ORACLE

Oracle Content (2 Months): -

Topic-5: - PL/SQL

->Introduction to PL/SQL

->Difference between SQL and PL/SQL

->Conditional & Looping statements

->Cursors

->Exception Handling

->Stored procedures

->Stored Functions

->Triggers

ORACLE

->In IT field users are interacting with two types of applications. Those are

1.Front End Applications

2.Back End Applications

1) <u>Front-End Application: -</u>

->FEA is an application where the end-users interact to an application directly.

Ex: - Register from, Login Form, View Profile forms, Home page, etc.

->UI technologies are HTML, CSS, JavaScript, Angular JS, React JS, etc.

2) <u>Back-End-Application: -</u>

->BEA is an application where we store the end-user's data/information.

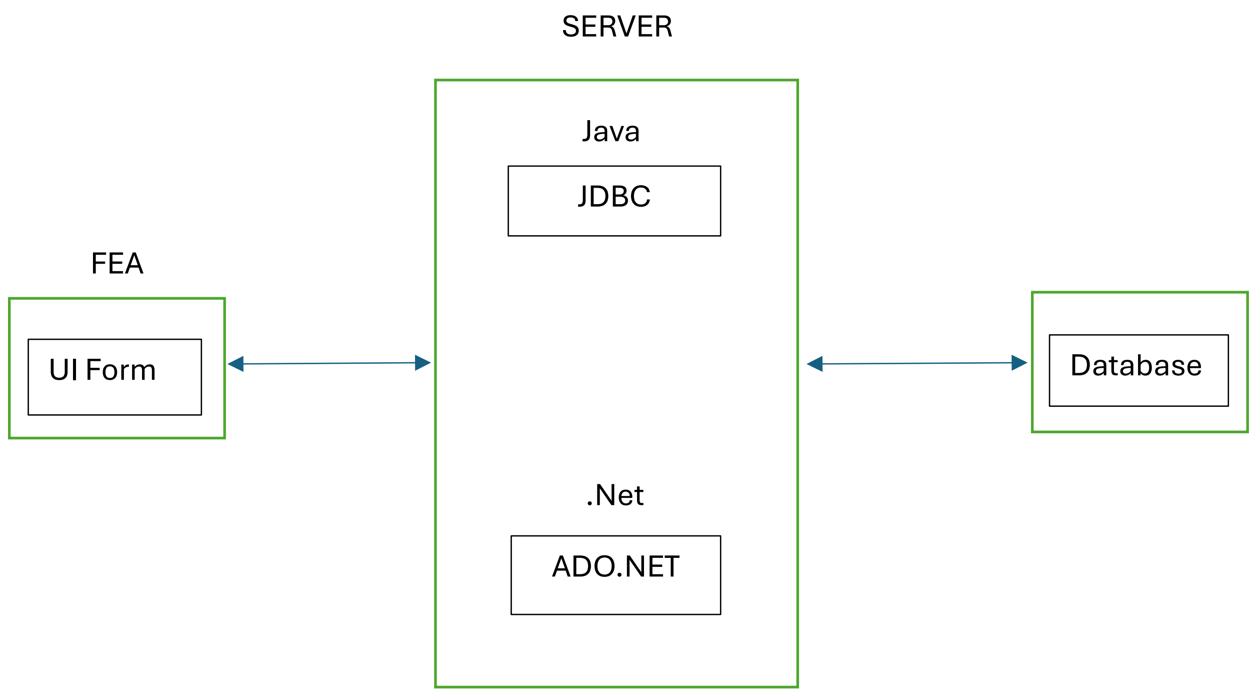
Ex: - Database.

->DB technologies are ORACLE, SQL SERVER, MYSQL, POSTGRESQL, DB2, etc.

=> <u>Server- Side Technologies: -</u>

-> These technologies are used to establish a connection between front end application and back- end application.

-> Server- Side Technologies are Java, .Net, Python, PHB, etc.



Topic – 1(DBMS)

=> What is Data?

->It is a raw fact. (i.e. Characters, Numbers, Special characters and Symbols)

->Data never gives meaningful statements to users.

Ex: -	SMITH is data	10021 is data
	MILLER is data	10022 is data
	ADAMS is data	10023 is data

=>What is information?

->Processing data is called information.

->Information always provides meaningful statements to users

Ex: -	Employee Name	Employee Id
	SMITH	10021
	MILLER	10022
	ADAMS	10023

=>What is Database?

->It is a memory which is used to store inter-related information of a particular organization.

=>What is inter-related information?

->Depends on each other.

Ex: - SBI Organization

-group of branches ----------------→Group of Customers

\>group of departments

\>group of employees.

Ex: - no department = no employees
no employees = no department
no customers = no products

no products = no customers.

=>Types of Databases?

->There are two types of databases in the real world. Those are

1.OLTP (Online Transaction Processing)

2.OLAP (Online Analytical Processing)

1.OLTP: -

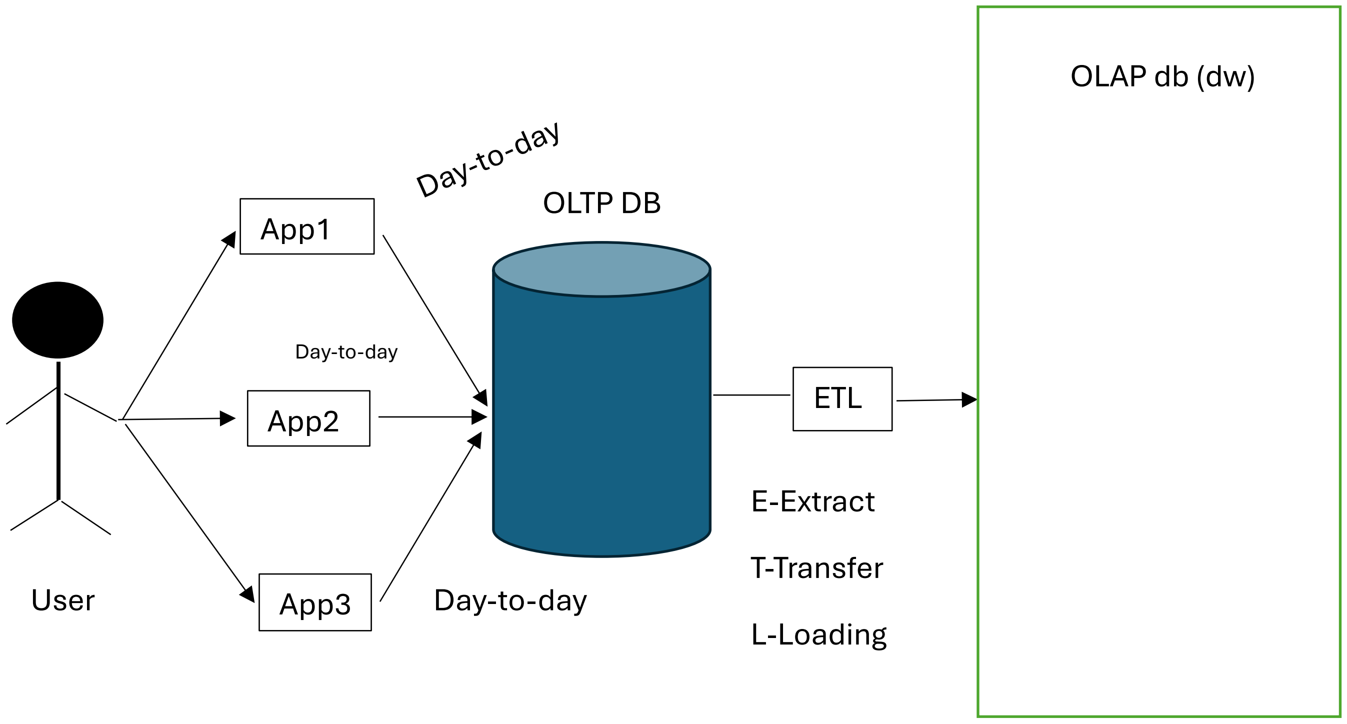
->These databases are used for saving/storing “day-to-day” transactional information.

Ex: - Oracle, SQL SERVER, MySQL, PostgreSQL, Db2, etc.

2.OLAP: -

->These databases are used for storing “historical data/information “. (i.e. Bigdata)

Ex: - Data Warehouse (DWH)



Ex: - Informatica, SSLS (msbi), power bi, Apache airflow.

=> What is DBMS?

->It is software which is used to manage and maintain data/information within the database memory.

->By using DBMS s/w we will perform the following operations are:

>Creating database
>Creating tables
>Inserting data
>Updating data
>Selecting data
>Deleting data
->Here DBMS s/w will act as an interface between user and database.

$$\begin{tikzcd}\text{USER} \arrow[r, leftrightarrow] & \text{DBMS s/w (Interface)} \arrow[r, leftrightarrow] & \text{Database}\end{tikzcd}$$

=> DBMS models?
->There are three types of DBMS models are there. Those are
1.Hierarchial Database Management System (HDBMS)
Ex: - IMS s/w (Information Management System).
2.Network Database Management System (NDBMS)
Ex: - IDBMS s/w (Integrated database management system)
NOTE: - HDBMS, NDBMS models are outdated in real time.
3. Relational Database Management System (RDBMS):
->There are two modules in RDBMS, those are
i. Object Relational Database Management System (ORDBMS)
ii. Object Oriented Database Management System (OODBMS)
i. ORDBMS: -
->Data can be stored/organized in the form of tables.
->A table is a collection of rows and columns
>Rows can be called as “record/tuples”.
>Columns can be called ad “attributes/fields”.
->A row is nothing but a group of columns in a table.
-> These databases are depending on “SQL”. So that these are called as “SQLDATABASES” in real-time.
Ex: - Oracle, SQL Server, MySQL, Db2, etc.
ii. OODBMS: -
->Data can be stored/organized in the form of “objects”.
->These databases are depending on “OOPS Concept “but not SQL. So that these are called as “NOSQL DATABASES” in real time.

TOPIC-2(ORACLE)

=> Introduction to Oracle: -

-> Oracle is an RDBMS product/ORDBMS module which was introduced by "Oracle Corporation" in 1979. Oracle is used to store data/information permanently and security.

->Oracle can be deployed in any platform like Windows, Linux, Unix, Solaris, MAC,. Etc.

->Oracle is a platform independent an RDBMS product.

=>What is platform?

->It is combination of operating systems and Micro Processor.

->There are two types of platforms. Those are

i. Platform dependent
ii. Platform independent

i. Platform dependent: -

->It supports only one operating system with the combination of any micro-processor.

Ex: - Cobal, Pascal, C++, C.

ii. Platform independent: -

->It supports any operating system with the combination of any micro-processor.

Ex: - Oracle, Java, .net core, Python, Etc.

=>Types of oracle software editions: -

There are two types of editions. Those are

i. Oracle Express Edition: -

->Supporting partial features of oracle.

Ex: - recycle bin, flash back, purge, partition, tables, are not allowed.

ii. Oracle Enterprise Edition: -

->Supporting all features of oracle.

Ex: - recycle bin, flashback, purge, partition, tables, are allowed.

=>Versions of oracle software's: -

->The first version of oracle software is "Oracle 1.0".

- Oracle 1.0
- Oracle 2.0
- Oracle 3.0
- Oracle 4.0

- Oracle 5.0
- Oracle 6.0
- Oracle 7.0
- Oracle 8.0
- Oracle 8i (Internet)
- Oracle 9i
- Oracle 10g (Grid technologies)
- Oracle 11g
- Oracle 12c (Cloud technologies)
- Oracle 18c
- Oracle 19c
- Oracle 20c
- Oracle 21c
- Oracle 23c (Beta Version)

=>How to download oracle 19c enterprise edition software: -

->Supports on WOS-10 or WOS-11. WOS [Windows Operating System]

->Supports on Hard Disk (HD)-500gb, SSD – 60, RAM – 1gb+.

->It's only valid for 90 days (3 months).

->For downloading oracle s/w we should create an account in oracle website.

->Download URL= (https://www.oracle.com/in/database/technologies/oracle 19c-windows-download.html).

=>How to install oracle 19c/21c enterprise edition software: -

->Follow the provided video in google classroom.

NOTE: -

->Once we installed oracle software there are two components installed in the system.

i. Client Component
ii. Server Component

i. Client Component: -

->By using client components, we will perform the following three steps:

Step-1: - User can connect to oracle server: -

Enter username : System (default username)
Enter password : LION (Created at installation level)
CONNECTED>>

Here, Username is not case-sensitive, but password is case-sensitive.

Step-2: - User can send requests to an oracle server: -

Request: SQL query/SQL Command.

Step-3: - User can get response from oracle server: -

Response: Result/Output.

Ex: - Client tools = {SQL PLUS [CUI (Character user interface) Environment, Manual Coding, using this s/w for whole oracle course], SQL Developer, Toad [These two are GUI (Graphical user interface) Environment]

ii. Server Components: -

->There are two more sub-components in Server.

a. Instance

b. Database

a. Instance: -

->It is a temporary memory which will be allocated from RAM (Random Access Memory).

->Here, data can be stored temporarily

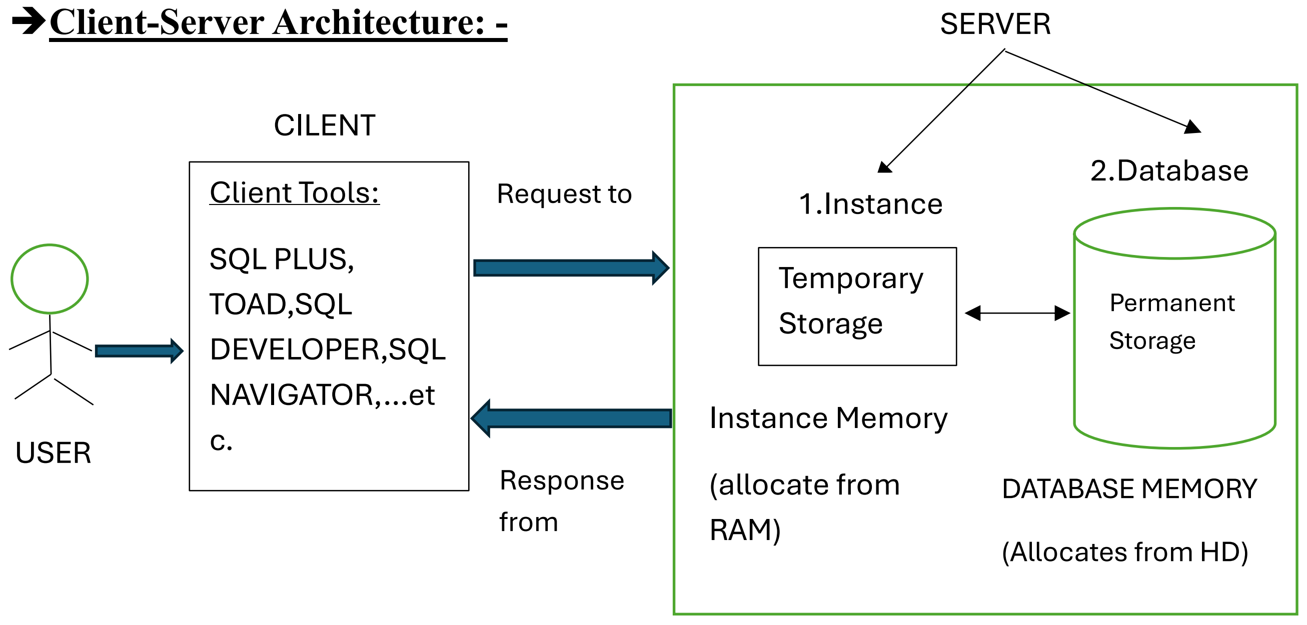
b. Database: -

->It is a permanent memory which will be allocated from Hard Disk.

->Here, data can be stored permanently.

Ex: - ATM (Automatic Teller Machine) is the real time example.

➔**<u>Client-Server Architecture: -</u>**



NOTE: -

->When we want to work with oracle database server. We need to follow the following two procedures

Step-1: Connect

Step-2: Communicate

Step-1: - Connect:

->When we want to connect to oracle, we need a client tool is known as "SQLPLUS"

Step-2: - Communicate:

->When we want to communicate with database we need a database language is known as "SQL".

➔SQL PLUS vs SQL: -

SQL PULS	SQL
1.It is a client tool which was introduced by "Oracle corporation".	1.It is a database language which was introduced by "IBM".
2.It is used to connect to an oracle server.	2.It is used to communicate with database.
3.It will act as an editor for writing & executing SQL queries and PL/SQL programs	3. SQL contains the following five sub-languages are (DDL, DML, DQL/DRL, TCL, DCL) used to perform some operations over database.

=> How to connect to oracle server: -

> Go to all programs

> Open oracle – oraDB19home | folder

> Click on SQLPLUS icon

Enter username: SYSTEM

Enter password: LION

Connected.

=> How to create a new username and password in oracle:

Syntax: -

Create user <username> identified by <password>;

Ex: -

```
SQL> CONN
        Enter username: SYSTEM/LION
        Connected.
SQL> CREATE USER MYDB9AM IDENTIFIED BY 123.
        User created.
SQL> CONN
        Enter username: MYDB9AM/123
Error:
ORA-01045: User MYDB(AM lacks CREATE SESSION privileges;
        Logon denied.
```

NOTE: -

->In oracle database server every new user is called as "dummy user" (i.e., no permission)

If the user want to connect to oracle server then the user need a permission from DBA (System).

=> How to give all permissions to user:

Syntax:

```
Grant <privilege name> to <username>;
```

Ex:

```
SQL> CONN
        Enter username: SYSTEM/LION
        Connected.
SQL> GRANT DBA TO MYDB9AM ;
        Grant succeeded.
SQL> CONN
        Enter username: MYDB9AM
        Enter password: 123
        Connected.
```

=> How to change password for user:

Syntax:

```
Password;
```

Ex:-

```
SQL> CONN
        Enter username: MYDB9AM/123
        Connected.
SQL> PASSWORD;
        Changing password for MYDB9AM
        Old password: 123
        New password: ABC
```

```
                    Retype new password: ABC
                    Password changed.
          SQL> CONN
                    Enter username; MYDB9AM/ABC
                    Connected.
```
=> How to re-create a new password if we forget it:

```
          Syntax:
                    Alter user <username> identified by <new password>;
          Ex:-
          SQL> CONN
                    Enter username: SYSTEM/LION
                    Connected.
          SQL>ALTER USER MYDB9AM IDENTIFIED BY MYDB9AM;
                    User altered.
          SQL> CONN
                    Enter username: MYDB9AM/MYDB9AM
                    Connected.
```
=> How to view usernames in oracle if we forget it:

```
          Syntax:
                    Select username from all_users;
          Ex: -
          SQL> CONN
                    Enter username: SYSTEM/LION
                    Connected.
          SQL> SELECT USERNAME FROM ALL_USERS;
```
=> How to drop a user:

```
          Syntax:
                    Drop user <username> cascade;
          Ex: -
          SQL> CONN
                    Enter username: SYSTEM/LION
                    Connected.
          SQL> DROP USER MYDB9AM CASCADE;
                    User dropped.
```
=> How to clear the screen of SQL plus editor:

```
          Syntax:
                    Cl scr;
```

Ex: -

SQL> CL SCR;

=> How to disconnect from oracle server:

Syntax:

Exit;

Ex: -

SQL> EXIT;

==

TOPIC-3(SQL)

➔Introduction to SQL: -

->SQL [Structured Query Language] is database language which was introduced by "IBM".

->SQL is used to communicate with any database in real-time.

Ex:- Oracle, MySQL, SQL server, db2, Sybase, PostgreSQL, etc.

-> Initially SQL is called as "SEQUEL" language and later renamed as "SQL".

-> SQL is not a case-sensitive language. i.e. user can write SQL queries in either upper/lower/combination of upper and lower case characters.

Ex: -

```
SELECT * FROM EMP; -----------→executed
Select * from emp; ------------------→executed
Select * From Emp; -----------------→executed
```

-> In oracle storage of data is a case-sensitive.

-> Every SQL query should ends with ";" (Semi-colon).

➔Sub – Languages of SQL :

1. Data Definition Language (DDL) :

-> CREATE

-> ALTER

> ALTER – MODIFY

> ALTER – ADD

> ALTER – RENAME

> ALTER – DROP

-> RENAME

-> TRUNCATE

-> DROP

==>New features in oracle-10g enterprise edition are :

-> RECYCLEBIN

-> FLASHBACK
->PURGE
2. Data Manipulation Language (DML) :
-> INSERT
-> UPDATE
-> DELETE
3. Data Query/Retrieval Language (DQL/DRL) :
-> SELECT (read only)
4. Transaction Control Language (TCL) :
-> COMMIT
-> ROLLBACK
-> SAVEPOINT
5. Data Control Language (DCL) :
-> GRANT
-> REVOKE
==

1. (DDL) :
=> CREATE: -
-> It is used to create a new database object such as Table, View sequence, Index, Procedure, Function, Triggers, etc.
* How to create a new table in oracle :
Syntax:
Create table <table name>(<Column name 1)<data type>[size], <Column name 2)<data type>[size], <Column name 3)<data type>[size],…);
* Data types in oracle:
-> Data type is an attribute which is used to store "What type data" into a column.
-> Oracle supports the following datatypes are,
>Number datatype
>Character/String datatype
>Long datatype
>Date datatype
> Raw & long raw datatype
> LOB datatype
i. Number datatype:
a. Number (p)
b. Number (p,s)

a. Number (p):

-> To store integer format data only.

b. Number (p,s):

-> To store integer and also float values.

Precision (p) :

-> Counting all digits including left and right sides of a decimal point in the expression.

-> The maximum size of precision is 38 digit.

Ex:-

126	➔	p=3
756323	➔	p=6
45.23	➔	p=4
9585.35	➔	p=6

Scale (s):

-> Counting the right-side digits of a decimal point in the expression.

-> There is no maximum size of scale because it is a part of precision value.

Ex: -

45.23	➔	p=4, s=2
9585.351	➔	p=7, s=3

NOTE: - Here the scale should be less than the precision value.

Ex: -

Sno number (3)	PRICE number (8,2)
0	0.0
1	25.18
2	85.17
.	.
.	.
.	.
.	.
999	999999.99
1000 – error	1000000(1000000.00) – error

ii. Character/String datatype: -

-> Storing string format data only.

-> In database string can be represents with '<string>'.

Ex:- <u>ENAME Char (10)</u>

Smith	->	Error
'Smith'	->	Smith (Accepted)
1234	->	Error

'1234'	->	1234
34.12	->	Error (Accepted)
'34.12'	->	34.12 (Accepted)

String Format
||

Characters only String Format	Alphanumeric String Format
||	||
[A-Z, a-z]	[A-Z, a-z, 0-9, @,$,#,.etc]
Ex:- 'smith', 'SMITH', .etc	Ex:-'Smith123@gmail.com' ,Pan Card, HTNO,.etc..

Types Of String Datatypes: -

-> These datatypes again classified into two types.

1. Non-unicode datatype: -

-> Supplying to store "localized data".(i.e., English Language only)

i. Char (size)
ii. Varchar2 (size)

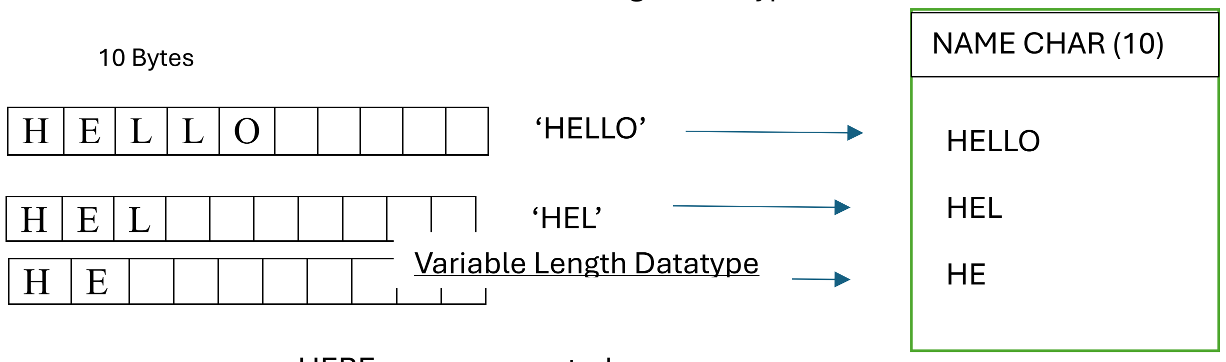
i. Char (size): -

->It is a fixed length datatype(i.e., Static memory)
-> It will store non-unicode character in the form of 1 char = 1 byte.
-> The maximum size of char datatype is 2000 bytes.
=> <u>Disadvantage: -</u>
-> Memory wasted because of it is static.

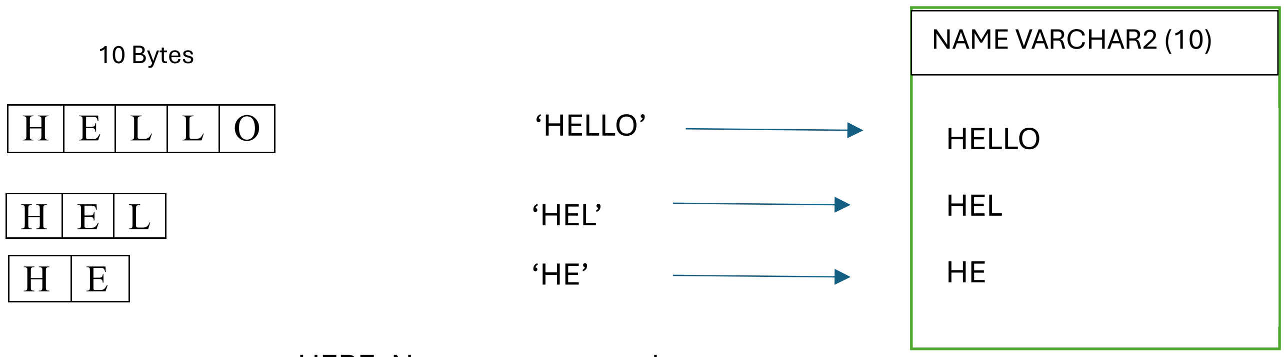
ii. Varchar2 (size): -

-> It is a variable length datatype (i.e dynamic memory)
-> It will store non-unicode characters in the form of 1 char = 1 byte
-> The maximum size of varchar2 datatype is 4000 bytes.
=><u>Advantage: -</u>
-> Memory saved not wasted.

<u>Fixed Length Datatype</u>



HERE, memory wasted.



HERE, No memory wasted.

2. Unicode datatypes: -

-> Supporting to store "globalized data" (i.e. All National Language)

i. NChar (size)

ii. NVarchar (size)

Here, N stands for 'National Language'-[196 languages]

i. NChar (size) :

-> It is a fixed length datatype (i.e, Static memory)

-> It will store Unicode characters in the form of 1 Char = 1 byte.

-> The maximum size of NChar datatype is 2000 bytes.

Disadvantages: -

->Memory wasted.

ii. NVarchar2 (size) :

-> It is a variable length datatype (i.e, dynamic memory)

-> It will store Unicode characters in the form of 1 char = 1 byte.

-> The maximum size of Nvarchar2 datatype is 4000 bytes.

Advantages: -

-> Memory saved.

* Long datatype: -

-> It is a variable length datatype(i.e, dynamic memory)

-> It will store both non-unicode & Unicode characters in the form of 1 char = 1 byte.

-> The maximum size of long datatypes is 2gb.

* Date datatype: -

-> To store date and time information of a particular day/transaction.

-> The range of date datatype in oracle is '01-JAN-4712'BC to '31-DEC-9999' AD.

-> There are two types of date datatypes

a. Date

b. Timestamp

a. Date :-

-> It will store date and time information but time is optional.

-> When user did not insert time in oracle will take time '00:00:00 am' by default.

-> By default format of oracle is 'DD:MON:YY/YYYY HH:MI:SS'

-> The maximum size is 7 bytes and it is a fixed memory.

Ex: - 'DD-MON-YY/YYYY HH:MI:SS'
'04-JAN-25/2025 00:00:00
1 1 2 1 1 1 →7 bytes(fixed)

-> Even if we not provide time value it will take 7 bytes.

b. Timestamp: -

-> It will store date and time information along with milliseconds.

-> The default format of timestamp is 'DD:MON-YY/YYYY HH:MI:SS:MS'

-> The maximum size is 11 bytes and it is a fixed memory.

Ex: - 'DD-MON-YY/YYYY HH:MI:SS:MS'
'01-JAN-25/2025 00:00:00:00'
1 1 2 1 1 1 4 →11 bytes

* Raw and Long Raw Datatypes: -

-> These datatypes are used for storing image file/audio file/video files in the form of 01010101010 binary format.

>Raw	- Static datatype	- 2000 bytes
>Long Raw	- Dynamic datatype	- 2gb

* LOB datatype: -

-> LOB stands for large objects datatype

i. BLOB
ii. CLOB
iii. NCLOB

i. BLOB :-

-> It stands for binary large object.

-> It will store image/audio/video files in the form of 010101010001 binary format.

-> It is dynamic datatype.

-> The maximum size is 4gb.

ii. CLOB :-

-> It stands for Character Large Object.
-> It will store non-unicode characters in the form of 1 char = 1 byte
-> It is a dynamic datatype.
-> The maximum size is 4gb.

iii. NCLOB :-
-> It stands for National Character Large Object.
-> It will store Unicode characters in the form of 1 char = 1 byte.
-> It is a dynamic datatype.
->The maximum size is 4gb.

➔Non-Unicode Characters:-

➢ Char (size)	-	2000 bytes
➢ Varchar (size)	-	4000 bytes
➢ Long	-	2gb
➢ CLOB	-	4gb

➔Unicode Characters:-

➢ NChar (size)	-	2000 bytes
➢ NVarchar (size)	-	4000 bytes
➢ Long	-	2gb
➢ NCLOB	-	4gb

➔Binary data:-

➢ Raw	-	2000 bytes
➢ Long Raw	-	2gb
➢ BLOB	-	4gb

* How to create a new table in oracle :-
Syntax:-
Create table <table name>(<column name1><datatype>[size],<column name2><datatype>[size],…);
Ex:-
```
SQL>CONN
Enter user-name: MYDB9AM/MYDB9AM
Connected
SQL> CREATE TABLE STUDENT(STID NUMBER(4),SNAME CHAR(8),SFEE(8,2));
Table created.
```

* How to view the structure of a table in oracle:-

Syntax:-

Desc <table name>; (describe command)

Ex:-

SQL> DESC STUDENT;

* How to view list of table in oracle window:-

Syntax:-

Select * from tab; (TAB is pre-defined table)

Ex:-

SQL> SELECT * FROM TAB;

==> ALTER:

-> It is used to modify/change the structure of a table.

-> There are four sub-commands of alter.

i. ALTER-MODIFY
ii. ALTER-ADD
iii. ALTER-RENAME
iv. ALTER-DROP

i. ALTER-MODIFY:

-> To change datatype from one datatype to another datatype and also the size of the datatype of a specific datatype.

Syntax:-

ALTER TABLE<TABLE NAME>MODIFY<COLUMN NAME><NEW DATATYPE>[NEW SIZE];

Ex:

SQL> ALTER TABLE STUDENT MODIFY SNAME VARCHAR2(20);

ii. ALTER-ADD:

->To add new column to an existing table

Syntax:

ALTER TABLE <TABLE NAME>ADD NEW<NEW COLUMN NAME><DATATYPE>[SIZE];

Ex:

ALTER TABLE STUDENT ADD SADDRESS VARCHAR2(50);

iii. ALTER-RENAME:

->To rename a column name in the table.

Syntax:

ALTER TABLE<TABLE NAME>RENAME COLUMN<OLD COLUMN NAME>TO<NEW COLUMN NAME>;

Ex:

ALTER TABLE STUDENT RENAME COLUMN SNAME TO STUDENT NAMES;

iv. ALTER-DROP:

->To drop/delete a column from an existing table.

Syntax:

ALTER TABLE<TABLE NAME>DROP COLUMN<COLUMN NAME>;

Ex:

ALTER TABLE STUDENT DROP COLUMN SFEE;

==> RENAME:

->To change a table name.

Syntax:

RENAME<OLD TABLE NAME>TO<NEW TABLE NAME>;

Ex:

SQL> RENAME STUDENT TO STUDENT_DETAILS;

SQL> RENAME STUDENT_DETAILS TO STUDENT;

==> TRUNCATE:

->Deleting all rows but not columns of table.

->We cannot delete a specific row from a table by using TRUNCATE because it does not allow "WHERE" clause condition.

->Deleting all rows from a table permanently

Syntax:

TRUNCATE TABLE<TABLE NAME>;

Ex:

SQL> TRUNCATE TABLE STUDENT WHERE STID=1001; --NOT ALLOWED.

SQL> TRUNCATE TABLE STUDENT; --ALLOWED.

==> DROP:

-> To drop the entire table. (i.e., collection of rows & columns)

Syntax:

DROP TABLE<TABLE NAME>;

Ex:

SQL> DROP TABLE STUDENT;

-> Before oracle-10g enterprise edition once we drop a table from a database then it was permanently deleted. Whereas from oracle-10g enterprise once we drop a table from a database then it was temporarily deleted.

* New features in the oracle-10g enterprise edition:

- RECYCLEBIN
- FLASHBACK
- PURGE

==> RECYCLEBIN:

-> It is a system defined table which is used to store the information about deleted tables from database.

-> It is a similar to recyclebin in a computer.

*. How to view deleted tables in recyclebin;

Syntax:

```
SELECT OBJECT NAME, ORIGINAL_NAME FROM RECYCLEBIN;
```

```
        OBJECT_NAME                        ORIGINAL_NAME
BIN$LMtpjn+7TWOIFBg2W+UBIQ==$O             STUDENT
```

==> FLASHBACK:

-> It is a DDL command which is used to restore a table from recyclebin to database.

Syntax;

```
FLASHBACK TABLE<TABLE NAME>TO BEFORE DROP;
```

Ex:

```
SQL> FLASHBACK TABLE STUDENT TO BEFORE DROP;
```

==> PURGE:

->It is DDL command which is used to drop a table permanently.

Syntax:

```
DROP TABLE<TABLE NAME>PURGE;
```

Ex:

```
SQL> DROP TABLE STUDENT PURGE;
```

2. DATA MANIPULATE LANGUAGE(DML):

==> INSERT:

-> To insert a new row.(i.e., data) into a table.

-> There are two methods to insert data into a table.

METHOD-1:

-> To insert values for all columns.

Syntax:

```
INSERT INTO<TABLE NAME>VALUES(VALUE 1,VALUE 2,…);
```

Ex:

```
SQL> INSERT INTO STUDENT VALUES(1021,’SMITH’,25000);
```

METHOD-2:

-> To insert values for required columns.

Syntax:

INSERT INTO<TABLE NAME>(REQUIRED COLUMN NAMES)VALUES(VALUE 1,VALUE 2,……);

Ex:

```
SQL> INSERT INTO STUDENT(STID) VALUES(1022);
SQL> INSERT INTO STUDENT (STID, SFEE) VALUES (1023,32000);
SQL> INSERT INTO STUDENT (STID, SNAME, SFEE) VALUES(‘1024’,’ALLEN’,45000);
SQL> INSERT INTO STUDENT (SNAME, SFEE, STID) VALUES(MILLER’,18000,1025);
```

*How to insert values into a table dynamically:

METHOD-1:

Syntax:

INSERT INTO<TABLE NAME>VALUES<&<COLUMN NAME1>,&<COLUMN NAME2>,……);

Ex:

```
SQL>INSERT INTO STUDENT VALUES(&STID,’&SNAME’,&SFEE);
Enter value for stid:1026
Enter value for sname:JAMES
Enter value for sfee: 58000
SQL>/
Enter value for stid: 1027
Enter value for sname:SCOTT
Enter value for sfee: 38000
```

METHOD-2:

Syntax:

INSERT INTO<TABLE NAME>(REQUIRED COLUMN NAMES)VALUES(&<COLUMN NAME1>,&<COLUMN NAME2.,…..);

Ex:

```
SQL>INSERT INTO STUDENT(STID)VALUES(&STID);
Enter value for stid: 1029
SQL>/
Enter value for stid: 1030
SQL>/
---------------------------------
```


==> UPDATE:

-> To update all rows data in a table at a time.

(or)

-> To update a specific row data in a table by using “WHERE” clause condition.

Syntax:

UPDATE<TABLE NAME>SET<COLUMN NAME1>=<VALUE1>,<COLUMN NAME2>=<VALUE2>,……..[WHERE <CONDITION>];

Ex:

SQL>UPDATE STUDENT SET SNAME=’SMITH’ WHERE STID=1023;

SQL>UPDATE STUDENT SET SNAME=’FORD’,SFEE=15000 WHERE STID=1021;

SQL>UPDATE STUDENT SET SFEE=25000;

==> DELETE:

->To delete all rows from a table at a time.

(or)

-> To delete a specific row from a table by using “WHERE” conditions.

Syntax:

DELETE FROM<TABLE NAME>[WHERE<CONDITION>];

Ex:

SQL>DELETE FROM STUDENT WHERE SNAME=’SMITH’;

SQL>DELETE FROM STUDENT;

NOTE: -Using DELETE we can delete all rows from table but not deleting the structure of the table.

***Difference between DELETE vs TRUNCATE:

DELETE	TRUNCATE
1.It is a DML operation.	1. It is a DDL operation.
2.Deleting specific row.	2.Cannot delete a specific row.
3.Supporting "WHERE" clause	3.Does not support "WHERE" clause.
4.Delete data temporarily.	4.Deleting data permanently.
5.We can restore deleted data by using "ROLLBACK".	5.We cannot restore deleted data by using "ROLLBACK'.
6.Deleting rows in one-by-one manner.	6.Deleting rows as a page wise.
7.Execution speed is slow.	7.Execution speed is fast.

3.(DQL/DRL)Data Query/Retrieval Language:
==>SELECT:
-> To retrieve all rows from a table at a time.
(or)
-> To retrieve a specific row from a table by using "WHERE" condition.
Syntax:
SELECT*/<LIST OF COLUMN NAMES>FROM<TABLE NAME>[WHERE<CONDITION>];
Ex:
SQL>SELECT * FROM EMP WHERE JOB='MANAGER';
SQL>SELECT * FROM DEPT WHERE DNAME='SALES';
SQL>SELECT * FROM DEPT;
SQL>SELECT * FROM EMP;
*Operators:
==>CONCATENATION OPERATOR(||):
-> To add two or more than two expressions.
Syntax:
<EXPRESSION 1>||<EXPRESSION 2>||<EXPRESSION 3>||,….
Ex:
SQL>SELECT 'THE EMPLOYEE'||' '||ENAME||' '||'IS WORKING AS A'||' '||JOB FROM EMP;
==>DISTINCT keyword:
->To eliminate duplicate values from a column temporarily.
Syntax:
DISTINCT<COLUMN NAME>

Ex:

SQL>SELECT DISTINCT JOB FROM EMP;

SQL>SELECT DISTINCT DEPTNO FROM EMP;

==> ALIAS NAME:

->It is a temporary name/alternate name for column and table.

->Alias names are created at two levels.

i.Column level alias names:

->In this level we are creating alias name for columns.

ii. Table level alias names:

->In this level we are creating alias name for table.

Syntax:

SELECT<COLUMN NAME>[AS]<COLUMN ALIAS NAME1>,<COLUMN NAME2>[AS]<COLUMN ALIAS NAME2>,…. FROM <TABLE NAME>[AS]<TABLE ALIAS NAME>;

Ex:

SQL>SELECT DEPTNO AS DEPARTMENT_ID,DNAME AS DEPARTMENT_NAME,LOC AS DEPARTMENT_LOCATION FROM DEPT DEPARTMENT;

(or)

SQL>SELECT DEPTNO DEPARTMENT_ID,DNAME AS DEPARTMENT_NAME,LOC DEPARTMENT_LOCATION FROM DEPT DEPARTMENT;

=>NOTE:

->When we want to display the large scale data tables in proper systematically in SQLPLUS editor then we must set the following two properties are,

1. PAGESIZE n
2. LINES n

1. PAGESIZE n:

->By default each page contains 14 rows in SQLPLUS editor.

->If we want to display more than 14 rows in a single page then we set pagesize property.

->Here ‘n’ is represent no of rows in a page and the maximum size of pagesize property is 5000 rows.

Ex:SQL>SET PAGESIZE 100;

2. LINES n:

->By default each line contains 80 bytes(i.e 80 chars) in SQLPLUS editor.

->If we want to display more than 80 characters in a single line then we set line property.

->Here 'n' is nothing but no of bytes/characters in a line.

-.Maximum bytes/characters of a line is 32767 bytes/characters.

Ex:SQL>SET LINES 160;

===

*OPERATORS IN ORACLE SQL:

->To perform some operations on the given operand value.

->Oracle supports the following operators those are,

➔Assignment operators	->	=	
➔Arithmetic operators	->	+,-,*,/	
➔Relational operators	->	<,>,<=,>=,!=,(or)<>	
➔Logical operators	->	AND,OR,NOT	
➔Set operators	->	UNION,UNION ALL, INTESECT,MINUS.	
➔Special operators	->	(+ve)	(-ve)
		IN	NOT IN
		BETWEEN	NOT BETWEEN
		IS NULL	IS NOT NULL
		LIKE	NOT LIKE

=> Assignment operators:

->To assign a values to an attribute/to a variable.

Syntax: <column name><assignment operator><value>

Ex:

SQL>UPDATE EMP SET SAL=34000;

SQL>UPDATE EMP SET JOB='HR' WHERE EMPNO=7788;

=>Arithmetic operators:

->To perform addition, subtraction, multiple and division.

Syntax:<column name><arithmetic operator><value>

Ex: Write a query to display all employees salaries after adding 1000/-?

SQL>SELECT ENAME, SAL AS OLD_SALARY, SAL+1000 AS NEW_SALARY FROM EMP;

Ex: Write a query to display EMPNO, ENAMES, DEPTNO, BASIC_SALARY and ANNUAL_SALARY of the employees who are working under deptno is 10?

```
SQL>SELECT EMPNO, ENAME, DEPTNO, SAL AS BASIC_SALARY,
2       SAL*12 AS ANNUAL_SALARY FROM EMP WHERE DEPTNO=10;
```

Ex: Write a query to display employee's salaries after increment of 5%?

```
SQL>SELECT ENAME, SAL AS BEFORE_INCREMENT,
2       SAL+SAL*5/100 AS AFTER_INCREMENT FROM EMP;
```

Ex: Write a query to display EMPNO, ENAME, BASIC_SALARY, 5% of HRA, 10% of DA, 5% of PF, GROSS_SALARY and NET_SALARY of employees who are working as a MANAGER?

```
SQL>SELECT EMPNO, ENAME, SAL AS BASIC_SALARY,
2       SAL*0.05 AS HRA, SAL*0.1 AS DA, SAL*0.05 AS PF,
3       SAL+SAL*0.05+SAL*0.1+SAL*0.05 AS GROSS_SALARY,
4       SAL+SAL*0.05+SAL*0.1-SAL*0.05 AS NET_SALARY,
5       FROM EMP WHERE JOB='MANAGER';
```

Ex: Write a query to display employee's salaries after decrement of 5%?

```
SQL> SELECT ENAME, SAL AS BEFORE_DECREMENT,
2       SAL-SAL*0.05 AS AFTER_DECREMENT FROM EMP;
```

=>Relational operators:

->Comparing a specific column value with user defined condition in the query.

Syntax:

Where<column name><relational operator><value>

Ex: Write a query to display the list of employees who are joined after 1981?

```
SQL>SELECT * FROM EMP WHERE HIREDATE>'31-DEC-1918';
```

Ex: Write a query to display the list of employees who are joined before 1981?

```
SQL>SELECT * FROM EMP WHERE HIREDATE<'01-JAN-1981';
```

=>Logical operator:

->To check more than one condition in the query.

->AND, OR, NOT.

=>> AND operator:

->It return a value if both conditions are TRUE in the query.

Condition-1	Condition-2		
T	T	=>	T
T	F	=>	F
F	T	=>	F
F	F	=>	F

Syntax: where<condition 1> and <condition 2>

Ex: Write a query to display employees whose name is 'SMITH' and working as 'CLERK'?

SQL>SELECT * FROM EMP WHERE ENAME='SMITH' AND JOB='CLERK';

=>>OR operator:

-> It return a value if any one condition is TRUE in the query.

Condition-1	Condition-2		
T	T	=>	T
T	F	=>	T
F	T	=>	T
F	F	=>	F

Syntax: where<condition 1>or<condition 2>

Ex: Write a query to display employees details who are working under the job is 'ANALYST','PRESIDENT'?

SQL> SELECT * FROM EMP WHERE JOB='ANALYST' OR JOB='PRESIDENT';

=>>NOT operator:

->It return all values from a table except the given conditional values in the query.

Syntax: not<condition 1>and not<condition 2>

Ex: Write a query to display employees who are not working under deptno 10 and 30?

SQL>SELECT * FROM EMP WHERE NOT DEPTNO=10 AND NOT DEPTNO=30;

=> SET operators:

->Set operators are used to combined the results of two select quries.

Syntax: <select query 1><set operator><select query 2>

Ex: A={10,20,30}, B={30,40,50}

-UNION:

->To combined two sets values without duplicates.

A U B={10,20,30,40,50}

-UNION ALL:

->To combined two sets values with duplicates.

A UL B={10,20,30,30,40,50}

-INTERSECT:

->It returns common values from both sets.

A | B={30}

-MINUS:

->It returns uncommon values from the left side but not the right side set.

A – B={10,20}

B – A={40,50}

<u>DEMO TABLES:</u>

SQL> SELECT * FROM EMP_HYD;

EID	ENAME	SAL
1021	SMITH	85000
1022	ALLEN	65000
1023	WARD	38000

SQL> SELECT * FROM EMP_MUMBAI;

EID	ENAME	SAL
1021	SMITH	85000
1024	MILLER	69000

Ex: Write a query to display employees who are working in both branches?

SQL>SELECT * FROM EMP_HYD INTERSECT SELECT * FROM EMP_MUMBAI;

Ex: Write a query to display employees who are working in HYD but not in MUMBAI?

SQL> SELECT * FROM EMP_HYD MINUS SELECT * FROM EMP_MUMBAI.

Ex: Write a query to display all employees details who are working in the organization?
SQL> SELECT * FROM EMP_HYD UNION ALL SELECT * FROM EMP_MUMBAI; (including duplicates).
SQL> SELECT * FROM EMP_HYD UNION SELECT * FROM EMP_MUMBAI; (excluding duplicates)

=> Special operators:
*IN operator:
->Comparing group of values with a single condition.
Syntax:
Where<column name> in (<list of values>);➔(+ve)
Where<column name>not in (<list of values>);➔(-ve)
Ex: Write a query to display the list of employees who are working as “CLERK”,”SALESMAN”,”MANAGER”? ➔(+VE)
SQL>SELECT * FROM EMP WHERE JON IN (‘CLERK’,’SALESMAN’,’MANAGER’);

Ex: Write a query to display the list of employees who are not working as “CLERK”,”SALESMAN”,”MANAGER”?
SQL>SELECT * FROM EMP WHERE JON IN (‘CLERK’,’SALESMAN’,’MANAGER’);

*BETWEEN Operator:
->Comparing a particular range value.
Syntax:
Where<column name>between<low value>and<high value>; ➔(+ve)
Where<column name>not between<low value>and<high value>; ➔(-ve)
NOTE:
->Between operator will return values including source and destination value from the given range value.
Ex: Write a query to display list of employees who are joined in 1981?
SQL>SELECT * FROM EMP WHERE HIREDATE BETWEEN ’01-JAN-1981’ AND ’31-DEC-1981’;➔(+ve)

Ex: Write a query to display list of employees who are not joined in 1981?
SQL>SELECT * FROM EMP WHERE HIREDATE NOT BETWEEN '01-JAN-1981' AND '31-DEC-1981';➔(-ve)

*IS Operator:

-> Comparing nulls in a table.

Syntax:

Where<column name>is null; ➔(+ve)

Where<column name>is not null; ➔(-ve)

Ex: Write a query to display the list of employees whose commission is undefined/unknown/null? ➔(+ve)
SQL>SELECT * FROM EMP WHERE COMM IS NULL;

Ex: Write a query to display the list of employees whose commission is not undefined/unknown/null? ➔(-ve)
SQL>SELECT * FROM EMP WHERE COMM IS NOT NULL;
** What is NULL?

->It is a unknown/a undefined value/a empty.

->Null! = 0 and Null! = space.

->When we perform arithmetic operations with null then it returns null only.

```
Ex:   If     X = 100;
      i)     X+NULL = 100+NULL = NULL
      ii)    X-NULL = 100-NULL = NULL
      iii)   X*NULL = 100*NULL = NULL
      iv)    X/NULL = 100/NULL = NULL
```

Ex: Write a query to display EMPNO, ENAME, SAL, COMM and SAL+COMM of the employees whose name is "SMITH"?
SQL>SELECT EMPNO, ENAME, SAL, COMM, SAL+COMM FROM EMP WHERE ENAME='SMITH';
Output: -

```
EMPNO    ENAME    SAL  COMM    SAL+COMM
7369     SMITH    800
```

->In the above example the employee "SMITH" is not taking any commission. So that commission + salary is 800 only but it returns NULL.

->To overcome the above problem then we must use a pre-defined function in oracle is “NVL ()”.

**What is NVL (Exp1, Exp2):

->NVL stands for NULL VALUE.

->It is used to replace a user defined value in place of NULL.

->This function is having two arguments those are expression 1 and expression 2.

>If Exp1 is NULL then it return Exp2 value (user defined value).

>If Exp1 is NOT NULL then it return Exp1 value.

Ex:

```
SQL>SELECT NVL(NULL,0) FROM DUAL; -----------------→0
SQL>SELECT NVL(NULL,500) FROM DUAL; --------------→500
SQL> SELECT NVL (0,500) FROM DUAL; -------------------→0
SQL> SELECT NVL (100,500) FROM DUAL; ----------------→100
```

SOLUTION:

```
SQL> SELECT EMPNO, ENAME, SAL, SAL + NVL (COMM,0) FROM EMP
WHERE ENAME = “SMITH”;
```

Output: -

```
    EMPNO     ENAME     SAL       SAL+COMM
    7369      SMITH     800       800
```

**NVL2(Exp 1, Exp 2, Exp 3):

-> It is an extension of NVL ().

-> This function is having three arguments are expression 1, expression 2 and expression 3.

>If Exp1 is NULL then it returns Exp3 (user defined value).

>If Exp1 is NOT NULL then it returns Exp2 value (user defined value).

Ex:

```
SQL>SELECT NVL2 (NULL,0,100) FROM DUAL; --------------→100
SQL>SELECT NVL2 (500,0,100) FROM DUAL; -----------------→0
```

Ex: Write a query to update employees commission in a table based on the following conditions:

Cond-1: - If employee commission is null then update those employees commission as 800.

Cond-2: - If employee commission is not null then update those employees commission as COMM+400.
SQL>UPDATE EMP SET COMM = NVL2(COMM, COMM+400, 800);

*LIKE Operator:
->Comparing a specific string character pattern.
->When we use LIKE operator, we must use the following two wildcard operators are,

i) %	->It represents the remaining group of characters after selected character.
ii) _	->Counting a single character from the expression.

Syntax:
(+ve) -> where<column name>like'[<wildcard operator>]<specific character pattern>[<wildcard operator>]';
(-ve) -> where<column name>not like'[<wildcard operator>]<specific character pattern>[<wildcard operator>]';

Ex: Write a query to fetch employees whose name start with "S" character?
SQL>SELECT * FROM EMP WHERE ENAME LIKE 'S%';

```
S        %
=        =
S        MITH
S        URESH
S        UMAN
S        OCTT
```

Ex: Write a query to fetch employees whose name is having 4 characters?
SQL>SELECT * FROM EMP WHERE ENAME LIKE '____';

```
ENAME
======
SMITH
                WARD
SURESH
                KING
SUMAN
                FORD
SCOTT
```

Ex: Write a query to fetch employees whose name having “I” character?
SQL> SELECT * FROM EMP WHERE ENAME LIKE ‘%I%’;

Ex: Write a query to fetch employees whose name having the second character is “O”?
SQL>SELECT * FROM EMP WHERE ENAME LIKE ‘_O%’;
Ex: Write a query to fetch employees who are joined in 1981?
SQL>SELECT * FROM EMP WHERE HIREDATE LIKE ‘%81’;

Ex: Write a query to fetch employees who are joined in the month of DECEMBER?
SQL>SELECT * FROM EMP WHERE HIREDATE LIKE ‘%DEC%’;

Ex: Write a query to fetch employees who are joined in the month of JUNE,DECEMBER?
SQL>SELECT * FROM EMP WHERE HIREDATE LIKE ‘%JUN%’ OR HIREDATE LIKE ‘%DEC%’;

Ex: Write a query to fetch employees who are joined in the month of DECEMBER in 1981?
SQL>SELECT * FROM EMP WHERE HIREDATE LIKE ‘%DEC%’ AND HIREDATE LIKE ‘%81’;

(OR)

SQL>SELECT * FROM EMP WHERE HIREDATE LIKE ‘%DEC-81’;

(OR)

SQL>SELECT * FROM EMP WHERE HIREDATE LIKE ‘%DEC_81’;

(OR)

SQL>SELECT * FROM EMP WHERE HIREDATE LIKE ‘%DEC%81’;

Ex: Write a query to fetch employees whose name not started with “S” character?
SQL>SELECT * FROM EMP WHERE ENAME NOT LIKE ‘S%’;

==

➔Functions in Oracle SQL:

->To perform some operation on the given operand values.

->Oracle supports two types of functions.

1. Pre-defined functions
   - > Use in SQL & PL/SQL
2. User-defined functions
   - >Use in PL/SQL

1. Pre-defined functions:

->These functions are also called as "built-in-functions" in oracle.

->It again divided into two types those are,

i. Single row functions (Scalar function)

ii. Multiple row functions (grouping function)

i. Single row functions (Scalar function) :

->These functions always return a single value.

>Numeric functions

>String/Character functions

>Date functions

>Null functions

>Conversion functions

>Analytical functions

* How to call a function in oracle:

Syntax: select <fname>(value(s)) from dual;

*What is dual?

->It is a pre-defined table in oracle.

->It is used to test functionalities of a function.

->It contains a single row and a single column.

->It is also called as "dummy table" in oracle.

*To view the structure of dual table:

Syntax: - desc<table name>

Ex: SQL>DESC DUAL;

*TO view data in dual table:

Syntax: select * from <table name>;

Ex: SQL>SELECT * FROM DUAL;

>Numeric function:
=>ABS() :
    ->To convert (-ve) sign values into (+ve) sign values.
Syntax: abs(n)
Ex:
SQL>SELECT ABS(-34) FROM DUAL; --------------→34
SQL>SELECT EMPNO,ENAME,SAL,COMM,ABS(COMM-SAL) AS TOTAL FROM EMP;

=>CEIL() :
    ->It return a value which is equal to (or) greater than to the given expression.
Syntax: ceil(n)
Ex:
SQL>SELECT CEIL(9.0) FROM DUAL; ------------------→9
SQL>SELECT CEIL(9.1) FROM DUAL; ------------------→10
Ex:
SQL>SELECT ENAME, SAL, CEIL(SAL*0.05) AS INCREMENT_AMOUNT FROM EMP;

=>FLOOR() :
    ->It return a value which is equals to (or) less than to the given expression.
Syntax: floor(n)
Ex:
SQL>SELECT FLOOR(9.0) FROM DUAL; -------------→9
SQL>SELECT FLOOR(9.9) FROM DUAL; -------------→9
Ex:
SQL>SELECT ENAME,SAL,FLOOR(SAL*0.05) AS INCREMENT_AMOUNT FROM EMP;

=>MOD() :
    ->It return the remainder value of the given expression.
Syntax: mod(m,n)
Ex:
SQL>SELECT MOD(10,2) FROM DUAL;

```
SQL>SELECT * FROM EMP WHERE MOD(EMPNO,2)=0; ->(even empno)
SQL>SELECT * FROM EMP WHERE MOD(EMPNO,2)=1; ->(odd empno)
```

=>POWER() :

->It return the power of the given expression.

Syntax: power(m,n)

Ex:

```
SQL>SELECT POWER(2,3) FROM DUAL; ----------------→8
SQL>SELECT ENAME,JOB,SAL,POWER(SAL,2) AS RESULT FROM EMP
WHERE JOB='CLERK';
```

=>ROUND() :

->It return the nearest value to the given expression.

->It return nearest value based on 0.5 expression.

>If expression is <0.5 then add 0 to the main expression

>If expression is =0.5 then add 1 to the main expression

>If expression is >0.5 then add 1 to the main expression

Syntax: round(<main expression>,[decimal places])

Ex:

```
SQL>SELECT ROUND(45.2) FROM DUAL; -----------→45
```

Solution:

```
        45.2 =>0.2<0.5 then add 0 to main expression
          +0
        ===
        45
        ===
```

```
SQL>SELECT ROUND(45.5) FROM DUAL; ------------→46
```

Solution:

```
        45.5 =>0.5=0.5 then add 1 to main expression
          +1
        ===
        46
        ===
```

```
SQL>SELECT ROUND(45.8) FROM DUAL; ------------→46
```

Solution:

```
        45.8 =>0.8>0.5 then add 1 to main expression
          +1
        ===
        46
        ===
```

SQL>SELECT ROUND(45.245,2) FROM DUAL; ------------→45.25

Solution:

```
        45.245 =>45.24=>0.5=0.5 then add 1 to main expression
                      +1
                     ===
                     45.25
                     ===
```

SQL>SELECT ROUND(45.240,2) FROM DUAL; ------------→45.24

Solution:

```
        45.240=>45.24=>0.0<0.5 then add 1 to main expression
                      +0
                     ===
                     45.24
                     ===
```

SQL>SELECT ROUND(45.249,2) FROM DUAL; ------------→45.25

Solution:

```
        45.249 =>45.24=>0.9>0.5 then add 1 to main expression
                      +1
                     ====
                     45.25
                     ====
```

Ex: SQL>SELECT ENAME,DEPTNO,SAL,ROUND(SAL*0.03) AS RESULT FROM EMP WHERE DEPTNO=30;

=>TRUNC () :

->It return an exact value from the given expression.

->It does not consider 0.5 value in the expression.

Syntax: trunc (<main expression>[,decimal places])
Ex:
SQL>SELECT TRUNC (45.3) FROM DUAL; -----------→45
SQL>SELECT TRUNC (45.8) FROM DUAL; -----------→45
SQL>SELECT TRUNC (45.82,1) FROM DUAL; -------→45.8
SQL>SELECT TRUNC (45.86,1) FROM DUAL; -------→45.8

>Character/String functions:
=>LENGTH () :
->It return the length of the given string.
Syntax : length(string)
Ex:
SQL>SELECT LENGTH('HELLO') FROM DUAL; ------------→5
SQL>SELECT LENGTH('WEL COME') FROM DUAL; ------→8 (count spaces also)
Ex:
SQL>SELECT ENAME,LENGTH(ENAME) AS NO_OF_CHARACTERS FROM EMP;
SQL>SELECT * FROM EMP WHERE LENGTH(ENAME)<5;
SQL>SELECT * FROM EMP WHERE LENGTH(ENAME)=5;
SQL>SELECT * FROM EMP WHERE LENGTH(ENAME)>5;

=>LOWER ();
->It converts upper case characters into lower case characters.
Syntax: lower(string)
Ex: SQL>SELECT LOWER ('HELLO') FROM DUAL; -------→hello
Ex:
SQL> UPDATE EMP SET ENAME=LOWER(ENAME) WHERE JOB='MANAGER';
SQL>UPDATE EMP SET ENAME-LOWER(ENAME);

=>UPPER ():
->It converts lower case characters into upper case characters.
Syntax: upper(string)
Ex: SQL>SELECT EMP SET ENAME=UPPER(ENAME);

=>INITCAP ():

->To convert the initial character is capital from the given string.

Syntax: initcap(string)

Ex:

SQL>SELECT INITCAP(‘hello’) FROM DUAL; --------→Hello

SQL>SELECT INITCAP(‘kamal kumar’) FROM DUAL; ----→Kamal Kumar

Ex:

SQL>UPDATE EMP SET ENAME=INITCAP(ENAME);

=>CONCAT ();

->To add two string expressions.

Syntax: concat(string1,string2)

Ex: SQL>SELECT CONCAT (‘GOOD’,’MORNING’) FROM DUAL;

→GOOD MORNING

Ex:

SQL>SELECT ENAME, CONCAT(‘Mr./Miss.’,ENAME) FROM DUAL;

SQL>UPDATE EMP SET ENAME=CONCAT(‘Mr.’,ENAME);

=>LTRIM ():

->To remove unwanted characters from the left side of the given string.

Syntax: ltrim(string, ‘trimming characters’)

Ex:

SQL>SELECT LTRIM(‘XXXSMITH’,’X’) FROM DUAL;

→SMITH

SQL>SELECT LTRIM(‘XYZSMITH’,’XYZ’) FROM DUAL;

→SMITH

=>RTRIM ():

->To remove unwanted characters from the right side of the given string.

Syntax: rtrim(string, ‘trimming characters’)

Ex:

SQL>SELECT RTRIM(‘SMITHXXX’,’X’) FROM DUAL;

→SMITH

SQL>SELECT RTRIM(‘SMITHXYZ’,’XYZ’) FROM DUAL;

→SMITH

=>TRIM ():

->To remove unwanted characters from both sides of the given string.

Syntax: trim('trimming character' from string)

Ex: SQL>SELECT TRIM('X' FROM 'XXSMITHXXX') FROM DUAL;

→SMITH

Ex:

SQL>SELECT TRIM('XY' FROM 'XYSMITHXY') FROM DUAL;

ERROR at line 1:

ORA-30001:trim set should have only one character.

=>LPAD ():

->Filling the required character on left side of the given string when string length is less than to user defined length.

Syntax: lpad(string,<user defined length>,'<required character>')

Ex:

SQL>SELECT LPAD('SMITH',1) FROM DUAL;

→S

SQL>SELECT LPAD('SMITH',1,'X') FROM DUAL;

→S

SQL>SELECT LPAD('SMITH',8,'X') FROM DUAL;

→XXXSMITH

=>RPAD ():

->Filling the required characters on right side of the given string when string length is less than to user defined length.

Syntax: rpad(string,<user defined length>,'<required character>')

Ex: SQL>SELECT RPAD('SMITH',8,'X') FROM DUAL;

→SMITHXXX

=>REPLACE ():

->To replace string to string/ string to character/ character to string.

Syntax: replace(string,'<old characters>','<new characters>')

Ex:

SQL>SELECT REPLACE('HELLO','ELLO','OD') FROM DUAL;

→HOD

SQL>SELECT REPLACE('HELLO','ELLO','D') FROM DUAL;

→HD

SQL>SELECT REPLACE('HELLO','L','OD') FROM DUAL;
→HEODODO
SQL>SELECT REPLACE('HELLO','L','D') FROM DUAL;
→HEDDO

=>TRANSLATE ():

->To translate character by character in the given string expression.

Syntax: translate(string,'<old characters>','<new characters?')

Ex:
SQL>SELECT TRANSLATE('HELLO','HLO','XYZ') FROM DUAL;
→XEYYZ

Here, H=X, L=Y, O=Z

SQL>SELECT TRANSLATE('HELLO','HLO','XY') FROM DUAL;
→XEYY

Here, H=X, L=Y, O=Removed.

=>SUBSTR ():

->fetching the required sub string from the given expression.

```
Syntax:     substr(string,<starting position of character>,<length of the
                                                           character>)
                                                                |
                                                           (+ve)-length
```

Expression:

```
-7 -6 -5 -4 -3 -2 -1
W  E  L  C  O  M  E
1  2  3  4  5  6  7
```

Ex:
SQL>SELECT SUBSTR('WELCOME',1,1) FROM DUAL; --------→W
SQL>SELECT SUBSTR('WELCOME',4,2) FROM DUAL; --------→CO
SQL>SELECT SUBSTR('WELCOME',5,4) FROM DUAL; --------→OME
SQL>SELECT SUBSTR('WELCOME',-4,4) FROM DUAL; --------→COME
SQL>SELECT SUBSTR('WELCOME',-6,2) FROM DUAL; --------→EL
SQL>SELECT SUBSTR('WELCOME',-6,-2) FROM DUAL; --------→

=>INSTR ():

->It remove the occurrence the position of character from the given string expression.

Syntax: instr(string,<specific character>,<string position of character>,<occurrence position of character>)

Expression:

```
 -13 -12 -11 -10 -9   -8 -7  -6  -5  -4 -3   -2    -1
  H    E    L    L   O     W   E   L   C  O   M    E
  1    2    3     4    5  6  7    8   9  10  11  12  13
```

Note:

->Counting characters from left to right (or) from right to left but the position of characters are fixed. (always positive positions)

Ex:

```
SQL>SELECT INSTR('HELLO WELCOME','L',1,2) FROM DUAL; -→4
SQL>SELECT INSTR('HELLO WELCOME','L',3,1) FROM DUAL; -→3
SQL>SELECT INSTR('HELLO WELCOME','E',4,3) FROM DUAL; -→0
SQL>SELECT INSTR('HELLO WELCOME','E',-1,1) FROM DUAL; -→13
SQL>SELECT INSTR('HELLO WELCOME','O',-7,1) FROM DUAL; -→5
SQL>SELECT INSTR('HELLO WELCOME',',',-2,1) FROM DUAL; -→12

SQL>SELECT INSTR('HELLO WELCOME','M',-2,-1) FROM DUAL;
        ERROR at line:1
        ORA-01428: argument '-1' is out of range.
```

>DATE FUNCTIONS:

=>SYSDATE:

->It return the current date information of the system.

Syntax: sysdate

Ex:

```
SQL>SELECT SYSDATE FROM DUAL; -------------→24-JAN-25
SQL>SELECT SYSDATE+5 FROM DUAL; -------------→29-JAN-25
SQL>SELECT SYSDATE-5 FROM DUAL; -------------→19-JAN-25
```

=>ADD_MONTHS ():

->To add/ subtract the number months to/ from the given date information.

Syntax: add_months(date,<no.of months>)

Ex:

```
SQL>SELECT ADD_MONTHS(SYSDATE,3) FROM DUAL; →24-APR-25
SQL>SELECT ADD_MONTHS(SYSDATE,-3) FROM DUAL; →24-OCT-24
```

Ex:
SQL>CREATE TABLE PRODUCT(PNAME VARCHAR2(10), MFG_DATE DATE, EXP_DATE DATE);
SQL>INSERT INTO PRODUCT (PNAME,MFG_DATE) VALUES(‘P1’,’09-SEP-2022’);
SQL>INSERT INTO PRODUCT (PNAME,MFG_DATE) VALUES(‘P2’,’25-DEC-2023’);
SQL>COMMIT;
SQL>SELECT * FROM PRODUCT; (before update)
SQL>UPDATE PRODUCT SET EXP_DATE=ADD_MONTHS(MFG_DATE,24);
SQL>SELECT * FROM PRODUCT; (after update)

=>LAST_DAY ():

->It return the last day from the given month in the date expression.

Syntax: last_day(date)

Ex:
SQL>SELECT LAST_DAY(SYSDATE) FROM DUAL; →31-JAN-25
SQL>SELECT LAST_DAY(’12-FEB-25’) FROM DUAL; →28-FEB-25

=>MONTHS_BETWEEN ():

->It return the number of months between the given two dates.

->Here, date 1 is always greater than to date 2 otherwise it returns (-ve) sign values.

Syntax: months_between(date1,date2)

Ex:
SQL>SELECT MONTHS_BETWEEN(’05-JUN-2024’,’05-JUN-2025’) FROM DUAL; →-12
SQL>SELECT MONTHS_BETWEEN(’05-JUN-2025’,’05-JUN-2024’) FROM DUAL; →12

>Conversion functions:

=>i) TO_CHAR () :

->To convert date type into character type and also display date in different formats.

Syntax: to_char(sysdate,<interval>)

Year formats:

YYYY	->Year in four digits format
YY	->Last two digits from year
YEAR	->TWENTY TWENTY-FIVE
CC	->Centaury 21
AD/BC	->AD year/BC year

Ex:

SQL>SELECT TO_CHAR(SYSDATE,’YYYY YY YEAR CC BC’) FROM DUAL;

Output: 2025 25 TWENTY TWENTY-FIVE 21 AD

Month functions:

MM	->Month in Number format
MON	->First three characters from month string
MONTH	->Full name of month.

Ex:

SQL>SELECT TO_CHAR(SYSDATE,’MM MON MONTH’) FROM DUAL;

Output: 01 JAN JANUARY

Day formats:

DDD	->Day of the year
DD	->Day of the month
D	->Day of the week

Sun	-	1
Mon	-	2
Tue	-	3
Wed	-	4
Thu	-	5
Fri	-	6
Sat	-	7

DAY	->Full name of the day
DY	->First three characters of the day string

Ex:

SQL>SELECT TO_CHAR(SYSDATE,’D DD DDD DY DAY’) FROM DUAL;

Output: 7 25 025 SAT SATURDAY

Quarter format:

Q	->One digit quarter of the year 1→JAN-MAR 2→APR-JUN 3→JUL-SEP 4→OCT-DEC

EX:

SQL>SELECT TO_CHAR(SYSDATE,'Q') FROM DUAL;

Output: 1

Week format:

WW	->Week of the year
W	->Week of the month

Ex:

SQL>SELECT TO_CHAR(SYSDATE,'W WW') FROM DUAL;

Output: 4 04

Time format:

HH	->Hour part in 12 hours format
HH24	->Hour part in 24 hours format
MI	->Minute part
SS	->Seconds part
AM/PM	->AM Time (or) PM Time

Ex:

SQL>SELECT TO_CHAR(SYSDATE,'HH HH24 MI SS PM') FROM DUAL;

Output: 11 11 45 29 PM

Ex: Write a query to display the list of employees details who are joined in 1981 by using TO_CHAR ()?

SQL>SELECT * FROM EMP WHERE
TO_CHAR(HIREDATE,'YYYY')='1981';

(OR)

SQL>SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,'YY')='81';

(OR)

SQL>SELECT * FROM EMP WHERE
TO_CHAR(HIREDATE,'YEAR')='NINETEEN EIGHT-ONE';

Ex: Write a query to display the list of employees details who are joined in 1980,1982,1983 by using TO_CHAR ()?

```
SQL>SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,’YYYY’)
IN(‘1980’,’1982’,’1983’);
                                    (OR)
SQL>SELECT * FROM EMP WHERE
TO_CHAR(HIREDATE,’YYYY’)=’1980’ OR
TO_CHAR(HIREDATE,’YYYY’)=’1982’ OR
TO_CHAR(HIREDATE,’YYYY’)=’1983’;
```

Ex: Write a query to display the list of employees who are joined in the month of DECEMBER?

```
SQL>SELECT * FROM EMP WHERE
TO_CHAR(HIREDATE,’MON’)=’DEC’;
                                    (OR)
SQL>SELECT * FROM EMP WHERE HIREDATE LIKE ‘%81’;
```

Ex: Write a query to display the list of employees details who are joined in the month of DECEMBER in 1982?

```
SQL>SELECT * FROM EMP WHERE
TO_CHAR(HIREDATE,’MON’)=’DEC’ AND
TO_CHAR(HIREDATE,’YYYY’)=’1982’;
                                    (OR)
SQL>SELECT * FROM EMP WHERE
TO_CHAR(HIREDATE,’MMYYYY’)=’121982’;
```

Ex: Write a query to display employees and their joined day from emp table?

```
SQL>SELECT ENAME, HIREDATE, TO_CHAR(HIREDATE,’DAY’) AS
        DAYS FROM EMP;
```

Ex: Write a query to display the list of employees who are joined on weekends?

```
SQL>SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,’DAY’) IN
(‘SATURDAY’,’SUNDAY’);
```

Ex: Write a query to display employees and their joined week from emp table?

SQL>SELECT ENAME, HIREDATE, TO_CHAR(HIREDATE,’WW’) AS WEEKS FROM EMP; (YEAR)

(OR)

SQL>SELECT ENAME, HIREDATE, TO_CHAR(HIREDATE,’W’) AS WEEKS FROM EMP; (MONTH)

Ex: Write a query to display employees details who are joined in 1st week of DECEMBER in 1981?

SQL>SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,’WMONYY’)=’1DEC81’;

(OR)

SQL>SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,’WMMYY’)=’11281’;

(OR)

SQL>SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,’MONYYW’) IN (DEC811’);

(OR)

```
SQL>SELECT * FROM EMP WHERE
2       TO_CHAR(HIREDATE,’W’)=’1’ AND
3       TO_CHAR(HIREDATE,’MON’)=’DEC’ AND
4       TO_CHAR(HIREDATE,’YYYY’)=’1981’;
```

=>ii) TO_DATE ():

->To convert string type to oracle default date type.

Syntax: to_date(string)

Ex:

SQL>SELECT TO_DATE(’12-DECEMBER-2024’) FROM DUAL;

→12-DEC-24

SQL>SELECT TO_DATE(‘12/DECEMBER/2024’) FROM DUAL;

→12-DEC-24

SQL>SELECT TO_DATE(‘12/DECEMBER/2024’) +10 FROM DUAL;

→22-DEC-24

SQL>SELECT TO_DATE(‘12/DECEMBER/2024’) -10 FROM DUAL;

→02-DEC-24

>Analytical functions:

->To generate rank numbers to each row wise/ to each group of rows wise from a table automatically.

(i)Rank () ➔Outdated

(ii)Dense_Rank () ➔Mostly used

->Rank() will assign same rank number to the same value but it will skip the next rank number in the order whereas Dense_Rank() will assign same rank number to the same value but it will not skip the next rank number in the order.

Ex:

ENAME	SALARY	RANK()	DENSE_RANK()
A	85000	1	1
B	72000	2	2
C	72000	2	2
D	68000	4	3
E	55000	5	4
F	55000	5	4
G	41000	6	5

Syntax: analytical function name() over ([partition by<column name>] order by<column name><asc/desc>)

Here,

Partition by clause is optional.

Order by clause is mandatory.

Ex: Write a query to display all employees salaries and their rank number from emp table?

SQL>SELECT ENAME, SAL, RANK()OVER(ORDER BY SAL DESC) AS RANKS FROM EMP; (no accuracy)

SQL>SELECT ENAME, SAL, DENSE_RANK() OVER (ORDER BY SAL DESC) AS RANKS FROM EMP; (with accuracy)

Ex: Write a query to display employees and their rank numbers based on each job wise from emp table?

SQL>SELECT ENAME, SAL, JOB, RANK() OVER(PARTITION BY JOB ORDER BY SAL DESC)AS RANKS FROM EMP;

(OR)

SQL>SELECT ENAME, SAL, JOB, DENSE_RANK() OVER(PARTITION BY JOB ORDER BY SAL DESC) AS RANKS FROM EMP;

2.Multiple Row Functions:

->These functions are also called as “grouping functions/aggregative functions” in database.

Syntax: <grouping function name>(column name)

=>SUM ():

->It returns total value.

Ex:

SQL>SELECT SUM(SAL) FROM EMP;

SQL>SELECT SUM(SAL) FROM EMP WHERE JOB=”MANAGER”;

=>AVG ():

->It returns average value.

Ex:

SQL>SELECT AVG(SAL) FROM EMP;

SQL>SELECT AVG(SAL) FROM EMP WHERE JOB=”MANAGER”;

=>MIN ():

->It returns minimum value.

Ex:

SQL>SELECT MIN(SAL) FROM EMP;

SQL>SELECT MIN(SAL) FROM EMP WHERE JOB=”MANAGER”;

=>MAX ():

->It returns maximum value.

Ex:

SQL>SELECT MAX(SAL) FROM EMP;

SQL>SELECT MAX(SAL) FROM EMP WHERE JOB=”MANAGER”;

=>COUNT ():

(i) Count(*):

->Counting all rows including duplicates and nulls from a table.

Ex:

SQL>SELECT COUNT(*) FROM EMP;

SQL>SELECT COUNT(*) FROM EMP WHERE JOB=”MANAGER”;

(ii) Count(column name):

->Counting all values including duplicate but not nulls from a column.

Ex:

SQL>SELECT COUNT(MGR) FROM EMP;

(iii) Count (distinct<column name>):

->Counting unique values from a column.(i.e.,no duplicates & no nulls)

Ex:

SQL>SELECT COUNT(DISTINCT MGR) FROM EMP;

===

➔Clauses:

->Clause is a statement which will add to SQL query for providing some facilities like ‘filtering rows, sorting values, grouping rows’ based on a particular column from a table automatically.

->Oracle supports the following clauses are,

>WHERE
>ORDER BY
>GROUP BY
>HAVING

Syntax: <sql query>+<clause statements>;

=>WHERE:

->Filtering rows before grouping the rows in a table.

->WHERE clause can be used in “SELECT, UPDATE, DELETE” commands only.

Syntax: where<filtering condition>;

Ex:

SQL>SELECT * FROM EMP WHERE EMPNO=7788;

SQL>UPDATE EMP SET SAL=45000 WHERE JOB=’MANAGER’;

SQL>DELETE FROM EMP WHERE DEPTNO=20;

=>ORDER BY:

->Order by clause is used for “sorting values of a particular column”.

->By default order by clause will arrange the values in “ascending” order. If we want to change the values in “descending” order then we must use “desc” keyword.

->It is used in “SELECT” query only.

Syntax:

Select */<list of columns>from<table name>order by<column name1><asc/desc>,<column name2><asc/desc>,…………………;

Ex:

```
SQL>SELECT * FROM EMP ORDER BY ENAME;
SQL>SELECT * FROM EMP ORDER BY ENAME DESC;
SQL>SELECT * FROM EMP ORDER BY HIREDATE;
SQL>SELECT * FROM EMP ORDER BY HIREDATE DESC;
```

Ex: Write a query to fetch employees details who are worked under deptno 20 and arrange their salaries in ascending order?

```
SQL>SELECT * FROM EMP WHERE DEPTNO=20 ORDER BY SAL;
```

Ex: Write a query to arrange the employes deptno’s in ascending order and their salaries in descending order from each deptno wise?

```
SQL>SELECT * FROM EMP ORDER BY DEPTNO, SAL DESC;
```

Note:

->Order by clause can be applied on not only column names eventhough we can apply on the position of column in the “SELECT” query.

Ex:

```
SQL>SELECT EMPNO, ENAME, SAL FROM EMP ORDER BY 1 DESC;
SQL>SELECT EMPNO, ENAME, SAL FROM EMP ORDER BY 2 DESC;
SQL>SELECT EMPNO, ENAME, SAL FROM EMP ORDER BY 3 DESC;
```

***ORDER BY with NULL Clauses:

->There are two types of NULL Clauses:

(i) NULLS FIRST

(ii) NULLS LAST

(i) NULLS FIRST:

->By default order by clause will arrange the nulls in ascending order is:

First	:	Values
Last	:	Nulls

Ex:
SQL>SELECT * FROM EMP ORDER BY COMM;
->To overcome the above problem in ascending order then we must use 'NULLS FIRST' Clause.
Solution:
SQL>SELECT * FROM EMP ORDER BY COMM NULLS FIRST;

(ii) NULLS LAST:
->By default order by clause will arrange the nulls in descending order is:

First	:	Nulls
Last	:	Values

Ex:
SQL>SELECT * FROM EMP ORDER BY COMM DESC;
->To overcome the above problem in descending order then we must use 'NULLS LAST" clause.
Solution:
SQL>SELECT * FROM EMP ORDER BY COMM DESC NULLS LAST;

=>GROUP BY:
->It is used to make groups based on columns from the table.
->When we use "group by" clause we must use "group functions/aggregative functions" are count(), sum(), avg(), min(), max().
->It is used in "SELECT" commands only.
Syntax:
Select<column name1>,<column name2>,……………………,<grouping function name1>,<grouping function name2>,…………….. from <table name>group by<column name1>,<column name2>,……………………;

Ex: Write a query to find out number of employees are working under in the organization?
SQL>SELECT COUNT(*) AS NO_OF_EMPLOYEES FROM EMP;

```
NO_OF_EMPLOYEES
----------------------------
                  14
```

Ex: Write a query to find out number of employees are working under deptno is 30?

```
SQL>SELECT COUNT(*) AS NO_OF_EMPLOYEES FROM EMP WHERE DEPTNO=30;
```

Output:

```
NO_OF_EMPLOYEES
----------------------------
                      6
```

Ex: Write a query to find out number of employees are working under each deptno ?

```
SQL>SELECT DEPTNO, COUNT(*) AS NO_OF_EMPLOYEES FROM EMP GROUP BY DEPTNO ORDER BY DEPTNO;
```

Output:

```
DEPTNO          NO_OF_EMPLOYEES
----------------     -----------------------------
10                   3
20                   5
30                   6
```

Ex: Write a query to find out number of employees, total amount and average amount from each deptno wise?

```
SQL>SELECT DEPTNO, COUNT(*) AS NO_OF_EMPLOYEES, SUM(SAL) AS TOTAL_SPENDING_AMOUNT, AVG(SAL) AS AVERAGE_AMOUNT FROM EMP GROUP BY DEPTNO ORDER BY DEPTNO;
```

Ex: Write a query to find out minimum, maximum salaries from each job wise?

```
SQL>SELECT JOB, MIN(SAL) AS MINIMUM_SALARY, MAX(SAL) AS MAXIMUM_SALARY FROM EMP GROUP BY JOB;
```

Ex: Write a query to display number of employees are working under each job along with their deptno?

SQL>SELECT JOB, DEPTNO, COUNT(*) AS NO_OF_EMPLOYEES FROM EMP GROUP BY JOB, DEPTNO ORDER BY JOB, DEPTNO;

IQ:
Ex: Write a query to find out number of employees are joined from each year wise?
SQL>SELECT TO_CHAR(HIREDATE,'YYYY') AS YEARS, COUNT(*) AS NO_OF_EMPLOYEES FROM EMP GROUP BY TO_CHAR(HIREDATE,'YYYY');

Ex: Write a query to find out number of employees are joined from each month wise?
SQL>SELECT TO_CHAR(HIREDATE,'MONTH') AS MONTHS, COUNT(*) AS NO_OF_EMPLOYEES FROM EMP GROUP BY TO_CHAR(HIREDATE,'MONTHS');

Ex: Write a query to find out number of employees are joined from each day wise?
SQL>SELECT TO_CHAR(HIREDATE,'DAY') AS DAYS, COUNT(*) AS NO_OF_EMPLOYEES FROM EMP GROUP BY TO_CHAR(HIREDATE,'DAY');

=>HAVING:
->Filtering rows after grouping data from a table.
->When we use "HAVING" clause we must use "GROUP BY" clause.

Syntax:
Select<column name1>,<column name2>,……………………………………, <grouping function name1>,<grouping function name2>,……………………., from<table name>group by<column name1>,<column name2>,……………… having <filter condition>;

Ex: Write a query to display YEARS in which year the number of employees are joined less than to 2?
SQL>SELECT TO_CHAR(HIREDATE,'YYYY') AS YEARS, COUNT(*) AS NO_OF_EMPLOYEES FROM EMP GROUP BY TO_CHAR(HIREDATE,'YYYY') HAVING COUNT(*)<2;

```
YEAR            NO_OF_EMPLOYEES
----------      ----------------------------
1980            1
1983            1
```

Ex: Write a query to display sum of salaries of jobs from emp table if the sum of salary of the jobs is more than 5000?

```
SQL>SELECT JOB, SUM(SAL) AS SUM_OF_SALARY FROM EMP
GROUP BY  JOB HAVING SUM(SAL)>5000;
```

<u>*** WHERE vs HAVING:</u>

WHERE	HAVING
1.Filtering rows before grouping data in the table.	1.Filtering rows after grouping data in the table.
2.WHERE conditions will work on each individual row wise	2.HAVING conditions will work on each group of rows wise.
3.It does not support "Aggregative functions".	3.It supports "Aggregative functions".
4.It will use before "GROUP BY" clause.	4.It will use after "GROUP BY" clause.
5.Without "GROUP BY" clause WHERE clause can be worked.	5.Without "GROUP BY" clause HAVING clause can not be worked.

**Using all clauses in a single selected query:

Syntax:

```
Select <column name1>,<column name2>,……………………………………..,
<grouping function name1>,<grouping function name2>,………………………,
from<table name>
                        [
                          Where<filtering condition>
                          Group by<column name1>,<column name2>,…………….
                          Having <filtering condition>
                          Order by<column name> <asc/desc>,……………………..
                        ];
```

Ex:

```
SQL>SELECT DEPTNO, COUNT(*) FROM EMP
                                                    WHERE SAL>1000
                                                    GROUP BY DEPTNO
                                                    HAVING COUNT(*)>3
                                                    ORDER BY DEPTNO;
```

Output:

```
DEPTNO          COUNT(*)
-----------     ----------------
20              4
30              5
```

==

JOINS:

->In RDBMS data can be organized in multiple tables. From those multiple tables if we want to retrieve the required data/information then we should use a technique is known as "JOINS".

->By using joins we are retrieving data/information from multiple tables at a time.

->In oracle joins statements can be written in two formats.

(i) NON-ANSI format
(ii) ANSI format

(i) NON-ANSI format:

->These statements are not portability statements.

i.e., we can't move a join statement from one platform to another platform.

->Here join conditions are preparing with "WHERE" clause.

Syntax:

Select * from<table name1>, <table name2>where<join condition>;

(ii) ANSI format:

->These statements are portability statements.

i.e., we can move a join statement from one platform to another platform.

->Here join conditions are preparing with "ON" clause.

Syntax:
Select * from<table name1><join key><table name2>on<join condition>;

=>Types of joins:

1. Inner joins
   - ->Equi join
   - ->Non-Equi join
   - ->Self join
2. Outer joins
   - ->Left outer join
   - ->Right outer join
   - ->Full outer join
3. Cross join
4. Natural join (not in MySQL)

1) Inner join:
=>EQUI JOIN:
->When we are retrieving data from multiple tables based on "=" operator join condition is known as Equi Join.
->When we use equi join we should maintain at least one commonness column name (Optional) in both tables for easy to identify join key column from tables.
->This commonness column must be have same datatype in both tables.
(IQ) ->Having relationship between tables is optional but not mandatory for joins.
->Retrieving matching rows from multiple tables.

JOIN CONDITION IN NON-ANSI:
Syntax:
Where<table name1/table alias name1>.<common column name> = <table name2/table alias name2>.<common column name>;
JOIN CONDITION IN ANSI:
Syntax:
On<table name1/table alias name1>.<common column name> = <table name2/table alias name2>.<common column name>;

DEM0-TABLES:

```
SQL>SELECT * FROM COURSES;
CID         CNAME           CFEE
===         =======         =====
1           ORACLE          3500
2           JAVA            5000
3           .NET            7000
SQL>SELECT * FROM STUDENTS;
SNAME           CID
=======         ====
ALLEN           1
JONES           1
ADAMS           2
MILLER          NULL
```

Ex: Write a query to retrieve students, courses details from the given tables who are confirmed their course?

NON-ANSI:

Without-alias:

===========

```
SQL>SELECT * FROM STUDENTS, COURSES WHERE
STUDENTS.CID=COURSES.CID;
```

(OR)

With - alias: (don’t use alias)

=========

```
SQL>SELECT * FROM STUDENTS S, COURSES C WHERE S.CID=C.CID;
```

ANSI: (We can use “join” or “inner join”)

Without-alias:

===========

```
SQL>SELECT * FROM STUDENTS JOIN/INNER JOIN COURSES ON
STUDENTS.CID=COURSES.CID;
```

(OR)

With – alias:
=========
SQL>SELECT * FROM STUDENTS S JOIN COURSES C ON S.CID=C.CID
;
Rule for JOINS:

<u>->A row in a table is compared with multiple rows of another table.</u>

Ex: Write a query to display students NAME, COURSE NAME from the given tables who are joined in "ORACLE" course?
NON-ANSI:

SQL>SELECT SNAME, CNAME FROM STUDENTS S, COURSES C
WHERE S.CID=C.CID AND CNAME='ORACLE';
ANSI:

SQL>SELECT SNAME, CNAME FROM STUDENTS S JOIN COURSES C
ON S.CID=C.CID AND CNAME='ORACLE';
(OR)
SQL>SELECT SNAME, CNAME FROM STUDENTS S JOIN COURSES C
<u>ON S.CID-C.CID</u> WHERE <u>CNAME='ORACLE'</u>;
Join Condition Additional Condition
(Mandatory) (Optional)

Ex: Write a query to display ename, location from the given tables who are working in "CHICAGO"?
ANSI:

SQL>SELECT ENAME, LOC FROM EMP JOIN DEPT ON
EMP.DEPTNO=DEPT.DEPTNO WHERE LOC='CHICAGO';

Ex: Write a query to display sum of salaries of each department name wise from the given tables?
SQL>SELECT DNAME, SUM(SAL) FROM EMP JOIN DEPT ON
EMP.DEPTNO=DEPT.DEPTNO GROUP BY DNAME;

Ex: Write a query to display deptno, sum of salaries of each department name from the given tables?

```
SQL>SELECT D.DEPTNO, DNAME, SUM(SAL) FROM EMP JOIN DEPT
ON EMP.DEPTNO=DEPT.DEPTNO GROUP BY D.DEPTNO, DNAME
ORDER BY DEPTNO;
```

Ex: Write a query to display number of employees are working under each department name from the given tables. If number of employees are more than 5?

```
SQL>SELECT DNAME, COUNT(*) AS NO_OF_EMPLOYEES FROM EMP
E JOIN DEPT D ON E.DEPTNO=D.DEPTNO GROUP BY DNAME HAVING
COUNT(*)>5;
```

=>NON-EQUI JOIN:

->Retrieving data from multiple tables based on any operator except an "=" operator is known as non-equi join.

->We can use the following operators are <, >, <=, >=, !=, BETWEEN, AND,…………, etc.

DEMO TABLES:
============

```
SQL>SELECT * FROM TEST1;
        SNO         NAME
        ---------   ------------
        1           ALLEN
        2           WARD
SQL>SELECT * FROM TEST2;
        SNO         FEE
        ----------  ------------
        1           3500
        2           5500
                    2400
```

Ex:

```
SQL>SELECT * FROM TEST1 T1 JOIN TEST2 T2 ON T1.SNO < T2.SNO;
SQL>SELECT * FROM TEST1 T1 JOIN TEST2 T2 ON T1.SNO <= T2.SNO;
SQL>SELECT * FROM TEST1 T1 JOIN TEST2 T2 ON T1.SNO > T2.SNO;
```

SQL>SELECT * FROM TEST1 T1 JOIN TEST2 T2 ON T1.SNO >=T2.SNO;
SQL>SELECT * FROM TEST1 T1 JOIN TEST2 T2 ON T1.SNO != T2.SNO;

Ex: Write a query to display ename, sal, low salary, high salary from the given tables whose salary is between low salary and high salary?
ANSI:

SQL>SELECT ENAME, SAL, LOSAL, HISAL FROM EMP JOIN SALGRADE ON SAL BETWEEN LOSAL AND HISAL;

NON-ANSI:

SQL>SELECT ENAME, SAL, LOSAL, HISAL FROM EMP, SALGRADE WHERE SAL BETWEEN LOSAL AND HISAL;

(OR)

SQL>SELECT ENAME, SAL, LOSAL, HISAL FROM EMP JOIN SALGRADE ON (SAL>=LOSAL) AND (SAL<=HISAL);

=>SELF JOIN:

->Joining a table by itself is known as “SELF JOIN”.

(OR)

->Comparing a table data by itself is known as “SELF JOIN”.

->It will work on a single table only.

->When we use self-join we must use alias names. Otherwise, we cannot implement self-join.

->When we create alias name on table name internally oracle server prepares virtual tables on each alias name in buffer memory.

->We can create any number of alias names on a single table but each alias name should be different name.

->Self join can be used at two levels in databases.

LEVEL 1: Comparing a single column values by itself with in the table.

LEVEL 2: Comparing multiple column values to each other with in the table.

*Example on comparing a single column values by itself with in the table:

DEMO TABLES:
=============

```
SQL>SELECT * FROM TEST;
        ENAME           LOC
        ----------      --------------
        SMITH           HYD
        JONES           MUMBAI
        ALLEN           HYD
        MILLER          DELHI
```

Ex: Write a query to display employees details who are working in the same location where the employee “ALLEN” is also working?

```
SQL>SELECT T1.ENAME,T1.LOC FROM TEST T1 JOIN TEST T2 ON
T1.LOC=T2.LOC AND T2.ENAME=’ALLEN’;
```

2) Outer join :

->There are three types of outer joins.

=> Left Outer Joins:

->Retrieving matching rows from both tables and unmatching rows from the left side of the table.

Ex:

ANSI:

```
SQL>SELECT * FROM STUDENTS S LEFT OUTER JOIN COURSES C ON
S.CID=C.CID;
```

NON-ANSI:

->When we want to write outer joins in non-ansi format then we must use a special join operator is (+).

Solution:

SQL>SELECT * FROM STUDENTS S, COURSES C WHERE S.CID=C.CID(+);

=> Right Outer Joins:

->Retrieving matching rows from both tables and unmatching rows from the right side of the table.

Ex:

ANSI:

SQL>SELECT * FROM STUDENTS S RIGHT OUTER JOIN COURSES C ON S.CID=C.CID;

NON-ANSI:

SQL>SELECT * FROM STUDENTS S, COURSES C WHERE S.CID (+)=C.CID;

=>Full Outer Join:

->Retrieving matching rows and unmatching rows from both tables.

Ex:

ANSI:

SQL>SELECT * FROM STUDENTS S FULL OUTER JOIN COURSES C ON S.CID=C.CID;

NON-ANSI:

->When we want to write full outer join statement in non-ansi format then we must use "UNION" operator.

3)CROSS JOIN:

->Retrieving data from multiple tables without any condition.

->In cross join mechanism each row of a table will join with each of another table.

For example, a table is having (M) number of rows and another table is having (N) number of rows then the result is (MXN) rows.

Ex:
ANSI:

```
SQL>SELECT * FROM STUDENTS S CROSS JOIN COURSES C;
```

NON-ANSI:

```
SQL>SELECT * FROM STUDENTS S, COURSES C;
```

Ex:
DEMO TABLES:
=============

```
SQL>SELECT * FROM ITEMS1;
        SNO          INAME             PRICE
        ------       -------------     ----------
        1            PIZZA             180
        2            BURGER            85
SQL>SELECT * FROM ITEMS2;
        SNO          INAME             PRICE
        ------       ----------        ----------
        101          PEPSI             20
        102          COCACOLA          25

SQL>SELECT I1.INAME, I1.PRICE, I2.INAME, I2.PRICE,
2       I1.PRICE+I2.PRICE AS COMBO_PRICE FROM
3       ITEMS1 I1 CROSS JOIN ITEMS2 I2;
```

Output:

```
INAME       PRICE       INAME       PRICE       COMBO_PRICE
---------   --------    ----------  --------    ---------------------
PIZZA       180         PEPSI       20          200
```

4)Natural Joins:

->It is similar to qui join.

->It always retrieve matching rows from tables based on an “=” operator just like equi join.

** NATURAL JOIN vs EQUI JOIN:

NATURAL JOIN	EQUI JOIN
1.Preparing a join condition by implicitly.	1.Preparing a join condition by explicitly.
2.We must maintain common column name.	2.Common column name is optional.
3.To avoid duplicate columns from the result set(output) automatically.	3.It does not avoid duplicate columns from the result set automatically.
4.Preparing join condition based on "USING" clause.	4.Preparing join condition based on "WHERE/ON" clause.

Ex:
SQL>SELECT * FROM STUDENTS NATURAL JOIN COURSES;

10-02-25

ORACLE BY MENTOR

=> OFFSET:
->Introduced in oracle 12 c.
->It is used to specify number of rows to be fetched.

Syntax:
OFFSET n Row/Rows

=>FETCH:
->Introduced in oracle 12 c.
->It is used to specify number of rows to be fetched.
->It is used to display top n rows.

Syntax:
FETCH FIRST/NEXT n Rows/Row ONLY.

Examples:
1) Write a query to display all employees records except first 5 rows.

```
SQL>SELECT ENAME, SAL
2       FROM EMP
3       OFFSET 5 ROWS;
```

2) Write a query to display first 5 rows only from emp table?

```
SQL>SELECT ENAME, SAL
2       FROM EMP
3       FETCH FIRST 5 ROWS ONLY;
```

3) Write a query to display top 3 salaried employee records from emp table?

```
SQL>SELECT ENAME, SAL
2       FROM EMP
3       ORDER BY SAL DESC
4       FETCH FIRST 3 ROWS ONLY;
```

4) Write a query to display senior most employee records from emp table by using fetch first clause?

```
SQL>SELECT ENAME, SAL, HIREDATE
2       FROM EMP
3       ORDER BY HIREDATE ASC
4       FETCH FIRST 1 ROW ONLY;
```

5) Write a query to display second maximum salary from emp table by using offset and fetch clause?

```
SQL>SELECT DISTINCT SAL
2       FROM EMP
3       ORDER BY SAL DESC
4       OFFSET 1 ROW
5       FETCH FIRST 1 ROW ONLY;
```

6) Display 6th row to 10th row?

```
SQL>SELECT ENAME, SAL, DEPTNO
2       FROM EMP
3       OFFSET 5 ROWS
4       FETCH NEXT 5 ROWS ONLY;
```

=>Joining more than 2 tables:

->If number of tables increases number of joining conditions are also increases.

->If we are joining n tables then we are writing n-1 joining conditions.

Syntax:

```
        SELECT COLUMN 1, COLUMN 2, COLUMN 3,………………
        FROM TABLE1 JOIN TABLE2
        ON TABLE1.KEY = TABLE2.KEY
                JOIN TABLE3
        ON TABLE2.KEY = TABLE3.KEY
                JOIN TABLE4
        ON TABLE3.KEY = TABLE4.KEY
```

7) Fetch Student_name and Course_name order by Student_id. Only first 20 records from the following tables (HCL interview question).

```
                    STUDENT TABLE
        -----------------------------------------------------------
        |       Student_id  |       Student_name               |
        -----------------------------------------------------------
                        FEE TABLE
        -----------------------------------------------------------
        |       Student_id  |       Course_fee                 |
        -----------------------------------------------------------
                    COURSE TABLE
        -----------------------------------------------------------
        |       Course_fee  |       Course_name                |
        -----------------------------------------------------------
```

Solution:

```
SQL>SELECT STUDENT_NAME, COURSE_NAME
2       FROM STUDENT JOIN FEE
3       ON STUDENT.STUDENT_ID = FEE.STUDENT_ID
4       JOIN COURSE
5       ON FEE.COURSE_FEE = COURSE.COURSE_FEE
6       ORDER BY STUDENT_ID
7       FETCH FIRST 20 ROWS ONLY;
```

11-02-25

SUB QUERIES

A query within another query is called Sub Query or Nested Query. Sub queries are used to retrieve data from single or multiple tables by using more than one step process.

All relational databases having two types of subqueries. Those are

1) Non – Correlated Subqueries
2) Correlated Subqueries

In Non - Correlated Subqueries child query is executed first then only parent query is executed, Whereas in correlated subqueries parent query is executed first then only child query is executed.

1) Non – Correlated Subqueries:

Non – correlated subqueries have two parts.

i) Child query
ii) Parent query

i) Child query:

A query which provides values to another query is called Child query or Inner query.

ii) Parent query:

A query which receives values from another query is called Parent query or Outer query.

In non – correlated sub query maximum limit is up to 255 queries.

Non – Correlated Sub Queries:

```
        |
        |--------------------> 1) Single row subquery
        |
        |--------------------> 2) Multiple row subquery
        |
        |--------------------> 3) Multiple column subquery
        |
        |--------------------> 4) Inline view or sub queries are used in from clause.
```

RULES:
----

1. 90% -----------→ Same Column
2. 5% ------------→ Group Function
3. 3% ------------→ Expression
4. 2% ------------→ Different Column

Ex: Write a query to display the employee who are getting more salary than the avg(sal) from emp table?
SQL>SELECT * FROM EMP WHERE SAL>(SELECT AVG(SAL) FROM DUAL);

Note : Don't use "*" in child query.

Execution:

Step 1:
SQL>SELECT AVG(SAL) FROM EMP;
2830.35714

Step 2:
SQL>SELECT * FROM EMP WHERE SAL>2830.35714;

Q) Write a query to display the employees who are working in SMITH department number from emp table?

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

Q) Write a query to display the employees who are getting more salary than the highest paid employee in 20th department from emp table?

```
SQL>SELECT * FROM EMP
2       WHERE SAL>
3       (SELECT MAX(SAL) FROM EMP WHERE DEPTNO=20);
```

Q) Write a query to display the employees records who are earning more salary than the BLAKE salary?

```
SQL>SELECT * FROM EMP
2       WHERE SAL>
3       (SELECT SAL FROM EMP WHERE ENAME='BLAKE');
```

Q) Write a query to display the employee records whose job title is same as ALLEN from emp table?

```
SQL>SELECT * FROM EMP
2       WHERE JOB=
3       (SELECT JOB FROM EMP WHERE ENAME='ALLEN');
```

Q) Write a query to display senior most employee details from emp table?

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

Q) Write a query to display the employees who are working in SALES department from emp,dept table?

```
SQL>SELECT ENAME, DNAME
2       FROM EMP E JOIN DEPT D
3       ON E.DEPTNO=D.DEPTNO
4       WHERE D.DEPTNO=
5       (SELECT DEPTNO FROM DEPT WHERE DNAME='SALES');
```

Q) Write a query to display second max(sal) or second highest salary from emp table?

```
SQL>SELECT MAX(SAL) FROM EMP
```

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

Q) Write a query to display second highest sal employee details from emp table?

12-02-25

Q) Write a query to display the employees who are getting more salary than the ALLEN salary and job is same as JONES job?

```
SQL>SELECT * FROM EMP WHERE SAL>
2       (SELECT SAL FROM EMP WHERE ENAME='ALLEN')
3       AND
4       JOB=(SELECT JOB FROM EMP WHERE ENAME='JONES');`
```

Q) Write a query to display the employee records who are senior to BLAKE?

```
SQL>SELECT * FROM EMP WHERE HIREDATE<
2       (SELECT HIREDATE FROM EMP WHERE ENAME='BLAKE');
```

Q) Write a query to display the employees who are working under BLAKE as a manager from emp table by using empno, mgr columns?

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

Q) Write a query to display highest number of employees department from emp table by using group by clause?

```
SQL>SELECT DEPTNO, COUNT(*) AS NO_OF_EMPLOYEES FROM EMP
GROUP BY DEPTNO HAVING COUNT(*) =
2       (SELECT MAX(COUNT(*) FROM EMP GROUP BY DEPTNO);
```

2) Multiple Row Sub Queries:
======================

When child query return multiple values then those queries are called Multiple row sub queries. In multiple row subqueries we are using in, all, any operators.

Q) Write a query to display the employee details who are getting maximum salary in each department from emp table?

```
SQL>SELECT * FROM EMP
2       WHERE SAL
3       IN
4       (SELECT MAX(SAL) FROM EMP GROUP BY DEPTNO);
```

NOTE: If we use “=” in the place of “IN” operator it provides an error.
This is a multiple row sub query because here query returns multiple values.

Q) Write a query to display the employee records whose job title is same as SMITH, BLAKE?

```
SQL>SELECT ENAME, JOB, SAL FROM EMP
2       WHERE SAL IN
3       (SELECT JOB FROM EMP WHERE ENAME IN(‘SMITH’,’BLAKE’);
```

IN --------------→It returns same values in list.
ALL -----------→ It returns all values in the list.
ANY ----------→ It returns any values in the list.

Whenever we are using subquery special operator “ALL” then database server internally uses logical operator AND. Where as when we are using subquery special operator “ANY” then database server internally uses logical operator OR.

Q) Display the employee records who are earning more salary than all managers from emp table?

```
SQL>SELECT * FROM EMP
2       WHERE SAL>ALL
3       (SELECT SAL FROM EMP WHERE JOB=’MANAGER’);
```

Q) Display the employee records who are earning more salary than any of the managers?

```
SQL>SELECT ENAME, SAL, FROM EMP
2       WHERE SAL>ANY
3       (SELECT SAL FROM EMP WHERE JOB=’MANAGER’);
```

Q) Display the male employees who are getting more salary than all female employees?

```
SQL>SELECT * FROM EMP
2       WHERE GENDER='M'
3       SAL>ALL
4       (SELECT SAL FROM EMP WHERE GENDER='F');
```

13-02-25

----

4) Inline-View:

===========

Oracle 7.2 introduced inline views. Inline views are special type of queries used in Top-n analysis. In place of table name we are writing subquery is called inline view.

Syntax:

Select * from (Subquery);

Generally, in all database systems we are not allowed to use "order by" clause in child. Why means reason also there parent query having its own order. Parent query never uses child query sorting order.

In Top-n analysis queries we are executing sorting order by clause is first. Then only other clauses then order by clauses queries we are putting in a inline views.

Q) Write a query to display the employees who are getting more than 30000 annual salary from emp table?

```
SQL>SELECT ENAME, SAL, SAL*12 AS ANNSAL FROM EMP WHERE
ANNSAL>30000;
```

ERROR:-Annsal :Invalid identifier

Note: In where clause we are not allowed to use alias names.

Generally in oracle we are not allowed to use column name alias name in where clause. To overcome this problem if we want to use column alias name in where clause then we must specify the alias name queries in place of table name within parent query. These type of queries are also called as Subqueries ae used in from clauses or inline views.

When we are using inline views then alias names internally behaves like a exact table column.
Solution:

SQL>SELECT * FROM(SELECT ENAME, SAL, SAL*12 ANNASL FROM EMP) WHERE ANNSAL>30000;

Execution:

```
                  |-----------------------------------------------------------|
Select * from (   |    Ename |        Sal       |          Annsal                  |) where
                  |-------------------------------------------------------------- |
Annsal>30000;
```

=>Analytical functions used in Inline Views:

Analytical functions are similar to group functions which work on group of data but group functions reduces number of rows in each group where as analytical functions does not reduces number of rows in each group.i.e server executes for each and every row in a table.

In all relational databases analytical functions performance is very high compared to all the other functions.

Syntax:
```
        Analytical function name() over
        (Partition by column name
        Order by column name[asc/desc])
```
Oracle having following analytical functions. Those are

1) Row_Number()
2) Rank()
3) Dense_Rank()

These 3 analytical functions automatically assigns ranks either group wise or rows wise in a table.

Row_number() analytical function automatically assigns different rank numbers when values are same whereas rank(), dense_rank() functions automatically assigns some rank numbers when values are same and also rank() skips next consecutive rank numbers where as dense_rank() does not skip next consecutive rank numbers.

Note: In these 3 analytical functions partition by clause is an optional clause.

Q) Write a query to display 2nd highest salary employee from emp table by using analytical function?
SQL>SELECT * FROM(SELECT DEPTNO, ENAME, SAL, DESNE_RANK() OVER (ORDER BY SAL DESC) R FROM EMP) WHERE R=2;

Q) Write a query to display 3rd highest salary employee in each department from emp table by using analytical function?
SQL>SELECT * FROM(SELECT DEPTNO, ENAME, SAL, DENSE_RANK() OVER(PARTITION BY DEPTNO ORDER BY SAL DESC)R FROM EMP) WHERE R=3;

Note: Partition by clause is an optional clause in all analytical functions. Whenever grouping you want means partition by. With in a table result means remove partition by.

Q) Write a query to display nth highest salary employee from emp table by using analytical function?
SQL>SELECT * FROM(SELECT DEPTNO,ENAME, SAL, DENSE_RANK() OVER(ORDER BY SAL DESC) R WHERE R=&n;

* ROWID:

Rowid is a pseudo column. It behaves like a table column. Rowid It behaves likes a table column. Rowid stores physical address of a row in a table. We are inserting a record into a table. Then oracle server internally automatically generates a unique identification number in hexadecimal format for identifying a record uniquely. This is called rowid.

Generally in oracle by using rowid we are retrieving data very fastly from oracle database. That's why rowid queries improve performance of the application.

In oracle by default rowid's are in ascending order . That's why we can also use min(), max() functions in rowid's.

Ex:

```
SQL>SELECT MIN(ROWID) FROM EMP;
SQL>SELECT MAX(ROWID) FROM EMP;
```

=>Deleting duplicate rows from a table:

In all relational databases delete duplicate rows excepts one row in each group in a table is also called as delete duplicate rows in a table. If we want ro delete duplicate rows from a table then we must use rowid.

Q) Write a query to delete duplicate rows from following table:

```
SQL>CREATE TABLE TEST(DEPTNO NUMBER(10));
SQL>INSERT INTO TEST VALUES(…………..);
SQL>SELECT * FROM TEST;
        DEPTNO
        ------------
        10
        10
        10
        20
        20
        20
        30
        30
        40
        50
```

Sol:

```
SQL>DELETE FROM TEST WHERE ROWID NOT IN(SELECT
MIN(ROWID) FROM TEST GROUP BY DEPTNO);
```

*ROWNUM:

Rownum is an pseudo column. It behaves like a table column. Generally rownum pseudo column is used to restrict rows in a table.i.e. used to filter the data in a table. Rownum pseudo column is also same as limit clause in MySQL database.

Whenever we are installing oracle server than automatically so many special columns are created with in oracle database. These columns are also called pseudo columns. These pseudo columns are rownum, rowid. These pseudo columns belong to all table in oracle database.

Generally rownum pseudo column automatically assigns numbers to each row in a table at the time of selection. Generally rownum having temporary values.

Ex:

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

Q) Write a query to display first row from emp table by using rownum?

```
SQL>SELECT * FROM EMP WHERE ROWNUM=1;
```

Q) Write a query to display 2nd record from emp table using rownum alias name?

```
SQL>SELECT * FROM(SELECT ROWNUM R, ENAME, SAL FROM EMP)
WHERE R=2;
```

Q) Write a query to display rows between 4 to 8 from emp table by using rownum alias name?

```
SQL>SELECT * FROM(SELECT ROWNUM R, ENAME, SAL FROM EMP)
WHERE R BETWEEN 4 AND 8;
```

Q) Write a query to display 2,5,6,9 records from emp table by using rownum alias name?

```
SQL>SELECT * FROM(SELECT ROWNUM R, ENAME, SAL FROM EMP)
WHERE R IN(2,5,6,9);
```

Q) Write a query to display even number of records from emp table by using rownum alias name?

```
SQL>SELECT * FROM(SELECT ROWNUM R, ENAME, SAL FROM EMP)
WHERE MOD(R,2)=0;
```

Q) Write a query to display 1ST and last row from emp table by using rownum alias name?

```
SQL>SELECT * FROM(SELECT ROWNUM R, ENAME, SAL FROM EMP)
WHERE R=1 OR R=(SELECT COUNT(*) FROM EMP);
```

Q) Write a query to display to skip 3 rows and then display remaining all rows from emp table by using rownum alias name?

SQL>SELECT * FROM(SELECT ROWNUM R, ENAME, SAL FROM EMP) WHERE R>3;

==

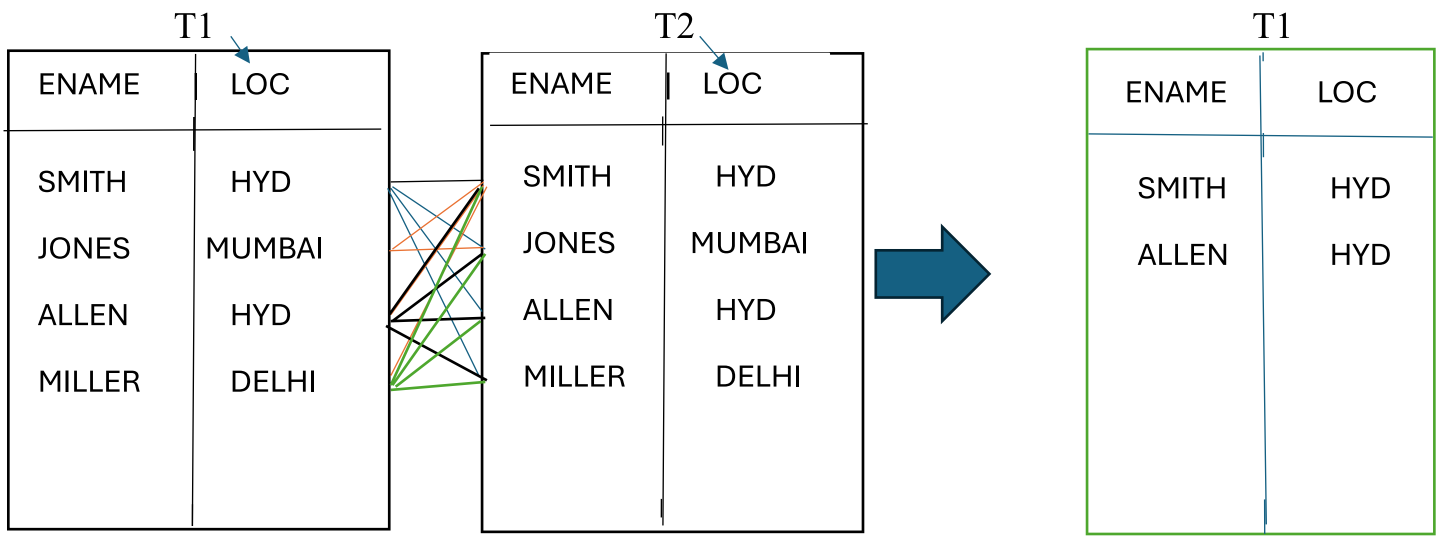
17-02-25 By Sudhakar Sir

---------- ---------------------

Self-Join:s

Explanation:

SELECT T1.ENAME, T1.LOC FROM TEST T1 JOIN TEST T2
ON <u>T1.LOC = T2.LOC</u> AND <u>T2.ENAME='ALLEN'</u>;



Non-Ansi:

SQL>SELECT T1.ENAME, T1.LOC FROM TEST T1, TEST T2 WHERE T1.LOC = T2.LOC AND T2.ENAME = 'ALLEN';

Ex: Write a query to display employees details whose salary is same as the employee "SCOTT" salary?

SQL>SELECT E1.ENAME, E1.SAL FROM EMP E1 JOIN EMP E2 ON E1.SAL = E2.SAL AND E2.ENAME = 'SCOTT';

=>Examples on comparing multiple columns values to each other with in the table:

Ex: Write a query to find out Managers and their employees from emp table?
SQL>SELECT M.ENAME AS MANAGERS, E.ENAME AS EMPLOYEES FROM EMP E JOIN EMP M ON M.EMPID = E.MGR;

Ex: Write a query to display employees who are working under "Blake" manager?

```
SQL>SELECT E.ENAME FROM EMP E JOIN EMP M ON
        M.EMPNO = E.MGR WHERE M.ENAME = 'BLAKE';
```

Ex: Write a query to display manager of the employee "Blake"?

```
SQL>SELECT M.ENAME FROM EMP E JOIN EMP M ON
        M.EMPNO = E.MGR WHERE E.ENAME = 'BLAKE';
```

Ex: Write a query to display employees who are joined before their manager?

```
SQL>SELECT E.ENAME FROM EMP E JOIN EMP M ON
        M.EMPNO = E.MGR WHERE E.HIREDATE < M.HIREDATE;
```

Ex: Write a query to display employees whose salary is more than to their managers salary?

```
SQL>SELECT E.ENAME FROM EMP E JOIN EMP M ON
        M.EMPNO = E.MGR WHERE E.SAL > M.SAL;
```

=>How to join more than two tables:

Non – Ansi Format:

Syntax:
Select * from<table name1>,<table name2>,<table name3>,………………
Where <condition1> and <condition2> and ,…………….;

Ansi Format:

Syntax:

Select * from <table name1><join><table name2> on <join condition1><join><table name3> on <join condition2><join><table name4> on <join condition3>……………<join><table nameN> on <join conditionN-1>;

Ex: Demo – Tables:

```
SQL>SELECT * FROM COURSES;       REGNO      REGDATE   CID
SQL>SELECT * FROM STUDENTS;      ---------- --------- ------
SQL>SELECT * FROM REGISTER;      1001       23-AUG-24 1
                                 1002       05-SEP-24 3
```

Ex: Write a query to retrieve Student, Course and Register from the given tables?

Non – Ansi:

--------------

```
SQL>SELECT SNAME, CNAME, REGDATE FROM STUDENT S,
COURSES C, REGISTER R WHERE S.CID = C.CID AND C.CID = R.CID;
```

Ansi:

-----

```
SQL>SELECT SNAME, CNAME, REGDATE FROM STUDENT S JOIN
COURSES C ON S.CID = C.CID JOIN REGISTER R ON C.CID = R.CID;
```

19-02-25

CONSTRAINTS

->Constraints are used to restrict/enforce unwanted data (i.e Invalid data) from a table.

->Oracle supports the following six types of constraints.

1> UNIQUE
2> NOT NULL
3>CHECK
4>PRIMARY KEY (UNIQUE + NOT NULL)
5>FOREIGN KEY
6>DEFAULT

->Constraints are applying on two levels.

1) Column – level:

->In this level we are applying a constraints on each individual column wise in the table.

Syntax:
Create table<table name>(<column name1><datatype>[size]<constraint type>, <column name2><datatype>[size]<constraint type>,………..;

2) Table – level:

-> In this level we are applying a constraint after all columns definition i.e the end of the table structure.

Syntax:
Create table<table name>(<column name1><datatype>[size],<column name2><datatype>[size],………..,<constraint>(<col1>,<col2>,……);

1> UNIQUE :
===========

->To restrict duplicate values but allowed Nulls.

Ex:
(Column-level):

SQL>CREATE TABLE USERS1(USERID VARCHAR2(20) UNIQUE, PASSWORD VARCHAR2(10) UNIQUE);

Testing :

SQL>INSERT INTO USERS1 VALUES(‘S@GMAIL.COM’,’ABC123’);-→Allowed
SQL>INSERT INTO USERS1 VALUES(‘S@GMAIL.COM’,’ABC123’);-→Not- Allowed
SQL>INSERT INTO USERS1 VALUES(NULL,NULL);-→Allowed

(Table – level):

SQL>CREATE TABLE MATCH_TABLE(TEAM1 VARCHAR2(10),TEAM2 VARCHAR2(10),UNIQUE(TEAM1,TEAM2));

Testing:

SQL>INSERT INTO MATCH_TABLE VALUES(‘IND’,’AUS’);----→Allowed
SQL>INSERT INTO MATCH_TABLE VALUES(‘IND’,’AUS’);→Not-Allow
SQL>INSERT INTO MATCH_TABLE VALUES(‘IND’,’SA’);----→Allowed
SQL>INSERT INTO MATCH_TABLE VALUES(’AUS’,’IND’);----→Allowed
SQL>INSERT INTO MATCH_TABLE VALUES(NULL,NULL);----→Allowed

2>NOT NULL:
===========

-> To restrict NULLs but not duplicates.
->It can be applied on column level but not table level.

Ex:
(Column – level):

SQL>CREATE TABLE TEST1(SNO NUMBER(2) NOT NULL, NAME VARCHAR2(20) NOT NULL);

Testing:

SQL>INSERT INTO TEST1 VALUES(1,'A');-----→Allowed
SQL>INSERT INTO TEST1 VALUES(1,'A');-----→Allowed
SQL>INSERT INTO TEST1 VALUES(null,null);-----→Not- Allowed

Note: Not Null is not allowed in table level.

3>CHECK:
-========

->To check the values with user defined condition before accepting into a column.

Ex:
(Column – level):

```
SQL>CREATE TABLE APPLICATION_FORM
2       (
3       REGNO NUMBER(4) UNIQUE NOT NULL,
4       CNAME VARCHAR2(10) NOT NULL,
5       ENTRY_FEE NUMBER(6,2) NOT NULL CHECK(ENTRY_FEE=500),
6       AGE_LIMIT NUMBER(2) NOT NULL CHECK(AGE_LIMIT
BETWEEN 18 AND 30),
7       LOC VARCHAR2(10) NOT NULL CHECK(LOC
IN('HYD','DELHI','MUMBAI'))
8       );
```

Testing:

SQL>INSERT INTO APPLICATION_FORM
VALUES(1001,'SMITH',499,17,'HYDERABAD');---------→Not – Allowed
SQL>INSERT INTO APPLICATION_FORM
VALUES(1001,'SMITH',500,18,'HYD');------------→Allowed

(Table – level):

SQL>CREATE TABLE TEST2(ENAME VARCHAR2(10), SAL
NUMBER(8,2),CHECK(ENAME = LOWER(ENAME) AND SAL>15000));

Testing:

SQL>INSERT INTO TEST2 VALUES('SMITH',15000);---→Not-Allowed
SQL>INSERT INTO TEST2 VALUES('smith',25000);---→Allowed

20-02-25

4>PRIMARY KEY:
==============

->It is a combination of unique and not null constraints.
->To restrict duplicate values and also Nulls from columns.
->A table is having only one primary key constraint.
->A primary key constraint is also called as candidate key /Super key in the table.

Ex:
(Column – level):

SQL>CREATE TABLE TEST3(USERID VARCHAR2(10) PRIMARY KEY, UNAME VARCHAR2(10), MBNO NUMBER(10) UNIQUE NOT NULL);

Testing:

SQL>INSERT INTO TEST3 VALUES('s@gmail','SMITH',9874563214);-→Allowed
SQL>INSERT INTO TEST3 VALUES('s@gmail','yuvi',9874563214);-→Not allowed
SQL>INSERT INTO TEST3 VALUES('y@gmail','yuvi',9874563215);→Allowed
SQL>INSERT INTO TEST3 VALUES(null,'yuvi',null)→Not allowed

=>COMPOSITE PRIMARY KEY (Table level):
====================================

->When we apply primary key constraint on multiple combinations of columns in the table then we called as "Composite Primary Key".
->In composite primary key mechanism individual columns are accepting duplicate values but combination of columns are never accept duplicate values.

Ex:

```
SQL>CREATE TABLE BRANCH(BCODE NUMBER(4),BNAME
VARCHAR2(10),BLOC VARCHAR2(10), PRIMARY
KEY(BCODE,BNAME);
```

Testing:

```
SQL>INSERT INTO BRANCH VALUES(1021,'SBI','AMMERPET');
→Allowed

SQL>INSERT INTO BRANCH VALUES(1021,'SBI','AMMERPET');-
→Not allowed

SQL>INSERT INTO BRANCH VALUES(1022,'SBI','MADHAPUR');-
→allowed

SQL>INSERT INTO BRANCH VALUES(1021,'HDFC','AMMERPET');-
→ allowed
```

5>FOREIGN KEY (reference):

=====================

->It is used to establish relationships between tables for taking some referential data (i.e., Identity) from one table to another table.

21-02-25

Basic Rules:

----

1. To maintain common column name(Optional) in both tables.
2. Those columns datatype must be matched.
3. One table should have "Primary key", another table should have "Foreign key".
4. A primary key table is called as "Parent table" and a foreign key is called as "Child table" in the relationship.
5. A foreign key column is allowed the values which should be there in primary key column.
6. By default a foreign key column is allowed duplicates and nulls.

Syntax:

<Common column name of child table><datatype>[size] references <parent table name>(Common column name)

Ex:
SQL>CREATE TABLE COLLEGE(COLLEGE_CODE NUMBER(4) PRIMARY KEY, COLLEGE_NAME VARCHAR2(10));→ Parent Table.
SQL>INSERT INTO COLLEGE VALUES(1021,’CBIT’);
SQL>INSERT INTO COLLEGE VALUES(1022,’VBIT’);
SQL>COMMIT;

SQL>CREATE TABLE STUDENT(STID NUMBER(3), SNAME VARCHAR2(10), COLLEGE_CODE NUMBER(4) REFERENCES COLLEGE(COLLEGE_CODE));-------→Child Table
SQL>INSERT INTO STUDENT VALUES(101,’SMITH’,1021);
SQL>INSERT INTO STUDENT VALUES(102,’ALLEN’,1021);
SQL>INSERT INTO STUDENT VALUES(103,’WARD’,1022);
SQL>INSERT INTO STUDENT VALUES(104,’JONES’,NULL);
SQL>COMMIT;

=>Updating NULLs in child table with references values of parent table:

Ex:
SQL>UPDATE STUDENT SET COLLEGE_CODE=1023 WHERE COLLEGE_CODE IS NULL;------→Not allowed
SQL>UPDATE STUDENT SET COLLEGE_CODE=1021 WHERE COLLEGE_CODE IS NULL;--------→Allowed
(Or)
SQL>UPDATE STUDENT SET COLLEGE_CODE=1022 WHERE COLLEGE_CODE IS NULL;--------→Allowed

Note:

->Once we establish relationships between tables there are two rules are come into picture.

Rule-1(INSERTION)

->We cannot insert values into foreign key column those values are not existing in primary key column of parent table.

[i.e No Parent = No Child]

Ex:
```
SQL>INSERT INTO STUDENT VALUES(105,'SCOTT',1023);
Error at line 1:
ORA – 02291:Integrity Constraint(____ C0081344) Violated parent key not found.
```

Rule-2(DELETION)

->We cannot delete a row from a parent table when those parent rows are having the corresponding child rows in child table without addressing to child.

Ex:
```
SQL>DELTE FROM COLLEGE WHERE COLLEGE_CODE = 1021;
Error at line 1:
ORA – 022392:Integrity Constraint(MYDB11AM.SYS_C008344) Violated child record found.
```

=>How to address to child table:

->If we want to address to child table then we have the following two types of" Cascade Rules".

i. ON DELETE CASCADE
ii. ON DELETE SET NULL

i. ON DELETE CASCADE:

->When we delete a row from parent table then the corresponding child rows also deleted from child table automatically.

Ex:
```
SQL>CREATE TABLE COLLEGE1(COLLEGE_CODE NUMBER(4)
PRIMARY KEY, COLLEGE_NAME VARCHAR2(10));-→Parent Table
SQL>INSERT INTO COLLEGE1(1021,'CBIT');
SQL>INSERT INTO COLLEGE1(1022,'VBIT');
SQL>COMMIT;

SQL>CREATE TABLE STUDENT1(STID NUMBER(3), SNAME
VARCHAR2(10), COLLEGE_CODE NUMBER(4) REFERENCES
COLLEGE1(COLLEGE_CODE) ON DELETE CASCADE);→Child Table
SQL>INSERT INTO STUDENT1 VALUES(101,'SMITH',1021);
SQL>INSERT INTO STUDENT1 VALUES(102,'ALLEN',1021);
SQL>INSERT INTO STUDENT1 VALUES(103,'WARD',1021);
SQL>COMMIT;
```

Testing:
```
SQL>DELETE FROM COLLEGE1 WHERE COLLEGE_CODE = 1022;
-------→Allowed
```

22-02-25
----
ii. ON DELETE SET NULL:

->When we delete a row from parent table the corresponding child rows foreign key column values are converting into NULLs in child table automatically.

Ex:
```
SQL>CREATE TABLE COLLEGE2(COLLEGE_CODE NUMBER(4)
PRIMARY KEY, COLLEGE_NAME VARCHAR2(10));→Parent table
SQL>INSERT INTO COLLEGE2 VALUES(1021,'CBIT');
SQL>INSERT INTO COLLEGE2 VALUES(1022,'VBIT');
SQL>COMMIT;

SQL>CREATE TABLE STUDENT2(STID NUMBER(4), SNAME
VARCHAR2(10), COLLEGE_CODE NUMBER(4) REFERENCES
COLLEGE2(COLLEGE_CODE) ON DELETE SET NULL);→Child table
```

SQL>INSERT INTO STUDENT2 VALUES(101,'SMITH',1021);
SQL>INSERT INTO STUDENT2 VALUES(102,'ALLEN',1021);

Testing:
SQL>DELETE FROM COLLEGE2 WHERE COLLEGE_CODE = 1021;→Allowed

***Data Dictionaries:

->Whenever we are installing oracle software internally system is preparing some pre-defined tables are called as "Data Dictionaries".

->These data dictionaries are used for storing the information about Tables, Constraints, Views, Sequences, Indexes, Procedures, Functions, Triggers,…etc.

->We cannot perform DML operations on data dictionaries but we can perform "DESC","SELECT" commands. So that data dictionaries are also called as "READ ONLY TABLES" in oracle.

->To view all data dictionaries in oracle database then we follow the following.

Syntax:

Select * from DICT;(Dictionary)

=>Pre – defined Constraint names:-

->When we apply constraint on a particular column internally system is created a unique identity number of constraint for identifying constraint type.

Ex:
SQL>CREATE TABLE TEST4(PCODE NUMBER(4) PRIMARY KEY, PNAME VARCHAR2(10) UNIQUE);

Note:

->If we want to view constraints name along with column name of specific table in oracle database then we use a dictionary is “USER_CONS_COLUMNS”.

Ex:
```
SQL>DESC USER_CONS_COLUMNS;
SQL>SELECT COLUMN_NAME, CONSTRAINT_NAME FROM
USER_CONS_CONSTRAINTS WHERE TABLE_NAME = ‘TEST4’;

COLUMN_NAME                    CONSTRAINT_NAME
----------------------         ----------------------------
PCODE                          SYS_C008349
PNAME                          SYS_C008350
```

=> How to create user defined constraint name(id):

Syntax:
<column name><datatype>[size]<constraint><user defined constraint name><constraint type>

Ex:
```
SQL>CREATE TABLE TEST5(PCODE NUMBER(4) CONSTRAINT
PCODE_PK PRIMARY KEY, PNAME VARCHAR2(10) CONSTRAINT
PNAME_UQ UNIQUE);
Table created.

SQL>SELECT COLUMN_NAME, CONSTRAINT_NAME FROM
USER_CONS_COLUMNS WHERE TABLE_NAME = ‘TEST5’;

COLUMN_NAME                    CONSTRAINT_NAME
----------------------         ----------------------------
PCODE                          PCODE_PK
PNAME                          PNAME_UQ
```

24-02-25

=>How to add constraint to an existing table:

Syntax:
Alter table<table name>add constraint <constraint key name><constraint type>(column name);

Ex:
SQL>CREATE TABLE PARENT(EID NUMBER(4), ENAME VARCHAR2(10), SAL NUMBER(8,2));
1. Adding primary key:

SQL>ALTER TABLE PARENT ADD CONSTRAINT PK_EID PRIMARY KEY(EID);

2.Adding Unique:

SQL>ALTER TABLE PARENT ADD CONSTRAINT UQ_ENAME UNIQUE(ENAME);

3.Adding Check:

SQL>ALTER TABLE PARENT ADD CONSTRAINT CHK_SAL CHECK (SAL>=15000);

Note:

->To view check constraint name along with conditional value of a specific table in oracle database then we use a data dictionary is “USER_CONSTRAINT”.

Ex:
SQL>DESC USER_CONSTRAINT;
SQL>SELECT CONSTRAINT_NAME, SEARCH_CONDITION FROM USER_CONSTRAINTS WHERE TABLE_NAME = ‘PARENT’;

```
CONSTRAINT_NAME                 SEARCH_CONDITION
-----------------------------   -----------------------------
CHK_SAL                         SAL>=15000
```

4.Adding Not Null:

Syntax:
Alter table<table name> modify <column name><constraint><constraint key name> Not Null;

Ex:
```
SQL>ALTER TABLE PARENT MODIFY ENAME CONSTRAINT
NN_ENAME NOT NULL;
```

5.Adding Foreign Key:

Syntax:
Alter table<table name>add<constraint><constraint key name>foreign key(common column of child table) references <parent table name>(common column name of parent table)on delete cascade/on delete set null;

Ex:
```
SQL>CREATE TABLE CHILD(DNAME VARCHAR2(10),EID
NUMBER(4));
SQL>ALTER TABLE CHILD ADD CONSTRAINT FK_EID FOREIGN
KEY(EID) REFERENCES PARENT(EID) ON DELETE CASCADE/ON
DELETE SET NULL;
```

➔How to remove/drop a constraint from an existing table:

Syntax:
Alter table <table name> drop <constraint><constraint key name>;

i. To drop a Primary key:

Case-1: Without relationship
```
SQL>ALTER TABLE PARENT DROP CONSTRAINT DK_EID;
```

Case-2: With relationship
Method-1:

SQL>ALTER TABLE CHILD DROP CONSTRAINT FK_EID;---→First
SQL>ALTER TABLE PARENT DROP CONSTRAINT PK_EID;-→Later

(Or)

Method-2:

SQL>ALTER TABLE PARENT DROP CONSTRAINT PK_EID CASCADE;

ii. To drop Unique, Check, Not null:

SQL>ALTER TABLE PARENT DROP CONSTRAINT UQ_ENAME;
SQL>ALTER TABLE PARENT DROP CONSTRAINT CHK_SAL;
SQL>ALTER TABLE PARENT DROP CONSTRAINT NN_ENAME;

→How to remove a constraint name:

Syntax:
Alter table<table name> rename <constraint><old constraint name> to <new constraint name>;

Ex:
SQL>CREATE TABLE TEST6(SNO NUMBER(2) PRIMARY KEY);
SQL>SELECT COLUMN_NAME, CONSTRAINT_NAME FROM USER_CONS_COLUMNS WHERE TABLE_NAME = 'TEST6';

```
        COLUMN_NAME              CONSTRAINT_NAME
        ----------------------   -----------------------------
        SNO                      SYS_C008358
```

SQL>ALTER TABLE TEST6 RENAME CONSTRAINT SYS_C008358 TO PK_SNO;

```
        COLUMN_NAME              CONSTRAINT_NAME
        ----------------------   -----------------------------
        SNO                      PK_SNO
```

6>DEFAULT CONSTRAINT:
========================

->To assign a user defined default values to a column.

Syntax:

<column name><data type>[size]default <value/expression>

Ex: CREATE TABLE TEST7(ENAME VARCHAR2(10), LOC VARCHAR2(10), DEFAULT ‘HYD’);

Testing:

SQL>INSERT INTO TEST7 (ENAME, LOC) VALUES (‘A’,’PUNE’);
SQL>INSERT INTO TEST7(ENAME) VALUES (‘B’);

25-02-25
****

➔How to add default constraint value to a column in the table:

Syntax:

Alter table<table name> modify <column name> default <value/expression>;

Ex:
CREATE TABLE TEST8(SNAME VARCHAR2(10), SFEE NUMBER(8,2));
SQL>ALTER TABLE TEST8 MODIFY SFEE DEFAULT 5000;

Testing:

SQL>INSERT INTO TEST8 (SNAME) VALUES (‘JONES’);
SQL>INSERT INTO TEST8 (SNAME,SFEE) VALUES (‘ALLEN’,12000);
SQL>SELECT * FROM TEST8;

Note:

->If we want to view default value of a column in the table then we use a data dictionary is “USER_TAB_COLUMNS”.
SQL>DESC USER_TAB_COLUMNS;

```
SQL>SELECT COLUMN_NAME, DATA_DEFAULT FROM
USER_TAB_COLUMNS WHERE TABLE_NAME = 'TEST8';
        COLUMN_NAME              DATA_DEFAULT
        ----------------------   ---------------------
        SFEE                     5000
```

➔How to remove default constraint value from a column:

Syntax: Same as adding constraint

```
SQL>ALTER TABLE TEST8 MODIFY SFEE DEFAULT NULL;
SQL>SELECT COLUMN_NAME, DATA_DEFAULT FROM
USER_TAB_COLUMNS WHERE TABLE_NAME = 'TEST8';
        COLUMN_NAME              DATA_DEFAULT
        ----------------------   ---------------------
        SFEE                     NULL
```

SUB QUERY (BY SUDHAKAR SIR)
****

*Sub Query:
=========

->A query inside another query is called as sub query/ nested query.

Syntax:

Select * from <table name> where <condition> (select * from ……(select * from………..(select * from………..))))));

->A subquery statement is having two more queries. Those are,

(i) Parent query/Main query/Outer query

(ii) Child query/Sub query/Inner query

->As per the execution process of subquery statement it again classified into two types.

(a) Non – Correlated Sub query

(b) Correlated Sub query

(a) Non – Correlated Sub query:

->In this mechanism first inner query is executed and later outer query will executed.

(b) Correlated Sub query:

->In this mechanism first outer query is executed and later inner query will execute.

** Non – Correlated Sub Query:

=====================

(i) Single Row Sub Query
(ii) Multiple Row Sub Query
(iii) Multiple Column Sub Query
(iv) In – Line View Sub Query

(i) Single Row Sub Query:

->When a subquery return a single value is known as Single Row Sub Query.

->In SRSQ we will use the following operators are "=,<,>,<=,>=,!=(or) <>".

Ex:

1. Write a query to find out maximum salary from emp table ?

SQL>SELECT MAX(SAL) FROM EMP;

2. Write a query to display employees details who are earning the first highest salary in the organization?

SUBQUERY = OUTER QUERY + INNER QUERY

Step -1: Inner Query:

SQL>SELECT MAX(SAL) FROM EMP; -→5000

Step -2: Outer Query:

SQL>SELECT * FROM EMP WHERE SAL<inner query return value column name> = (inner query);

Step -3: Sub Query Statement = (Outer + Inner)

SQL>SELECT * FROM EMP WHERE SAL = (SELECT MAX(SAL) FROM EMP);

26-02-25
****
Ex: Write a query to display the senior most employees details from emp table?
SQL>SELECT * FROM EMP WHERE HIREDATE = (SELECT MIN(HIREDATE) FROM EMP);

Ex: Write a query to display employees details whose salary is more than to the maximum salary of “salesman”?
SQL>SELECT * FROM EMP WHERE SAL>(SELECT MAX(SAL) FROM EMP WHERE JOB = ‘SALESMAN’);

Ex: Write a query to find out the second maximum salary from emp table?
SQL>SELECT MAX(SAL) FROM EMP WHERE SAL>(SELECT MAX(SAL) FROM EMP);

Ex: Write a query to display employees details who are getting the 2nd highest salary from emp table?
SQL>SELECT * FROM EMP WHERE SAL=(SELECT MAX(SAL) FROM EMP WHERE SAL<(SELECT MAX(SAL) FROM EMP));

Ex: Write a query to display employees details who are getting the 3rd highest salary from emp table?
SQL>SELECT * FROM EMP WHERE SAL= (SELECT MAX(SAL) FROM EMP WHERE SAL<(SELECT MAX(SAL) FROM EMP WHERE SAL<(SELECT MAX(SAL) FROM EMP));

(ii) Multiple Row Sub Query:

->When a subquery return more than one value is known as “Multiple Row Sub Query”.
->In this subquery we will use the following operators are “IN”ANY”,”ALL”.

Ex: Write a query to display employees details who’s job is same as the employee “MARTIN”,”ADAMS” jobs?
SQL>SELECT * FROM EMP WHERE JOB IN (SELECT JOB FROM EMP WHERE ENAME IN(“MARTIN”,”ADAMS”);

(OR)

SQL>SELECT * FROM EMP WHERE JOB IN(SELECT JOB FROM EMP WHERE ENAME = ‘MARTIN’ OR ENAME = ‘ADAMS’);

Ex: Write a query to display employees details who are getting minimum , maximum salaries from emp table?

SQL>SELECT * FROM EMP WHERE SAL IN((SELECT MIN(SAL) FROM EMP),(SELECT MAX(SAL) FROM EMP));

(OR)

SQL>SELECT * FROM EMP WHERE SAL IN((SELECT MIN(SAL) FROM EMP) UNION (SELECT MAX(SAL) FROM EMP));

Assignment:

1.Write a query to display the senior most employees details from each department number wise?

```
SQL>SELECT * FROM EMP WHERE HIREDATE
2       IN
3       (SELECT MIN(HIREDATE) FROM EMP GROUP BY DEPTNO);
```

2.Write a query to display employees details who are getting maximum salary from each job wise?

```
SQL>SELECT * FROM EMP WHERE SAL
2       IN
3       (SELECT MAX(SAL) FROM EMP GROUP BY JOB);
```

=> ANY Operator:

->It returns TRUE if any one value is satisfied with the given conditional value.

```
Ex:   if X (10) > ANY (10,20,30)
            X (10) = FALSE
            X (20) = TRUE
            X (40) = TRUE
```

=> ALL Operator:

->It returns TRUE if all one value is satisfied with the given conditional value.

```
Ex:	if X (10) > ALL (10,20,30)
		X (10) = FALSE
		X (20) = FALSE
		X (40) = TRUE
```

Ex: Write a query to display the list of employees whose salary is more than all "SALESMAN" salaries?

```
SQL> SELECT * FROM EMP WHERE SAL
2	> ALL
3	(SELECT SAL FROM EMP WHERE JOB = 'SALESMAN');
```

Ex: Write a query to display list of employees details whose salary is more than any "SALESMAN" salaries?

```
SQL> SELECT * FROM EMP WHERE SAL
2	> ANY
3	(SELECT SAL FROM EMP WHERE JOB = 'SALESMAN');
```

ANY Operator	ALL Operator
X > ANY (<List of values>)	X>All (<List of values>)
X >=ANY (<List of values>)	X >= All (<List of values>)
X < ANY (<List of values>)	X < All (<List of values>)
X <= ANY (<List of values>)	X <= All (<List of values>)
X = ANY (<List of values>)	X = All (<List of values>)
X! = ANY (<List of values>)	X!= All (<List of values>)

(iii). Multiple Column Sub Query:

-> Comparing multiple column values of inner query with multiple columns values of outer query is known as Multiple Column Sub Query.

Syntax:

Select * from <table name> where (<column name1>, <column name2>, ………) in (select <column name1>, <column name2>, ……….. from <table name>……);

=>Why we need Multiple Column Sub Query:

->Generally in multiple row subquery mechanism whenever we are comparing group of values of a column there is more possibility to give wrong values.

->To overcome the above problem then we should use "Multiple Column Sub Query" mechanism.

Ex: Write a query to display the employees details who are getting maximum salary from each job wise?

```
SQL> SELECT * FROM EMP WHERE SAL IN
2       (SELECT JOB, MAX(SAL) FROM EMP GROUP BY JOB);
```

Output:

```
JOB                 MAX(SAL)
-----               ---------------
SALESMAN            1600
MANAGER             2975
ANALYST             3000
PRESIDENT           5000            WRONG RESULT
ANALYST             3000
CLERK               1300
```

Solution:

```
SQL> SELECT * FROM EMP WHERE (JOB, SAL) IN
2       (SELECT JOB, MAX(SAL) FROM EMP GROUP BY JOB);
```

Output:

```
JOB                MAX(SAL)
-----              ----------------
SALESMAN           1600
MANAGER            2975
ANALYST            3000
PRESIDENT          5000
CLERK              1300
```

Ex: Write a query to display the list of employees details whose manager salary columns values are same as the employee “MARTIN” mgr, salary from emp table?

```
SQL> SELECT * FROM EMP WHERE (MGR, SAL) IN
2       (SELECT MGR, SAL FROM EMP WHERE ENAME = 'MARTIN');
```

* PSEUDO COLUMNS:
****

->Pseudo columns are working just like table columns in database.

1. ROWID
2. ROWNUM

1. ROWID:
========

->When we insert a new row into a table internally oracle server is creating a permanent row address row identify for each row wise automatically.

Ex:

```
SQL>SELECT ENAME, ROWID FROM EMP;
SQL>SELECT EMP.*, ROWID FROM EMP;
SQL>SELECT ENAME, DEPTNO, ROWID FROM EMP WHERE DEPTNO
= 10;
```

Ex:

```
SQL>SELECT MIN(ROWID) FROM EMP;
AAATH4AAHAAAAFzAA
SQL>SELECT MAX(ROWID) FROM EMP;
AAATH4AAHAAAAFzAN
```

=>How to delete multiple duplicate rows except one duplicate row from a table:

Demo Table:
=========

```
SQL>SELECT * FROM TEST;
            SNO             NAME
            -----           ---------
            1               A
            1               A
            1               A
            2               B
            3               C
            3               C
            4               D
            4               D
            4               D
            5               E
            5               E
```

Solution:

```
SQL>DELETE FROM TEST WHERE ROWID NOT IN
2       (SELECT MAX(ROWID) FROM TEST GROUP BY SNO);
```

Output:

```
SQL>SELECT * FROM TEST;
            SNO             NAME
            -----           ---------
            1               A
            2               B
            3               C
            4               D
            5               E
```

01/03/25
****
2.ROWNUM:
==========

->To generate row numbers to each row wise.to each group of row wise on a table automatically. These row numbers are not saved in databases so that these are temporary numbers.
Ex:
```
SQL>SELECT ENAME, ROWNUM FROM EMP;
SQL>SELECT ENAME, DEPTNO, ROWNUM FROM EMP WHERE
DEPTNO = 10;
```
Note:

->Generally ROWNUM pseudo column is used to perform “Nth” and “Top n” operations in a table.

Ex: Write a query to fetch the 1st position row from emp table by using ROWNUM?
```
SQL>SELECT * FROM EMP WHERE ROWNUM = 1;
```

Ex: Write a query to fetch the 2st position row from emp table by using ROWNUM?
```
SQL>SELECT * FROM EMP WHERE ROWNUM = 2;
No rows selected
```

Note:

->Generally, ROWNUM always starts with 1 for every selected row from a table. So, to overcome this problem we should use the operators are “<, <= and MINUS”.
Solution:

```
SQL>SELECT * FROM EMP WHERE ROWNUM <=2
2       MINUS
3       SELECT * FROM EMP WHERE ROWNUM =1;
```

Ex: Write a query to fetch 5th row from emp table by using ROWNUM?

```
SQL>SELECT * FROM EMP WHERE ROWNUM <=5
2       MINUS
3       SELECT * FROM EMP WHERE ROWNUM < 5;
                                                      (OR)
                                                  ROWNUM <=4;
```

Ex: Write a query to fetch the rows from 4th row to 8th row from emp table by using ROWNUM?

```
SQL>SELECT * FROM EMP WHERE ROWNUM <=8
2       MINUS
3       SELECT * FROM EMP WHERE ROWNUM <4;
```

Ex: Write a query to fetch last two rows from emp table by using ROWNUM?

```
SQL>SELECT * FROM EMP
2       MINUS
3       SELECT * FROM EMP WHERE <=(SELECT COUNT(*) -2 FROM
        EMP);
```

Ex: Write a query to fetch Top 5 rows from emp table by using ROWNUM?

```
SQL>SELECT * FROM EMP WHERE ROWNUM <=5;
```

(iv) Inline View Sub Query (Inline View)

->Providing a select query in place of table name in SELECT statement is known as "INLINE VIEW/INLINE VIEW SUB QUERY".

(OR)

->Providing a subquery under FROM clause in SELECT statement is known as "INLINE VIEW/INLINE VIEW SUB QUERY".

Syntax:

Select * from <table name>; →select query

Syntax:

Select * from (<select query/sub query>); →inline view

Note:

->In INLINE VIEW SUB QUERY mechanism the result of inner query will act as TABLE for outer query.
(IQ)Why we need INLINE VIEW?
****
->Inline Views was introduced for two reasons:
1. Generally, columns alias names cannot use under WHERE clause condition. If we want to use "Column Alias Name" under WHERE clause then we required INLINE VIEW technique.
2. Generally, subqueries are not supporting to use "ORDER BY" clause. If we want to use ORDER BY clause in subqueries then we need INLINE VIEW.

Ex: Write a query to display employees who's annual salary is more than 25000?
Error: SELECT ENAME, SAL, SAL*12 AS ANN_SAL FROM EMP WHERE ANN_SAL>25000;
"ANN_SAL": Invalid identifier
03/03/25
****
Ex: Write a query to fetch 3rd position row from emp table by using rownum alias name along with inline view?
SQL>SELECT * FROM (SELECT EMP.*, ROWNUM R FROM EMP) WHERE R =3;
(OR)
SQL>SELECT * FROM (SELECT EMPNO, ENAME, SAL, ROWNUM R FROM EMP) WHERE R = 3;

Ex: Write a query to fetch 1st , 5th , 9th , 13th row from emp table by using ROWNUM alias name along with inline view?
SQL>SELECT * FROM (SELECT EMP.*, ROWNUM R FROM EMP) WHERE R IN (1,5,9,13);

Ex: Write a query to fetch even rows from emp table by using ROWNUM alias name along with inline view?

SQL>SELECT * FROM (SELECT EMP.*, ROWNUM R FROM EMP)
WHERE MOD(R,2) = 0;

Ex: Write a query to fetch first and last row from emp table by using ROWNUM alias name along with inline view?

SQL>SELECT * FROM (SELECT EMP.*, ROWNUM R FROM EMP)
WHERE R =1 OR R = (SELECT COUNT(*) FROM EMP);

(OR)

SQL>SELECT * FROM (SELECT EMP.*, ROWNUM R FROM EMP)
WHERE R IN(1,(SELECT COUNT(*) FROM EMP));

=>Sub query with Order By Clause:

Ex: Write a query fetch the first 3 highest salaries employees rows from emp table by using INLINE VIEW?

SQL>SELECT * FROM (SELECT EMPNO, ENAME, SAL FROM EMP
ORDER BY SAL) WHERE ROWNUM <=3;

(OR)

SQL>SELECT * FROM (SELECT EMP.* FROM EMP) WHERE ROWNUM
<=3;

Ex: Write a query to fetch the 5th highest salaried employee row from emp table by using INLINE VIEW?

```
SQL>SELECT * FROM (SELECT EMP.* FROM EMP ORDER BY SAL)
        WHERE ROWNUM <=5;
2       MINUS
3       SELECT * FROM (SELECT EMP.* FROM EMP ORDER BY SAL)
        WHERE ROWNUM <=4;
```

b. Correlated Sub Query:

->In co-related subquery mechanism first outer query is executed and later inner query will execute to get the final result.

=>How to find out “Nth” highest/lowest salary:

Syntax:

Select * from <table name><table alias name> where n-1 = (select count(distinct <column name>) from <table name><table alias name>.<column name>[<(or)>]<table alias name1>.<column name>);

Here,

```
        <       - For low salary
        >       - For high salary
```

Ex: Write a query to find out the 1st highest salary employees details?
Solution:

```
        N = 1 => N-1 => 1-1 =>0
SQL>SELECT * FROM TEST T1 WHERE 0 = (SELECT COUNT(DISTINCT SAL) FROM TEST T2 WHERE T2.SAL>T1.SAL);
```

Ex: Write a query to find out the 3rd highest salary employer details?
Solution:

```
        N = 3 => N-1 => 3-1 =>2
SQL>SELECT * FROM TEST T1 WHERE 2 = (SELECT COUNT(DISTINCT SAL) FROM TEST T2 WHERE T2.SAL>T1.SAL);
```

4/03/25
******
Ex: Write a query to find out the first lowest salary employer details?
Solution:

```
        N = 1 => N-1 => 1-1 =>0
SQL>SELECT * FROM EMP E1 WHERE 0 = (SELECT COUNT(DISTINCT SAL) FROM EMP  E2 WHERE E2.SAL>E1.SAL);
```

=>How to find out “Top n” high/low salaries:

Syntax:

Select * from <table name><table alias name 1> where n>(select count(distinct<column name>) from <table name><table alias name2> where <table alias name2>.<column name>[<(or)>]<table alias name1>.<column name>;

Here,

<	- For low salary
>	- For high salary

Ex: Write a query to display Top 3 highest salary employee details from emp table?

Solution:

N = 3 => N> => 3>

```
SQL>SELECT * FROM EMP E1 WHERE 3>
2       (SELECT COUNT(DISTINCT SAL) FROM EMP E2 WHERE
3       E2.SAL>E1.SAL);
```

Ex: Write a query to display Top 3 lowest salary employee details from emp table?

Solution:

N = 3 => N> => 3>

```
SQL>SELECT * FROM EMP E1 WHERE 3>
2       (SELECT COUNT(DISTINCT SAL) FROM EMP E2 WHERE
3       E2.SAL<E1.SAL);
```

Note:

----

1. To find out “Nth” high/low salary : N-1
2. To display “Top n” high/low salary: N>

* EXISTS OPERATOR:

==================

->It is a special operator which will use in co-related subquery only.

->It is used to check the required row is existing in a table or not

>If a row is existing in a table then it return “TRUE”.

>If a row is not existing in a table then it returns “FALSE”.

Syntax:

        Where exists(<subquery/inner query>);

Ex: Write a query to display departments details in which department the employees are working?

```
SQL>SELECT * FROM DEPT D WHERE EXITS
2       (SELECT DEPTNO FROM EMP E WHERE E.DEPTNO =
        D.DEPTNO);
```

Ex: Write a query to display departments details in which department the employees are not working?

```
SQL>SELECT * FROM DEPT D WHERE NOT EXITS
2       (SELECT DEPTNO FROM EMP E WHERE E.DEPTNO =
        D.DEPTNO);
```

===

* Transactional Control Language (TCL):

=>What is Transaction:

->To perform some operation against database.

->There are two types of transactions

(i) Implicit transaction

(ii) Explicit transaction

(i) Implicit Transaction :

->These transactions are committed by oracle server by default.

Ex: DDL Commands

(ii) Explicit Transaction:

->These transactions are committed by user.

Ex: DML Commands

->To control types over these DML transactions over a table in database then user will use TCL commands. These are

(a) COMMIT
(b) ROLLBACK
(c) SAVEPOINT

(a) COMMIT:

->To make a transaction is permanent

Syntax:

Commit;

Ex:
```
SQL>CREATE TABLE TEST (EID NUMBER(4), ENAME VARCHAR2(10));
```
Testing:

```
SQL>INSERT INTO TEST VALUES(1,'SMITH');
COMMIT;
SQL>UPDATE TEST SET EID = 101 WHERE EID = 1;
COMMIT;
SQL>DELETE FROM TEST WHERE EID = 101;
COMMIT;
```

(b) ROLLBACK:

->To cancel a transaction.

->Once we commit a transaction then we cannot rollback it.

Syntax:

Rollback;

Ex:
```
SQL>INSERT INTO TEST VALUES(1,'SMITH');
SQL>UPDATE TEST SET EID = 101 WHERE EID =1;
SQL>DELETE FROM TEST WHERE EID = 101;
ROLLBACK;
```

(c) SAVEPOINT:

->Whenever we create a save point internally oracle server is allocating a special memory for a save point for storing the information about a row/rows which we want to rollback (i.e , cancel) in the future.

Syntax to create a save point:

```
SAVEPOINT <POINTER NAME>;
```
Syntax to roll back a save point:

```
ROLLBACK TO <SAVEPOINT NAME>;
```
DEMO_TABLE:

```
SQL>SELECT * FROM TEST;
            EID         ENAME
            ----        ------------
            1           SMITH
            2           ALLEN
            3           JONES
            4           ADAMS
            5           MILLER
```
Ex:1
```
SQL>DELETE FROM TEST WHERE EID = 1;
SQL>DELETE FROM TEST WHERE EID = 5;
SQL>SAVEPOINT P1;   =>Savepoint created
SQL>DELETE FROM TEST WHERE EID = 3;
```
Case-1:
```
SQL>ROLLBACK TO P1;       (We cancelled 3rd row)
```
Case-2:
```
SQL>ROLLBACK/COMMIT;  (For remaining 1,5 rows)
```

Ex:2
```
SQL>DELETE FROM TEST WHERE EID = 1;
SQL>SAVEPOINT P1;   =>Savepoint created
SQL>DELETE FROM TEST WHERE EID IN(3,5);
```
Case:1
```
SQL>ROLLBACK TO P1;       (We cancelled 3 and 5 rows)
```

Case: 2
SQL>ROLLBACK/COMMIT; (For remaining 1st row)

=>ACID Properties:
================

->By default all relational databases are having ACID properties for manage and maintain accurate and consistency data/information in database

> A	-	Automicity
> C	-	Consistency
> I	-	Isolation
>D	-	Durability

Automicity:

->All operations in a transaction will treat as a single transaction in database. (Automic = Singel)

Ex: Withdraw Transaction

ATM

Step1: Insert ATM Card

Step2: Select Language

Step3: Click On Banking Option

Step4: Click On Withdraw Option

Step5: Enter Amount: 500

Step6: Select Saving/Current Account

Strp7: Enter PinNo: xxxx

Step8: Yes (Commit)/No (Rollback)

→ As a Single Transaction.

05-03-25
****

Consistency:

->To maintain accurate information after/before a transaction.

Ex:

X-Customer	Y-Customer
---------------	---------------
Main bal:7000 (before transaction)	Main bal:2000 (before transaction)
Transfer:4000 (debit)	Received:4000 (Credit)
==========================	==========================
New bal :3000	New bal :6000
Debit amount:4000	Credit amount:4000
==========================	==========================
Main bal :7000(After Trans)	Main bal :2000 (Before Trans)
==========================	==========================

Isolation:

->Every transaction is an independent transaction i.e. one transaction will never be involved in any other transaction.

Ex: Withdraw Transaction

Withdraw 1: 500 →Trid: xxxxxxxxxxxx986 (Unique)
Withdraw 2: 1000 →Trid: xxxxxxxxxxxx987 (Unique)
Withdraw 3: 400 →Trid: xxxxxxxxxxxx988 (Unique)

Durability:

->Once we committed a transaction then we can’t roll back at any level.

==

06-03-25

****

VIEWS

***VIEWS:

=========

->It is a subset/virtual table of base table.

->It does not store data/information but stores “SELECT” query.

->The main advantage of view is:

>Providing security for data.

(i) Column Level:

-> In this level we are hiding the required columns from users.

(ii) Row Level:

->In this level we are hiding the required rows from users.

>To convert complex queries into simple query.

=>Types of Views:

----

1. Simple Views
2. Complex Views

1. Simple Views:

->When we created view based on a single base table is known as simple views.

->By default, simple views are allowed all DML operations on base table.

Syntax:

Create view<view name> as <select query>;

Ex: Create a view to access the data from dept table?

SQL>SELECT * FROM DEPT; →Base table

SQL>CREATE VIEW V1 AS SELECT * FROM DEPT;

Testing:

SQL>INSERT INTO V1 VALUES(50,’ORACLE’,’HYD’);

SQL>UPDATE V1 SET LOC = ‘PUNE’ WHERE DEPTNO = 50;

SQL>DELETE FROM V1 WHERE DEPTNO = 50;

SQL>SELECT * FROM V1;

Ex: Create a view to access EMPNO, ENAME, SAL from emp table? (Hiding Columns)

SQL>CREATE VIEW V1 AS SELECT EMPNO,ENAME,SAL FROM EMP;

Testing:

SQL>INSERT INTO V1 VALUES (1122,’YUVIN’,5000);

Ex: Create a view to access employees details who are working under deptno 20? (Hiding Rows)

SQL>CREATE VIEW V3 AS SELECT * FROM EMP WHERE DEPTNO = 20;

SQL>SELECT * FROM V3; →(Showing required department rows)

Ex: Create a view to access ENAME, HIREDATE and JOB details from emp table who are working as a “MANAGER”?

SQL>CREATE VIEW V4 AS SELECT ENAME, HIREDATE, JOB FROM EMP WHERE JOB = ‘MANAGER’;

SQL>SELECT * FROM V4;

=> View Options:

->View can be created with two options in oracle database.

i. With Check Option

ii. With Read Only

07-03-25

****

i. With Check Option:

->To restrict data on a base table through a view object.

Ex: Create a view to display and accept the employees details whose salary is 3000?

SQL>CREATE VIEW V5 AS SELECT * FROM EMP WHERE SAL = 3000 WITH CHECK OPTION ;

Testing:

SQL>INSERT INTO V5 VALUES (1122,'YUVIN','HR',NULL,NULL,8000,NULL,20);→Not allowed

SQL>INSERT INTO V5 VALUES (1122,'YUVIN','HR',NULL,NULL,2000,NULL,20);→Not allowed

SQL>INSERT INTO V5 VALUES (1122,'YUVIN','HR',NULL,NULL,3000,NULL,20);→Allowed

ii. With Read Only:

->To restrict DML operations on base table through a view object.

Ex: Create a view to restrict DML operations on DEPT table?

SQL>CREATE VIEW V6 AS SELECT * FROM DEPT WITH READ ONLY;

Testing:

SQL>INSERT INTO V6 VALUES (50,'ORACLE','HYD');→Not allowed

SQL>UPDATE V6 SET LOC = 'PUNE' WHERE DEPTNO = 10;→Not allowed

SQL>DELETE FROM V6 WHERE DEPTNO = 30;→Not allowed

SQL>SELECT * FROM V6; →Allowed

2. Complex Views:

==============

->When we create a view based on:

>Multiple tables

>By using group by

>By using having

>By using distinct

>By using set operator

>By using joins

>By using sequences

>By using grouping functions

->By default, complex views are not allowed DML operations on base table.

Syntax:

Create view <view name> as <select query>;

Ex: Create a view to display sum of salaries of each deptno wise from emp table?

SQL>CREATE VIEW V7 AS SELECT DEPTNO, SUM(SAL) FROM EMP GROUP BY DEPTNO;

Testing:

SQL>SELECT * FROM V7; (Before updating data in emp)

->We cannot perform DML operations on base table through complex views.

SQL>UPDATE EMP SET SAL = SAL+1000 WHERE DEPTNO = 10;

SQL>SELECT * FROM V7; (After updating data in emp);

Note:

->If we want to see all views in oracle database then we use "USER_VIEWS" datadictionary.

Ex:

SQL>DESC USER_VIEWS;

SQL>SELECT VIEW_NAME FROM USER_VIEWS;

=>To see "Select Query" in a views:

SQL>SELECT TEXT FROM USER_VIEWS WHERE VIEW_NAME='V7';

=>How to drop a view:

Syntax:

Drop view <view name>;

Ex:

SQL>DROP VIEW V1;

SQL>DROP VIEW V7;

SEQUENCE

***Sequence:

==========

->It is a database object which is used to generate sequence numbers on a specific column in the table automatically.

->It will provide a facility is called as “Auto Incremental Value”.

Syntax:

```
Create sequence <sequence name>
[Start with n]
[Min value n]
[Increment by n]
[Max value n]
[No cycle/cycle]
[No Cache/Cache n];
```

08-03-25

****

=>Start With n:

->To specify the starting value of sequence. Here “n” is a number.

=>Min Value n:

->To show minimum value in the sequence. Here “n” is a number.

=>Increment By n:

->To specify incremental value in between sequence numbers. Here “n” is a number.

=>Max Value n:

->To show maximum value from a sequence. Here “n” is a number.

=>No Cycle:

->It is a default attribute of sequence object.

->When we create a sequence object with "No Cycle" then the set of sequence numbers are not repeat again and again.

=>Cycle:

->When we create a sequence with "Cycle" then the set of sequence numbers are repeated again and again.

=>No Cache:

->It is a default attribute of sequence object.

->Cache is a temporary memory.

->When we created a sequence object with "No Cache" then the set of sequence numbers are saved in database memory directly. So that every user request wise go to database and retrieving the required data from database and send to client application. By this reason the burden on database will increase and degrade the performance of database.

=>Cache:

->When we created a sequence object with "Cache" then the set of sequence numbers are saved in database memory and also the copy of data is saved in cache memory. So that how every request will go to cache instead of database and retrieving the required data from cache memory and send to client application. So that we reduce the burden on database and improve the performance of database. Here "n" is the size of each file and will create with 2kb.

=>Note:

->When we generate sequence numbers on a column then we must use a pseudo column is "NEXTVAL".

NEXTVAL:

->It is a pseudo column of a sequence.

->it is used to generate next by next numbers.

Syntax:

```
<sequence by>.NEXTVAL
```

Ex:

```
SQL>CREATE SEQUENCE SQ1
    START WITH 1
    MINVALUE 1
    INCREMENT BY 1
    MAXVALUE 3;
```

Testing:

```
SQL>CREATE TABLE TEST55(SNO NUMBER(3),SNAME
VARCHAR2(10));
SQL>INSERT INTO TEST55 VALUES(SA1.NEXTVAL,’&SNAME’);
    Enter value for name : A
SQL>/
    Enter value for name: B
SQL>/
    Enter value for name: C
SQL>/
    Enter value for name: D
    Error at line:1
    ORA-08004 : Sequence SQ1.NEXTVAL exceeds MAXVALUE and
cannot be instantiated.
```

=>Altering a sequence:

Syntax:

```
Alter Sequence<sequence name><attribute name>n;
```

Ex:

```
SQL>ALTER SEQUENCE SQ1 MAXVALUE 5;
SQL>INSERT INTO TEST55 VALUES(SQ1.NEXTVAL,'&SNAME');
        Enter value for name: D
SQL>/
        Enter value for name: E
SQL>SELECT * FROM TEST55;
```

Ex:2

```
SQL>CREATE SEQUENCE SQ2
        START WITH 3
        MINVALUE 1
        INCREMENT BY 1
        MAXVALUE 5
        CYCLE
        CACHE 2;
```

Testing:

```
SQL>CREATE TABLE TEST56(SNO NUMBER(2), NAME
VARCHAR2(10));
SQL>INSERT INTO TEST56 VALUES(SQ2.NEXTVAL,'&NAME');
        Enter value for Nam: Q
SQL>/
        Enter value for Name: ……. Etc (By Cycle)
SQL>SELECT * FROM TEST56;
```

NOTE:

->Once we created a sequence object then we can apply that sequence on multiple tables. That mean's reusability.

=>Note:

->If we want to view all sequence objects in oracle database then use a data dictionary is “USER_SEQUENCES”.

Ex:

SQL>DESC USER_SEQUENCES;

SQL>SELECT SEQUENCE_NAME FROM USER_SEQUENCES;

=>How to drop sequence object from a table:

Syntax:

Drop sequence<sequence name>;

Ex:

SQL>DROP SEQUENCE SQ1;

10-03-25

****

INDEXES

***Indexes:

=========

->It is a database object which is used to retrieve the required row/rows from a table fastly.

->Database index is similar to book index page in a text book. By using book index page how we are retrieving the required topic from a text book fastly same as by using database index we are retrieving the required row/rows from a table fastly.

->Database index can be created on a particular column in the table and this column is called as ”INDEX KEY COLUMN”.

->Whenever we want to retrieve the required row/rows from a table then we must use INDEXED KEY COLUMNS under WHERE clause condition then only indexes are working.

->All databases are supporting the following two types of searching mechanisms.

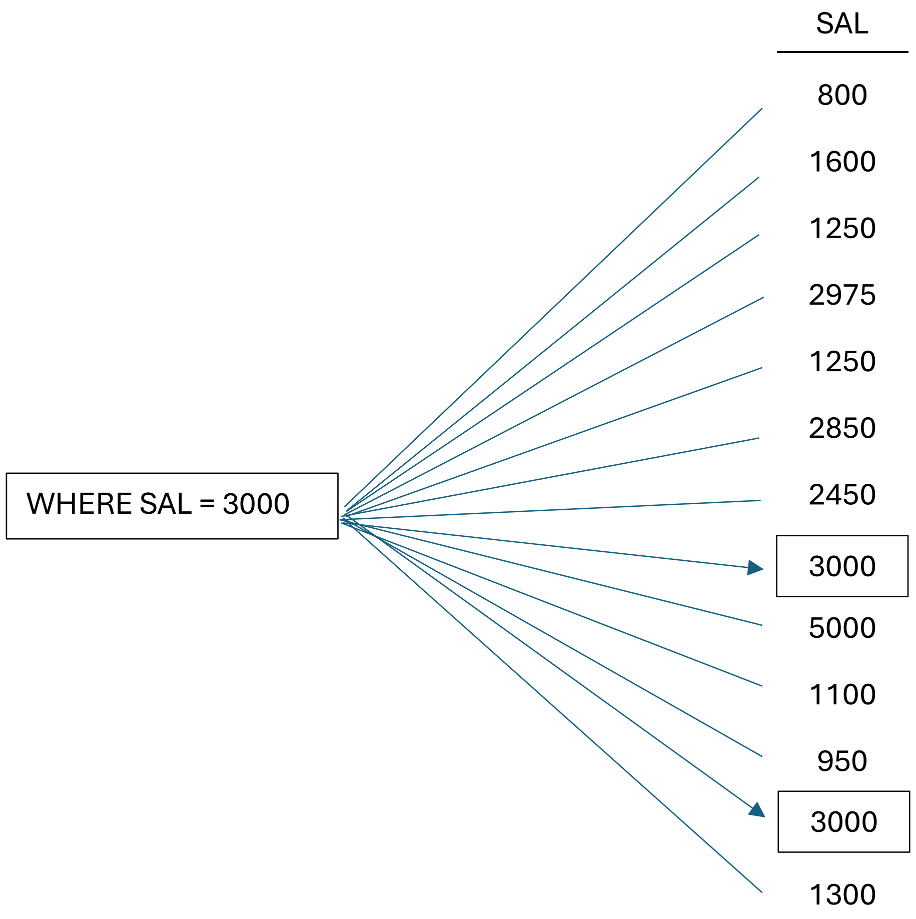
1. Table Scan (Default Scan)
2. Index Scan

1. Table Scan:

->In this scan, Oracle server is searching the entire table for required data. So that it takes much time to give the required data to user.

Ex:

SQL>SELECT * FROM EMP WHERE SAL = 3000;



2. Index Scan:

->In this scan oracle server is searching the required row based on a indexed column wise.

->These are two types of indexes in oracle

a. B-Tree Index

b. Bitmap Index

a. B-Tree Index:

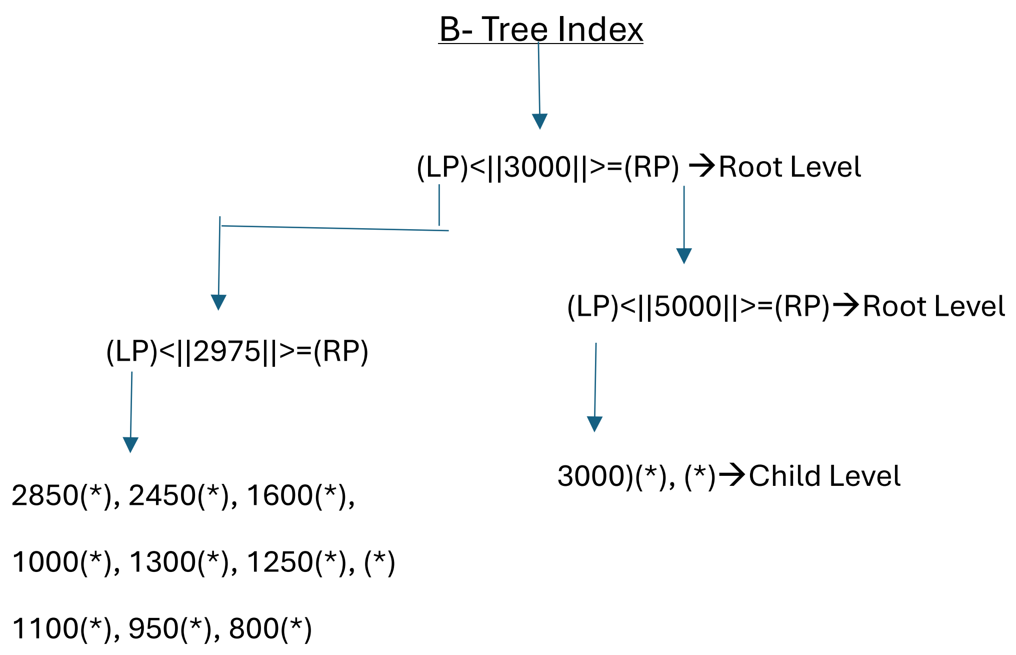
->In this mechanism data can be organized the form "Tree" structure by the system.

Syntax:

Create index<index name>on<table name>(column name);

Ex:

```
SQL>CREATE INDEX I1 ON EMP(SAL);
SQL>SELECT * FROM EMP WHERE SAL = 3000;
```



Here,

LP	=	Left Pointer
RP	=	Right Pointer
*	=	Rowid/Row Address

b. Bitmap Index:

->In this mechanism data can be organized in the form “Table” format by the system based on bit numbers are 0,1.

Here,

0 is representing when condition is false

1 represents when condition is true

Syntax:

Create bitmap index<index name> on <table name>(column name);

Ex:

SQL>CREATE BITMAP INDEX BIT1 ON EMP(JOB);

SQL>SELECT * FROM EMP WHERE JOB = ‘CLERK’;

Ex: Bitmap Indexed Table Format

JOB	1	2	3	4	5	6	7	8	9	10	11	12	13	14
CLERK	1	0	0	0	0	0	0	0	0	0	1	1	0	1

Here, “*” represents ROWID.

=>To view all indexes in oracle:

SQL>DESC USER_IND_COLUMNS;

SQL>SELECT COLUMN_NAME, INDEX_NAME FROM USER_IND_COLUMNS WHERE TABLE_NAME = ‘EMP’;

=>To view type of index in oracle:

SQL>DESC USER_INDEXES;

SQL>SELECT INDEX_NAME, INDEX_TYPE FROM USER_INDEXES WHERE TABLE_NAME = ‘EMP’;

INDEX_NAME	INDEX_TYPE
I1	Normal (B- Tree index)
B1	Bitmap

=>How to drop index:

Syntax:

Drop index<index name>;

Ex:

```
SQL>DROP INDEX I1;

SQL>DROP INDEX B1;
```

NORMALIZATION

***Normalization:

=============

->Normalization is a technique which is used to decompose a table data into multiple tables.

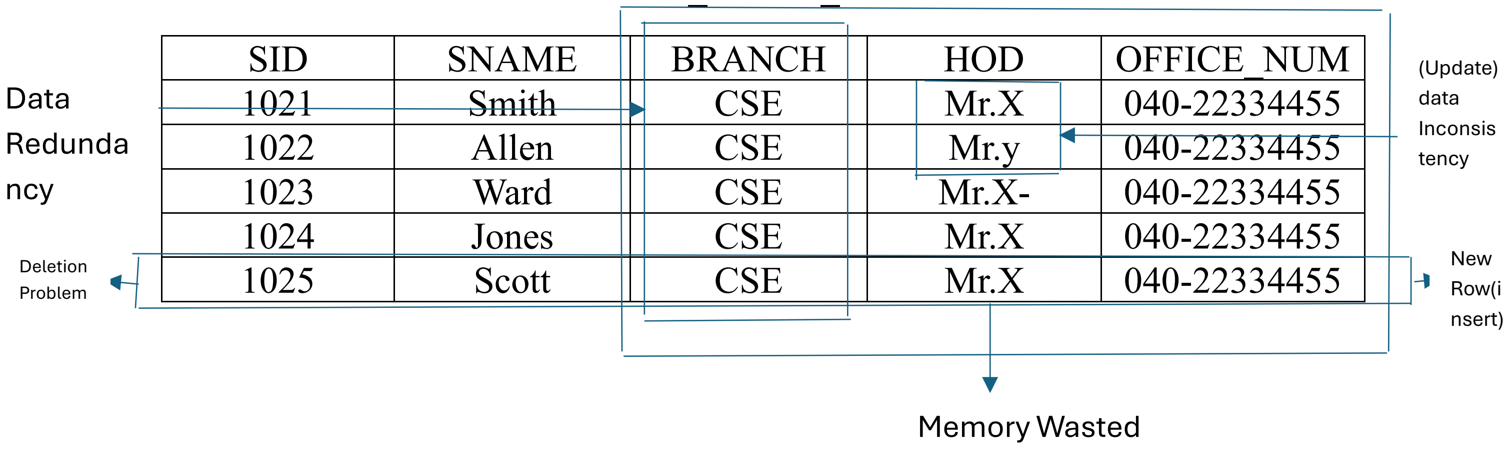
*Where we want to use normalization:

->Database designing level.

*Why we need normalization:

Ex:

Branch_Student_Details



SID	SNAME	BRANCH	HOD	OFFICE_NUM
1021	Smith	CSE	Mr.X	040-22334455
1022	Allen	CSE	Mr.y	040-22334455
1023	Ward	CSE	Mr.X-	040-22334455
1024	Jones	CSE	Mr.X	040-22334455
1025	Scott	CSE	Mr.X	040-22334455

Disadvantages:

->Data redundancy problem. (i.e duplicate data)

->Memory Wasted

->Data Inconsistency problem. (i.e Irregular data)

->Insertion problem

->Updation problem

->Deletion problem

-> To overcome the above problem on table then we must use a technique is known as "Normalization".

Using Normalization:

****

|--------------------------------Relationship--------------------------------|

(PK) Branch_Details

Bcode	Branch	HOD	Office_Num
1	CSE	Mr.X	040-22334455

Student_Details (FK)

STID	Sname	Bcode
1021	Smith	1
1022	Allen	1
1023	Ward	1
1024	Jones	1
1025	Scott	1
1026	Miller	1

Here,

PK = Primary Key

FK = Foreign Key

Advantages:

->To avoid data redundancy problem. (i.e. duplicate data)

->Memory saved.

->To avoid data inconsistency problem. (i.e. irregular data)

->To avoid Insertion problem.

->To avoid updation problem.

->To avoid deletion problem.

*Types of Normalization Forms:

$$\left.\begin{array}{l}\text{1. First Normal Form (1NF)}\\ \text{2. Second Normal Form (2NF)}\\ \text{3. Third Normal Form (3NF)}\end{array}\right\}\text{Mandatory}$$

$$\left.\begin{array}{l}\text{4. Boyce – Codd Normal Form (BCNF)}\\ \text{5. Fourth Normal Form (4NF)}\\ \text{6. Fifth Normal Form (5NF)}\end{array}\right\}\text{Optional}$$

1.First Normal Form (1NF):

->For a table to be in the first normal form, it should follow the following 4 rules:

i. Each column should contain atomic value. (Atomic = Single Value)

ii. A column should contain values that are the same datatype.

Iii. All the columns in a table should have unique names.

iv. The order in which the data is stored, does not matter.

Ex:

STID	SNAME	REGDATE	BRANCH
1	Smith	12-May-2024	IT
2	Clark	23-Mar-2024	CSE

3	Allen	02-Jan-2025	EEE
4	Blake	24-Sep-2023	ECE

2. Second Normal Form (2NF):

->For a table to be in the second normal form, it must specify two conditions:

i. The table should be in the first normal form.

ii. There should be no partial dependency.

=>What is dependency:

->In a table if non-key columns (non-primary key) are depending on key column (primary key) then it is called as fully dependency/functional dependency.

Ex: (PK)

STID SNAME BRANCH ADDRESS

Here, "STID" is a key column and "SNAME","BRANCH","ADDRESS" are non-key columns.

These non-key columns are linked with the key column is STID. So that in this table there is no partial dependency columns.

=>What is Partial dependency:

-> In a table if non-key column depends on part of the key column, then it is called Partial dependency.

Ex: <Primary key(Stu_id/Sub_id)/composite primary key>

STU_ID SUB_ID STU_MARKS TEACHER

->Here, "STU_ID" and "SUB_ID" is a key columns "STU_MARKS","TEACHER" are non-key columns. Then "TEACHER" depends on "SUB_ID" but not "STU_ID" column.

->Here we found a partial dependency column is "TEACHER" so that we need to do decompose a table like below,

Solution:

SUBJECT_TABLE STUDENT_TABLE

(PK) (PK) (FK)

SUB_ID SUB_NAME TEACHER STU_ID STU_MARKS SUB_ID

3.Third Normal Form (3NF):

->For a table to be in the third normal form there is two conditions:

i. It should be in the second normal form.

ii. And it should not have Transitive dependency.

=>Transitive dependency:

->In a table if non-key column depends on another non-key column, then it is called as “Transitive Dependency”

12-03-25

****

```
Ex:    |-----------CPK---------|
  STUDENT_ID        SUBJECT_ID   STU_MARKS      EXAM_NAME       TOTAL_MARKS
 ==========         =========   ===========    ===========      ============
```

->Here, ”STU_ID” and “SUB_ID” are key columns,”EXAM_NAME”,”TOTAL_MARKS” are non-key columns. Then “TOTAL_MARKS” depends on “EXAM_NAME” but not “STU_ID” and “SUB_ID” columns.

->Here we found transitive dependency columns are “EXAM_NAME” and “TOTAL_MARKS” so that we need to do decompose the above table into multiple tables.

Solution:

(PK) EXAM_TABLE (CPK) SCORE_TABLE (FK)

EXAM_ID||EXAM_NAME||TOTAL_MARKS STU_ID||SUB_ID||STU_MARKS||EXAM_ID

4. Boyce – Codd Normal Form (BCNF):

->For a table to satisfy the Boyce – Codd Normal Form, it should satisfy the following two conditions

i. It should be in the Third Normal Form.

ii. And, for any dependency A->B, A should be a super key.

=>Super Key:

->A column (or) combination of columns which are uniquely identifying a row in a table is called as "Super Key".

=>Candidate Key:

->A minimal super key which is uniquely identifying a row in a table is called as "Candidate Key".

(Or)

->A super key which is subset of another super key, but the combination of super key are not a candidate key.

Ex: STUDENT_TABLE

STUDENT_ID || NAME || BRANCH || EMAIL_ID || REG_NUMBER

Super Key Columns:

Student_id	Student_id+mail_id	
Mail_id	mail_id+reg_number	Student_id+mail_id+
Reg_number	reg_number+Student_id	reg_number

Candidate Key Columns:

Student_id

Mail_id

Reg_number

Ex: Professor_Table

|------------------CPK---------------------|

Professor_Id	Subject (B)	Professor (A)
1	Java	P.Java
2	Java	P.Java

->Here, Professor column depends on subject so that professor should be super key but it is not a super key.

->Now to make professor column is super key and subject is non-super key column in the table then we do some changes in a table like below.

Solution:

Professor_Table

|----------------CPK-----------------|

Professor_Id	Professor (A)	Subject (B)
1	P.Java	Java
2	P.Java	Java

5. Fourth Normal Form (5NF):

->For a table to satisfy the fourth normal form, it should satisfy the following two conditions
i. It should be in the Boyce – Codd Normal Form
ii. A table does not contain more than one independent multi valued attribute/Multi valued dependency.

=>Multi Valued Dependency:

->In a table one column same value mapping with multiple values of another column is called as "Multiple Valued Dependency".

Ex:

Multi Valued Columns

COLLEGE ENROLLMENT TABLE (5NF)

Student_Id	COURSE	HOBBY
1	ORACLE	Cricket
1	JAVA	Reading
1	HTML	Hockey

Mapping with multiple values of columns after decomposing the table:

Course_Details (4NF)

Student_ID	Course
1	Oracle
1	Java
1	HTML

Hobbies_Details (4NF)

Student_ID	Hobby
1	Cricket
1	Reading
1	Hockey

6. Fifth Normal Form (5NF):

->If a table is having multi valued attributes and also that table cannot decomposed into multiple tables are called as "Fifth Normal Form".

Ex: COLLEGE ENROLLEMENT TABLE (5NF)

STUDENT_ID	COURSE	HOBBY
1	Oracle	Cricket
1	Java	Reading
1	HTML	Hockey