename' if you pass like this it will take as a string.

(Q3) ⇒ WAQTD the string like Hi' I am SMith'.

⇒ select concat ('Hi I am', Ename)
From EMP;

(Q) WAQTD the string like My name is SMITh with Employee id 1234 and Joining date is '14-FEB-24' and Salary is 3000 and Commission is 400;

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10)

Concat(concat(concat(concat(concat(concat(concat(concat(concat('My name is', Ename), with Employee id), empno), 'and Joining date is'), hiredate), 'And salary is'), SAL), 'and commission is'), Comm);

③ Lower( ) ⇒ This lower ( ) is mainly used to convert a string into lower case.

Syntax :⇒

lower ('str');

⇒ WAQTD all the name of Employee in lower case.

⇒ Select lower (Ename)
From EMP;

~~Upper~~

④ Upper( ) ⇒ Upper( ) is mainly used to convert a string in upper case.

Syntax :⇒ upper ('str');

⇒ WAQTD all the name of Employee in upper case.

⇒ Select upper (Ename)
From EMP;

⑤ INITCAP ( ) ⇒ It is mainly used to convert a string in which 1st letter will be in capital case and remaining letters are in smaller case.

Syntax :⇒ INITCAP ('str');

(Q) WAQTD name of the Employee 1st character in upper case and remaining in lower case.

⇒ Select INITCAP (Ename)
From EMP;

⑥ Reverse ( ) :=> Reverse ( ) is mainly used to revers a string.

=> Syntax :- Reverse ('str');

(a) WAQTD the name of Employee in Reverse order.

=> Select REVERSE (ENAME)
From EMP;

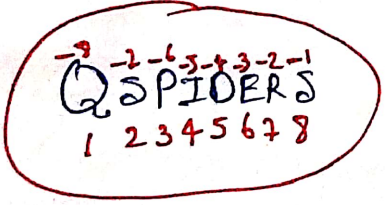
⑦ SUBSTR ( ) :=> It is mainly used to extract a particular string from the original string.

=> Syntax :- SUBSTAR ('originalstring', Position [, length]);
optional.

∴ (if you not passing the length it will start ~~from~~ from the position and print till the End.)

(a) => WAQTD spiders from the string @SPIDERS.

=> select SUBSTR ('@SPIDERS', 2);
From DUAL;



(b) WAQTD 1st 3characters of a string.

=> select SUBSTAR ('@SPIDERS', 1, 3)
From DUAL;

(5) WAQTD 1st two characters of your Employee Name.

=> Select SUBSTR (Ename, 1, 2)
From EMP;

(6) WAQTD last 3 characters of an Employee Name.

=> ~~Select (Ename, -3)~~

Select SUBSTR (Ename, -3)

From EMP;

(7) WAQTD Name of Employee whose Job consist of 1st 3 characters as 'MAN'.

=> Select Ename, JOB
From EMP
where SUBSTR (Ename, 1, 3) = 'MAN';

(8) WAQTD Name of Employee whose Job consist of last 3 characters as 'MAN'.

=> Select Ename, JOB
From EMP
where SUBSTR (Ename, -3, 3) = 'MAN';

(9) WAQTD 1st half of the Employee Name.

=> Select SUBSTR (Ename, 1, length (Ename)/2)
From EMP;

(10) WAQTD the 2nd half of Employee Name.

=> select SUBSTR (Ename, length (Ename)/2+1)
From EMP)

⑧ INSTR( ) :⇒ It is mainly used to obtain the position of a given string in a original string, if string is present in a original string then it will give the position otherwise we will get '0' as an output.

Syntax :⇒ INSTR('Original String', 'Str', Position, [occurance])

'Str' → Checking string

occurance → the string how many times repeated.

EX :⇒

BANANA
123456
Index No.

⇒ INSTR('BANANA', 'A', 1)

O/P ⇒ (2)

⇒ if occurance = 2.

⇒ INSTR('BANANA', 'A', 1, 2)

O/P = (4)

⇒ INSTR('BANANA', 'A', 1);

O/P = 0

(Q) ⇒ WAQTD Name of ~~An~~ an Employee which consist of 'A' present inside the Name.

⇒ Select Ename
From EMP
where (Ename, 'A') >= 1;

~~where (Ename, 'A') > 0~~

where
(Ename, 'A', 1) > 0

(8) => WAQTD Ename. if it consist of At least '2A'.

=> Select Ename
From EMP
where INSTR(Ename, 'A', 2) > 0;

(9) Replace :=>

It is mainly used to Replace a given string with New string in a original string.

Syntax :=>

Replace ('Original string', 'Str' ['New string'])

↓ String to be Replace.

Ex :=>

(1) Replace ('BANANA', 'A', 'C');

o/p => BCNCNC

(2) Replace ('BANANA', 'NA', 'M');

o/p => BAMM

(3) Replace ('BANANA', 'NA')

Replace by Empty string.

o/p => BA

(Q) WAQTD a string in which Employee names first two characters should be in capital case 3rd characters will be as it is and remaing character will be in lower case.

⇒ Select Replace (Replace (Lower (EName),
SUBSTR (Lower (Ename) 1, 2),
UPPER (SUBSTR (EName) 1, 2))),
SUBSTR (Lower (EName), 4),
Lower (SUBSTR (Ename, 4)))
From EMP;

(Q) ⇒ WAQTD total Number of times A present in Employee name.

⇒ ~~Select Sum (Length (ENAME) - Length (Replace (Ename, 'A',~~

Select (Length (ENAME) - Length (Replace (Ename, 'A', '')))
+ Length (Ename) - Length (Replace (Ename, 'a', '')))
AS Total-A
From EMP;

⇒ Upper (SubSTR (Cname, 1, 2))
|| (SubstR (ename, 3, 1))
|| Lower (substR (Ename, 4))
From EMP;

(10) MOD() :=> It is mainly used to obtain the remainders between two value.

Syntax :=> mod (value1, value2)

(Q1) :=> WAQD Even Employee ID of an Employee.

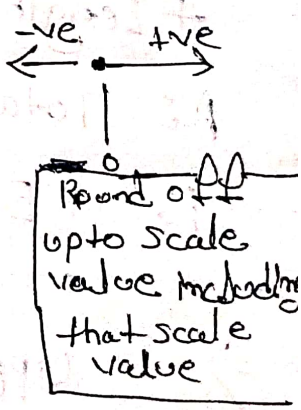
=> Select Empno
From EMP
where MOD (Empno, 2) = 0;

(Q2) => WAQTD name & salary of an employee if emp is getting odd salary.

=> Select ename, sal
From Emp
where mod (sal, 2) = 1;

(11) ROUND() => It is mainly used to obtain round off value for a number according to the scale value.

Syntax :=> Round (value [, scale]);



Eg :=> Select ROUND (420.56) → 421
From Dual;

ROUND (786.453, 0) → 786

5 00 → 000

ROUND (786.9543, 2) → 786.95

ROUND (7864.943, -2) → 7900

65 → 50 → 100

7900

ROUND (420.534, -2) → 400

• 5 → 10
• 50 → 100
• 500 → 1000

(12) TRUNC ( ) :⇒ It is mainly used to obtain the round of value to the smallest value for a given numbers.

Syntax :⇒ Trunc(value[,scale])

Ex:⇒ Trunc(420.56) → 420

Trunc(420.1234) → 420

Trunc(786.453, 0) → 786

Trunc(786.95|43, 2) → 786.95

Trunc(78|64.943, -2) → 7800

(13) MONTHS-BETWEEN :⇒

It is mainly used to obtain numbers in of months present between given two dates.

Syntax :⇒ Months_Between(Date1, Date2)

Date functions :⇒ 1) sysdate
2) Current_Date
3) Systimestamp

(Q1) ⇒ WAQTD experience of an emp in a month.

```
select Months_Between (current_Date, Hiredate)
From emp;
```

(Q2) ⇒ WAQTD experience in years

```
⇒ select (round (Months_Between(Current_Date, Hiredate)/12)
        From EMP;
```

(14) TO-CHAR() ⇒

⇒ It is mainly used to convert a date into string in a particular format model.

Syntax ⇒ `TO_CHAR(Date, format_Model)`

1) year -
   - YYYY - 1980
   - YY - 80
2) MONTH - FEBRUARY
   - MON - FEB
   - MM - 02
3) DAY - MONDAY
4) DD - 14
5) D ⇒ S(1) M(2) T(3) ---- S(7)
6) HH 24 → format in 24 hours.
8) SS → format in second.
7) HH12 → month hour format in 12 hours.
9) MI ⇒ format for minutes
10) 'YY - MM - DD'

(Q) => WAQTD name. of & hiredate of emp who are hired on wednesday.

=> select ename, hiredate
from emp.
where To_char(Hiredate, 'Day') = 'WEDNESDAY';

(15) LAST_DAY() => It is mainly used to obtain the last day of a particular month.

Syntax - Last_Day (Date)

(16) NVL (Null value Logic) => NVL function is mainly used to prevent or to eliminate the drawbacks of Null.

Syntax :- NVL (colName, value)

Ex1=> select sal + NVL(comm, 0)
From emp;

∴ => (1) DQL is compelet.

(2) DDL (Data Defination Language) =>

DDL is mainly used to create or to manage the structure.

↳ (nothing but the Tables.)

(1) Create
(2) Alter
(3) Rename
(4) Truncate
(5) Drop

① **Create** :=> Create is mainly used to create new Table structure.

Syntax :=> Create Table TableName
(
col1 datatype Constraint,
col2 datatype Constraint,
- - - - - - - - - -
);

② **Rename** :=> Rename is mainly used to change the name of a table.

Syntax :=> Rename old_Table_Name To New_Table_Name;

③ **Alter** => Alter is mainly used to modify the table structure.

=> Alter Table Table_Name

1. To add a column
2. To rename a column
3. To drop a column
4. To modify datatype
5. To modify constraints
6. To add a constraints seperately
7. To add foreign key.
8. To drop a constraints.

(1) To add a column :=>

Syntax :=> Alter Table TableName
ADD Colname datatype constraints;

(2) Rename a column =>

Syntax => Alter Table TableName
Rename column old_col_name to new_col_name;

(3) To drop a column =>

Syntax => Alter Table TableName
Drop Column Column_Name;

(4) To Modify data type =>

Syntax :=> Alter Table TableName
Modify Column_Name New_datatype.

(5) To Modify a Constraints :=>

Syntax :=> Alter Table TableName
Modify Colname existing_datatype
new_Constraints;

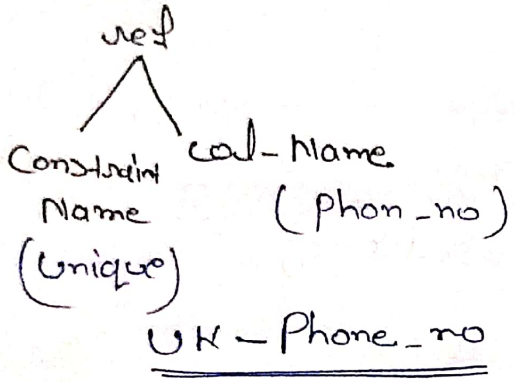
∴ You can Modify data type and Constraints at a

=> Syntax :=> Alter Table TableName
Modify Col_Name New_datatype
New_constraints;

⑥ To add a constraints seperately :=>

Syntax :=> Alter Table TableName
Add Constraint constraint_reference_name
constraint_name(colName);

ex :=> Alter Table Table_Name
ADD constraint UK_Phone_no UNIQUE(Phon_no);



⑦ To add a Foreign Key :=>

① while creating a Table :=>

```
Create Table Trainer
(
  Tid Number (5) primary key
      check (Tid > 0) check (length(Tid) = 5),
      Tname varchar (10),
      Course Varchar (10),
      id Number (5),
  Constraint FK_id Foreign Key (id)
      reference student (id)
);
```

=> To add Foreign Key in allready created table

① Create a column in a Table :=>

```
Alter table TableName
Add id number(5);
```

② Add a Foregin key :=>

```
Alter Table TableName
Add Constraint FK_id Foregin Key(id)
    reference student(id);
```

⑧ => Drop a Constraints :=>

```
Alter Table TableName
Drop Constraint Constraint_ref_Name;
```

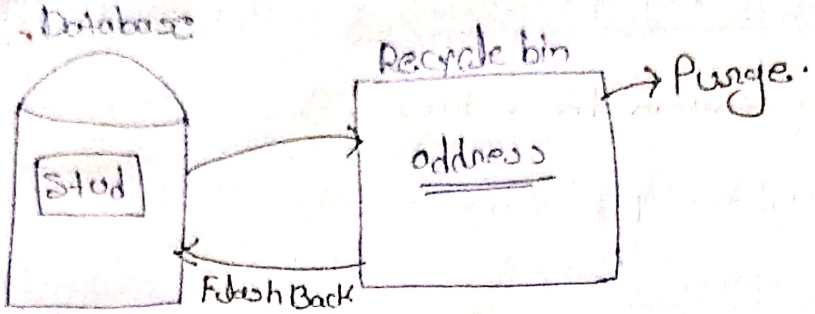
Truncate :=> It is mainly used to delete the data from the table without affecting table structure.

Syntax :=> `Truncate Table TableName;`

(Can't access data if you doing truncate)

Drop :=> Drop is mainly used to delete the table (it will delete the data as well as ~~well~~ Table structure.)

Syntax :=> `Drop Table TableName;`



① Permantly Delet !=>

```
Purge table TableName
```

② Recycle Table !=>

```
Flash back Table TableName;
To before Drop
```

DML (Data Manipulation Language) !=>

DML is mainly used to deals with data with help DML we can Insert, update and Delet the Data.

=> Insert !=> It is mainly used to Insert the data in the table.

① To Insert all data in a Table !=>

```
Insert into TableName values (V1, V2, ---Vn);
```

② To Insert data in particular column !=>

```
Insert into TableName (Col1, Col2, Coln ---)
                values (V1, V2, V3, Vn ---);
```

Ex :=> Insert into STUDENT (ID, Name, DOB, Course)
values (10003, 'RIYA', '17-SEP-2002',
'Java');

③ To Insert data through user Input :=>

=> Insert into TableName value (&id, '&Name', '&course',
$phon_no, '&Gender',
'&DOB');

(or)

=> Insert Table Name (id, Name, course)
values (&id, '&names', '&course');

④ Insert Multiple Records in Table :=>

```
Insert all
into STUD values (v1, v2, ---)
into STUD values (_ _ _ _ _ _)
into STUD values (         )
Select * From DUAL;
```

Update :=> Update is mainly used to modify the existing records.

Syntax :=>

```
Update TableName
Set Col1 = V1 , Col2 = V2
Where <Condition>;
```

(Q) => WAQT Update the Job of Miller as 'Developer'.

=> Update EMP
Set Job = 'Developer'
Where ename = 'MILLER';

Delete :=> Delete is mainly used to Delete a particular records.

Syntax :=>

```
Delete From TableName
Where <Condition>;
```

(Q) WAQTD Delete the Data of BLAKE.

=> Delete EMP
Where ename = 'BLAKE';

TCL (Transaction control language) :=>

It is mainly used to save the transaction.

① Commit :=> It is mainly used to save all the transaction permanently.

Syntax :=> `Commit;`

Ex① =>
```
insert
insert
insert
update
update
Delete
Commit
```
} this all record are save parmanently.

Ex② =>
```
insert
insert
insert
update
update
Delete
Create / Rename
```
} This all record are save automatically.

Create / Rename => this is statement of DDL.

∴ DDL is auto Committed.

NOTE :=>

① only DML is not auto committed.

② All statement is auto committed other than DML.

② **Savepoint** :⇒ Savepoint is mainly used to Save the location.

Syntax :⇒

```
Savepoint savepointName;
```

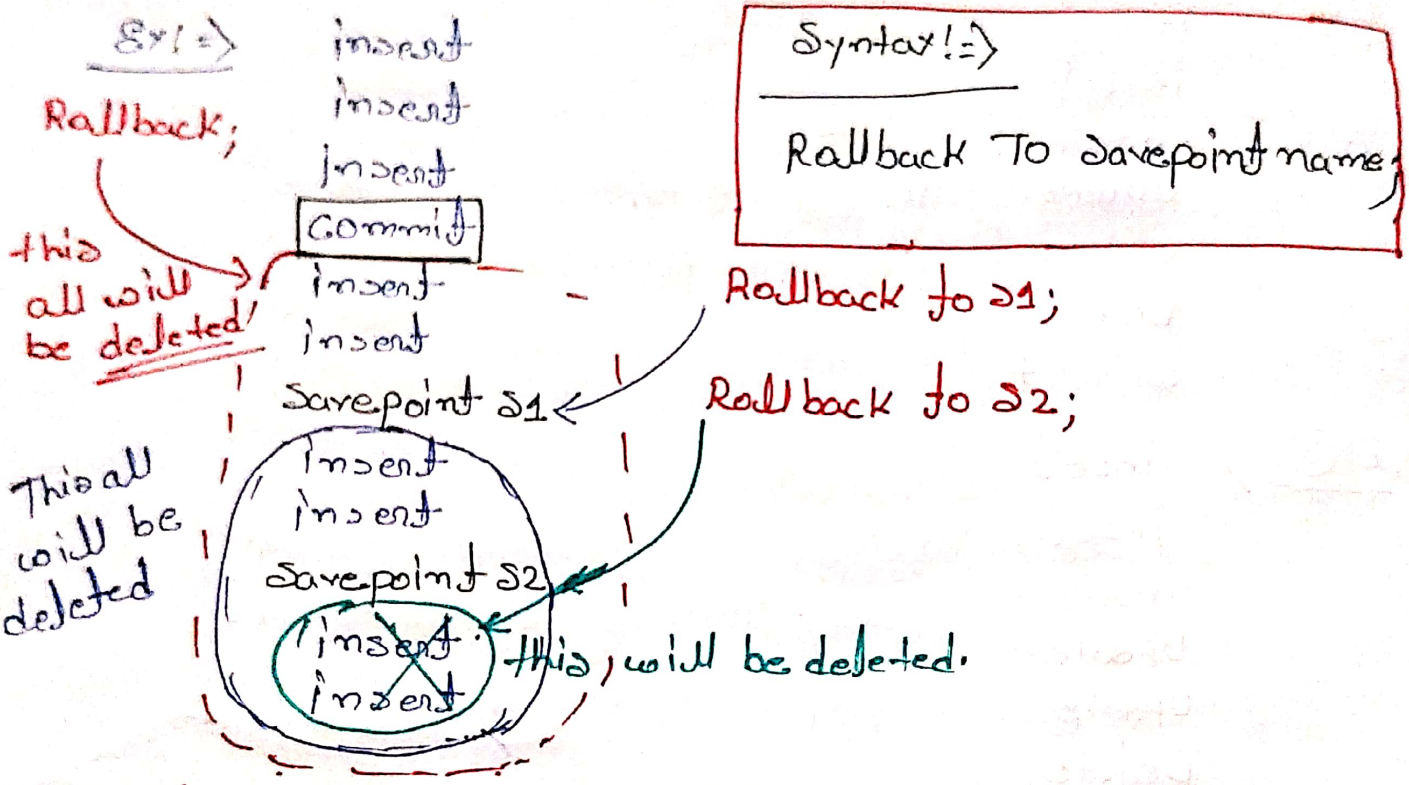
③ **Rollback** :⇒ Rollback is mainly used to reach latest saved location.

Ex :⇒

```
Rollback;
```

Syntax :⇒

```
Rollback To savepoint name;
```



NOTE :⇒

∴ Save_point is temporarily.

⇓

Transaction going to save but Savepoint remove after turn of the system. if no again login and write query Rollback To s2; we face the Error message "there is no s2".

So Savepoint is Temporary.

DCL (Data control Languag) :=>

It is mainly used to control the data flow b/w the users.

1. Grant
2. Revoke

=> **Grant** :=> Grant is mainly used to give an access of the data to the user.

Syntax :=>

```
Grant Command_Name
ON Table_Name
To UserName;
```

Eg => Grant Select
ON EMP
TO HR;

2. **Revoke** => Revoke is mainly used to back the access on data from the users.

Syntax :=>

```
Revoke Command_Name
ON Table_Name
From UserName;
```

⇒ Types of Keys :⇒

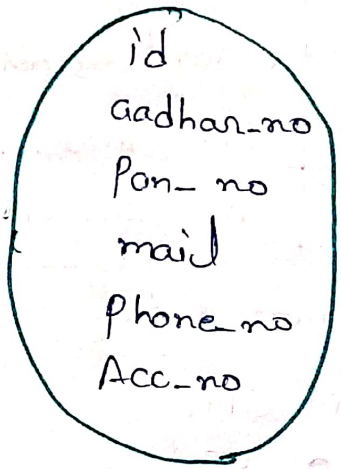
① Key attribute / candidate key :⇒

The key attribute is an attribute which can identify the another attributes uniquely is called key attributes.

Ex :⇒ Students

- id
- name
- age
- Course
- DOB
- Phone_no
- mail
- gender
- Acc_no
- Aadhar_no
- Pan_no

Key attribute / candidate key



② Non_Key attributes :⇒ All the attributes other then the key attributes is called is non_key attributes.

Ex :⇒
- name
- age
- Course
- DOB
- gender
- add

③ **Prime Key attributes** :⇒ Among all the key attributes any one key attribut is choosen to identify another attributes uniquely is called as prime key attributes.

Ex:⇒ (id) is prime key attribute.

④ **Non_Prime Key attributes** :⇒ the key attributes other than prime key attributes is known as non prime key attributes.

Ex:⇒ adhar, phon, pan, acc_no } non_prime key attributes.

⑤ **Composite key attributes** ⇒ The combination of more than two non_key attributes which can identify an attributes uniquely is called as composite key attributes.

Ex:⇒ (name, age, DOB)

⑥ **Super Key** :⇒ The set of all key attributes is called as super key attributes.

eg ⇒ (id, aadhar, phone_no, mail, Pan, Acc_no).

⑦ **Foregin key attribute** :⇒ An attribute which behaves as a part of an another entity as called as Foregin key attribute. | eg ⇒ (id).

Normalization :=>

It is a process of reducing the given tables into smaller tables in order to remove redundancy and anomalies, by identifying functional dependencies is called as Normalization.

Redundancies :=> Redundancies are nothing but the repetition of unwanted records.

Anamolies :=> Anamolies are nothing but the side effects of DML operations.

* Insertion anamolies
* ~~Updation~~
* Updatation anamolies
* deletion anamolies

Functional Dependencies :=> There exists dependencies such that on attributes in a relation can identifies another attribute unique is known as functional ~~depen~~ dependencies.

Ex :=> $R \rightarrow (A, B)$ $A \rightarrow B$

$\therefore$ B = dependent

A = determiner

⇒ There are three type of Functional Dependencies !⇒

① Total Functional Dependencies !⇒ There exists a dependencies such that only a key attributes can identify another attributes uniquely is known a Total Functional dependencies.

Ex !⇒ R -> (A, B, C, D)

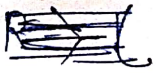
A -> B
A -> C
A -> D

② Partial Functional Dependencies !⇒ There are exists dependencies such that a part of a composite key can identify any other non-key attributes uniquely is called as partial Functional Dependencies.

Ex : R -> (A, B, X, Y)

Composite key (A, B) -> (X, Y)

part of composite key ⇒ B -> Y



③ Transitive Functional Dependencies !⇒

There exists a dependencies such that a non-key attributes can identify another non-key attribute uniquely but there is one key attribute which can identify another attributes uniquely.

Ex :>

Interdependent on Each other.

NKA1 (Brother) → NKA2 (Sister) → KA (Father)

∴ NKA Non-Key-Attribute
KA Key-Attribute

Normal Forms :=>

the tables is said to be in a normal form if it does not consist of any redundancies and anomalies.

First Normal Form (1NF) :=>

A ~~tabe~~ table is said to be in 1NF if does not have any redundancies or multi-value data.

Second Normal Form (2NF) :=>

A table is said to be in 2NF if

* It should to be 1NF.
* It should not have Partial Functional Dependencies.

Third Normal Form (3NF) :=>

A table is said to be in 3NF if

* It should be in 2NF.
* It should not have Transitive Functional Dependencies

Boycee Codd NF (BCNF / 3.5NF) !=>

A table is said to be in BCNF / 3.5NF if it in 3NF.

NOTE !=>

A table said to be Normal if table is ~~that~~ third Normal Form.