Demo Sessions:

Day-1 https://youtu.be/v3smWCRgDEg
Day-2 https://youtu.be/Y6AnObIfndE
Day-3 https://youtu.be/BIMc-kxffuE
Day-4 https://youtu.be/kQ10nrz9Fzw
Day-5 https://youtu.be/AfNi7spt5Tg
Day-6 https://youtu.be/ZnSDP0J-zqU
Day-7 https://youtu.be/lvZno54Z_s4
Day-8 https://youtu.be/EwaDKMlSkeA
Day-9 https://youtu.be/zkHIqWabVNw

==
==
23-01-2023: ORACLE19c
==
==
DEMO-1:
=======
What is Oracle?
- It is a DB software / back end tool / RDBMS product.

Where we want to use oracle?
Using in:
> Banking
> Educational
> Hospitals
> HR management
> Transports
> Sales & Production
> Finance

Who want to learn oracle?
> Any platform

Ex:
java,.net,python,php,aws,devops,testing,powerBI,MSBI,.....etc

What about oracle job's?
> DB developer (SQL + PL/SQL)
> SQL developer
> PL/SQL programmers
==
ORACLE COURSE SCHEDULE
==========================

COURSE NAME	: ORACLE19c
DURATION	: 50 - 60 SESSIONS / CLASSES
TIME	: 4:15PM TO 5:45PM (1.30PM - CLASS) + 15mints(DOUBTS)
COURSE TYPE	: OFFLINE & ONLINE

COURSE CONTENT:
===============

- TOPIC-1 : DBMS
- TOPIC-2 : ORACLE TOOL
- TOPIC-3 : SQL
- TOPIC-4 : DE-NORMALIZATION & NORMALIZATION
- TOPIC-5 : PL/SQL
- TOPIC-6 : DYNAMIC SQL

END OF THE ORACLE COURSE:
=========================
- COMPLETE ORACLE19C MATERIAL (.pdf)
- ORACLE IQ’s (200+)
- ORACLE MORE EXAMPLES (300+ and 300+ = 500+ EXAMPLES)
- RESUME MODELS (fresh & expr)
- ORACLE COURSE CERTIFICATE ---------> BY NARESHIT MANAGEMENT.

==
25-01-2023 TOPIC-1: DBMS
==

What is Data?

- It is raw fact (i.e. char's, numbers, special char's, symbols, etc.)

Ex:

SMITH	10021
WARNER	10022
MILLER	10023
SCOTT	10024

What is Information?

- processing data is called as "information".
- is always give meaningful statements.

Ex:

Customer_name	Customer_id
SMITH	10021
WARNER	10022
MILLER	10023
SCOTT	10024

What is Database?

- It is a collection of inter-related data / information.

Ex:

```
SBI_DB (organization)
        > Group of branches
                > Group of dept.
                        > Group of employees
                                > Group of customers
```

Ex:

No departments = no employees
No employees = no departments

No customers = no products
No products = no customers

Types of Databases?

- There are two types of databases in real world.
    1. OLTP (online transaction processing)
    2. OLAP (online analytical processing)

1. OLTP:

- For storing "day - to - day" transactional data/information. (I.e 24x7 transactional).
- supporting insert, update, delete and select.

Ex:

Oracle, sqlserver, MySQL, db2, PostgreSQL, etc

2. OLAP:

- For storing "historical data/information". (i.e. big data)
- supporting "select" operation only. (Reading data)

Ex:
Data warehouse (DWH).

What is DBMS?

- It is a software which is used to maintain and manage data / information in database.
- By using DBMS s/w we can,
    - create database memory
    - create tables
    - inserting data
    - updating data
    - selecting data
    - deleting data.
- It will act as an interface between User and Database memory.

Models DBMS s/w?

1) Hierarchical Database management system (HDBMS)
   Ex: IMS s/w (Information Management System)
2) Networks Database management system (NDBMS)
   Ex: IDBMS s/w (Integrated Database Management System)

Note: HDBMS & NDBMS models are outdated models in real time.

3) Relational Database management system (RDBMS)

i) Object Relational DBMS(ORDBMS):
==============================

- Completely depends on "SQL".
- Data can be stored in "table format".

A table = collection rows & columns
A row = group of columns
A database = group of tables.
A row / a record / a tuple
A column / an attribute / a field
Ex:
Oracle, Sqlserver, MySQL, PostgreSQL, etc.

ii) Object Oriented DBMS (OODBMS):
================================

- These are not depends on "SQL". So that these databases are
called as "NO SQL" databases.
- Completely depends on "OOPS" concept.
- Data can be stored in "objects" format.

Ex:
MongoDB, Cassandra, etc.

==
26-01-2023: Topic-2: ORACLE19c
==
Session-1:

- Oracle is an rdbms (ordbms) product which was introduced by "Oracle Corporation" In 1979.it is used to store data / information permanently along with security manner.
- The first version of oracle s/w is Oracle1.0 version to Oracle19c version.
- Very latest version is Oracle21c.
- When we install / deploys oracle s/w we need a platform whereas platform is nothing it is combination of operating system and microprocessor.
- There are two types of platforms:

i) Platform Independent:
=====================
- It supports any OS and any Microprocessor.

Ex: oracle, MySQL, java, .net, python, etc.

ii) Platform Dependent:
====================
- it supports only one OS with any Microprocessor.

Ex: C-language

Working with Oracle:

- Oracle s/w is available in two editions in real world.

I) Oracle expression edition

- supporting partial features but not completely.

Ex: recyclebin, flashback, purge, partition tables, etc. are not allowed into oracle express edition.

II) Oracle enterprise edition:

- supports all features completely.

Ex: recyclebin, flashback, purge, partition tables, etc. are allowed into oracle enterprise edition.

Note:
=====
- To work with oracle DB server there are two steps to follow,

step1: installing oracle s/w
step2: connecting to oracle DB server.

step1: How to installing oracle enterprise19c edition:
===

Hardware Req:

Minimum ----- 500 GB HD (or) above / SSD120 (or) above
Minimum ----- 1 GB ram (or) above
OS -------------- any operating system (windows, Linux, Solaries, mac, etc.)
Windows10 (or) 11.

Software Req:

Oracle19c enterprise edition.

How to download oracle19c enterprise edition:
===
https://www.oracle.com/in/database/technologies/oracle19c-windows-downloads.html

HOW TO GRANTING CREATE TABLE PERMISSION TO USERS:

```
SQL> CONN SYSTEM/TIGER
SQL> GRANT CREATE TABLE TO MYDB2P
```

NOTE:

- When we installed oracle s/w internally there are two components are installed in the system those are,
  1) Client component
  2) Server component

1) Client component:
=================

- By using of this client tool we can perform the following operations are:
  - step1: can connect to oracle db server.
  - step2: can send request to oracle db server.
  - step3: can get response from oracle server.

Ex: sqlplus, sql developer, toad, etc.

2) Server component:
==================

- It is having two more sub components,
  - I) Instance
  - II) Database

I) Instance:
==========

- It is a temporary memory.
- This component memory is allocating by system from RAM.
- Data can be stored temporarily.

II) Database:
===========

- It is permanent memory.
- This component memory is allocating by system from HD.
- Data can be stored permanently.

Note:

Now, we want to work on oracle db by using the following two methods are,
- i) Connect to oracle server.
- ii) Communicate with oracle db.

i) Connect to oracle server:
=======================

- When user want to connect to oracle server we need a client tool is called as "Sqlplus".

ii) Communicate with oracle db:
===========================

- When user want to communicate with oracle database we need a db language is called as "sql".

SQLPLUS vs SQL:

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

HOW TO CONNECT TO ORACLE DB SERVER:

> go to all programs
    > go to oracle-oradb19home1 folder
        > click on sqlplus icon
            Enter username: system (default username)
            Enter password: tiger (created at installation time)
            Connected.

NOTE:
    Here, username is not a case sensitive but password is a case sensitive.

CONNECTIVITY PROBLEMS:

Problem-1:
Enter user-name: system/tiger
ERROR:
ORA-12560: TNS: protocol adapter error
Solution:
Go to services
    > go to oracleserviceORCL and click on it
        > Select startup type is "Automatic"
            > click on start button
                > click on apply and ok.

Go to open sqlplus editor and type,
    > Enter user-name: system/tiger
    Connected.

Problem-2:
Enter user-name: system/tiger
ERROR:
ORA-28000: The account is locked.
Solution:
Enter username: \sys as sysdba (default username)
Enter password: sys (default password)
Connected.

HOW TO LOCK / UNLOCK A USER ACCOUNT IN ORACLE DB:

SYNTAX:
ALTER USER <USERNAME> ACCOUNT LOCK/UNLOCK;

EX:
ALTER USER SYSTEM ACCOUNT UNLOCK;
Enter username: system /tiger
Connected.

Problem-3:
Enter username: \sys as sysdba (default username)
Enter password: sys (default password)
Error: unable to connect to oracle db server.
Solution:
Re-installation of oracle s/w.

HOW TO CREATE A NEW USERNAME AND PASSWORD IN ORACLE DB:

SYNTAX:
CREATE USER <USERNAME> IDENTIFIED BY <PASSWORD>;

EX:
STEP1:
Enter username: system/tiger
Connected

STEP2:
CREATE USER MYDB4PM IDENTIFIED BY 123;

STEP3:
Enter username: MYDB4PM/123

ERROR:
ORA-01045: user MYDB4PM lacks CREATE SESSION privilege; logon denied

Note:
- Every newly created user is called as "dummy user" in oracle db.it means that the user does not have any privileges(permissions).if this user want to connect to oracle db server then the user required a permission from DBA(system).

HOW TO GRANT CONNECTIVITY PERMISSION TO MYDB4PM USER:

SYNTAX:
GRANT <PRIVILEGE NAME> TO <USER NAME>;

EX:
SQL> CONN
Enter user-name: system/tiger
Connected.

```
SQL> GRANT CONNECT TO MYDB4PM;
Grant succeeded.

SQL> CONN
Enter user-name: MYDB4PM/123
Connected.
```

HOW TO CHANGE A USER PASSWORD IN ORACLE DB:

Ex:

```
SQL> CONN
Enter user-name: MYDB4PM/123
Connected.

SQL> PASSWORD;
Changing password for MYDB4PM
Old password: 123
New password: ABC
Retype new password: ABC
Password changed.

SQL> CONN
Enter user-name: MYDB4PM/ABC
Connected.
```

HOW TO RE-CREATE A NEW PASSWORD FOR A USER, IF WE FORGOT:

SYNTAX:

```
ALTER USER <USERNAME> IDENTIFIED BY <NEW PASSWORD>;
```

EX:

```
Enter user-name: system/tiger
Connected.
SQL> ALTER USER MYDB4PM IDENTIFIED BY MYDB4PM;
User altered.

SQL> CONN
Enter user-name: MYDB4PM/MYDB4PM
Connected.
```

HOW TO VIEW USERNAME IN ORACLE DB, IF WE FORGOT:

SYNTAX:

```
SELECT USERNAME FROM ALL_USERS;
```

EX:

```
SELECT USERNAME FROM ALL_USERS;
```

HOW TO DELETE A USER FROM ORACLE DB:

SYNTAX:
DROP USER <USERNAME> CASCADE;

EX:
DROP USER MYDB4PM CASCADE;

==
31-01-2023 TOPIC-3: SQL
==

- SQL stands for "structure query language" which is used to communicate with a database.It was introduced by "IBM" and the initial name is "sequel" and later it was renamed as a "SQL".
- SQL is not case-sensitive language i.e. we can write sql queries either in upper or lower case characters.

```
Ex:     SELECT * FROM EMP; ------executed
        select * from emp; -------executed
        SelecT * From Emp; -----executed
```

- Every sql query should ends with a semicolon " ; ".

SUB-LANGUAGES OF SQL:

1) **DATA DEFINITION LANGUAGE (DDL):**

> create
> alter
>> > alter - modify
>> > alter - add
>> > alter - rename
>> > alter - drop
> rename
> truncate
> drop

New Features:
> Recyclebin
> Flashback
> Purge

2) **DATA MANIPULATION LANGUAGE (DML):**

> insert
> update
> delete

New Features:
> insert all
> merge

3) **DATA RETRIVAL / QUERY LANGUAGE(DRL / DQL):**

> select

4) **TRANSACTION CONTROL LANGUAGE (TCL):**

> commit
> rollback
> savepoint

5) **DATA CONTROL LANGUAGE (DCL):**

> grant
> revoke

DATA DEFINITION LANGUAGE (DDL)

CREATE:

- To create a new object in oracle db.

Ex: Tables, Synonyms, Views, Sequences, Procedure, Functions, Triggers, Indexes, etc.

DATATYPES IN ORACLE:

- It is an attribute to specify what type of data is storing into a column.
- Oracle supports the following datatypes are,

i. numeric datatypes
ii. string / character datatypes
iii. long datatype
iv. date datatypes
v. raw & long raw datatypes
vi. lob datatypes

NUMERIC DATATYPES:

- int
- number(p,s)

i) int:

- storing integer values only.

Ex:

```
SNO int -----> converting -----> SNO number (38)
         int = number(38)

SNO int              SNO number (4)
======               ==============
0                    0
99                   99
100                  100
999                  999
1000                 1000
9999                 9999
10000                x
```

ii) number (p,s):

- storing integer values and also float values.

> When we give number (p) ------> storing integer only

> When we give number (p, s) ----> storing float values.

Precision (p):

- counting all digits including left and right side of a decimal point.

Ex:

i) 67.23
Precision = 4

ii) 9869.35
Precision = 6

Scale(s):

- counting the right side digits only.

Ex:

i) 67.23
Precision = 4
Scale = 2

ii) 986890.234
Precision = 9
Scale = 3

Ex:

Price number (7,2)
===============

0.0 --------> 0.0
99.99 -----> 99.99
100 --------> 100.00
999.99------> 999.99
1000 --------> 1000.00
9999.99 -----> 9999.99
99999.99-----> 99999.99

STRING / CHARACTER DATATYPES:

- storing string format of data only.
- In db string can be represented with '<string>'.

Ex: EmpName char (10)
================

smith-------> error
'smith' ----> allowed
1021 -------> error
'1021'------> allowed

```
                        String format of data
                                   |
            Character string data                  Alphanumeric string data
                        |                                       |
             (A-Z (or) a - z)                      ( A-Z (or) a-z , 0 - 9 , @,#,$,%,&,_,.....etc.)
                        |                                                   |
            Ex: 'SMITH','smith',....etc.                Ex: userid, password, pancard number, etc.
                                                              'smith123@gmail.com','ADF45NT6', etc.
```

Types of character datatypes:

- These are again two types:
  1. Non-Unicode datatypes
     > char (size)
     > varchar2 (size)
  2. Unicode datatypes
     > Nchar (size)
     > Nvarchar2 (size)

1. Non-Unicode datatypes:

- are storing "localized data".(i.e. English language only)
  > char (size)
  > varchar2 (size)

<u>Char (size):</u>

- It is fixed length datatype. (Static)
- It will store non-Unicode char's in the form 1 char = 1 byte.
- Maximum size is 2000 bytes.

Ex:

```
            Ename char (10)
'smith'-----> allowed ----------> 5 bytes/ 10 bytes ----> 5 bytes wasted.
'miller'-----> allowed ----------> 6 bytes / 10 bytes ----> 4 bytes “""
'ward' -----> allowed -----------> 4 bytes/ 10 bytes -----> 6 bytes  ''''
```

Disadvantage:

- wasted memory

<u>varchar2 (size):</u>

- It is dynamic datatype.
- It allowed non-Unicode char's in the form 1 char - 1 byte.
- Maximum size is 4000 bytes.

Ex:

```
            Ename varchar2(10)
'smith'-----> allowed ----------> 5 bytes/ 5 bytes ----> 0 bytes wasted.
'miller'-----> allowed ----------> 6 bytes / 6 bytes ----> 0 bytes  """
'ward' -----> allowed -----------> 4 bytes/ 4 bytes -----> 0 bytes  ''''
```

Advantage:

- Memory saved

2. Unicode datatypes:

- storing "globalized data"(i.e. all National languages only)
  - > Nchar (size)
  - > Nvarchar2 (size)
- "N" stands for National language.

Nchar (size):

- It is fixed length datatype.(static)
- It will store Unicode char's in the form 1 char = 1 byte.
- Maximum size is 2000 bytes.

Disadvantage:

- Memory wasted.

Nvarchar2 (size):

- It is dynamic datatype.
- It allowed Unicode char's in the form 1 char - 1 byte.
- Maximum size is 4000 bytes.

Advantage:

- Memory saved.

LONG DATATYPE:

- It is a dynamic datatype.
- storing non-Unicode & Unicode char's in the form of 1 char = 1 byte.
- Maximum size is 2gb.

Ex:

```
Eaddress long
============
H.NO:12-04-234,
Madhapur,
Hyderabad.
```

DATE DATATYPES:

- Storing date and time information of a particular day.
- Range of date expression is from '01-jan-4712 bc' to '31-dec-9999 ad'.
  - i) date
  - ii) timestamp

i) date:

- storing date and time information but time is optional.

If user not enter time then oracle db server will take time '12:00:00 am ' (or) '00:00:00 am' by default.

- Oracle default date format is 'dd-mon-yy/yoyo hh:mi:ss am/pm '

Ex:

```
'01-feb-23/2023  17:16:SSpm '
 1  1    2    1  1  1 --------> 7 bytes(fixed memory)
```

ii) timestamp:

- storing date and time information along with milliseconds.
- Allocating 11 bytes of memory i.e. fixed memory.

Ex:

```
 'dd-mon-yy/yyyy  hh:mi:ss.ms am'
   1  1    2      1  1  1  4 ----------> 11 bytes
```

RAW & LONG RAW DATATYPES:

- storing image file / audio file / video file in the form of 01001010101010 binary format. These datatypes are also called as binary datatypes in oracle db.
- RAW maximum size is 2000 bytes.
- LONG RAW maximum size is 2 gb.

LOB datatypes:

- It stands for large objects.
  - clob
  - nclob
  - blob

clob:

- It stands for character large object.
- It is a dynamic datatype.
- It will store non-Unicode char's.
- Maximum size is 4 gb.

nclob:

- It stands for national character large object.
- It is a dynamic datatype.
- It will store Unicode char's.
- Maximum size is 4 gb.

blob:

- It stands for binary large object.
- It is a dynamic memory.
- Will store image file / audio file / video file in the form of 0101000101010 binary format.
- Maximum size is 4 gb.

Non-Unicode char's:

- char (size)	**- 2000 bytes**
- varchar2 (size)	**- 4000 bytes**
- long	**- 2 GB**
- clob	**- 4 GB**

Unicode char's:

- Nchar (size)	**- 2000 bytes**
- Nvarchar2 (size)	**- 4000 bytes**
- long	**- 2 GB**
- Nclob	**- 4 GB**

Binary data:

- Raw	**- 2000 bytes**
- Long raw	**- 2 GB**
- blob	**- 4 GB**

1) CREATE

CREATING A TABLE:

SYNTAX:

CREATE TABLE <TABLE NAME>(<COLUMN NAME1> <DATATYPE>[SIZE], <COLUMN NAME2> <DATATYPE>[SIZE],................................);

EX:

CREATE TABLE STUDENT(STID int,SNAME char(10),SFEE number(6,2));

HOW TO GRANTING CREATE TABLE PERMISSION TO USERS:

EX:

SQL> CONN SYSTEM/TIGER
SQL> GRANT CREATE TABLE TO MYDB2PM;

HOW TO VIEW THE STRUCTURE OF A TABLE IN ORACLE:

SYNTAX:

DESC <TABLE NAME> (DESCRIBE)

EX:

DESC STUDENT;

TO VIEW THE LIST OF TABLES IN ORACLE DB:

SYNTAX:

SELECT * FROM TAB;

EX:

SELECT * FROM TAB (TAB is predefined table name)

2) ALTER

- This command is used to modify the structure of a table.
- It is having four more commands those are,
    - i) ALTER - MODIFY
    - ii) ALTER - ADD
    - iii) ALTER - RENAME
    - iv) ALTER -DROP

ALTER - MODIFY:

- To change / modify column datatype and also the size of the datatype.

SYNTAX:

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

EX:

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

ALTER - ADD:

- To add a new column to an existing table.

SYNTAX:

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

EX:
SQL> ALTER TABLE STUDENT ADD SADDRESS VARCHAR2(50);

ALTER - RENAME:

- To change a column name in a table.

SYNTAX:
ALTER TABLE <TN> RENAME <COLUMN> <OLD COLUMN NAME> TO <NEW COLUMN NAME>;

EX:
SQL> ALTER TABLE STUDENT RENAME COLUMN SNAME TO STUDENT_NAMES;

ALTER - DROP:

- To delete / drop a column from a table.

SYNTAX:
ALTER TABLE <TN> DROP <COLUMN> <COLUMN NAME>;

EX:
SQL> ALTER TABLE STUDENT DROP COLUMN STID;

3) RENAME

- To change a table name.

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

EX:
SQL> RENAME STUDENT TO SDETAILS;
SQL> RENAME SDETAILS TO STUDENT;

4) TRUNCATE

- To delete all rows but not columns from a table.
- It is permanent data deletion.(i.e we cannot restore)
- We cannot delete a specific row from a table because it does not supports “where" clause condition.

TRUNCATE
SYNTAX:
TRUNCATE TABLE <TABLE NAME>;

EX:
SQL> TRUNCATE TABLE STUDENT;

5) DROP

- To delete / drop a table (rows and columns) from a database memory.

DROP
SYNTAX:
DROP TABLE <TABLE NAME>;

EX:
SQL> DROP TABLE STUDENT;

NOTE:
- BEFORE ORACLE10g ENTERPRISE EDITION ONCE WE DROP A TABLE FROM A DATABASE THAT IS PERMANENT WHEREAS FROM ORACLE10g ENTERPRISE EDITION ONCE WE DROP A TABLE FROM A DATABASE THAT IS TEMPORARY.

NEW FEATURES OF ORACLE10g ENTERPRISE EDITION:

- RECYCLEBIN
- FLASHBACK
- PURGE

6) RECYCLEBIN

- IT IS A PRE-DEFINED TABLE.
- IT WILL STORE THE INFORMATION ABOUT DELETED TABLES FROM DB.
- IT WILL WORK JUST LIKE WINDOWS RECYCLEBIN IN COMPUTER.

TO VIEW THE STRUCTURE OF RECYCLEBIN:
SYNTAX:
DESC RECYCLEBIN;

EX:
DESC RECYCLEBIN;

TO VIEW THE INFORMATION ABOUT DELETED TABLES IN RECYCLEBIN:
SYNTAX:
SQL> SELECT OBJECT_NAME,ORIGINAL_NAME FROM RECYCLEBIN;

```
OBJECT_NAME                                        ORIGINAL_NAME
------------------------------------------------------------          ---------------------
BIN$t7wU7ikOTg23619JsfBJmg==$0                        STUDENT
```

7) FLASHBACK

- It is a command which is used to restore a deleted table from recyclebin to database memory.

FLASHBACK
SYNTAX:
FLASHBACK TABLE <TN> TO BEFORE DROP;

EX:
SQL> FLASHBACK TABLE STUDENT TO BEFORE DROP;

8) PURGE

- This command is used to delete a table permanently.

CASE-1: TO DELETE A SPECIFIC TABLE FROM RECYCLEBIN PERMANENTLY:
SYNTAX:
PURGE TABLE <TABLE NAME>;

EX:
CREATE TABLE T1(SNO INT);
CREATE TABLE T2(NAME VARCHAR2(10));

DROP TABLE T1;
DROP TABLE T2;

PURGE TABLE T2;

CASE-2: TO DELETE ALL TABLES FROM RECYCLEBIN PERMANENTLY:
SYNTAX:
PURGE RECYCLEBIN;

EX:
PURGE RECYCLEBIN;

CASE-3: TO DELETE A TABLE FROM DATABASE PERMANENTLY:
SYNTAX:
DROP TABLE <TABLE NAME> PURGE;

EX:
CREATE TABLE TEST1(SNO INT);

DROP TABLE TEST1 PURGE;

==
03-02-2023:
==

DML COMMANDS

INSERT:

- TO INSERT A NEW ROW DATA INTO A TABLE.

SYNTAX1:
INSERT INTO <TN> VALUES(VALUE1,VALUE2,.........);

EX:
SQL> INSERT INTO STUDENT VALUES(1021,'SMITH',3500);

- IN THIS METHOD NO.OF VALUES MUST BE EQUALS TO NO.OF COLUMNS IN A TABLE.

SYNTAX2:
INSERT INTO <TN>(<LIST OF COLUMN NAMES>) VALUES(VALUE1,VALUE2,........);

EX:
SQL> INSERT INTO STUDENT(SNAME)VALUES('JONES');
SQL> INSERT INTO STUDENT(STID,SNAME)VALUES(1023,'ALLEN');
SQL> INSERT INTO STUDENT(STID,SNAME,SFEE)VALUES(1024,'ADAMS',2500);
SQL> INSERT INTO STUDENT(SFEE,STID,SNAME)VALUES(5000,1025,'WARD');

- IN THIS METHOD USER CAN INSERT VALUES FOR REQUIRED COLUMNS ONLY AND REMAINING COLUMNS WILL TAKE "NULLS" BY DEFAULT.

HOW TO INSERT MULTIPLE ROWS INTO A TABLE BY DYNAMICALLY:

&: THIS OPERATOR IS USED TO INSERT VALUES INTO A TABLE DYNAMICALLY.

SYNTAX1:

INSERT INTO <TN> VALUES(<&COLUMN NAME1>,<&COLUMN NAME2>,............);

EX:

```
SQL> INSERT INTO STUDENT VALUES(&STID,'&SNAME',&SFEE);
Enter value for stid: 1026
Enter value for sname: SCOTT
Enter value for sfee: 6500
```

SQL> / (TO RE-EXECUTE THE LAST EXECUTED SQL QUERY IN SQLPLUS ENVIRONMENT)

SYNTAX2:

INSERT INTO <TN>(<LIST COLUMN NAMES>)VALUES(<&COLUMN NAME1>,<&COLUMN NAME2>,.......);

EX:

```
SQL> INSERT INTO STUDENT(SFEE)VALUES(&SFEE);
Enter value for sfee: 2000
SQL> /
.....................................
```

UPDATE

- TO UPDATE ALL ROWS DATA IN A TABLE (OR) A SPECIFIC ROW DATA IN A TABLE BY USING "WHERE" CONDITION.

SYNTAX:

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

EX:

```
SQL> UPDATE STUDENT SET STID=1022,SFEE=7000 WHERE SNAME='JONES';
SQL> UPDATE STUDENT SET SFEE=NULL WHERE SFEE=6000;
SQL> UPDATE STUDENT SET STID=1122,SNAME='WARNER',SFEE=500 WHERE STID=1025;
SQL> UPDATE STUDENT SET STID=NULL,SNAME=NULL,SFEE=NULL WHERE STID=1122;
SQL> UPDATE STUDENT SET SFEE=5000;
SQL> UPDATE STUDENT SET SFEE=NULL;
```

DELETE:

- TO DELETE TO ALL ROWS FROM A TABLE (OR) A SPECIFIC A ROW FROM A TABLE BY USING "WHERE" CONDITION.

SYNTAX:

DELETE FROM <TN> [WHERE <CONDITION>];

EX:

```
SQL> DELETE FROM STUDENT WHERE STID=1027;
SQL> DELETE FROM STUDENT WHERE STID IS NULL;
SQL> DELETE FROM STUDENT;
```

DELETE vs TRUNCATE:

DELETE	TRUNCATE
1. TO DELETE A SPECIFIC ROW.	1. NO
2. IT SUPPORTS "WHERE" CLAUSE CONDITION.	2. NOT ALLOWED
3. TEMPORARY DATA DELETION.	3. PERMANENT DATA DELETION.
4. WE CAN RESTORE DATA BY USING "ROLLBACK" COMMAND.	4. NOT ALLOWED
5. EXECUTION SPEED IS SLOW. (deleting rows in one by one)	5. EXECUTION SPEED IS FAST. (deleting all rows as a page)

===
04-02-2023:
===

DQL / DRL

SELECT

- TO RETRIEVE ALL ROWS FROM A TABLE (OR) A SPECIFIC ROW FROM A TABLE BY USING "WHERE" CONDITION.

SYNTAX:
SELECT * / <LIST OF COLUMN NAMES> FROM <TABLE NAME> [WHERE <CONDITION>];

Here, " * " -----------------> ALL COLUMNS IN A TABLE.

EX:
SQL> SELECT * FROM DEPT;
(OR)
SQL> SELECT DEPTNO,DNAME,LOC FROM DEPT;

SQL> SELECT * FROM DEPT WHERE DEPTNO=30;
SQL> SELECT * FROM EMP WHERE JOB='MANAGER';

ALIAS NAMES

- IT IS A TEMPORARY NAME FOR COLUMNS AND TABLE NAME.

i) COLUMN LEVEL ALIAS NAME:

- IN THIS LEVEL WE ARE CREATED ALIAS NAMES FOR COLUMNS.

ii) TABLE LEVEL ALIAS NAME:

- IN THIS LEVEL WE ARE CREATED ALIAS NAMES FOR TABLE NAME.

SYNTAX:
SELECT <COLUMN NAME1> <COLUMN ALIAS NAME1>,<COLUMN NAME2> <COLUMN ALIAS NAME2>,.........................FROM <TN> <TABLE ALIAS NAME>;

EX:
SQL> SELECT DEPTNO X,DNAME Y,LOC Z FROM DEPT D;

CONCATENATION OPERATOR (||)

- TO ADD TWO OR MORE THAN TWO STRING EXPRESSIONS.

SYNTAX:

<STRING1> || <STRING2> || <STRING3> || ....;

EX:

SQL> SELECT 'Mr.'||ENAME||' '||'IS WORKING AS A'||' '||JOB FROM EMP;

OUTPUT:

Mr.SMITH IS WORKING AS A CLERK

DISTINCT KEYWORD

- TO ELIMINATE DULPLICATE VALUES FROM A COLUMN TEMPORARLY.

SYNTAX:

DISTINCT <COLUMN NAME>

EX:

SQL> SELECT DISTINCT JOB FROM EMP;
SQL> SELECT DISTINCT DEPTNO FROM EMP;

NOTE:

- TO DISPLAY A BIG DATA TABLES IN PROPER SYSTEMATICALLY THEN WE SHOULD SET SOME PROPERTIES THOSE ARE,

1) PAGESIZE 100:

- IT USED TO DISPLAY NO.OF ROWS IN A PAGE.
- BY DEFAULT A PAGE CONTAINS 14 ROWS.
- THE MAXIMUM NO.OF ROWS PER A PAGE IS 50000 ROWS.

SYNTAX:

SET PAGESIZE n;

EX:

SET PAGESIZE 100;

2. LINES n:

- IT IS USED TO DISPLAY NO.OF BYTES PER A LINE.
- BY DEFAULT EACH LINE IS HAVING 80 BYTES (80 CHAR's).
- THE MAXIMUM SIZE IS 32767 BYTES.

SYNTAX:

SET LINES n;

EX:

SET LINES 150;

HOW TO CREATE A NEW TABLE FROM THE OLD TABLE:

CASE-1: WITHOUT ROWS:

SYNTAX:

CREATE TABLE <NEW TABLE NAME> AS SELECT * FROM <OLD TN> WHERE <FALSE CONDITION>;

EX:
SQL> CREATE TABLE NEWDEPT AS SELECT * FROM DEPT WHERE 1=2;

CASE-2: WITH ROWS:
SYNTAX:
CREATE TABLE <NEW TABLE NAME> AS SELECT * FROM <OLD TN> WHERE <TRUE CONDITION>;

EX:
SQL> CREATE TABLE NEWDEPT1 AS SELECT * FROM DEPT WHERE 1=1;

CASE-3: WITH COPY OF SPECIFIC COLUMNS:
SYNTAX:
CREATE TABLE <NEW TN> AS SELECT <SPECIFIC COLUMN NAMES> FROM <OLD TN> WHERE <TRUE CONDITION / FALSE CONDITION>;

EX:
SQL> CREATE TABLE NEWEMP AS SELECT EMPNO,ENAME,SAL FROM EMP WHERE 2=2;

CASE-4: WITH COPY OF SPECIFIC ROWS:
SYNTAX:
CREATE TABLE <NEW TN> AS SELECT * FROM <OLD TN> WHERE <FILTERING CONDITION>;

EX:
SQL> CREATE TABLE NEWEMP1 AS SELECT * FROM EMP WHERE JOB='MANAGER';

HOW TO COPY DATA FROM ONE TABLE TO ANOTHER TABLE:
SYNTAX:
INSERT INTO <DESTINATION TABLE NAME> SELECT * FROM <SOURCE TABLE NAME>;

EX:
SQL> INSERT INTO NEWDEPT SELECT * FROM DEPT;

INSERT ALL

- IT IS DML COMMAND WHICH IS USED TO INSERT ROWS(DATA) INTO MULTIPLE TABLES BUT THE DATA SHOULD BE AN EXISTING TABLE.

SYNTAX:
```
INSERT ALL INTO <TN1> VALUES(<COLUMN NAME1>,<COLUMN NAME2>,.........)
INTO <TN2> VALUES(<COLUMN NAME1>,<COLUMN NAME2>,...........................)
INTO <TN3> VALUES(<COLUMN NAME1>,<COLUMN NAME2>,...........................)
......................................................................................................)
......................................................................................................)
INTO <TN n> VALUES(<COLUMN NAME1>,<COLUMN NAME2>,..........................)
SELECT * FROM <OLD TABLE NAME>;
```

EX:
STEP1: CREATE EMPTY TABLES:
SQL> CREATE TABLE TEST1 AS SELECT * FROM DEPT WHERE 1=0;
SQL> CREATE TABLE TEST2 AS SELECT * FROM DEPT WHERE 1=0;
SQL> CREATE TABLE TEST3 AS SELECT * FROM DEPT WHERE 1=0;

STEP2: INSERT DATA INTO EMPTY TABLES:

```
SQL> INSERT ALL INTO TEST1 VALUES(DEPTNO,DNAME,LOC)
  2  INTO TEST2 VALUES(DEPTNO,DNAME,LOC)
  3  INTO TEST3 VALUES(DEPTNO,DNAME,LOC)
  4  SELECT * FROM DEPT;
```

MERGE: (AFTER JOINS)

==
(06-02-2023)
==

OPERATORS

- TO PERFORM SOME OPERATION ON THE GIVEN OPERAND VALUES.
- ORACLE SUPPORTS THE FOLLOWING OPERATORS ARE,

1. ASSIGNMENT OPERATOR	=>	=
2. ARITHMETICA OPERATORS	=>	+ , - , * , /
3. RELATIONAL OPERATORS	=>	< , > , <= , >= ,!=(OR) < >
4. LOGICAL OPERATORS	=>	AND,OR,NOT
5. SET OPERATORS	=>	UNION,UNION ALL,INTERSECT,MINUS
6. SPECIAL OPERATORS	=>(+ve)	(-ve)

(+ve)	(-ve)
=====	=====
IN	NOT IN
BETWEEN	NOT BETWEEN
IS NULL	IS NOT NULL
LIKE	NOT LIKE

ASSIGNMENT OPERATOR (=):

- TO ASSIGN A VALUE TO A VARIABLE (OR) ATTRIBUTE.

SYNTAX:

<COLUMN NAME> <ASSIGNMENT OPERATOR> <VALUE>

EX:

```
SELECT * FROM EMP WHERE ENAME='SMITH';
UPDATE EMP SET SAL=24000 WHERE DEPTNO=10;
DELETE FROM EMP WHERE SAL=1250;
```

ARITHMETIC OPERATORS :

- TO PERFORM ADDITION,SUBTRACTION,MULTIPLE AND DIVISION.

SYNTAX:

<COLUMN NAME> <ARITHMETIC OPERATOR> <VALUE>

EX:

WAQ TO DISPLAY ALL EMPLOYEES SALARIES AFTER ADDING 1000 /-?

```
SQL> SELECT SAL BASIC_SALARY,SAL+1000 NEW_SALARY FROM EMP;
```

EX:
WAQ TO DISPLAY EMPNO,ENAME,SALARY AND ANNUAL SALARY OF EMPLOYEES WHO ARE WORKING UNDER DEPTNO=20?

```
SQL> SELECT EMPNO,ENAME,SAL BASIC_SALARY,SAL*12 ANNUAL_SALARY
  2  FROM EMP WHERE DEPTNO=20;
```

EX:
WAQ TO DISPLAY EMPLOYEES AFTER INCREMENT OF 10% ON THEIR BASIC SALARY?

```
SQL> SELECT ENAME,SAL BASIC_SALARY,SAL*0.1 INCREMENT_SALARY,
  2  SAL+SAL*0.1 TOTAL_SALARY FROM EMP;
```

EX:
WAQ TO UPDATE EMPLOYEES SALARIES WITH INCREMENT OF 5% WHO ARE WORKING AS A MANAGER?

```
SQL> UPDATE EMP SET SAL=SAL+SAL*0.05 WHERE JOB='MANAGER';
```

RELATIONAL OPERATORS:

- COMPARING A SPECIFIC COLUMN VALUES WITH USER DEFINED CONDITION IN THE QUERY.

SYNTAX:
WHERE <COLUMN NAME> <RELATIONAL OPERATOR> <VALUE>

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE JOINED AFTER 1981?

```
SQL> SELECT * FROM EMP WHERE HIREDATE > '31-DEC-81';
```

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE JOINED BEFORE 1981?

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

LOGICAL OPERATOR:

- TO CHECK MORE THAN ONE CONDITION.
- AND, OR, NOT

AND

- IT RETURN A VALUE IF BOTH CONDITIONS ARE TRUE.

SYNTAX:
WHERE <CONDITION1> AND <CONDITION2> AND ....;

```
COND1  COND2
====== ======
T      T      ---> T
T      F      ---> F
F      T      ---> F
F      F      ---> F
```

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE WORKING AS A "CLERK" AND WHOSE NAME IS "SMITH"?
SQL> SELECT * FROM EMP WHERE JOB='CLERK' AND ENAME='SMITH';

OR

- TO RETURN A VALUE IF ANY ONE CONDITION TRUE.

SYNTAX:

WHERE <CONDITION1> OR <CONDITION2> OR ....;

COND1	COND2	
=====	======	
T	T	---> T
T	F	---> T
F	T	---> T
F	F	---> F

EX:
WAQ TO DISPLAY EMPLOYEES WHOSE EMPNO IS 7788 OR WHOSE NAME IS "MILLER"?
SQL> SELECT * FROM EMP WHERE EMPNO=7788 OR ENAME='MILLER';

NOTE:
"AND" OPERATOR WORKING ON SAME ROW IN A TABLE.
"OR" OPERATOR WORKING ON DIFFERENT ROWS IN A TABLE.

NOT

- IT RETURNS ALL VALUES EXCEPT THE GIVEN CONDITIONAL VALUES IN THE QUERY.

SYNTAX:

WHERE NOT <CONDITION1> AND NOT <CONDITION2> AND NOT ....;

EX:
WAQ TO DISPLAY LIST OF EMPLOYEES WHO ARE NOT WORKING AS A "CLERK" AND "SALESMAN" ?
SQL> SELECT * FROM EMP WHERE NOT JOB='CLERK' AND NOT JOB='SALESMAN';

SET OPERATORS:

- ARE USED TO COMBINE THE RESULTS OF TWO SELECT STATEMENTS.

SYNTAX:

<SELECT QUERY1> <SET OPERATOR> <SELECT QUERY2>;

UNION:
- TO COMBINED THE RESULTS OF TWO SELECT STATEMENTS WITHOUT DUPLICATES.

UNION ALL:
- TO COMBINED THE RESULTS OF TWO SELECT STATEMENTS WITH DUPLICATES.

INTERSECT:

- IT RETURNS THE COMMON VALUES FROM BOTH SELECT STATEMENTS.

MINUS:

- IT RETURNS UNCOMMON VALUES FROM THE LEFT SIDE BUT NOT RIGHT SIDE OF SELECT QUERY.

==
DEMO_TABLES: (07-02-2023)
==

SQL> SELECT * FROM EMP_HYD;

```
       EID ENAME             SAL
---------- ---------- ----------
      1021 SMITH           85000
      1022 JONES           67000
      1023 WARD            43000
```

SQL> SELECT * FROM EMP_CHENNAI;

```
       EID ENAME             SAL
---------- ---------- ----------
      1021 SMITH           85000
      1024 MILLER          60000
```

EX:
WAQ TO DISPLAY EMPLOYEES DETAILS WHO ARE WORKING IN BOTH BRANCHES?
SQL> SELECT * FROM EMP_HYD INTERSECT SELECT * FROM EMP_CHENNAI;

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE WORKING IN CHENNAI BUT NOT IN HYD BRANCH?
SQL> SELECT * FROM EMP_CHENNAI MINUS SELECT * FROM EMP_HYD;

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE WORKING IN THE ORGANIZATION?
SQL> SELECT * FROM EMP_HYD UNION ALL SELECT * FROM EMP_CHENNAI;---WITH DUPLICATES
SQL> SELECT * FROM EMP_HYD UNION SELECT * FROM EMP_CHENNAI;---WITHOUT DUPLICATES

SPECIAL OPERATORS:

IN

- COMPARING THE LIST OF VALUES WITH A SINGLE CONDITION.

SYNTAX:

WHERE <COLUMN NAME> IN(V1,V2,V3,..........);

EX:
WAQ TO LIST OUT EMPLOYEES DETAILS WHOSE EMPNO IS 7369,7566,7788,7900?
SQL> SELECT * FROM EMP WHERE EMPNO IN(7369,7566,7788,7900);

BETWEEN

- WORKING ON A PARTICULAR RANGE VALUE.

BASIC RULES:
1. IT RETURNS ALL VALUES INCLUDING SOURCE AND DESTINATION VALUES FROM THE GIVEN RANGE.

2. IT IS ALWAYS WORK ON LOW VALUE TO HIGH VALUE.

SYNTAX:

WHERE <COLUMN NAME> BETWEEN <LOW VALUE> AND <HIGH VALUE>;

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE JOINED IN 1981?
SQL> SELECT * FROM EMP WHERE HIREDATE BETWEEN '01-JAN-81' AND '31-DEC-81';

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE NOT JOINED IN 1981?
SQL> SELECT * FROM EMP WHERE HIREDATE NOT BETWEEN '01-JAN-81' AND '31-DEC-81';

IS NULL

- COMPARING NULLS IN A TABLE.

SYNTAX:

WHERE <COLUMN NAME> IS NULL;

EX:
WAQ TO DISPLAY EMPLOYEES WHOSE COMMISSION IS NULL / EMPTY ?
SQL> SELECT * FROM EMP WHERE COMM IS NULL;

EX:
WAQ TO DISPLAY EMPLOYEES WHOSE COMMISSION IS NOT NULL / EMPTY ?
SQL> SELECT * FROM EMP WHERE COMM IS NOT NULL;

WORKING NULL:

- IT IS UNKNOW VALUE/ UNDEFINED VALUE/ EMPTY.
- NULL != 0 & NULL != SPACE

EX:
WAQ TO DISPLAY EMPNO,ENAME,SALARY,COMMISSION AND SALARY+COMMISSION OF THE EMPLOYEE IS "SMITH"?

```
SQL> SELECT EMPNO,ENAME,SAL,COMM,SAL+COMM TOTAL_SALARY
     FROM EMP WHERE ENAME='SMITH';
```

```
EMPNO              ENAME       SAL        COMM      TOTAL_SALARY
----------------     ----------          ----------   ----------   ------------
7369               SMITH       800
```

NOTE:

IF ANY ARITHMETIC OPERATOR WILL PERFORM SOME OPERATION WITH "null" THEN IT AGAIN RETURNS "null" ONLY.

Ex: IF X = 1000;

i) X+NULL ===> 1000+NULL ====> NULL
ii) X-NULL ===> 1000-NULL ====> NULL
iii) X*NULL ===> 1000*NULL ===> NULL
iv) X/NULL ===> 1000/NULL ====> NULL

- TO OVERCOME THE ABOVE PROBLEM WE USE A PRE-DEFINED FUNCTION IN ORACLE DB i.e "NVL()".

WHAT IS NVL()

- IT IS PRE-DEFINED FUNCTION.
- IT STANDS FOR NULL VALUE.
- IT IS USED TO REPLACE A USER DEFINED VALUE INPLACE OF NULL.

SYNTAX:

NVL(EXPRESSION1,EXPRESSION2)

- IF EXP1 IS NULL THEN IT RETURNS EXP2 VALUE(USER DEFINED VALUE)
- IF EXP1 IS NOT NULL IT RETURNS EXP1 VALUE ONLY.

EX:

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

SQL> SELECT NVL(NULL,100) FROM DUAL;
NVL(NULL,100)
-------------
         100

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

SOLUTION:

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



EMPNO                ENAME      SAL       COMM      TOTAL_SALARY
----------------     ----------         ----------   ----------   ------------
7369                 SMITH      800                  800
```

NVL2(EXP1,EXP2,EXP3):

- IT IS AN EXTENSION OF NVL().
- IS HAVING 3 ARGUMENTS ARE "EXP1,EXP2,EXP3".
  > IF EXP1 IS NULL -----------> RETURN EXP3 VALUE(UD VALUE)
  > IF EXP1 IS NOT NULL ----> RETURN EXP2 VALUE(UD VALUE)

SYNTAX:

NVL2(EXP1,EXP2,EXP3)

EX:

```
SQL> SELECT NVL2(NULL,100,200) FROM DUAL;
NVL2(NULL,100,200)
------------------
200

SQL> SELECT NVL2(0,100,200) FROM DUAL;
NVL2(0,100,200)
---------------
100
```

EX:

WAQ TO UPDATE ALL EMPLOYEES COMMISSION IN A TABLE AS PER THE FOLLOWING CONDITIONS ARE,

i) IF THE EMPLOYEE COMM IS NULL THEN UPDATE THOSE EMPLOYEES COMM AS 700.

ii) IF THE EMPLOYEE COMM IS NOT NULL THEN UPDATE THOSE EMPLOYEES COMM AS COMM+700.

SQL> UPDATE EMP SET COMM=NVL2(COMM,COMM+700,700);

LIKE

- COMPARING A SPECIFIC STRING CHARACTER PATTERN WISE.
- WHEN WE LIKE OPERATOR WE SHOULD USE THE FOLLOWING WILDCARD OPERATORS ARE " % , _ ".

WILDCARD OPERATORS:

i) %	- IT REPRESENT THE REMAINING GROUP OF CHAR's AFTER SELECTED CHARACTER.
ii) _	- COUNTING A SINGLE CHARACTER IN THE EXPRESSION.

SYNTAX:

WHERE <COLUMN NAME> LIKE '<SPECIFIC STRING CHARACTER PATTERN>';

EX:

WAQ TO DISPLAY EMPLOYEES WHOSE NAME STARTS WITH "S" ?

SQL> SELECT * FROM EMP WHERE ENAME LIKE 'S%';

S	%
S	AI
S	UMAN
S	COTT
S	MITH
S	URESH

EX:
WAQ TO DISPLAY EMPLOYEES WHOSE NAME IS HAVING 3 CHARACTERS?
SQL> SELECT * FROM EMP WHERE ENAME LIKE '___';

SAI
SUMAN
SCOTT
SMITH
SURESH
ANU

EX:
WAQ TO FETCH EMPLOYEES WHOSE NAME ENDS WITH 'N' ?
SQL> SELECT * FROM EMP WHERE ENAME LIKE '%N';

EX:
WAQ TO FETCH EMPLOYEES WHOSE NAME IS HAVING " I " ?
SQL> SELECT * FROM EMP WHERE ENAME LIKE '%I%';

EX:
WAQ TO FETCH EMPLOYEES WHOSE NAME STARTS WITH "M" AND ENDS WITH "N" ?
SQL> SELECT * FROM EMP WHERE ENAME LIKE 'M%N';

EX:
WAQ TO FETCH EMPLOYEES WHOSE NAME IS HAVING 4 CHAR's?
SQL> SELECT * FROM EMP WHERE ENAME LIKE '____';

EX:
WAQ TO FETCH EMPLOYEES WHOSE NAME IS HAVING 2ND CHARACTER IS "O" ?
SQL> SELECT * FROM EMP WHERE ENAME LIKE '_O%';

EX:
WAQ TO FETCH EMPLOYEES WHO ARE JOINED IN 1981?
SQL> SELECT * FROM EMP WHERE HIREDATE LIKE'%81';

EX:
WAQ TO FETCH EMPLOYEES WHO ARE JOINED IN THE MONTH OF "DECEMBER"?
SQL> SELECT * FROM EMP WHERE HIREDATE LIKE'%DEC%';

EX:
WAQ TO FETCH EMPLOYEES WHO ARE JOINED IN THE MONTH OF "JUNE" OR "DECEMBER"?
SQL> SELECT * FROM EMP WHERE HIREDATE LIKE'%JUN%' OR HIREDATE LIKE '%DEC%';

EX:
WAQ TO FETCH EMPLOYEES WHOSE EMPNO STARTS WITH 7 AND ENDS WITH 8?
SQL> SELECT * FROM EMP WHERE EMPNO LIKE '7%8';

LIKE OPERATOR ON SPECIAL CHAR's:

DEMO TABLE:
SQL> SELECT * FROM CUSTOMER;

```
      CID CNAME
----------             ----------
     1021 _ALLEN
     1022 WAR@NER
     1023 MILL#ER
     1024 KAMAL_ROY
     1025 MAR%TIN
```

EX:
WAQ TO FETCH CUSTOMER WHOSE NAME IS HAVING '@' SYMBOL?
SQL> SELECT * FROM CUSTOMER WHERE CNAME LIKE '%@%';

EX:
WAQ TO FETCH CUSTOMER WHOSE NAME IS HAVING '#' SYMBOL?
SQL> SELECT * FROM CUSTOMER WHERE CNAME LIKE '%#%';

EX:
WAQ TO FETCH CUSTOMER WHOSE NAME IS HAVING '_' SYMBOL?
SQL> SELECT * FROM CUSTOMER WHERE CNAME LIKE '%_%';

- WHEN WE EXECUTED THE ABOVE QUERY IT RETURN WRONG RESULT BECAUSE BY DEFAULT ORACLE DB SERVER WILL TREAT "_ ,%" AS WILDCARD OPERATORS BUT NOT SPECIAL CHAR's.SO TO OVERCOME THIS PROBLEM WE USE A PRE-DEFINED KEYWORD IS CALLED AS " ESCAPE ' \ ' ".

SOLUTION:
SQL> SELECT * FROM CUSTOMER WHERE CNAME LIKE %_%' ESCAPE'\';

EX:
WAQ TO FETCH CUSTOMER WHOSE NAME IS HAVING '%' SYMBOL?
SQL> SELECT * FROM CUSTOMER WHERE CNAME LIKE '%\%%'ESCAPE'\';

EX:
WAQ TO FETCH EMPLOYEES DETAILS WHOSE NAME IS NOT STARTS WITH "S"?
SQL> SELECT * FROM EMP WHERE ENAME NOT LIKE'S%';

==
(09-02-2023) **FUNCTIONS**
==

- TO PERFORM SOME TASK AND IT MUST BE RETURN A VALUE.
- ORACLE DB SUPPORTS THE FOLLOWING 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 ARE BUILT IN / IN BUILT FUNCTIONS IN ORACLE.
  i) SINGLE ROW FUNCTIONS(SCALAR FUNCTIONS)
  ii) MULTIPLE ROW FUNCTIONS(GROUPING FUNCTIONS)

SINGLE ROW FUNCTIONS:

- THESE FUNCTIONS ARE ALWAYS RETURNS A SINGLE VALUE ONLY.
  > NUMERIC FUNCTIONS
  > STRING FUNCTIONS
  > DATE FUNCTIONS
  > CONVERSION FUNCTIONS
  > ANALYTICAL FUNCTIONS (USED IN SUBQUERY)

HOW TO CALL A PRE-DEFINED FUNCTION:

SYNTAX:

SELECT <FNAME>(VALUES) FROM DUAL;

WHAT IS DUAL?

- IT IS A PRE-DEFINED TABLE.
- IT IS HAVING A SINGLE COLUMN & A SINGLE ROW.
- IS USED TO TEST FUNCTION FUNCTIONALITIES.
- IS ALSO CALLED AS "DUMMY" TABLE IN ORACLE.

TO VIEW THE STRUCTURE OF DUAL TABLE:

SQL> DESC DUAL;

EX:

```
SQL> DESC DUAL;
 Name                                                    Null?    Type
 ------------------------------------------------------- -------- ----------------------------------------------------------
 DUMMY                                                            VARCHAR2(1)
```

TO VIEW THE INFORMATION ABOUT DUAL TABLE:

SYNTAX:

SELECT * FROM DUAL;

```
D
--
X
```

NUMERIC FUNCTIONS:

ABS():

- TO CONVERT (-ve) SIGN VALUES INTO (+ve) SIGN VALUES.

SYNTAX:

ABS(n)

EX:

```
SQL> SELECT ABS(-12) FROM DUAL;--------> 12
SQL> SELECT EMPNO,ENAME,SAL,COMM,ABS(COMM-SAL) RESULT FROM EMP;
```

CEIL():

- IT RETURNS THE UPPER BOUND VALUE OF GIVEN EXPRESSION.

SYNTAX:

CEIL(n)

EX:

```
SQL> SELECT CEIL(9.3) FROM DUAL;
 CEIL(9.3)
----------
        10
```

FLOOR():

- IT RETURNS THE LOWER BOUND VALUE OF GIVEN EXPRESSION.

SYNTAX:

FLOOR(n)

EX:

```
SQL> SELECT FLOOR(9.8) FROM DUAL;
FLOOR(9.8)
----------
         9
```

MOD():

- IT RETURNS THE REMAINDER VALUE OF GIVEN EXPRESSION.

SYNTAX:

MOD(m,n)

EX:

```
SQL> SELECT MOD(10,2) FROM DUAL;
MOD(10,2)
----------
         0

SQL> SELECT * FROM EMP WHERE MOD(EMPNO,2)=0;
```

POWER():

- IT RETURNS THE POWER OF GIVEN EXPRESSION.

SYNTAX:

POWER(m,n)

EX:

```
SQL> SELECT POWER(2,3) FROM DUAL;
POWER(2,3)
----------
       8
```

ROUND():

- IT RETURNS THE NEAREST VALUE OF GIVEN EXPRESSION.
- THIS FUNCTION IS CONSIDER 0.5 EXPRESSION.
    - IF AN EXPRESSION >= 0.5 -------> ADD "1" TO THE EXPRESSION.
    - IF AN EXPRESSION <0.5 ----------> ADD "0" TO THE EXPRESSION.

SYNTAX:

ROUND(EXPRESSION,[DECIMAL PLACES])

EX:

```
SQL> SELECT ROUND(56.50) FROM DUAL;
ROUND(56.50)
------------
        57

SQL> SELECT ROUND(56.45) FROM DUAL;
ROUND(56.45)
------------
        56

SQL> SELECT ROUND(45.684,2) FROM DUAL;
ROUND(45.684,2)
---------------
        45.68

SQL> SELECT ROUND(45.686,2) FROM DUAL;
ROUND(45.686,2)
---------------
        45.69
```

TRUNC():

- IT RETURN AN EXACT VALUES BASED ON THE SPECIFIED DECIMAL PLACES.
- IT IS NOT CONSIDER 0.5 EXPRESSION.

SYNTAX:

TRUNC(EXPRESSION,[DECIMAL PLACES])

EX:

SQL> SELECT TRUNC(56.45) FROM DUAL;
```
TRUNC(56.45)
------------
        56
```

SQL> SELECT TRUNC(56.457,2) FROM DUAL;
```
TRUNC(56.457,2)
---------------
        56.45
```

STRING FUNCTIONS:

LENGTH():

- IT RETURNS THE NO.OF CHARACTERS IN THE GIVEN STRING EXPRESSION.

SYNTAX:

LENGTH(n)

EX:

SQL> SELECT LENGTH('HELLO') FROM DUAL;
```
LENGTH('HELLO')
---------------
           5
```

SQL> SELECT LENGTH('WEL COME') FROM DUAL;
```
LENGTH('WELCOME')
-----------------
             8
```

SQL> SELECT ENAME,LENGTH(ENAME) FROM EMP;
SQL> SELECT * FROM EMP WHERE LENGTH(ENAME)=6;

INITCAP():

- TO MAKE THE FIRST CHARACTER AS A CAPITAL IN THE GIVEN STRING.

SYNTAX:

INITCAP(STRING)

EX:

SQL> SELECT ENAME,INITCAP(ENAME) FROM EMP;
SQL> UPDATE EMP SET ENAME=INITCAP(ENAME) WHERE DEPTNO=20;

LOWER():

- IT CONVERTS UPPER CASE CHARACTERS INTO LOWER CASE CHARACTERS.

SYNTAX:

LOWER(STRING)

EX:
```
SQL> SELECT LOWER('HELLO') FROM DUAL;
LOWER
-----------
hello

SQL> UPDATE EMP SET ENAME=LOWER(ENAME) WHERE JOB='SALESMAN';
SQL> UPDATE EMP SET ENAME=LOWER(ENAME);
```

UPPER():

- IT CONVERTS LOWER CASE CHARACTERS INTO UPPER CASE CHARACTERS.

SYNTAX:

UPPER(STRING)

EX:
```
SQL> UPDATE EMP SET ENAME=UPPER(ENAME);
```

LTRIM():

- TO REMOVE UNWANTED CHARACTERS FROM THE GIVEN STRING ON LEFT SIDE.

SYNTAX:

LTRIM(STRING1,STRING2)

EX:
```
SQL> SELECT LTRIM('XXXXSAI','X') FROM DUAL;
LTR
---
SAI

SQL> SELECT LTRIM('123SAI','123') FROM DUAL;
LTR
---
SAI
```

RTRIM():

- TO REMOVE UNWANTED CHARACTERS FROM THE GIVEN STRING ON RIGHT SIDE.

SYNTAX:

RTRIM(STRING)

EX:
```
SQL> SELECT RTRIM('SAIXXXXX','X') FROM DUAL;
RTR
---
SAI
```

TRIM():

- TO REMOVE UNWANTED CHARACTERS FROM BOTH SIDES OF GIVEN STRING.

SYNTAX:

TRIM('trimming character' FROM STRING)

EX:
```
SQL> SELECT TRIM('X'FROM 'XXXSAIXXXXX') FROM DUAL;
TRI
---
SAI
```

LPAD():

- PADDING THE SPECIFIC CHARACTER TO THE GIVEN STRING ON LEFT SIDE.

SYNTAX:

LPAD(STRING,<LENGTH>,<SPECIFIC CHARACTER>)

EX:
```
SQL> SELECT LPAD('SAI',10,'@') FROM DUAL;
LPAD('SAI'
----------
@@@@@@@SAI
```

RPAD():

- PADDING THE SPECIFIC CHARACTER TO THE GIVEN STRING ON RIGHT SIDE.

SYNTAX:

RPAD(STRING,<LENGTH>,<SPECIFIC CHARACTER>)

EX:
```
SQL> SELECT RPAD('SAI',10,'@') FROM DUAL;
RPAD('SAI'
----------
SAI@@@@@@@
```

CONCAT():

- ADDING TWO STRING EXPRESSION.

SYNTAX:

CONCAT(STRING1,STRING2)

EX:
```
SQL> SELECT CONCAT('Mr.',ENAME) FROM EMP;
CONCAT('MR.',
-------------
Mr.SMITH
```

REPLACE():

- TO REPLACE A STRING WITH ANOTHER STRING.

SYNTAX:

REPALCE(STRING,<OLD CHAR's>,<NEW CHAR's>)

EX:

```
SQL> SELECT REPLACE('HELLO','ELL','XYZ') FROM DUAL;
REPLA
-----
HXYZO
```

```
SQL> SELECT REPLACE('JACK AND JUE','J','BL') FROM DUAL;
REPLACE('JACKA
--------------
BLACK AND BLUE
```

TRANSLATE():

- TO TRANSLATE CHARACTER BY CHARACTER.

SYNTAX:

TRANSLATE(STRING1,STRING2,STRING3)

EX:

```
SQL> SELECT TRANSLATE('HELLO','ELO','XYZ') FROM DUAL;
TRANS
-----
HXYYZ
```

SUBSTR():

- IT RETURN THE REQUIRED SUB STRING FROM THE GIVEN STRING EXPRESSION.

SYNTAX:

SUBSTR(STRING,<STARTING POSITION OF CHARACTER>,<LENGTH OF CHAR's>)

EX:

```
SQL> SELECT SUBSTR('WELCOME',2,4) FROM DUAL;
SUBS
----
ELCO
```

```
SQL> SELECT SUBSTR('WELCOME',1,1) FROM DUAL;
S
-
W
```

```
SQL> SELECT SUBSTR('WELCOME',-4,2) FROM DUAL;
SU
--
CO
```

INSTR():

- IT RETURNS THE OCCURRENCE POSITION OF CHARACTER FROM THE GIVEN STRING EXPRESSION.

SYNTAX:
INSTR(STRING,<SPECIFIC CHARACTER>,<STARTING POSITION OF CHAR>,<OCCURRENCE POSITION OF CHAR>)

NOTE:
- WHEN WE COUNT CHARACTERS FROM LEFT TO RIGHT (OR) RIGHT TO LEFT THE POSITION OF CHARCTER IS FIXED POSITION.

```
'HELLO WELCOME'
 12345 6 7890123 ----------FIXED POSITIONS
```

EX:

SQL> SELECT INSTR('HELLO WELCOME','O',7,1) FROM DUAL;
```
INSTR('HELLOWELCOME','O',7,1)
-----------------------------
                   11
```

SQL> SELECT INSTR('HELLO WELCOME','O',7,2) FROM DUAL;
```
INSTR('HELLOWELCOME','O',7,2)
-----------------------------
                    0
```

SQL> SELECT INSTR('HELLO WELCOME','E',1,1) FROM DUAL;
```
INSTR('HELLOWELCOME','E',1,1)
-----------------------------
                    2
```

SQL> SELECT INSTR('HELLO WELCOME','E',1,2) FROM DUAL;
```
INSTR('HELLOWELCOME','E',1,2)
-----------------------------
                    8
```

SQL> SELECT INSTR('HELLO WELCOME','E',1,3) FROM DUAL;
```
INSTR('HELLOWELCOME','E',1,3)
-----------------------------
                   13
```

SQL> SELECT INSTR('HELLO WELCOME','E',1,4) FROM DUAL;
```
INSTR('HELLOWELCOME','E',1,4)
-----------------------------
                    0
```

SQL> SELECT INSTR('HELLO WELCOME','E',8,1) FROM DUAL;
```
INSTR('HELLOWELCOME','E',8,1)
-----------------------------
                    8
```

SQL> SELECT INSTR('HELLO WELCOME','E',-6,1) FROM DUAL;
```
INSTR('HELLOWELCOME','E',-6,1)
------------------------------
                    8
```

SQL> SELECT INSTR('HELLO WELCOME','E',-6,3) FROM DUAL;

```
INSTR('HELLOWELCOME','E',-6,3)
------------------------------
                     0
```

==

11-02-2023 **DATE FUNCTIONS**

==

SYSDATE:

- IT RETURNS THE CURRENT DATE INFORMATION OF THE SYSTEM.

SYNTAX:

SYSDATE

EX:

SQL> SELECT SYSDATE FROM DUAL;

```
SYSDATE
---------
11-FEB-23
```

SQL> SELECT SYSDATE+10 FROM DUAL;

```
SYSDATE+1
---------
21-FEB-23
```

SQL> SELECT SYSDATE-10 FROM DUAL;

```
SYSDATE-1
---------
01-FEB-23
```

LAST_DAY():

- IT RETURNS THE LAST DAY OF GIVE DATE EXPRESSION.

SYNTAX:

LAST_DAY(DATE)

EX:

SQL> SELECT LAST_DAY(SYSDATE) FROM DUAL;

```
LAST_DAY(
---------
28-FEB-23
```

ADD_MONTHS():

- TO ADD / SUBTRACT NO.OF MONTHS TO THE GIVEN DATE EXPRESSION.

SYNTAX:

ADD_MONTH(DATE,<NO.OF MONTHS>)

EX:

```
SQL> SELECT SYSDATE CURRENT_DATE,ADD_MONTHS(SYSDATE,5) NEW_DATE  FROM DUAL;
CURRENT_D     NEW_DATE
---------     ---------
11-FEB-23     11-JUL-23

SQL> SELECT SYSDATE CURRENT_DATE,ADD_MONTHS(SYSDATE,-5) OLD_DATE  FROM DUAL;
CURRENT_D     OLD_DATE
---------     ---------
11-FEB-23     11-SEP-22
```

EX:

```
SQL> CREATE TABLE PRODUCT(PCODE INT,PNAME VARCHAR2(10),MFG DATE,EXP DATE);
Table created.

SQL> INSERT INTO PRODUCT(PCODE,PNAME,MFG)VALUES(1021,'X','15-MAY-2022');
SQL> INSERT INTO PRODUCT(PCODE,PNAME,MFG)VALUES(1022,'Y','08-JAN-2023');

SQL> SELECT * FROM PRODUCT;

   PCODE       PNAME        MFG          EXP
   ---------   ----------   ---------    ---------
    1021       X            15-MAY-22
    1022       Y            08-JAN-23

SQL> UPDATE PRODUCT SET EXP=ADD_MONTHS(MFG,24);
2 rows updated.

SQL> SELECT * FROM PRODUCT;

   PCODE       PNAME        MFG          EXP
   ----------  ----------   ---------    ---------
    1021       X            15-MAY-22    15-MAY-24
    1022       Y            08-JAN-23    08-JAN-25
```

MONTHS_BETWEEN():

- IT RETURNS THE NO.OF MONTHS BETWEEN THE GIVEN TWO DATE EXPRESSIONS.

SYNTAX:

MONTHS_BETWEEN(DATE1,DATE2)

EX:

```
SQL> SELECT MONTHS_BETWEEN('03-JAN-23','03-JAN-24') FROM DUAL;
MONTHS_BETWEEN('03-JAN-23','03-JAN-24')
---------------------------------------
                              -12

SQL> SELECT MONTHS_BETWEEN('03-JAN-24','03-JAN-23') FROM DUAL;
MONTHS_BETWEEN('03-JAN-24','03-JAN-23')
---------------------------------------
                               12
```

NOTE:
    - DATE1 MUST BE GREATER THAN DATE2 OTHERWISE IT RETURNS NEGATIVE(-ve) VALUES.

CONVERSION FUNCTIONS

- TO_CHAR()
- TO_DATE()

i) TO_CHAR():

    - TO CONVERT DATE INTO CHARACTER(STRING) TYPE AND ALSO DISPLAY DATE EXPRESSION IN DIFFERENT FORMATS.

SYNTAX:

    TO_CHAR(SYSDATE,'<INTERVALS>')

Year Formats:

YYYY	- IT RETURNS FOUR DIGITS FORMAT(2023)
YY	- IT RETURNS THE LAST TWO DIGITS FORMAT (23)
YEAR	- IT RETURNS CHARACTER FORMAT (Twenty Twenty Three)
CC	- Centuary 21
AD / Bc	- Ad YEAR / Bc YEAR

EX:

```
SQL> SELECT TO_CHAR(SYSDATE,'YYYY YY YEAR CC BC') FROM DUAL;
TO_CHAR(SYSDATE,'YYYYYYYEARCCBC')
----------------------------------------------------------
2023 23 TWENTY TWENTY-THREE 21 AD
```

Month Format:

MM	- IT RETURNS Month in Number format
MON	- First Three Char's From Month Spelling
Month	- It retruns the Full Name Of Month

EX:

```
SQL> SELECT TO_CHAR(SYSDATE,'MM MON MONTH') FROM DUAL;
TO_CHAR(SYSDATE,'MMMONMONTH')
------------------------------------------------------
02 FEB FEBRUARY
```

Day Formats:

DDD	- Day Of The Year.
DD	- Day Of The Month.
D	- Day Of The Week

Sun - 1
Mon - 2
Tue - 3
Wen - 4

Thu - 5
Fri - 6
Sat - 7

DAY	- Full Name Of The Day
DY	- First Three Char's Of Day Spelling

EX:

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

```
TO_CHAR(SYSDATE,'DDDDDDDAYDY')
----------------------------------------------------------
042 11 7 SATURDAY  SAT
```

Quater Format:

Q - One Digit Quater Of The Year

1 - Jan - Mar
2 - Apr - Jun
3 - Jul - Sep
4 - Oct - Dec

EX:

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

```
T
-
1
```

Week Format:

WW	- Week Of The Year
W	- Week Of Month

EX:

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

```
TO_C
----
06 2
```

Time Format:

HH	- Retrun Hour Part in 12hrs Format
HH24	- 24 Hrs Fromat
MI	- Minute Part
SS	- Seconds Part
AM / PM	- Am Time (Or) Pm Time

EX:

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

```
TO_CHAR(SYSDAT
--------------
05 17 18 38 PM
```

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE JOINED IN 1981 BY USING TO_CHAR() FUNCTION?

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 EIGHTY-ONE';

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE JOINED IN 1980,1982,1983?

SQL> SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,'YY') IN('80','82','83');

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE JOINED IN MONTH OF MARCH,JUNE,DECEMBER BY USING TO_CHAR()?

SQL> SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,'MM') IN('03','06','12');

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE JOINED IN MONTH OF DECEMBER IN 1981?

SQL> SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,'MMYYYY')='121981';

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE JOINED IN 2ND QUATER OF 1981?

SQL> SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,'QYYYY')='21981';

(OR)

SQL> SELECT * FROM EMP WHERE TO_CHAR(HIREDATE,'Q')='2' AND
TO_CHAR(HIREDATE,'YYYY')='1981';

ii) TO_DATE():

- TO CONVERT CHARACTER TYPE TO ORACLE DEFAULT DATE TYPE.

SYNTAX:

TO_DATE(STRING)

EX:

```
SQL> SELECT TO_DATE('23/AUGUST/2022') FROM DUAL;
TO_DATE('
---------
23-AUG-22

SQL> SELECT TO_DATE('23/AUGUST/2022')+5 FROM DUAL;
TO_DATE('
---------
28-AUG-22

SQL> SELECT TO_DATE('23/AUGUST/2022')-5 FROM DUAL;
TO_DATE('
---------
18-AUG-22
```

MULTIPLE ROW FUNCTIONS

- THESE FUNCTIONS ARE ALSO CALLED AS "GROUPING FUNCTIONS / AGGREGATIVE FUNCTIONS".
- THESE FUNCTIONS ARE "SUM(),AVG(),MIN(),MAX(),COUNT()".

SUM():

- IT RETURNS TOTAL VALUE.

EX:
```
SQL> SELECT SUM(SAL) FROM EMP;
  SUM(SAL)
----------
     29025

SQL> SELECT SUM(SAL) FROM EMP WHERE JOB='MANAGER';
  SUM(SAL)
----------
      8275
```

AVG():

- IT RETURNS THE AVEARAGE OF TOTAL VALUE.

EX:
```
SQL> SELECT AVG(SAL) FROM EMP;
  AVG(SAL)
----------
2073.21429
```

MIN():

- IT RETURNS THE MINIMUM VALUE.

EX:
```
SQL> SELECT MIN(SAL) FROM EMP;
  MIN(SAL)
----------
       800

SQL> SELECT MIN(HIREDATE) FROM EMP;
MIN(HIRED
---------
17-DEC-80

SQL> SELECT MIN(SAL) FROM EMP WHERE DEPTNO=30;
  MIN(SAL)
----------
       950
```

MAX():

- IT RETURNS MAXIMUM VALUE.

EX:

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

COUNT():

i) COUNT(*)
ii) COUNT(COLUMN NAME)
iii) COUNT(DISTINCT <COLUMN NAME>)

i) COUNT(*):

- COUNTING ALL ROWS INCLUDING DUPLICATES AND NULLS.

EX:

```
SQL> SELECT * FROM TEST;

       SNO NAME
---------- ----------
         1 A
         2 B
         3
         4 C
         5 A
         6 C

SQL> SELECT COUNT(*) FROM TEST;
  COUNT(*)
----------
         6
```

ii) COUNT(COLUMN NAME):

- COUNTING ALL ROWS INCLUDING DUPLICATES BUT NOT NULLS.

EX:

```
SQL> SELECT COUNT(NAME) FROM TEST;
COUNT(NAME)
-----------
          5
```

iii) COUNT(DISTINCT <COLUMN NAME>):

- COUNTING UNIQUE ROWS ONLY.(NO DUPLICATES & NO NULLS)

EX:

```
SQL> SELECT COUNT(DISTINCT NAME) FROM TEST;
```

```
COUNT(DISTINCTNAME)
-------------------
               3
```

==
(13-02-2023) **CLAUSES**
==

- IS A STATEMENT WHICH IS USED TO ADD TO SQL QUERY FOR PROVIDING THE ADDITIONAL FACILITIES ARE "FILTERING ROWS,SORTING VALUES, GROUPING SIMILAR DATA,FINDING SUB TOTAL AND GRAND TOTALS" BASED ON COLUMNS IN A TABLE.
- ORACLE SUPPORTS THE FOLLOWING CLAUSES ARE,
  > WHERE
  > ORDER BY
  > GROUP BY
  > HAVING
  > ROLLUP
  > CUBE

SYNTAX:

<SQL QUERY> + <CLAUSE STATEMENT>;

WHERE:

- FILTERING ROWS BEFORE GROUPING DATA IN A TABLE.

SYNTAX:

WHERE <FILTERING CONDITION>

NOTE:

- IT CAN BE USED IN "SELECT,UPDATE,DELETE" COMMANDS ONLY.

EX:
```
SELECT * FROM EMP WHERE EMPNO=7788;
UPDATE EMP SET SAL=45000 WHERE DEPTNO=20;
DELETE FROM EMP WHERE JOB='CLERK';
```

ORDER BY:

- TO ARRANGE A SPECIFIC COLUMN VALUES IN ASCENDING OR DESCENDING ORDER.BY DEFAULT ORDER BY CLAUSE WILL ARRANGE VALUES IN ASCENDING ORDER IF WE WANT TO ARRANGE VALUES IN DESCENDING ORDER THE USE "DESC" KEYWORD.

SYNTAX:

SELECT * / <LIST OF COLUMNS> FROM <TN> ORDER BY <COLUMN NAME1> <ASC/DESC>, <COLUMN NAME2> <ASC/DESC>,.....................................;

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

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE WORKING IN DEPTNO IS 20 AND ARRANGE THOSE EMPLOYEES SALARIES IN DESCENDING ORDER?
SQL> SELECT * FROM EMP WHERE DEPTNO=20 ORDER BY SAL DESC;

EX:
WAQ TO ARRANGE EMPLOYEES DEPTNO's IN ASCENDING ORDER AND THEIR SALARIES IN DESCENDING FROM EACH DEPTNO WISE?
SQL> SELECT * FROM EMP ORDER BY DEPTNO,SAL DESC;

NOTE:
- ORDER BY CLAUSE NOT ONLY ON COLUMN NAMES EVEN THOUGH WE CAN APPLY THE POSITION OF COLUMNS IN SELECT QUERY.

EX:
SQL> SELECT * FROM EMP ORDER BY 6 DESC;
SQL> SELECT ENAME,SAL FROM EMP ORDER BY 2 ;
SQL> SELECT SAL FROM EMP ORDER BY 1 DESC;

"NULL" CLAUSES IN ORDER BY:
- NULLS FIRST
- NULLS LAST

- BY DEFAULT ORACLE BY CLAUSE WILL ARRANGE "NULLS" IN ASCENDING ORDER
  i) FIRST VALUES
  ii) LAST NULLS
- TO CHANGE THE ABOVE ORDER THEN WE USE "NULLS FIRST" CLAUSE ALONG WITH ORDER BY,
EX:
SQL> SELECT * FROM EMP ORDER BY COMM;
SQL> SELECT * FROM EMP ORDER BY COMM NULLS FIRST;

- BY DEFAULT ORACLE BY CLAUSE WILL ARRANGE "NULLS" IN DESCENDING ORDER
  i) FIRST NULLS
  ii) LAST VALUES
- TO CHANGE THE ABOVE ORDER THEN WE USE "NULLS LAST" CLAUSE ALONG WITH ORDER BY,
EX:
SQL> SELECT * FROM EMP ORDER BY COMM DESC ;
SQL> SELECT * FROM EMP ORDER BY COMM DESC NULLS LAST;

GROUP BY:

- IT IS USED TO DIVIDING GROUPS BASED ON A SPECIFIC COLUMN/(S) WISE.
- WHEN WE GROUP BY WE SHOULD USE GROUPING FUNCTIONS ARE " SUM(),AVG(),MIN(),MAX(),COUNT() ".

SYNTAX:
SELECT <COLUMN NAME1>,....................,<AGGREGATIVE FUNCTION NAME1>,...........
FROM <TN> GROUP BY <COLUMN NAME1>,<COLUMN NAME2>,...............;

EX:
SQL> SELECT * FROM CUSTOMER;

```
      CID  CNAME            GENDER
       ---- ----------       -------------
        1   SMITH            M
        2   ALLEN            F
        3   WARD             F
        4   WARNER           M
        5   MILLER           M
```

WAQ TO FIND OUT THE NO.OF MALE AND FEMALE CUSTOMERS?
SQL> SELECT GENDER,COUNT(*) FROM CUSTOMER GROUP BY GENDER;
(OR)
SQL> SELECT GENDER,COUNT(GENDER) FROM CUSTOMER GROUP BY GENDER;

```
GENDER          COUNT(GENDER)
-------------   -------------
M               3
F               2
```

EX:
WAQ TO FIND OUT THE NO.OF EMPLOYEES ARE WORKING IN EACH JOB?
SQL> SELECT JOB,COUNT(JOB) NO_OF_EMPLOYEES FROM EMP GROUP BY JOB;

EX:
WAQ TO DISPLAY NO.OF EMPLOYEES WORKING IN EACH JOB ALONG WITH DEPTNO WISE?
```
SQL> SELECT JOB,DEPTNO,COUNT(JOB) FROM EMP
        GROUP BY JOB,DEPTNO;
```

EX:
WAQ TO DISPLAY SUM OF SALARIES OF EACH DEPTNO WISE?
```
QL> SELECT DEPTNO,SUM(SAL) FROM EMP GROUP BY DEPTNO
  2  ORDER BY DEPTNO;
```

EX:
WAQ TO DISPLAY NO.OF EMPLOYEES,SUM OF SALARIES,AVERAGE SALARY,MINIMUM AND MAXIMUM SALARY FROM EACH DEPTNO WISE?
```
SQL> SELECT DEPTNO,COUNT(*) NO_OF_EMPLOYEES,
  2  SUM(SAL) SUM_SALARY,AVG(SAL) AVG_SALARY,
  3  MIN(SAL) MIN_SALARY,MAX(SAL) MAX_SALARY
  4  FROM EMP GROUP BY DEPTNO ORDER BY DEPTNO;
```

HAVING:

- FILTERING ROWS AFTER GROUPING DATA IN A TABLE.
- IT CAN BE USED ALONG WITH GROUP BY CLAUSE ONLY.

SYNTAX:
**SELECT <COLUMN NAME1>,....................,<AGGREGATIVE FUNCTION NAME1>,...........
FROM <TN> GROUP BY <COLUMN NAME1>,<COLUMN NAME2>,........HAVING<CONDITION>;**

EX:
WAQ TO DISPLAY DEPTNO's IN WHICH DEPTNO MORE THAN 3 EMPLOYEES?

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

EX:
WAQ TO DISPLAY JOB's IN WHICH JOB THE SUM OF SALARY IS LESS THAN 5000?

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

USING ALL CLAUSES IN A SINGLE SELECT STATEMENT:

SYNTAX:

```
SELECT <COLUMN NAME1>,.........,<AGGREGATIVE FUNCTION NAME1>,..............
FROM <TN> [ WHERE <FILTERING CONDITION>
          GROUP BY <COLUMN NAME1>,..............
          HAVING <FILTERING CONDITION>
          ORDER BY <COLUMN NAME1> <ASC/DESC>,............
         ];
```

EX:

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

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

ROLLUP & CUBE

- THESE ARE TWO SPECIAL CLAUSES WHICH ARE USED TO DESCRIBE SUB TOTALS AND GRAND TOTAL BASED ON COLUMNS AUTOMATICALLY.
- THESE CLAUSES ARE USED ALONG WITH GROUP BY CLAUSE ONLY.
  > ROLLUP : FIND OUT SUB & GRAND TOTAL BASED ON SINGLE COLUMN.
  > CUBE : FIND OUT SUB & GRAND TOTAL BASED ON MULTIPLE COLUMNS.

SYNTAX FOR ROLLUP:

```
SELECT <COLUMN NAME1>,.........,<AGGREGATIVE FUNCTION NAME1>,.........................
FROM <TN> GROUP BY  ROLLUP(<COLUMN NAME1>,<COLUMN NAME2>.............);
                                  |              ........................................
                          operational column                      |
                                                        supporting columns for
                                                         operational column
```

ROLLUP WITH A SINGLE COLUMN:

EX:

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

ROLLUP WITH MULTIPLE COLUMNS:

EX:

```
SQL> SELECT DEPTNO,JOB,COUNT(*) FROM EMP
  2  GROUP BY ROLLUP(DEPTNO,JOB);
```

EX:

```
SQL> SELECT JOB,DEPTNO,COUNT(*) FROM EMP
  2  GROUP BY ROLLUP(JOB,DEPTNO);
```

SYNTAX FOR CUBE:

```
SELECT <COLUMN NAME1>,.........,<AGGREGATIVE FUNCTION NAME1>,.........................
FROM <TN> GROUP BY  CUBE(<COLUMN NAME1>,<COLUMN NAME2>.............);
                                        |
                              all columns are operational columns
```

CUBE WITH A SINGLE COLUMN:

EX:

```
SQL> SELECT DEPTNO,SUM(SAL) FROM EMP
  2  GROUP BY CUBE(DEPTNO) ORDER BY DEPTNO;
```

CUBE WITH MULTIPLE COLUMNS:

EX:

```
SQL> SELECT DEPTNO,JOB,SUM(SAL) FROM EMP
  2  GROUP BY CUBE(DEPTNO,JOB) ORDER BY DEPTNO,JOB;
```

GROUPING_ID()

- IT IS PRE-DEFINED FUNCTION.WHICH IS USED TO GENERATE GROUPING ID's FOR EACH GROUPING COLUMN SUB ROWS AND ALSO GRAND TOTAL ROW.

SYNTAX:

```
GROUPING_ID(COL1,COL2,COL3,......)
```

EX:

```
1ST GROUPING COLUMN SUB TOTAL ROWS --------->   1
2ND GROUPING COLUMN SUB TOTAL ROWS --------->   2
3RD GROUPING COLUMN SUB TOTAL ROWS ---------->  3
GRAND TOTAL ROW-------------------------------------------------> 4
```

EX:

```
SQL> SELECT DEPTNO,JOB,SUM(SAL),GROUPING_ID(DEPTNO,JOB)
  2  FROM EMP GROUP BY CUBE(DEPTNO,JOB)
  3  ORDER BY DEPTNO,JOB;
```

==

(16-02-2023) **JOINS**

==

- IN RDBMS DATA CAN BE STORED IN MULTIPLE TABLES FROM THOSE MULTIPLE TABLES IF WE WANT TO RETRIEVE THE REQUIRED DATA THEN WE USE A TECHNIQUE IS CALLED AS "JOINS".
- JOINS ARE USED TO RETRIEVING SPECIFIC DATA FROM MULTIPLE TABLE AT A TIME.
- THESE JOINS STATEMENTS CAN WRITTEN IN TWO WAYS,
    1. NON-ANSI JOINS (ORACLE8i)
        - EQUI JOIN
        - NON-EQUI JOIN
        - SELF JOIN
    2. ANSI JOINS(ORACLE9i)
        - INNER JOIN
        - OUTER JOIN
            > LEFT OUTER JOIN
            > RIGHT OUTER JOIN
            > FULL OUTER JOIN
        - CROSS JOIN
        - NATURAL JOIN

WHY ANSI JOINS:

- ANSI JOINS ARE PORTABILTY STATEMENTS i.e WE CAN MOVE ANSI JOIN STATEMENT FROM ONE DATABASE TO ANOTHER DATABASE WITHOUT ANY CHANGES.

SYNTAX FOR NON-ASNI JOINS:

SELECT * FROM <TABLE NAME1>,<TABLE NAME2> WHERE <JOINING CONDITION>;

SYNTAX FOR ANSI JOINS:

SELECT * FROM <TABLE NAME1> <JOIN KEY> <TABLE NAME2> ON <JOINING CONDITION>;

DEMO TABLES:

```
SQL> SELECT * FROM COURSE;

  CID CNAME          CFEE
----- ----------     ----------
    1  ORACLE        2500
    2  JAVA          3000
    3  .NET          5000
```

```
SQL> SELECT * FROM STUDENT;

     STID          SNAME      CRID
   ---------       ----------  ---------
     1021          SMITH          1
     1022          ALLEN          2
     1023          WARD           1
     1024          JONES
```

EQUI JOIN / INNER JOIN:

- RETRIEVING DATA FROM MULTIPLE TABLES BASED ON AN " = " OPERATOR.
- WHEN WE USE EQUI JOIN WE SHOULD HAVE A COMMON COLUMN IN BOTH TABLES AND THOSE COMMON COLUMN DATATYPES MUST BE MATCH.
- WHEN WE PERFORM JOINS ON TABLES THERE IS NO NEED TO HAVE A RELATIONSHIP BETWEEN TABLES.HERE RELATIONSHIP BETWEEN TABLES ARE OPTIONAL.
- RETRIEVING MATCHING ROWS(DATA) ONLY FROM MULTIPLE TABLES.

SYNTAX:
NON-ANSI:
WHERE <TN1>.<COMMON COLUMN NAME> = <TN2>.<COMMON COLUMN NAME>;

ANSI:
ON <TN1>.<COMMON COLUMN NAME> = <TN2>.<COMMON COLUMN NAME>;

EX:
WAQ TO FETCH STUDENTS AND THE CORRESPONDING COURSE DETAILS FROM STUDENT,COURSE TABLES BY USING EQUI / INNER JOIN?

NON-ANSI:
```
SQL> SELECT * FROM STUDENT ,COURSE WHERE STUDENT.CID=COURSE.CID;
                              (OR)
SQL> SELECT * FROM STUDENT S ,COURSE C WHERE  S.CID=C.CID;
```

ANSI:
```
SQL> SELECT * FROM STUDENT INNER JOIN COURSE ON STUDENT.CID=COURSE.CID;
                              (OR)
SQL> SELECT * FROM STUDENT S INNER JOIN COURSE  C ON S.CID=C.CID;
```

EX:
WAQ TO DISPLAY STUDENTS WHO ARE SELECTED "ORACLE" COURSE?

NON-ANSI:
```
SQL> SELECT SNAME,CNAME FROM STUDENT S,COURSE C
  2  WHERE S.CID=C.CID AND CNAME='ORACLE';
```

ANSI:
```
SQL> SELECT SNAME,CNAME FROM STUDENT S INNER JOIN
  2  COURSE C ON S.CID=C.CID AND CNAME='ORACLE';
                    (OR)
SQL> SELECT SNAME,CNAME FROM STUDENT S INNER JOIN
```

```
  2  COURSE C ON S.CID=C.CID WHERE CNAME='ORACLE';
```

RULE OF JOINS:

A ROW IN A TABLE IS COMPARING WITH ALL ROWS IN ANOTHER TABLE.

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE WORKING IN "CHICAGO" LOCATION?

NON-ANSI:
```
SQL> SELECT * FROM EMP E,DEPT D WHERE E.DEPTNO=D.DEPTNO AND LOC='CHICAGO';
```

ANSI:
```
SQL> SELECT * FROM EMP E INNER JOIN DEPT D ON E.DEPTNO=D.DEPTNO AND LOC='CHICAGO';
```

EX:
WAQ TO DISPLAY NO.OF EMPLOYEES WORKING IN EACH DEPARTMENT BY USING
EQUI / INNER JOIN?

NON-ANSI:
```
SQL> SELECT DNAME,COUNT(*) FROM EMP E,DEPT D
  2  WHERE E.DEPTNO=D.DEPTNO GROUP BY DNAME;
```

ANSI:
```
SQL> SELECT DNAME,COUNT(*) FROM EMP E INNER JOIN DEPT D
  2  ON E.DEPTNO=D.DEPTNO GROUP BY DNAME;
```

EX:
WAQ TO DISPLAY SUM OF SALARIES OF DEPARTMENTS IN WHICH DEPARTMENT THE
SUM OF SALARY IS MORE THAN 10000?

NON-ANSI:
```
SQL> SELECT DNAME,SUM(SAL) FROM EMP E,DEPT D
    WHERE E.DEPTNO=D.DEPTNO GROUP BY DNAME HAVING SUM(SAL)>10000;
```

ANSI:
```
SQL> SELECT DNAME,SUM(SAL) FROM EMP E INNER JOIN DEPT D
    ON E.DEPTNO=D.DEPTNO GROUP BY DNAME HAVING SUM(SAL)>10000;
```

EX:
WAQ TO DISPLAY NO.OF EMPLOYEES WORKING IN EACH LOCATION FROM EMP,DEPT
TABLES BY USING EQUI / INNER JOIN?

NON-ANSI:
```
SQL> SELECT LOC,COUNT(*) FROM EMP E,DEPT D
  2  WHERE E.DEPTNO=D.DEPTNO GROUP BY LOC;
```

ANSI:
```
SQL> SELECT LOC,COUNT(*) FROM EMP E INNER JOIN DEPT D
  2  ON E.DEPTNO=D.DEPTNO GROUP BY LOC;
```

OUTER JOINS:

- THESE ARE AGAIN THREE TYPES,
  - I) LEFT OUTER JOIN
  - II) RIGHT OUTER JOIN
  - III) FULL OUTER JOIN

I) LEFT OUTER JOIN:

- RETRIEVING ALL MATCHING ROWS FROM BOTH TABLES AND ALSO RETRIEVING UNMATCHING ROWS FROM THE LEFT SIDE TABLE ONLY.

EX:

ANSI:

```
SQL> SELECT * FROM STUDENT S LEFT OUTER JOIN COURSE C
  2  ON S.CID=C.CID;
```

NON-ANSI:

- WHEN WE IMPLEMENT OUTER JOINS IN NON-ANSI FORMAT THEN WE SHOULD USE A JOIN OPERATOR (+) ON A TABLE IN JOIN CONDITION.

```
SQL> SELECT * FROM STUDENT S,COURSE C WHERE S.CID=C.CID(+);
```

II) RIGHT OUTER JOIN:

- RETRIEVING ALL MATCHING ROWS FROM BOTH TABLES AND ALSO RETRIEVING UNMATCHING ROWS FROM THE RIGHT SIDE TABLE ONLY.

EX:

ANSI:

```
SQL> SELECT * FROM STUDENT S RIGHT OUTER JOIN COURSE C ON S.CID=C.CID;
```

NON-ANSI:

```
SQL> SELECT * FROM STUDENT S,COURSE C WHERE S.CID(+)=C.CID;
```

III) FULL OUTER JOIN:

- IT COMBINATION OF LEFT OUTER AND RIGHT OUTER JOINS.
- RETRIEVING ALL MATCHING AND UNMACHING ROWS FROM MULTIPLE TABLES AT A TIME.

EX:

ANSI:

```
SQL> SELECT * FROM STUDENT S FULL OUTER JOIN COURSE C ON S.CID=C.CID;
```

NON-ANSI:

```
SQL> SELECT * FROM STUDENT S,COURSE C WHERE S.CID(+)=C.CID(+);

ERROR at line 1:
ORA-01468: a predicate may reference only one outer-joined table.
```

- TO OVERCOME THE ABOVE ERROR THEN WE USE "UNION" OPERATOR TO FULLFILL FULL OUTER JOIN MECHANISM LIKE BELOW,

SOLUTION:

```
SQL> SELECT * FROM STUDENT S,COURSE C WHERE S.CID=C.CID(+)
  2  UNION  SELECT * FROM STUDENT S,COURSE C WHERE S.CID(+)=C.CID;
```

CROSS JOIN:

- RETRIEVING DATA FROM MULTIPLE TABLES WITHOUT ANY JOIN CONDITION.
- IN CROSS JOIN MECHANISM EACH ROW OF THE FIRST TABLE WILL JOINS WITH EACH ROW OF THE SECOND TABLE FOR EXAMPLE A TABLE IS HAVING (m) NO.OF ROWS AND ANOTHER TABLE IS HAVNG (n) NO.OF ROWS THEN THE RESULT IS (mXn) ROWS.

EX:

ANSI:

SQL> SELECT * FROM STUDENT CROSS JOIN COURSE;

NON-ANSI:

SQL> SELECT * FROM STUDENT,COURSE;

EX:

```
SQL> SELECT * FROM ITEMS1;

      SNO          INAME            PRICE
-------------      ----------       ----------
        1          PIZZA            160
        2          BURGER           80
```

```
SQL> SELECT * FROM ITEMS2;

      SNO          INAME              PRICE
----------         --------------------  ----------
      101          PEPSI               25
      102          COCACOLA            20
```

ANSI:

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

NON-ANSI:

```
SQL> SELECT I1.INAME,I1.PRICE,I2.INAME,I2.PRICE,
  2  I1.PRICE+I2.PRICE TOTAL_AMOUNT FROM ITEMS1 I1 ,ITEMS2 I2;
```

```
INAME     PRICE          INAME        PRICE TOTAL_AMOUNT
----------         ----------         -------------------- ---------- ------------
PIZZA      160    PEPSI          25        185
```

NON-EQUI JOIN:

- RETRIEVING DATA FROM MULTIPLE TABLES BASED ON ANY CONDITION EXCEPT AN "=" OPERATOR.

EX:

```
SQL> SELECT * FROM TEST1;

      SNO NAME
----------     ----------
       10   SMITH
       20   ALLEN
```

```
SQL> SELECT * FROM TEST2;

      SNO    SAL
----------       ----------
       10     25000
       30     34000
```

NON-ANSI:

```
SQL> SELECT * FROM TEST1 T1,TEST2 T2 WHERE T1.SNO>T2.SNO;
```

ANSI:

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

      SNO NAME   SNO       SAL
----------     ---------- ---------- ----------
       20    ALLEN  10      25000
```

EX:
WAQ TO DISPLAY EMPLOYEES WHOSE SALARY IS BETWEEN LOW SALARY AND HIGH SALARY?

NON-ANSI:

```
SQL> SELECT ENAME,SAL,LOSAL,HISAL FROM EMP,SALGRADE
  2  WHERE SAL BETWEEN LOSAL AND HISAL;
                        (OR)
SQL> SELECT ENAME,SAL,LOSAL,HISAL FROM EMP,SALGRADE
  2  WHERE (SAL>=LOSAL) AND (SAL<=HISAL);
```

ANSI:

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

NATURAL JOIN:

- RETRIEVING MATCHING ROWS FROM MULTIPLE TABLES JUST LIKE EQUI JOIN.

EQUI JOIN	**NATURAL JOIN**
- RETRIEVING DATA INCLUDING DUPLICATE COLUMNS.	- RETRIEVING DATA WITHOUT DUPLICATE COLUMNS.
- USER CAN DEFINED A JOIN CONDITION BY EXPLICITLY.	- SYSTEM CAN DEFINED A JOIN CONDITION BY IMPLICITLY.
- HERE COMMON COLUMN NAME IS OPTIONAL	- COMMON COLUMN NAME IS MANDATORY

EX:

```
SQL> SELECT * FROM STUDENT S NATURAL JOIN COURSE C;----ANSI
```

SELF JOIN:

- A TABLE JOINING BY ITSELF.
- WORKING ON A SINGLE TABLE ONLY.
- WHEN WE WORK WITH SELF JOIN WE SHOULD CREATE ALIAS NAMES ON A TABLE.WHENEVER WE CREATE ALIAS NAME ON A TABLE NAME INTERNALLY SYSTEM WILL PREPARE VIRTUAL TABLE ON EACH ALIAS NAME.
- WE CAN CREATE ANY NO.OF ALIAS NAMES ON A SINGLE TABLE BUT EACH ALIAS NAME SHOULD BE DIFFERENT.
- WE CAN IMPLEMENT SELF JOIN IN TWO CASES:

CASE-1: COMPARING A SINGLE COLUMN VALUES BY ITSELF WITH IN THE TABLE.
CASE-2: COMPARING TWO DIFFERENT COLUMNS VALUES TO EACH OTHER WITH IN THE TABLE.

CASE-1: COMPARING A SINGLE COLUMN VALUES BY ITSELF WITH IN THE TABLE:

EX:
WAQ TO DISPLAY EMPLOYEES WHO ARE WORKING IN THE SAME LOCATION THE EMPLOYEE "SMITH" IS ALSO WORKING?

DEMO_TABLE:

```
SQL> SELECT * FROM TEST;

ENAME     LOC
----------          ----------
SMITH    HYD
ALLEN    MUMBAI
JONES    HYD
ADAMS    CHENNAI
```

NON-ANSI:

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

(OR)

ANSI:

```
SQL> SELECT T1.ENAME,T1.LOC FROM TEST T1 JOIN TEST T2
  2  ON T1.LOC=T2.LOC AND T2.ENAME='SMITH';

ENAME     LOC
----------   ----------
SMITH      HYD
JONES      HYD
```

EX:
WAQ TO DISPLAY EMPLOYEES WHOSE SALARY IS SAME AS THE EMPLOYEE "SCOTT" SALARY?

NON-ANSI:

```
SQL> SELECT E1.ENAME,E1.SAL FROM EMP E1,EMP E2
  2  WHERE E1.SAL=E2.SAL AND E2.ENAME='SCOTT';
```

ANSI:

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

ENAME     SAL
----------          ----------
SCOTT      3000
FORD       3000
```

CASE-2: COMPARING TWO DIFFERENT COLUMNS VALUES TO EACH OTHER WITH IN THE TABLE.

EX:

WAQ TO DISPLAY MANAGERS AND THEIR EMPLOYEES?

NON-ANSI:

```
SQL> SELECT M.ENAME MANAGER,E.ENAME EMPLOYEES
  2  FROM EMP E,EMP M WHERE M.EMPNO=E.MGR;
```

ANSI:

```
SQL> SELECT M.ENAME MANAGER,E.ENAME EMPLOYEES
  2  FROM EMP E JOIN EMP M ON M.EMPNO=E.MGR;
```

EX:

WAQ TO DISPLAY EMPLOYEES WHO ARE JOINED BEFORE THEIR MANAGER?

NON-ANSI:

```
SQL> SELECT E.ENAME EMPLOYEES,E.HIREDATE E_DOJ,M.ENAME MANAGER,
  2  M.HIREDATE M_DOJ FROM EMP E,EMP M WHERE M.EMPNO=E.MGR
  3  AND E.HIREDATE<M.HIREDATE;
```

ANSI:

```
SQL> SELECT E.ENAME EMPLOYEES,E.HIREDATE E_DOJ,M.ENAME MANAGER,
  2  M.HIREDATE M_DOJ FROM EMP E JOIN EMP M ON M.EMPNO=E.MGR
  3  AND E.HIREDATE<M.HIREDATE;
```

EX:

WAQ TO DISPLAY EMPLOYEES WHOSE SALARY IS MORE THAN THEIR MANAGER SALARY?

NON-ANSI:

```
SQL> SELECT E.ENAME EMPLOYEES,E.SAL E_SALARY,
  2  M.ENAME MANAGER,M.SAL M_SALARY FROM EMP E,EMP M
  3  WHERE M.EMPNO=E.MGR AND E.SAL>M.SAL;
```

ANSI:

```
SQL> SELECT E.ENAME EMPLOYEES,E.SAL E_SALARY,
  2  M.ENAME MANAGER,M.SAL M_SALARY FROM EMP E JOIN EMP M
  3  ON M.EMPNO=E.MGR AND E.SAL>M.SAL;
```

HOW TO JOIN MORE THAN TWO TABLES:

SYNTAX FOR NON-ANSI:

```
SELECT * FROM <TN1>,<TN2>,<TN3>,..........................................................
WHERE <CONDITION1> AND <CONDITION2> AND ................;
```

EX ON EQUI JOIN:

```
SQL> SELECT * FROM STUDENT S,COURSE C,REGISTER R
  2  WHERE S.CID=C.CID AND C.CID=R.CID;
```

SYNTAX FOR ANSI:

```
SELECT * FROM <TN1> <JOIN KEY> <TN2> ON <JOIN CONDITION1>
<JOIN KEY> <TN3> ON <JOIN CONDITION2>
<JOIN KEY> <TN4> ON <JOIN CONDITION3>
......................................................................
.....................................................................
<JOIN KEY> <TN n> ON <JOIN CONDITION n-1>;
```

EX ON INNER JOIN:

```
SQL> SELECT * FROM STUDENT S INNER JOIN COURSE C
  2  ON S.CID=C.CID INNER JOIN REGISTER R ON C.CID=R.CID;
```

==

(22-02-2023) **DATAINTEGRITY**

==

- TO MAINTAIN ACCURATE & CONSISTENCY DATA IN DATABASE TABLES.
  1. DECLARATIVE DATAINTEGRITY
     > PRE-DEFINED INTEGRITY RULES.
     > BY USING "CONSTRAINTS" (SQL)
  2. PROCEDURAL DATAINTEGRITY
     > USER-DEFINED INTEGRITY RULES.
     > BY USING "TRIGGERS" (PL/SQL)

1. DECLARATIVE DATAINTEGRITY:

- IT CAN BE IMPLEMENTED BY USING "CONSTRAINTS".
  - i) ENTITY INTEGRITY
  - ii) REFERENCIAL INTEGRITY
  - iii) DOMAIN INTEGRITY

i) ENTITY INTEGRITY:

- IT ENSURE THAT TO IDENTIFY UNIQUE IDENTITY FOR A ROW IN A TABLE.
- IT CAN BE IMPLEMENTED BY USING "UNIQUE / PRIMARY KEY" CONSTRAINTS.

ii) REFERENCIAL INTEGRITY:

- TO MAKE RELATIONSHIP BETWEEN TABLES.
- IT CAN IMPLEMENT BY USING "FOREIGN KEY"(REFERENCES KEY)

iii) DOMAIN INTEGRITY:

- TO CHECK DATA BEFORE ALLOWED INTO A COLUMN.
- IT CAN BE IMPLEMENTED BY USING "CHECK,NOT NULL,DEFAULT" CONSTRAINTS.

CONSTRAINTS:

- ARE USED TO RESTRICTED / ENFORCE UNWANTED DATA INTO TABLES.
  > UNIQUE
  > NOT NULL
  > CHECK

> PRIMARY KEY
> FOREIGN KEY(REFERENCES)
> DEFAULT

- ALL DATABASES ARE SUPPORTING TWO TYPES OF METHODS TO APPLY CONSTRAINTS ON TABLE COLUMNS.

i) COLUMN LEVEL:

- IN THIS LEVEL WE CAN DEFINE A CONSTRAINT ON EACH COLUMN WISE IN A TABLE.

SYNTAX:

CREATE TABLE <TN>(<COLUMN NAME1> <DT>[SIZE] <CONSTRAINT TYPE>, <COLUMN NAME2> <DT>[SIZE] <CONSTRAINT TYPE>,..........................);

ii) TABLE LEVEL:

- IN THIS LEVEL CONSTRAINT CAN BE APPLIED AFTER ALL COLUMNS ARE DECLARED.

SYNTAX:

CREATE TABLE <TN>(<COLUMN NAME1> <DT>[SIZE],<COLUMN NAME2> <DT>[SIZE],........... ....,<CONSTRAINT TYPE>(<COLUMN NAME1>,<COLUMN NAME2>,......));

NOTE:

- TABLE LEVEL CONSTRAINTS ARE ALSO CALLED AS "COMPOSITE CONSTRAINTS".

UNIQUE

- TO RESTRICTED DUPLICATE VALUES BUT ALLOWED NULLS.

EX:

COLUMN LEVEL:

```
SQL> CREATE TABLE TEST1(SNO INT UNIQUE,NAME VARCHAR2(10) UNIQUE);
```

TESTING:

```
SQL> INSERT INTO TEST1 VALUES(1,'A');----ALLOWED
SQL> INSERT INTO TEST1 VALUES(1,'A');----NOT ALLOWED
SQL> INSERT INTO TEST1 VALUES(NULL,NULL);----ALLOWED
```

TABLE LEVEL:

```
SQL> CREATE TABLE TEST2(SNO INT,NAME VARCHAR2(10),UNIQUE(SNO,NAME));
```

TESTING:

```
SQL> INSERT INTO TEST2 VALUES(1,'A');----ALLOWED
SQL> INSERT INTO TEST2 VALUES(1,'A');----NOT ALLOWED
SQL> INSERT INTO TEST2 VALUES(NULL,NULL);----ALLOWED
```

NOT NULL

- TO RESTRICTED NULLS BUT ALLOWED DUPLICATE VALUES.
- THIS CONSTRAINT CAN BE DEFINED AT COLUMN LEVEL ONLY.

EX:

COLUMN LEVEL:

```
SQL> CREATE TABLE TEST3(STID INT NOT NULL,SNAME VARCHAR2(10) NOT NULL);
```

TESTING:

```
SQL> INSERT INTO TEST3 VALUES(1,'A');----ALLOWED
SQL> INSERT INTO TEST3 VALUES(1,'A');---ALLOWED
SQL> INSERT INTO TEST3 VALUES(NULL,NULL);---NOT ALLOWED
```

CHECK

- TO CHECK VALUES WITH USER DEFINED CONDITION BEFORE ACCEPTING INTO COLUMN.

EX:

COLUMN LEVEL:

```
SQL> CREATE TABLE TEST4(EID INT UNIQUE NOT NULL,SAL NUMBER(10) NOT NULL
CHECK(SAL>=15000));
```

TESTING:

```
SQL> INSERT INTO TEST4 VALUES(1,14000);----NOT ALLOWED
SQL> INSERT INTO TEST4 VALUES(1,15000);----ALLOWED
```

TABLE LEVEL:

```
SQL> CREATE TABLE TEST5(ENAME VARCHAR2(10),
     SAL NUMBER(10),CHECK(ENAME=LOWER(ENAME) AND SAL>10000));
```

TESTING:

```
SQL> INSERT INTO TEST5 VALUES('SMITH',2000);        ----NOT ALLOWED
SQL> INSERT INTO TEST5 VALUES('smith',12000);       -----ALLOWED
```

PRIMARY KEY

- TO RESTRICTED DUPLICATE VALUES AND ALSO NULLS.
- IS NOTHING BUT IT IS A COMBINATION OF "UNIQUE AND NOT NULL" CONSTRAINTS.
- A TABLE IS HAVING ONLY ONE PRIMARY KEY CONSTRAINT.
- PRIMARY KEY IS ALSO CALLED AS A "CANDIDATE KEY".

EX:

COLUMN LEVEL:

```
SQL> CREATE TABLE TEST6(BCODE INT PRIMARY KEY,BNAME VARCHAR2(10));
```

TESTING:

```
SQL> INSERT INTO TEST6 VALUES(1021,'SBI');----ALLOWED
SQL> INSERT INTO TEST6 VALUES(1021,'ICICI');---NOT ALLOWED
SQL> INSERT INTO TEST6 VALUES(NULL,'ICICI');----NOT ALLOWED
```

TABLE LEVEL:

```
SQL> CREATE TABLE TEST7
  2  (
  3  BCODE INT,BNAME VARCHAR2(10),
  4  PRIMARY KEY(BCODE,BNAME)
  5  );
```

TESTING:

```
SQL> INSERT INTO TEST7 VALUES(1021,'SBI');----ALLOWED
SQL> INSERT INTO TEST7 VALUES(1021,'SBI');----NOT
SQL> INSERT INTO TEST7 VALUES(1021,'ICICI');---ALLOWED
```

FOREIGN KEY

- IT IS USED TO ESTABLISH RELATIONSHIP BETWEEN TABLES TO GET A REFERENCIAL IDENTITY FROM ONE TABLE TO ANOTHER TABLE.

BASIC RULES:
1. WE HAVE A COMMON COLUMN NAME(OPTIONAL).
2. DATATYPES OF COMMON COLUMN MUST BE MATCH.
3. WHEN WE CREATE RELATIONSHIP BETWEEN TABLES ONE TABLE SHOULD HAVE A PRIMARY KEY CONSTRAINT AND ANOTHER TABLE SHOULD HAVA A FOREIGN KEY CONSTRAINT.
4. A PRIMARY KEY CONSTRAINT TABLE IS CALLED AS "PARENT TABLE" AND A FOREIGN KEY CONSTRAINT TABLE IS CALLED AS "CHILD TABLE".
5. A FOREIGN KEY COLUMN IS ALLOWED THE VALUES OF PRIMARY KEY COLUMN ONLY.
6. BY DEFAULT A FOREIGN KEY COLUMN IS ALLOWED DUPLICATES AND NULLS.

SYNTAX:

```
<COMMON COLUMN OF CHILD TABLE> <DATATYPE>[SIZE] REFERENCES
<PARENT TABLE NAME>(COMMON COLUMN OF PARENT TABLE)
```

EX:

```
SQL> CREATE TABLE BRANCH(BCODE INT PRIMARY KEY,BNAME VARCHAR2(10));---PARENT TABLE

SQL> INSERT INTO BRANCH VALUES(1021,'CSE');
SQL> INSERT INTO BRANCH VALUES(1022,'EEE');

SQL> CREATE TABLE STUDENT(STID INT PRIMARY KEY,
  2  SNAME VARCHAR2(10),BCODE INT REFERENCES
  3  BRANCH(BCODE)); ------ CHILD TABLE

SQL> INSERT INTO STUDENT VALUES(1,'SMITH',1021);
SQL> INSERT INTO STUDENT VALUES(2,'ALLEN',1022);
SQL> INSERT INTO STUDENT VALUES(3,'WARD',NULL);
```

NOTE:

- ONCE WE ESTABLISHED RELATIONSHIP BETWEEN TABLES THERE ARE TWO RULES ARE COME INTO PICTURE.

RULES-1: (INSERTION RULE)

- WE CANNOT INSERT VALUES INTO A FOREIGN KEY COLUMN WHICH ARE NOT FOUND IN PRIMARY KEY COLUMN.

EX:

```
SQL> INSERT INTO STUDENT VALUES(4,'MILLER',1023);
ERROR at line 1:
ORA-02291: integrity constraint (MYDB4PM.SYS_C007881) violated - parent key not found
```

RULE-2: (DELETION RULE):

- WE CANNOT DELETE A ROW FROM PARENT TABLE IF THOSE PARENT ROWS ARE HAVING CHILD ROWS IN CHILD TABLE WITHOUT ADDRESSING TO CHILD TABLE.

EX:

```
SQL> DELETE FROM BRANCH WHERE BCODE=1021
ERROR at line 1:
ORA-02292: integrity constraint (MYDB4PM.SYS_C007881) violated - child record found
```

HOW TO ADDRESSING TO CHILD TABLE?

1. ON DELETE CASCADE
2. ON DELETE SET NULL

1. ON DELETE CASCADE:

- ONCE WE DELETE A ROW FROM THE PARENT TABLE THEN THE CORRESPONDING CHILD ROWS ARE ALSO DELETED FROM CHILD TABLE AUTOMATICALLY.

EX:

```
SQL> CREATE TABLE BRANCH1(BCODE INT PRIMARY KEY,BNAME VARCHAR2(10));---PARENT TABLE

SQL> INSERT INTO BRANCH1 VALUES(1021,'CSE');
SQL> INSERT INTO BRANCH1 VALUES(1022,'EEE');

SQL> CREATE TABLE STUDENT1(STID INT PRIMARY KEY,
   SNAME VARCHAR2(10),BCODE INT REFERENCES
   BRANCH1(BCODE) ON DELETE CASCADE); ------ CHILD TABLE

SQL> INSERT INTO STUDENT1 VALUES(1,'SMITH',1021);
SQL> INSERT INTO STUDENT1 VALUES(2,'ALLEN',1021);
SQL> INSERT INTO STUDENT1 VALUES(3,'WARD',1022);
```

2. ON DELETE SET NULL:

- ONCE WE DELETE A ROW FROM THE PARENT TABLE THEN THE CORRESPONDING CHILD TABLE FOREIGN KEY COLUMN VALUES ARE CONVERTING INTO NULLS AUTOMATICALLY.

EX:

```
SQL> CREATE TABLE BRANCH2(BCODE INT PRIMARY KEY,BNAME VARCHAR2(10));---PARENT TABLE

SQL> INSERT INTO BRANCH2 VALUES(1021,'CSE');
SQL> INSERT INTO BRANCH2 VALUES(1022,'EEE');
SQL> CREATE TABLE STUDENT2(STID INT PRIMARY KEY,
  SNAME VARCHAR2(10),BCODE INT REFERENCES
  BRANCH2(BCODE) ON DELETE SET NULL); ------ CHILD TABLE
```

```
SQL> INSERT INTO STUDENT2 VALUES(1,'SMITH',1021);
SQL> INSERT INTO STUDENT2 VALUES(2,'ALLEN',1021);
SQL> INSERT INTO STUDENT2 VALUES(3,'WARD',1022);
```

TESTING:

```
SQL> DELETE FROM BRANCH2 WHERE BCODE=1021;-----ALLOWED
```

DATADICTIONARY / READ ONLY TABLES

- WHEN WE ARE INSTALLING ORACLE S/W INTERNALLY SYSTEM WILL CREATE SOME PRE-DEFINED TABLES THESE TABLES ARE CALLED AS "DATADICTIONARIES".
- DATADICTIONARIES ARE USED TO STORE THE INFORMATION ABOUT THE TABLES,VIEWS,SYNONYMS,SEQUENCES,INDEXES,CLUSTERS,CONSTRAINTS.......etc
- DATADICTIONARIES ARE NOT ALLOWED DML OPERATIONS BUT ALLOWED "SELECT" COMMAND ONLY. SO THAT DATADICTIONARIES ARE CALLED AS "READ ONLY TABLES".
- IF WE WANT TO VIEW ALL DATADICTIONARIES IN ORACLE DB THEN WE FOLLOW THE FOLLOWING,

SYNTAX:

```
SQL> SELECT * FROM DICT; (DICTIONARY)
```

EX:

```
SQL> CREATE TABLE TEST8
  2  (
  3  EID INT PRIMARY KEY,
  4  ENAME VARCHAR2(10) UNIQUE NOT NULL,
  5  SAL NUMBER(10) CHECK(SAL>=15000),
  6  LOC VARCHAR2(10) UNIQUE NOT NULL CHECK(LOC IN('HYD','MUMBAI'))
  7  );
```

NOTE:

- IF WE WANT TO VIEW COLUMN NAME ALONG WITH CONSTRAINT NAME OF A PARTICULAR TABLE IN ORACLE DB THEN WE USE A DATADICTIONARY IS CALLED AS "USER_CONS_COLUMNS".

EX:

```
SQL> DESC USER_CONS_COLUMNS;
SQL> SELECT COLUMN_NAME,CONSTRAINT_NAME FROM USER_CONS_COLUMNS
  2  WHERE TABLE_NAME='TEST8';
```

```
COLUMN_NAME                              CONSTRAINT_NAME
------------------------------------------  ---------------------------------------------------------
ENAME                                    SYS_C007888
LOC                                      SYS_C007889
SAL                                      SYS_C007890
LOC                                      SYS_C007891
EID                                      SYS_C007892
ENAME                                    SYS_C007893
LOC                                      SYS_C007894
```

HOW TO CREATE A USER DEFINED CONSTRAINT NAME:

SYNTAX:

<COLUMN NAME><DT>[SIZE] CONSTRAINT <USER DEFINED CONSTRAINT KEY NAME> <CONSTRAINT TYPE>,.....................

EX:

```
SQL> CREATE TABLE TEST9
  2  (
  3  EID INT CONSTRAINT EID_PK PRIMARY KEY,
  4  ENAME VARCHAR2(10) CONSTRAINT ENAME_UQ UNIQUE,
  5  SAL NUMBER(10) CONSTRAINT SAL_CHK CHECK(SAL>=8000)
  6  );

Table created.

SQL> SELECT COLUMN_NAME,CONSTRAINT_NAME FROM USER_CONS_COLUMNS
  2  WHERE TABLE_NAME='TEST9';

COLUMN_NAME                              CONSTRAINT_NAME
---------------------------------------------------  ---------------------------------------------------------------------------
SAL                                      SAL_CHK
EID                                      EID_PK
ENAME                                    ENAME_UQ
```

HOW TO ADD CONSTRAINT TO AN EXISTING TABLE:

SYNTAX:

ALTER TABLE <TN> ADD CONSTRAINT <CONSTRAINT KEY NAME> <CONSTRAINT TYPE>(COLUMN NAME);

I) ADDING A PRIMARY KEY:

```
SQL> CREATE TABLE PARENT(EID INT,ENAME VARCHAR2(10),SAL NUMBER(10));
Table created.
SQL> ALTER TABLE PARENT ADD CONSTRAINT PK_EID PRIMARY KEY(EID);
```

II) ADDING UNIQUE , CHECK CONSTRAINT:

```
SQL> ALTER TABLE PARENT ADD CONSTRAINT UQ_NAME UNIQUE(ENAME);
SQL> ALTER TABLE PARENT ADD CONSTRAINT CHK_SAL CHECK(SAL>=15000);
```

NOTE:

- TO VIEW A CHECK CONSTRAINT CONDITIONAL VALUE OF A PARTICULAR

TABLE IN ORACLE DB THEN USE A DATADICTIONARY IS "USER_CONSTRAINTS".
EX:

```
SQL> DESC USER_CONSTRAINTS;
SQL> SELECT SEARCH_CONDITION FROM USER_CONSTRAINTS
        WHERE TABLE_NAME='PARENT';

SEARCH_CONDITION
--------------------------------------------------------------------------------
SAL>=15000
```

III) ADDING "NOT NULL" CONSTRAINT:
SYNTAX:
ALTER TABLE <TN> MODIFY <COLUMN NAME> CONSTRAINT <CONSTRAINT KEY NAME> NOT NULL;

EX:

```
SQL> ALTER TABLE PARENT MODIFY SAL CONSTRAINT SAL_NN NOT NULL;
```

IV) ADDING A FOREIGN KEY REFERENCES TO AN EXISTING TABLE:
SYNTAX:

ALTER TABLE <TN> ADD CONSTRAINT <CONSTRAINT KEY NAME>
FOREIGN KEY(COMMON COLUMN OF CHILD TABLE) REFERENCES
<PARENT TABLE NAME>(COMMON NAME OF PARENT TABLE)
ON DELETE CASCADE / ON DELETE SET NULL;

EX:

```
SQL> CREATE TABLE CHILD(DNAME VARCHAR2(10),EID INT);
Table created.

SQL> ALTER TABLE CHILD ADD CONSTRAINT EID_FK
  2  FOREIGN KEY(EID) REFERENCES PARENT(EID)
  3  ON DELETE CASCADE;
```

HOW TO DROP A CONSTRAINT FROM AN EXISTING TABLE:
SYNTAX:
ALTER TABLE <TN> DROP CONSTRAINT <CONSTRAINT KEY NAME>;

I) TO DROP A PRIMARY KEY:
CASE-1: WITHOUT RELATIONSHIP:

```
SQL> ALTER TABLE PARENT DROP CONSTRAINT PK_EID;
```

CASE-2: WITH RELATIONSHIP:

```
SQL> ALTER TABLE PARENT DROP CONSTRAINT PK_EID CASCADE;
```

II) TO DROP UNIQUE,CHECK AND NOT NULL CONSTRAINT:

```
SQL> ALTER TABLE PARENT DROP CONSTRAINT UQ_NAME;
SQL> ALTER TABLE PARENT DROP CONSTRAINT CHK_SAL;
SQL> ALTER TABLE PARENT DROP CONSTRAINT SAL_NN;
```

HOW TO RENAME A CONSTRAINT KEY NAME IN A TABLE:

SYNTAX:

ALTER TABLE <TN> RENAME CONSTRAINT <OLD CONSTRAINT KEY NAME> TO <NEW CONSTRAINT KEY NAME>;

EX:

```
SQL> CREATE TABLE TEST10(SNO INT PRIMARY KEY);

SQL> SELECT COLUMN_NAME,CONSTRAINT_NAME FROM USER_CONS_COLUMNS
  2  WHERE TABLE_NAME='TEST10';

COLUMN_NAME                              CONSTRAINT_NAME
---------------------------------------- ------------------------------------------
SNO                                      SYS_C007903


SQL> ALTER TABLE TEST10 RENAME CONSTRAINT SYS_C007903 TO SNO_PK;
Table altered.

COLUMN_NAME                              CONSTRAINT_NAME
---------------------------------------- ------------------------------------------
SNO                                      SNO_PK
```

HOW TO DISABLE AND ENABLE A CONSTRAINT ON A TABLE:

- BY DEFAULT ALL CONSTRAINTS ARE ENABLE MODE(i.e WORKING).IF WE WANT TO DISABLE CONSTRAINT THEN USE "DISABLE" KEYWORD.

SYNTAX:

ALTER TABLE <TN> DISABLE / ENABLE <CONSTRAINT> <CONSTRAINT KEY NAME>;

EX:

```
SQL> CREATE TABLE TEST11(ENAME VARCHAR2(10),SAL NUMBER(10) CHECK(SAL>=9000));
```

TESTING:

```
SQL> INSERT INTO TEST11 VALUES('SMITH',8000);----NOT ALLOWED
SQL> INSERT INTO TEST11 VALUES('SMITH',9000);----ALLOWED
```

TO DISABLE A CONSTRAINT:

EX:

```
ALTER TABLE TEST11 DISABLE CONSTRAINT SYS_C007904;
```

TESTING:

```
SQL> INSERT INTO TEST11 VALUES('WARD',8000);----ALLOWED
```

TO ENABLE A CONSTRAINT:

EX:

```
SQL> ALTER TABLE TEST11 ENABLE CONSTRAINT SYS_C007904
ERROR at line 1:
ORA-02293: cannot validate (MYDB4PM.SYS_C007904) - check constraint violated.
```

- TO OVERCOME THE ABOVE PROBLEM WE SHOULD USE A KEYWORD IS CALLED AS "NOVALIDATE" AT THE TIME OF ENABLE A CONSTRAINT.

- ONCE WE USE "NOVALIDATE" KEYWORD THEN ORACLE SERVER WILL NOT CHECK AN EXISTING DATA IN A COLUMN BUT CHECKING NEWLY INSERTING DATA.

EX:

SQL> ALTER TABLE TEST11 ENABLE NOVALIDATE CONSTRAINT SYS_C007904;
Table altered.

TESTING:

SQL> INSERT INTO TEST11 VALUES('SCOTT',5000);----NOT ALLOWED
SQL> INSERT INTO TEST11 VALUES('SCOTT',11000);----ALLOWED

DEFAULT CONSTRAINT

- IT IS USED TO ASSIGN A USER DEFINED DEFAULT VALUE TO A COLUMN.

EX:

SQL> CREATE TABLE TEST12(SNAME VARCHAR2(10),LOC VARCHAR2(10) DEFAULT 'HYD');

TESTING:

SQL> INSERT INTO TEST12(SNAME,LOC) VALUES('A','KERALA');
SQL> INSERT INTO TEST12(SNAME) VALUES('B');

HOW TO ASSIGN DEFAULT VALUE TO AN EXISTING COLUMN IN A TABLE:

SYNTAX:

ALTER TABLE <TN> MODIFY <COLUMN NAME> DEFAULT <VALUE>;

EX:

SQL> CREATE TABLE TEST13(SNO INT,SFEE NUMBER(10));
SQL> ALTER TABLE TEST13 MODIFY SFEE DEFAULT 500;

TESTING:

SQL> INSERT INTO TEST13(SNO)VALUES(1);

NOTE:

- TO VIEW A DEFAULT CONSTRAINT VALUE ALONG WITH A COLUMN OF A PARTICULAR TABLE THEN USE A DATADICTIONARY IS "USER_TAB_COLUMNS".

EX:

```
SQL> DESC USER_TAB_COLUMNS;
SQL> SELECT COLUMN_NAME,DATA_DEFAULT FROM USER_TAB_COLUMNS
        WHERE TABLE_NAME='TEST13';

COLUMN_NAME                              DATA_DEFAULT
----------------------------------------  ----------------------------------------
SFEE                                     500
```

HOW TO REMOVE A DEFAULT CONSTRAINT VALUE FROM A COLUMN:

SYNTAX:

ALTER TABLE <TN> MODIFY <COLUMN NAME> DEFAULT NULL;

EX:
```
SQL> ALTER TABLE TEST13 MODIFY SFEE DEFAULT NULL;
```

TESTING:
```
SQL> INSERT INTO TEST13(SNO)VALUES(3);

COLUMN_NAME                              DATA_DEFAULT
---------------------------------------- ----------------------------------------
SFEE                                     NULL
```

==

TRANSACTION CONTROL LANGUAGE(TCL)

==

- TRANSACTIO IS NOTHING BUT TO PERFORM SOME OPERATION OVER DATABASE / TABLE.
  1. COMMIT
  2. ROLLBACK
  3. SAVEPOINT

1. COMMIT:

- TO MAKE A TRANSACTION IS PERMANENT.
  - i) IMPLICIT COMMIT TRANSACTIONS
  - ii) EXPLICIT COMMIT TRANSACTIONS

i) IMPLICIT COMMIT:

- THESE TRAANSACTIONS ARE COMMITTED BY SYSTEM BY DEFAULT.
  EX: DDL COMMANDS.
- IMPLICIT COMMIT TRANSACTIONS ARE CALLED AS "AUTO COMMIT TRANSACTIONS".

ii) EXPLICIT COMMIT:

- THESE TRANSACTIONS ARE COMMITTED BY USER.
  EX: DML COMMANDS.

SYNTAX:
COMMIT;

EX:
```
SQL> CREATE TABLE BRANCH(BCODE INT,BNAME VARCHAR2(10),LOC VARCHAR2(10));

SQL> INSERT INTO BRANCH VALUES(1021,'SBI','HYD');
SQL> COMMIT;

SQL> UPDATE BRANCH SET LOC='PUNE' WHERE BCODE=1021;
SQL> COMMIT;

SQL> DELETE FROM BRANCH WHERE BCODE=1021;
SQL> COMMIT;
        (OR)
SQL> INSERT INTO BRANCH VALUES(1021,'SBI','HYD');
SQL> UPDATE BRANCH SET LOC='PUNE' WHERE BCODE=1021;
SQL> DELETE FROM BRANCH WHERE BCODE=1021;
SQL> COMMIT;
```

ROLLBACK:

- TO CANCEL A TRANSACTION.
- IT IS SIMILAR TO "UNDO" OPERATION IN FILES.
- ONCE WE COMMIT A TRANSACTION THEN WE CANNOT ROLLBACK.

SYNTAX:

ROLLBACK;

EX:

```
SQL> DELETE FROM BRANCH WHERE BCODE=1021;
SQL> ROLLBACK;
```

SAVEPOINT:

- WHEN WE CREATED A SAVEPOINTER INTERNALLY SYSTEM IS ALLOCATING A SPECIAL MEMORY TO A POINTER UNDER RAM.IN THIS POINTER MEMORY WE WILL BE SAVED A SPECIFIC ROW / ROWS WHICH WE WANT TO ROLLBACK IN THE FEATURE.

HOW TO CREATE A SAVEPOINTER:

SYNTAX:

SAVEPOINT <POINTER NAME>;

EX:

SAVEPOINT P1;

HOW TO ROLLBACK A SAVEPOINTER:

SYNTAX:

ROLLBACK TO <POINTER NAME>;

EX:

ROLLBACK TO P1;

EX:

```
SQL> SELECT * FROM BRANCHS;

     BCODE             BNAME        LOC
----------             ----------   ----------
      1021             SBI          HYD
      1022             HDFC         MUMBAI
      1023             BOI          CHENNAI
      1024             AXIS         KERALA
      1025             ICICI        PUNE

SQL> DELETE FROM BRANCHS WHERE BCODE=1021;
SQL> DELETE FROM BRANCHS WHERE BCODE=1025;

SQL> SAVEPOINT P1;
Savepoint created.
SQL> DELETE FROM BRANCHS WHERE BCODE=1023;
```

CASE-1:
ROLLBACK TO P1;-------RESTORE 1023 RECORD ONLY

CASE-2:
COMMIT / ROLLBACK ;

EX:
SQL> DELETE FROM BRANCHS WHERE BCODE=1021;

SQL> SAVEPOINT P1;
Savepoint created.
SQL> DELETE FROM BRANCHS WHERE BCODE IN(1023,1025);

CASE-1:
ROLLBACK TO P1;-------RESTORE 1023,1025 RECORDS ONLY

CASE-2:
COMMIT / ROLLBACK ;

ACID PROPERTIES:

- BY DEFAULT ALL DBMS ARE HAVING "ACID" PROPERTIES TO MANAGE AND MAINTAIN ACCURATE AND CONSISTENCY DATA / INFORMATION IN DATABASES.

A - ATOMICITY
C - CONSISTENCY
I - ISOLATION
D - DURABILITY

A - ATOMICITY:

- ATOMIC = A SINGLE TRANSACTION.
- IN A TRANSACTION,ALL INTERNAL OPERATIONS ARE MAKING AS A SINGLE TRANSACTION AND
THAT TRANSACTION MAY BE SUCCESS OR FAIL.

EX:

```
                              X-CUSTOMER  A/C BAL : 10000
                                          |
WITHDRAW TRANSACTION:                    ATM
...........................................        |
                              > INSERT ATM CARD
                              > SELECT LANGUAGE
                              > SELECT BANKING
                              > CLICK ON WITHDRAW
                              > ENTER AMOUNT : 4000
                              > SELECT SAVE / CURRENT
                              > ENTER PIN NO: XXXX
                              > CLICK ON YES / NO
```

C - CONSISTENCY:

- DATABASE SHOULD MAINTAIN ACCURATE INFORMATION BEFORE / AFTER A TRANSACTION.

EX:

```
                    X-PERSON -------> TRANSFER --------------->          Y-PERSON
                    =========                               =========
          A/C BAL : 10000                  A/C BAL : 4000 -----------------> BEFORETRANSACTION

                      4000(DEBIT)              (CREDIT): 4000
                    ===========              =============
                    6000                     8000 ----------------------------> AFTER TRANSACTION
```

I - ISOLATION:

- EVERY TRANSACTION IS INDEPENDENT.
- IT NEVER INTERFERENCE ONE TRANSACTION IN ANOTHER TRANSACTION.

D - DURABILITY:

- ONCE WE COMMIT A TRANSACTION WE CANNOT ROLLBACK EVEN IF ANY SYSTEM FAILURE, POWER LOSS,ABNORMAL TERIMINATIONS,....etc

MERGE COMMAND:

- IT IS DML COMMAND WHICH IS USED TO TRANSFER DATA FROM SOURCE TABLE TO DESTINATION
TABLE.
- WHEN WE FOUND MATCHING ROWS IN BOTH TABLES THEN THOSE MATCHING ROWS ARE OVERRIDE ON DESTINATION TABLE BY USING "UPDATE" COMMAND WHEREAS IF WE FOUND UNMATCHING
ROWS THEN THOSE UNMATCHING ROWS ARE TRANSFER FROM ONE SOURCE TABLE TO DESTINATION TABLE
BY USING "INSERT" COMMAND.
- MERGE COMMAND IS A COMBINATION OF UPDATE AND INSERT COMMAND.

SYNTAX:

```
MERGE INTO <DESTINATION TABLE NAME> <TABLE ALIAS NAME> USING
<SOURCE TABLE NAME> <TABLE ALIAS NAME> ON (JOIN CONDITION)
WHEN MATCHED THEN
UPDATE SET <DESTINATION TABLE NAME>.<COLUMN NAME1>=<SOURCE TABLE
NAME>.<COLUMN NAME1>,
...............................................................................................................................
..........
WHEN NOT MATCHED THEN
INSERT(DESTINATION TABLE COLUMNS) VALUES(SOURCE TABLE COLUMN);
```

EX:

```
SQL> CREATE TABLE SDEPT AS SELECT * FROM DEPT;

SQL> SELECT * FROM SDEPT;
SQL> INSERT INTO SDEPT VALUES(50,'DBA','HYD');
SQL> INSERT INTO SDEPT VALUES(60,'SAP','MUMBAI');
SQL> COMMIT;
SQL> SELECT * FROM SDEPT;----------SOURCE TABLE
SQL> SELECT * FROM DEPT;-----------DESTINATION TABLE

SQL> MERGE INTO DEPT D USING SDEPT S ON(D.DEPTNO=S.DEPTNO)
  2  WHEN MATCHED THEN
  3  UPDATE SET D.DNAME=S.DNAME,D.LOC=S.LOC
  4  WHEN NOT MATCHED THEN
  5  INSERT(D.DEPTNO,D.DNAME,D.LOC)VALUES(S.DEPTNO,S.DNAME,S.LOC);
```

SUBQUERY

A query inside another query is called as "subquery".

SYNTAX:

SELECT * FROM <TN> WHERE <CONDITION>(SELECT * FROM.....(SELECT * FROM ....));

- A subquery statement is having two more queries those are,
  - I) Outer query / main query / parent query
  - Ii) Inner query / sub query / child query
- As per the execution process of a subquery statement it again classified Into two types those are,
  1. Non co-related subquery
  2. Co-related subquery
- In Non co-related mechanism first inner query is executed and later outer Query will execute whereas in co-related subquery first outer query is executed And later inner query will execute.

1. Non co-related subquery:
   - i) Simple / single row subquery
   - ii) Multiple - row subquery
   - iii) Multiple - column subquery
   - iv) Inline view

I) SIMPLE / SINGLE ROW SUBQUERY:

- when a subquery return a single value is called as "SRSQ".
- in this subquery we can use the following operators are

  " = , < , > , <= , >= , != ".

Ex:

WAQ TO DISPLAY EMPLOYEES DETAILS WHO ARE EARNING THE 1ST HIGHEST SALARY?

Solution:

subquery statement = outer query + inner query

STEP1: INNER QUERY:

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

STEP2: OUTER QUERY:

SELECT * FROM EMP WHERE (CONDITION);

STEP3: SUBQUERY(OUTER+INNER):

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

Ex:

WAQ TO DISPLAY THE SENIOR MOST EMPLOYEE DETAILS FROM EMP TABLE?

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

Ex:

WAQ TO DISPLAY EMPLOYEES DETAILS WHOSE JOB IS SAME AS THE EMPLOYEE "SCOTT" JOB?

```
SQL> SELECT * FROM EMP WHERE JOB=(SELECT JOB FROM EMP
        WHERE ENAME='SCOTT');
```

Ex:

WAQ TO DISPLAY EMPLOYEES DETAILS WHOSE SALARY IS MORE THAN THE MAXIMUM SALARY OF THE JOB IS "SALESMAN"?

```
SQL> SELECT * FROM EMP WHERE SAL>(SELECT MAX(SAL) FROM EMP
  2  WHERE JOB='SALESMAN');
```

Ex:

WAQ TO FIND OUT 2ND HIGHEST SALARY FROM EMP TABLE?

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

Ex:

WAQ TO DISPLAY 2ND HIGHEST SALARY EMPLOYEE DETAILS?

```
SQL> SELECT * FROM EMP WHERE SAL=(SELECT MAX(SAL) FROM EMP
WHERE SAL<(SELECT MAX(SAL)FROM EMP));
```

Ex:

WAQ TO DISPLAY 3RD HIGHEST SALARY EMPLOYEE DETAILS?

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

Nth	N+1
====	===
1ST	2Q
2ND	3Q
3RD	4Q
30TH	31Q
150THE	151Q

HOW TO OVERCOME THE ABOVE PROBLEM?

ii) MULTIPLE ROW SUBQUERY:

- When a subquery returns more than one value is called as MRSQ.
- We will use the following operators are "IN,ANY,ALL".

Ex:
WAQ TO DISPLAY EMPLOYEES WHOSE JOB IS SAME AS THE JOB OF THE EMPLOYEES "SMITH","CLARK" ?

```
SQL> SELECT * FROM EMP WHERE JOB IN(SELECT JOB FROM EMP
  2  WHERE ENAME='SMITH' OR ENAME='CLARK');
                        (OR)
SQL> SELECT * FROM EMP WHERE JOB IN(SELECT JOB FROM EMP
  2  WHERE ENAME IN('SMITH''CLARK'));
```

Ex:
WAQ TO DISPLAY EMPLOYEES DETAILS WHO ARE EARNING MINIMUM,MAXIMUM SALARY FROM EMP TABLE?

```
SQL> SELECT * FROM EMP WHERE SAL IN
  2  (
  3  SELECT MIN(SAL) FROM EMP
  4  UNION
  5  SELECT MAX(SAL) FROM EMP
  6  );
```

Ex:
WAQ TO DISPLAY THE SENIOR MOST EMPLOYEE DETAILS FROM EACH DEPTNO WISE?

```
SQL> SELECT * FROM EMP WHERE HIREDATE IN(SELECT MIN(HIREDATE)
  2  FROM EMP GROUP BY DEPTNO);
```

Ex:
WAQ TO DISPLAY EMPLOYEES WHO ARE EARNING MAXIMUM SALARY FROM EACH JOB WISE?

```
SQL> SELECT * FROM EMP WHERE SAL IN(SELECT MAX(SAL) FROM
       EMP GROUP BY JOB);
```

ANY :

- It returns a value if any one value is satisfied with the given expression from the given list of values.

Ex:

X >ANY(10,20,30)

```
X=40 ---- TRUE
X=09 --- FALSE
X=25 ---- TRUE
```

ALL:

- It returns a value if all values are satisfied with the given expression from the given list of values.

Ex:

X >ALL(10,20,30)

```
X=40 ----TRUE
X=09 --- FALSE
X=25 ----FALSE
```

Ex:
WAQ TO DISPLAY EMPLOYEES WHOSE SALARY IS MORE THAN ANY "SALESMAN" SALARY?
```
SQL> SELECT * FROM EMP WHERE SAL>ANY(SELECT SAL FROM EMP
  2  WHERE JOB='SALESMAN');
```

Ex:
WAQ TO DISPLAY EMPLOYEES WHOSE SALARY IS MORE THAN ALL "SALESMAN" SALARIES?
```
SQL> SELECT * FROM EMP WHERE SAL>ALL(SELECT SAL FROM EMP
         WHERE JOB='SALESMAN');
```

iii) MULTIPLE COLUMN SUBQUERY:

- Multiple columns values of inner query is comparing with multiple columns values of outer query is called as "MCSQ".

SYNTAX:
```
SELECT * FROM <TN> WHERE (<COLUMN NAME1>,<COLUMN NAME2>,.....) IN(SELECT
<COLUMN NAME1>,<COLUMN NAME2>,......FROM <TN>);
```

TESTING:
```
SQL> UPDATE EMP SET SAL=1300 WHERE ENAME='FORD';
```

Ex:
WAQ TO DISPLAY EMPLOYEES WHO ARE EARNING MAXIMUM SALARY FROM EACH JOB WISE?
```
SQL> SELECT * FROM EMP WHERE SAL IN(SELECT MAX(SAL) FROM
       EMP GROUP BY JOB);
```

OUTPUT:
```
JOB             SAL
====            ====
SALESMAN        1600
MANAGER         2975
ANALYST         3000
PRESIDENT       5000
ANALYST         1300
CLERK           1300
```

- When we are comparing multiple values by using multiple row subquery mechanism it returns wrong result.so to overcome the this problem we should use a technique is called as "multiple column subquery".

SOLUTION:
```
SQL> SELECT * FROM EMP WHERE(JOB,SAL) IN(SELECT
  2  JOB,MAX(SAL) FROM EMP GROUP BY JOB);
```

```
JOB             MAX(SAL)
---------       ----------
CLERK           1300
SALESMAN        1600
ANALYST         3000
MANAGER         2975
PRESIDENT       5000
```

Ex:

WAQ TO DISPLAY EMPLOYEES WHOSE JOB,MGR IS SAME AS THE JOB,MGR OF THE EMPLOYEE "MARTIN" ?

```
SQL> SELECT * FROM EMP WHERE(JOB,MGR)IN(SELECT JOB,MGR
  2  FROM EMP WHERE ENAME='MARTIN');
```

PSEUDO COLUMNS:

- These are working just like a table columns.
  - i) ROWID
  - ii) ROWNUM

i) ROWID:

- Whenever we are inserting a new row data into a table internally system will create a unique row identity / row address for each row wise and saved in DB directly.so that rowid's are permanent id's.

Ex:

```
SQL> SELECT ROWID,ENAME FROM EMP;
SQL> SELECT ROWID,ENAME,DEPTNO FROM EMP WHERE DEPTNO=10;
```

EX:

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

MIN(ROWID)
------------------
AAATJtAAHAAAAFeAAA

SQL> SELECT MAX(ROWID) FROM EMP;

MAX(ROWID)
------------------
AAATJtAAHAAAAFeAAN
```

HOW TO DELETE MULTIPLE DUPLICATE ROWS EXCEPT ONE DUPLICATE ROW FROM A TABLE:

EX:

```
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(SELECT
  2  MAX(ROWID) FROM TEST GROUP BY SNO);
```

```
SQL> SELECT * FROM TEST;

       SNO NAME
---------- ----------
         1 A
         2 B
         3 C
         4 D
         5 E
```

ii) ROWNUM:

- To generate row numbers to each row wise (or) to each group of rows wise on a table automatically.these row numbers are not saved in DB so that these numbers are temporary. this column is used to perform "Top N / Nth" operations over a table.

Ex:

```
SQL> SELECT ROWNUM,ENAME FROM EMP;
SQL> SELECT ROWNUM,ENAME,DEPTNO FROM EMP WHERE DEPTNO=10;
```

Ex:

WAQ TO FETCH 1ST ROW FROM EMP TABLE BY USING ROWNUM?

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

Ex:

WAQ TO FETCH 2ND ROW FROM EMP TABLE BY USING ROWNUM?

```
SQL> SELECT * FROM EMP WHERE ROWNUM=2;
no rows selected
```

Note:

- Rownum is always starts with "1" for every selected row from a table. to overcome this problem we should use the following operator are " < , <= ".

SOLUTION:

```
SQL> SELECT * FROM EMP WHERE ROWNUM<=2
  2  MINUS
  3  SELECT * FROM EMP WHERE ROWNUM=1;
```

Ex:

WAQ TO FETCH TOP 5 ROWS FROM EMP TABLE BY USING ROWNUM?

```
SQL> SELECT * FROM EMP WHERE ROWNUM<=5;
```

Ex:

WAQ TO FETCH FROM 5TH ROW TO 10TH ROW FROM EMP TABLE BY USING ROWNUM?

```
SQL>  SELECT * FROM EMP WHERE ROWNUM<=10
  2     MINUS
  3     SELECT * FROM EMP WHERE ROWNUM<5;
```

Ex:
WAQ TO DISPLAY THE LAST TWO ROWS FROM EMP TABLE BY USING ROWNUM?
```
SQL> SELECT * FROM EMP WHERE  ROWNUM<=14
  2  MINUS
  3  SELECT * FROM EMP WHERE ROWNUM<=12;
```

(OR)

```
SQL> SELECT * FROM EMP
  2  MINUS
  3  SELECT * FROM EMP WHERE ROWNUM<=(SELECT COUNT(*)-2 FROM EMP);
```

INLINE VIEW:

- Providing a select query inplace of table name in select statement.

(or)

- Providing a select query in from clause.

SYNTAX:
SELECT * FROM (<SELECT QUERY>);------INLINE VIEW

PURPOSE OF INLINE VIEW:
1. Generally subquery is not allowed "order by" clause so to overcome this problem we should use "inline view" technique.

2. Generally column alias name is not allowed under "where clause" condition so to overcome this problem we should use "inline view".

Ex:
WAQ TO DISPLAY EMPLOYEES WHOSE ANNUAL SALARY IS MORE THAN 25000?
```
SQL> SELECT EMPNO,ENAME,SAL,SAL*12 ANNSAL  FROM EMP
        WHERE ANNSAL>25000;
ERROR at line 2:
ORA-00904: "ANNSAL": invalid identifier.
```

SOLUTION:
```
SQL> SELECT * FROM(SELECT EMPNO,ENAME,SAL,SAL*12 ANNSAL FROM EMP) WHERE
ANNSAL>25000;
```

Ex:
**WAQ TO DISPLAY 5TH ROW EMPLOYEE DETAILS FROM EMP BY USING ROWNUM ALIAS NAME ALONG
WITH INLINE VIEW ?**
```
SQL> SELECT * FROM(SELECT ROWNUM R,EMPNO,ENAME,SAL FROM EMP) WHERE R=5;
```
(OR)
```
SQL> SELECT * FROM(SELECT ROWNUM R,EMP.* FROM EMP) WHERE R=5;
```

Ex:
**WAQ TO FETCH 1ST,3RD,7TH,10TH ROWS FROM EMP TABLE BY USING ROWNUM ALIAS NAME ALONG
WITH INLINE VIEW?**
```
SQL> SELECT * FROM(SELECT ROWNUM R,EMP.* FROM EMP) WHERE R IN(1,3,7,10);
```

Ex:

WAQ TO FETCH THE EVEN POSITION ROWS FROM EMP TABLE BY USING ROWNUM ALIAS NAME ALONG WITH INLINE VIEW?

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

Ex:

WAQ TO FETCH 1ST ROW AND THE LAST ROW FROM EMP TABLE BY USING ROWNUM ALIAS NAME ALONG WITH INLINE VIEW?

```
SQL> SELECT * FROM(SELECT ROWNUM R,EMP.* FROM EMP)
  2  WHERE R=1 OR R=14;
                    (OR)
SQL> SELECT * FROM(SELECT ROWNUM R,EMP.* FROM EMP)
  2  WHERE R=1 OR R=(SELECT COUNT(*) FROM EMP);
                    (OR)
SQL> SELECT * FROM(SELECT ROWNUM R,EMP.* FROM EMP)
  2  WHERE R IN(1,14);
                    (OR)
SQL> SELECT * FROM(SELECT ROWNUM R,EMP.* FROM EMP)
  2  WHERE R IN(1,(SELECT COUNT(*) FROM EMP));
```

SUBQUERY WITH "ORDER BY" CLAUSE:

EX:

WAQ TO FETCH THE FIVE HIGHEST SALARY ROWS FROM EMP TABLE BY USING INLINE VIEW ALONG
WITH ORDER BY CLAUSE?

```
SQL> SELECT * FROM(SELECT * FROM EMP ORDER BY SAL DESC)
  2  WHERE ROWNUM<=5;
```

EX:

WAQ TO FETCH THE 5TH HIGHEST SALARY ROW FROM EMP TABLE BY USING INLINE VIEW ALONG WITH ORDER BY CLAUSE?

```
SQL> SELECT * FROM(SELECT * FROM EMP ORDER BY SAL DESC)
  2  WHERE ROWNUM<=5
  3  MINUS
  4  SELECT * FROM(SELECT * FROM EMP ORDER BY SAL DESC)
  5  WHERE ROWNUM<=4;
```

ANALYTICAL FUNCTIONS:

I) RANK()
II) DENSE_RANK()

- RANK() AND DENSE_RANK() FUNCTIONS ARE USED TO ASSIGN RANKING NUMBERS TO EACH ROW WISE (OR) TO EACH GROUP OF ROWS WISE BUT RANK() WILL SKIP THE NEXT SEQUENCE RANK NUMBER WHEN IT REPEAT DUPLICATE VALUES WHEREAS DENSE_RANK() WILL NOT SKIP THE NEXT SEQUENCE RANK NUMBER EVEN REPEAT DUPLICATE VALUES.

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	43000	7	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

WITHOUT PARTITION BY CLAUSE:

EX:

```
SQL> SELECT ENAME,SAL,RANK() OVER(ORDER BY SAL DESC) FROM EMP;
SQL> SELECT ENAME,SAL,DENSE_RANK()OVER(ORDER BY SAL DESC) FROM EMP;
```

WITH PARTITION BY CLAUSE:

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

EX:

WAQ TO DISPLAY EMPLOYEES WHO ARE EARNING 4TH HIGHEST SALARY FROM EACH JOB WISE BY USING DENSE_RANK() ALONG WITH INLINE VIEW?

```
SQL> SELECT * FROM(SELECT EMPNO,ENAME,JOB,SAL,
  2  DENSE_RANK()OVER(PARTITION BY JOB ORDER BY SAL DESC)
  3  RANKS FROM EMP)WHERE RANKS=4;
```

EX:

WAQ TO DISPLAY THE 3RD SENIOR MOST EMPLOYEE DETAILS FROM EACH DEPTNO WISE BY USING DENSE_RANK() ALONG WITH INLINE VIEW?

```
SQL> SELECT * FROM(SELECT EMPNO,ENAME,DEPTNO,HIREDATE,
  2  DENSE_RANK()OVER(PARTITION BY DEPTNO ORDER BY
  3  HIREDATE) RANKS FROM EMP) WHERE RANKS=3;
```

2) CO-RELATED SUBQUERY:

- In this mechanism first outer query is executed and later inner query will execute.

SYNTAX TO FIND OUT "NTH" HIGH /LOW SALARY:

**SELECT * FROM <TN> <TABLE ALIAS NAME1> WHERE N-1=(SELECT COUNT(DISTINCT <COLUMN NAME> FROM <TN> <TABLE ALIAS NAME2>
WHERE <TABLE ALIAS NAME2>.<COLUMN NAME> < / > <TABLE ALIAS NAME1>.<COLUMN NAME>);**

Here,

< ----------- LOW SALARIES
> ----------- HIGH SALARIES

EX:
WAQ TO FIND OUT THE FIRST HIGHEST SALARY EMPLOYEE DETAILS?

SOLUTION:
DEMO_TABLE:
SQL> SELECT * FROM EMPLOYEE;

```
ENAME          SAL
----------             ----------
SMITH        56000
ALLEN        85000
WARD          15000
JONES        85000
MILLER       67000
SCOTT        32000
```

TO FIND N-1 VALUE:

> N=1
> N-1 ===> 1-1 ===> 0

**SQL> SELECT * FROM EMPLOYEE E1 WHERE 0=(SELECT
    COUNT(DISTINCT SAL) FROM EMPLOYEE E2
    WHERE E2.SAL > E1.SAL);**

EX:
WAQ TO FIND OUT 4TH HIGHEST SALARY EMPLOYEE DETAILS?
SOLUTION:

> N=4 ----> N-1 ----> 4-1 ----> 3

**SQL> SELECT * FROM EMPLOYEE E1 WHERE 3=(SELECT
    COUNT(DISTINCT SAL) FROM EMPLOYEE E2
    WHERE E2.SAL > E1.SAL);**

EX:
WAQ TO FIND OUT THE 1ST LOWEST SALARY EMPLOYEE DETAILS?
SOLUTION:

> N=1 ---> N-1 ----> 1-1 ---> 0

**SQL> SELECT * FROM EMPLOYEE E1 WHERE 0=(SELECT
    COUNT(DISTINCT SAL) FROM EMPLOYEE E2
    WHERE E2.SAL < E1.SAL);**

SYNTAX TO DISPLAY "TOP N" HIGH /LOW SALARIES:
**SELECT * FROM <TN> <TABLE ALIAS NAME1> WHERE N>(SELECT COUNT(DISTINCT
<COLUMN NAME> FROM <TN> <TABLE ALIAS NAME2>
WHERE <TABLE ALIAS NAME2>.<COLUMN NAME> < / > <TABLE ALIAS NAME1>.<COLUMN
NAME>);**

Here,

< ----------- LOW SALARIES
> ----------- HIGH SALARIES

EX:
WAQ TO DISPLAY TOP 3 HIGHEST SALARIES EMPLOYEE DETAILS FROM EMP TABLE?

Solution:
if n = 3 -----> n > ------> 3 >

```
SQL> SELECT * FROM EMPLOYEE E1 WHERE 3>(SELECT
  2  COUNT(DISTINCT SAL) FROM EMPLOYEE E2
  3  WHERE E2.SAL > E1.SAL);
```

EX:
WAQ TO DISPLAY TOP 3 LOWEST SALARIES EMPLOYEE DETAILS FROM EMP TABLE?

```
SQL> SELECT * FROM EMPLOYEE E1 WHERE 3>(SELECT
   COUNT(DISTINCT SAL) FROM EMPLOYEE E2
   WHERE E2.SAL < E1.SAL);
```

NOTE:
> TO FIND OUT "Nth" HIGH / LOW SALARY ---------------> N-1
> TO DISPLAY "TOP n" HIGH / LOW SALARIES ----------> N>

EXISTS OPERATOR:

- IT IS A SPECIAL OPERATOR WHICH IS USED IN CO-RELATED SUBQUERY ONLY.IT IS USED TO CHECK THE REQUIRED ROW(DATA) IS EXISTING IN A TABLE OR NOT.
- IF A ROW IS EXISTS IN A TABLE THEN RETURN "TRUE" OTHERWISE "FALSE".

SYNTAX:

```
WHERE EXISTS(INNER QUERY);
```

EX:
WAQ TO DISPLAY DEPARTMENT DETAILS IN WHICH DEPARTMENT EMPLOYEES ARE WORKING ?

```
SQL> SELECT * FROM DEPT D WHERE EXISTS(SELECT DEPTNO
  2  FROM EMP E WHERE E.DEPTNO=D.DEPTNO);
```

EX:
WAQ TO DISPLAY DEPARTMENT DETAILS IN WHICH DEPARTMENT EMPLOYEES ARE NOT WORKING ?

```
SQL> SELECT * FROM DEPT D WHERE NOT EXISTS(SELECT DEPTNO
     FROM EMP E WHERE E.DEPTNO=D.DEPTNO);
```

SCALAR SUBQUERY:

- PROVIDING SUBQUERIES UNDER "SELECT" CLAUSE IS CALLED AS SCALAR SUBQUERY.

- IN SCALAR SUBQUERY MECHANISM THE RESULT OF EACH SUBQUERY WILL ACT AS A COLUMN IN RESULT SET.

SYNTAX:
SELECT (<SUBQUERY1>),(<SUBQUERY2>),(<SUBQUERY3>),.......... FROM <TN>;

EX:
```
SQL> SELECT (SELECT COUNT(*) FROM DEPT),
  2  (SELECT COUNT(*) FROM EMP) FROM DUAL;

(SELECTCOUNT(*)FROMDEPT)    (SELECTCOUNT(*)FROMEMP)
------------------------                ----------------------
                        4                                     14
```

```
SQL> SELECT (SELECT SUM(SAL) FROM EMP WHERE DEPTNO=10) "10",
  2  (SELECT SUM(SAL) FROM EMP WHERE DEPTNO=20) "20",
  3  (SELECT SUM(SAL) FROM EMP WHERE DEPTNO=30) "30" FROM DUAL;

      10             20                  30
----------         ----------         ----------
     8750          9175                9400
```

DB SECURTIY

- All databases are supporting the following two types of security mechanisms are,
  1. Authentication
  2. Authorization

1. AUTHENTICATION:

- To verify user credentials (username & password) before connect /login into system / database.
- These username & password is created by DBA only.

SYNTAX TO CREATE USERNAME & PASSWORD BY DBA (SYSTEM):
CREATE USER <USER NAME> IDENTIFIED BY <PASSWORD>;

Ex:
CREATE USER U1 IDENTIFIED BY U1;

```
SQL> CONN
Enter user-name: U1/U1
ERROR:
ORA-01045: user U1 lacks CREATE SESSION privilege; logon denied
```

2. AUTHORIZATION:

- To give permissions to users to perform some operations over database.
- These permissions are also giving by DBA only by using "DCL" commands.

I) GRANT:

- Grant permissions to user.

SYNTAX:
GRANT <PRIVILEGE NAME> TO <USER NAME>;

II) REVOKE:

- To cancel permissions of a user.

SYNTAX:

REVOKE <PRIVILEGE NAME> FROM <USER NAME>;

PRIVILEGE:

- It is a right to give permission to users.
  1. System privileges
  2. Object privileges

1. SYSTEM PRIVILEGES:

- These privileges are giving by DBA only.

Ex: connect, create table, unlimited tablespace, create synonym, create view, create materialized view, create sequence, create index,..........etc.

SYNTAX:

GRANT <SYSTEM PRIVILEGE NAME> TO <USER NAME>;

EX:

```
Enter user-name: system/tiger
SQL> GRANT CONNECT TO U1;
SQL> GRANT CREATE TABLE,UNLIMITED TABLESPACE TO U1;

SQL> CONN
Enter user-name: U1/U1
Connected.

SQL> CREATE TABLE T1(SNO INT,NAME VARCHAR2(10));
Grant succeeded.

SQL> INSERT INTO T1 VALUES(1,'A');
SQL> UPDATE T1 SET SNO=1021 WHERE SNO=1;
SQL> DELETE FROM T1 WHERE SNO=1021;
SQL> SELECT * FROM T1;
```

HOW TO CANCEL CONNECT PERMISSION OF A USER:

SYNTAX:

REVOKE <SYSTEM PRIVILEGE NAME> FROM <USERNAME>;

EX:

```
SQL> CONN
Enter user-name: system/tiger
Connected.

SQL> REVOKE CONNECT FROM U1;
Revoke succeeded.

SQL> CONN
Enter user-name: U1/U1
ERROR:
ORA-01045: user U1 lacks CREATE SESSION privilege; logon denied.
```

2) OBJECT PRIVILEGES:

- These privileges are giving by DBA and also USER.

Ex: select, insert, update, delete (or) "all" keyword.

SYNTAX:

GRANT <OBJECT PRIVILEGE NAME> ON <TABLE NAME> TO <USERNAME>;

CASE-1: DBA TO USER:

```
SQL> CONN
Enter user-name: system/tiger
Connected.
SQL> GRANT SELECT,INSERT,UPDATE,DELETE ON DEPT TO U1;

SQL> CONN
Enter user-name: U1/U1
Connected.
SQL> SELECT * FROM SYSTEM.DEPT;
SQL> INSERT INTO SYSTEM.DEPT VALUES(80,'DBA','HYD');
SQL> UPDATE SYSTEM.DEPT SET LOC='PUNE' WHERE DEPTNO=80;
SQL> DELETE FROM SYSTEM.DEPT WHERE DEPTNO=80;
```

HOW TO CANCEL OBJECT PRIVILEGES OF A USER:

```
SQL> CONN
Enter user-name: system/tiger
Connected.

SQL> REVOKE ALL ON DEPT FROM U1;
Revoke succeeded.

SQL> CONN
Enter user-name: U1/U1
Connected.
SQL> SELECT * FROM SYSTEM.DEPT;
ERROR at line 1:
ORA-00942: table or view does not exist
```

CASE-2: USER TO USER:

```
SQL> CONN
Enter user-name: system/tiger
Connected.

SQL> SQL> CREATE USER U2 IDENTIFIED BY U2;
User created.
SQL> GRANT CONNECT TO U2;
SQL> GRANT SELECT ON DEPT TO U1;

SQL> CONN
Enter user-name: U1/U1
Connected.
SQL> SELECT * FROM SYSTEM.DEPT;---ALLOWED

SQL> GRANT SELECT ON SYSTEM.DEPT TO U2
```

```
ERROR at line 1:
ORA-01031: insufficient privileges
```

NOTE:
A USER(U1) WANT TO GIVE OBJECT PRIVILEGES TO ANOTHER USER(U2) FIRST USER(U1) SHOULD PERMISSION FROM DBA WITH "WITH GRANT OPTION" STATEMENT OTHER WISE A USER(U1) UNABLE TO GIVE OBJECT PRIVILEGES TO ANOTHER USER(U2).

```
SQL> CONN
Enter user-name: system/tiger
Connected.

SQL> GRANT SELECT ON DEPT TO U1 WITH GRANT OPTION;
Grant succeeded.

SQL> CONN
Enter user-name: U1/U1
Connected.
SQL> SELECT * FROM SYSTEM.DEPT;-----ALLOWED

SQL> GRANT SELECT ON SYSTEM.DEPT TO U2;
Grant succeeded.

SQL> CONN
Enter user-name: U2/U2
Connected.
SQL> SELECT * FROM SYSTEM.DEPT;-----ALLOWED
```

TO VIEW ALL USERNAMES IN ORACLE DB:
SYNTAX:
SELECT USERNAME FROM ALL_USERS;

HOW TO DROP A USER:
SYNTAX:
DROP USER <USERNAME> CASCADE;

EX:
DROP USER U1 CASCADE;

ROLE:
- It is a collection of privileges which are assigning to group of users at a time who are working on same project.
- These roles are created by DBA only.

STEP1: CREATE A ROLE:
SYNTAX:
CREATE ROLE <ROLE NAME>;

STEP2: TO ASSIGN PRIVILEGES TO ROLE:
SYNTAX:

GRANT <PRIVILEGES NAME> TO <ROLE NAME>;

STEP3: TO ASSIGN A ROLE TO MULTIPLE USERS:
SYNTAX:
GRANT <ROLE NAME> TO <USERS>;

EX:
```
Enter user-name: system/tiger
SQL> CREATE USER U1 IDENTIFIED BY U1;
SQL> CREATE USER U2 IDENTIFIED BY U2;
SQL> CREATE USER U3 IDENTIFIED BY U3;
User created.

SQL> CONN
Enter user-name: U1/U1
ERROR:
ORA-01045: user U1 lacks CREATE SESSION privilege; logon denied

SQL> CONN
Enter user-name: U2/U2
ERROR:
ORA-01045: user U2 lacks CREATE SESSION privilege; logon denied

SQL> CONN
Enter user-name: U3/U3
ERROR:
ORA-01045: user U3 lacks CREATE SESSION privilege; logon denied

SQL> CONN
Enter user-name: system/tiger
Connected.

SQL> CREATE ROLE R1;
Role created.

SQL> GRANT CONNECT,CREATE TABLE TO R1;
Grant succeeded.

SQL> GRANT R1 TO U1,U2,U3;
Grant succeeded.

SQL> CONN
Enter user-name: U1/U1
Connected.
SQL> CREATE TABLE T1(SNO INT);

SQL> CONN
Enter user-name: U2/U2
Connected.
SQL> CREATE TABLE T1(SNO INT);
SQL> CONN
Enter user-name: U3/U3
Connected.
SQL> CREATE TABLE T1(SNO INT);
```

HOW TO DROP A ROLE:

SYNTAX:

DROP ROLE <ROLE NAME>;

EX:

DROP ROLE R1;

SQL:

```
        > DDL
        > DML            DEVELOPERS
        > DQL / DRL
        > TCL
        ==============================
        > DCL            DBA ONLY
        ==============================
```

SYNONYM

- It is a db object which is used to create a permanent alias name for tables.

PURPOSE OF SYNONYM:

1. To reduced lengthy table name:
2. To hide owner name & table name.

TYPES OF SYNONYMS:

1. Private synonyms
2. Public synonyms

1. PRIVATE SYNONYMS:

- These synonyms are created by user who are having permission.

SYNTAX:

CREATE SYNONYM <SYNONYM NAME> FOR <TABLE NAME>;

Ex:

```
SQL> conn
Enter user-name: system/tiger
Connected.

SQL> CREATE USER U50 IDENTIFIED BY U50;
User created.

SQL> GRANT CONNECT,CREATE TABLE,UNLIMITED TABLESPACE TO U50;
Grant succeeded.

SQL> CONN
Enter user-name: U50/U50
Connected.
```

```
SQL> CREATE TABLE COLLEGE_ENROLLMENT_DETAILS
  2  (STID INT,SNAME VARCHAR2(10),BRANCH VARCHAR2(10));
Table created.

SQL> INSERT INTO COLLEGE_ENROLLMENT_DETAILS
  2  VALUES(1021,'SMITH','CSE');

SQL> CREATE SYNONYM CED FOR COLLEGE_ENROLLMENT_DETAILS
ERROR at line 1:
ORA-01031: insufficient privileges

SQL> CONN
Enter user-name: system/tiger
Connected.

SQL> GRANT CREATE SYNONYM TO U50;
Grant succeeded.

SQL> CONN
Enter user-name: U50/U50
Connected.

SQL> CREATE SYNONYM CED FOR COLLEGE_ENROLLMENT_DETAILS;
Synonym created.

SQL> SELECT * FROM CED;
```

NOTE:

- To view private synonyms along with table name in oracle db then use a data dictionary is called as "user_synonyms".

EX:

```
SQL> DESC USER_SYNONYMS;
SQL> SELECT TABLE_OWNER,TABLE_NAME,SYNONYM_NAME
  2  FROM USER_SYNONYMS;

TABLE_OWNER         TABLE_NAME                     SYNONYM_NAME
----------------    ----------------               ----------------
U50                 COLLEGE_ENROLLMENT_DETAILS     CED
```

HOW TO DROP A PRIVATE SYNONYM:

SYNTAX:

DROP SYNONYM <SYNONYM NAME>;

EX:

DROP SYNONYM CED;

2) PUBLIC SYNONYMS:

- These synonyms are created by DBA only

SYNTAX:

CREATE PUBLIC SYNONYM <SYNONYM NAME> FOR [OWNER NAME].<TABLE NAME>;

EX:

```
SQL> conn
Enter user-name: system/tiger
Connected.

SQL> CREATE PUBLIC SYNONYM PS1 FOR SYSTEM.DEPT;
Synonym created.

SQL> GRANT SELECT ON PS1 TO U50;
Grant succeeded.

SQL> CONN
Enter user-name: U50/U50
Connected.
SQL> SELECT * FROM PS1;---ALLOWED
```

NOTE:

- To view all public synonyms in oracle db then use a datadictionary is "all_synonyms".

EX:

```
SQL> DESC ALL_SYNONYMS;
SQL> SELECT OWNER,TABLE_OWNER,TABLE_NAME,
  2  SYNONYM_NAME FROM ALL_SYNONYMS WHERE TABLE_NAME='DEPT';
```

HOW TO DROP PUBLIC SYNONYMS:

SYNTAX:

DROP PUBLIC SYNONYM <SYNONYM NAME>;

EX:

DROP PUBLIC SYNONYM PS1;

VIEWS

- IT IS A VIRTUAL (OR) LOGICAL TABLE OF A BASE TABLE.
- VIEW DOES NOT STORE DATA BUT IT CAN EXTRACT THE REQUIRED DATA FROM A BASE TABLE AND DISPLAY TO USER.
- WHENEVER WE PERFORM DML OPERATIONS ON A VIEW INTERNALLY THOSE OPERATIONS ARE EXECUTED ON BASE TABLE AND REFELECTED INTO VIEW TABLE.
- VIEW WILL ACT AS AN INTERFACE BETWEEN USER AND DATABASE TABLES.

PURPOSE OF VIEWS:

1. SECURITY:
   > COLUMN LEVEL SECURITY
   > ROW LEVEL SECURITY
2. CHECKING INTEGRITY RULES
3. CONVERT COMPLEX QUERY INTO SIMPLE QUERY.
4. VIEW CAN BE STORED SELECT QUERY.

TYPES OF VIEWS:

I) SIMPLE VIEWS
II) COMPLEX VIEWS
III) FORCE VIEWS

I) SIMPLE VIEWS:

- WHEN WE CREATED A VIEW TO ACCESS THE REQUIRED DATA FROM A SINGLE BASE TABLE IS CALLED AS "SMIPLE VIEW".
- SUPPORITNG DML OPERATIONS THROUGH A VIEW ON BASE TABLE BY DEFAULT.

SYNTAX:
CREATE VIEW <VIEW NAME> AS <SELECT * FROM <TABLE NAME> [WHERE <CONDITION>]>;

EX:
CREATE A VIEW TO ACCESS THE DATA FROM DEPT TABLE?
SQL> CREATE VIEW V1 AS SELECT * FROM DEPT;

TESTING:
INSERT INTO V1 VALUES(50,'DBA','HYD');
UPDATE V1 SET LOC='PUNE' WHERE DEPTNO=50;
DELETE FROM V1 WHERE DEPTNO=50;
SELECT * FROM V1;

EX:
CREATE A VIEW TO ACCESS EMPNO,ENAME,SAL DETAILS FROM EMP TABLE?
SQL> CREATE VIEW V2 AS SELECT EMPNO,ENAME,SAL FROM EMP;

TESTING:
INSERT INTO V2 VALUES(1122,'YUVIN',8000);

EX:
CREATE A VIEW TO ACCESS EMPLOYEES DETAILS WHO ARE WORKING UNDER DEPTNO=20?
SQL> CREATE VIEW V3 AS SELECT * FROM EMP WHERE DEPTNO=20;

- **VIEW OPTIONS:**
  I) WITH CHECK OPTION
  II) WITH READ ONLY

I) WITH CHECK OPTION:

- WHEN WE WANT TO RESTRICTED DATA ON A BASE TABLE THROUGH A VIEW THEN WE USE "WITH CHECK OPTION" STATEMENT.

EX:
CREATE A VIEW TO DISPLAY AND ALSO ACCEPT EMPLOYEE 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(1123,'YUVIN','HR',7566,'15-MAR-23',2500,NULL,10);--NOT ALLOWED

SQL> INSERT INTO V5 VALUES(1123,'YUVIN','HR',7566,'15-MAR-23',5500,NULL,10);---NOT ALLOWED
SQL> INSERT INTO V5 VALUES(1123,'YUVIN','HR',7566,'15-MAR-23',3000,NULL,10);---ALLOWED

II) WITH READ ONLY:

- WHEN WE WANT TO RESTRICTED DML OPERATIONS ON A BASE TABLE THROUGH A VIEW OBJECT THEN WE USE "WITH READ ONLY" STATEMENT.
- READ ONLY VIEWS ARE SUPPORTING "SELECT" COMMAND BUT NOT ALLOWED DML OPERATIONS.

EX:

```
SQL> CREATE VIEW V6 AS SELECT * FROM DEPT WITH READ ONLY;
View created.
```

TESTING:

```
SQL> SELECT * FROM V6;----ALLOWED
SQL> INSERT INTO V6 VALUES(50,'SAP','HYD');--------------NOT ALLOWED
SQL> UPDATE V6 SET LOC='PUNE' WHERE DEPTNO=30;--NOT ALLOWED
SQL> DELETE FROM V6 WHERE DEPTNO=10;------NOT ALLOWED
```

2) COMPLEX VIEWS:

- WHEN WE CREATED A VIEW BASED ON:
  - > MULTIPLE BASE TABLES
  - > BY USING GROUP BY
  - > BY USING HAVING
  - > BY USING AGGREGATIVE / GROUPING FUNCTIONS
  - > BY USING SET OPERATORS
  - > BY USING DISTINCT KEYWORD
  - > BY USING JOINS
  - > BY USING SUBQUERY
- BY DEFAULT COMPLEX VIEWS ARE NOT ALLOWED DML OPERATIONS.THESE ARE READ ONLY VIEWS.

EX1:

```
SQL> CREATE VIEW V7 AS
  2  SELECT * FROM EMP_HYD
  3  UNION
  4  SELECT * FROM EMP_CHENNAI;
ERROR: DML OPERATIONS ARE NOT ALLOWED.
```

EX2:

```
SQL> CREATE VIEW V8 AS
  2  SELECT DEPTNO,SUM(SAL)
  3  FROM EMP GROUP BY DEPTNO;
ERROR at line 2:
ORA-00998: must name this expression with a column alias
```

SOLUTION:

```
SQL> CREATE VIEW V8 AS
  2  SELECT DEPTNO,SUM(SAL) TOTAL_SALARY
  3  FROM EMP GROUP BY DEPTNO;
View created.
```

TESTING:
SELECT * FROM V8;(BEFORE UPDATE)
UPDATE EMP SET SAL=SAL+1000 WHERE DEPTNO=10;
SELET * FROM V8;(AFTER UPDATE)

3) FORCE VIEW:
- GENERALLY VIEWS ARE CREATED BASED ON BASE TABLES BUT FORCE VIEWS ARE CREATED WITHOUT BASE TABLES.

SYNTAX:
CREATE FORCE VIEW <VIEW NAME> AS <SELECT QUERY>;

EX:
SQL> CREATE FORCE VIEW FV1 AS SELECT * FROM TEST100;
Warning: View created with compilation errors.

TESTING:
SELECT * FROM FV1 ; ----- NOT WORKING
ERROR at line 1:
ORA-04063: view "MYDB4PM.FV1" has errors

SOLUTION:
SQL> CREATE TABLE TEST100(SNO INT,NAME VARCHAR2(10));
Table created.

SQL> INSERT INTO TEST100 VALUES(1,'A');
1 row created.

SQL> SELECT * FROM FV1; ----- WORKING

NOTE:
- TO SEE ALL VIEWS IN ORACLE DB THEN USE A DATADICTIONARY IS "USER_VIEWS".

EX:
SQL> DESC USER_VIEWS;
SQL> SELECT VIEW_NAME FROM USER_VIEWS;

- TO VIEW TEXT(SELECT QUERY) OF A PARTICULAR VIEW IN ORALE DB THEN FOLLOW THE FOLLOWING SYNTAX,

EX:
SELECT TEXT FROM USER_VIEWS WHERE VIEW_NAME='V1';

HOW TO DROP A VIEW:
SYNTAX:
DROP VIEW <VIEW NAME>;

EX:
DROP VIEW V1;

MATERIALIZED VIEWS

- MVIEWS ARE CREATED BASED ON BASE TABLE JUST LIKE VIEWS.

VIEWS =======	MVIEWS ========
1. VIRTUAL / LOGICAL TABLE OF A BASE TABLE.	1. IT IS A TABLE OF A TABLE.
2. IT DOESNOT STORE DATA.	2. IT WILL STORE DATA.
3. WE CANNOT ACCESS VIEW WITHOUT BASE TABLE.	3. WE CAN ACCESS MVIEW WITHOUT BASE TABLE.
4. IT IS A DEPENDENT OBJECT.	4. IT IS AN INDEPENDENT OBJECT.
5. VIEWS ARE SUPPORTING DML OPERATIONS.	5. MVIEWS ARE NOT ALLOWED DML OPERATIONS.

SYNTAX:
CREATE MATERIALIZED VIEW <VIEW NAME> AS <SELECT QUERY>;

EX:
```
SQL> CREATE TABLE TEST91(SNO INT,NAME VARCHAR2(10));
Table created.

SQL> CREATE VIEW V91 AS SELECT * FROM TEST91;
View created.

SQL> CREATE MATERIALIZED VIEW MV91 AS SELECT * FROM TEST91;
Materialized view created.
```

TESTING:
INSERT INTO TEST91 VALUES(101,'SMITH');

- WHEN WE INSERT DATA INTO A BASE TABLE(TEST91) THAT DATA CAN SEE IN VIEW(V91) BUT NOT IN MVIEW(MV91).IF WE WANT TO SEE DATA IN MVIEW(91) THEN WE SHOULD FOLLOW THE FOLLOWING REFRESHING METHODS ARE,

I) ON DEMAND
II) ON COMMIT

<u>I) ON DEMAND:</u>

- IT IS A DEFAULT REFRESHING METHOD OF MVIEW.

SYNTAX:
EXECUTE DBMS_MVIEW.REFRESH('<MVIEW NAME>');

EX:
```
SQL> EXECUTE DBMS_MVIEW.REFRESH('MV91');
PL/SQL procedure successfully completed.
SQL> SELECT * FROM MV91;
```

II) ON COMMIT:

- WHEN WE PERFORM DML OPERATIONS ON A BASE TABLE THOSE OPERATIONAL DATA WANT TO VIEW IN MVIEW THEN WE SHOULD USE "COMMIT" COMMAND TO COMMIT A DML OPERATION.

SYNTAX:

```
CREATE MATERIALIZED VIEW  <VIEW NAME>
REFRESH ON COMMIT
AS
SELECT QUERY;
```

EX:

```
SQL> CREATE TABLE TEST92(EID INT,ENAME VARCHAR2(10));
Table created.

SQL> CREATE VIEW V92 AS SELECT * FROM TEST92;
View created.

SQL> CREATE MATERIALIZED VIEW MV92
  2  REFRESH ON COMMIT
  3  AS
  4  SELECT * FROM TEST92;
ERROR at line 4:
ORA-12054: cannot set the ON COMMIT refresh attribute for the materialized view
```

- TO OVERCOME THE ABOVE PROBLEM WE MUST HAVE A PRIMARY KEY CONSTRAINT ON BASE TABLE(TEST92).

ADDING PRIMARY KEY CONSTRAINT:

```
SQL> ALTER TABLE TEST92 ADD CONSTRAINT PK_EID PRIMARY KEY(EID);
Table altered.

SQL> CREATE MATERIALIZED VIEW MV92
  2  REFRESH ON COMMIT
  3  AS
  4  SELECT * FROM TEST92;
Materialized view created.
```

TESTING:

```
SQL> INSERT INTO TEST92 VALUES(1,'A');
SQL> COMMIT;
SQL> SELECT * FROM MV92;
```

NOTE:

- TO VIEW ALL MVIEWS IN ORACLE DB THEN USE A DATADICTIONARY IS "USER_MVIEWS".

EX:

```
SQL> DESC USER_MVIEWS;
SQL> SELECT  MVIEW_NAME FROM USER_MVIEWS;
```

HOW TO DROP MATERIALIZED VIEW:

SYNTAX:

DROP MATERIALIZED VIEW <VIEW NAME>;

EX:

DROP MATERIALIZED VIEW MV91;

SEQUENCE

- IT IS A DB OBJECT WHICH IS USED TO GENERATE SEQUENCE NUMBERS ON A PARTICULAR COLUMN IN A TABLE AUTOMATICALLY.
- SEQUENCE OBJECT WILL PROVIDE "AUTO INCREMENTAL VALUES" FACILITY ON A TABLE.

SYNTAX:

```
CREATE SEQUENCE <SEQUENCE NAME>
[START WITH n]
[MINVALUE n]
[INCREMENT BY n]
[MAXVALUE n]
[NO CYCLE / CYCLE]
[NO CACHE / CACHE n];
```

START WITH n:

- IT SPECIFY THE STARTING VALUE OF THE SEQUENCE.

HERE "n" - NUMBER.

MINVALUE n:

- IT SHOWS THE MINIMUM VALUE IN THE SEQUENCE.

HERE "n" - NUMBER.

INCREMENT BY n:

- IT SPECIFY THE INCREMENTAL VALUE IN BETWEEN SEQUENCE NUMBERS.

HERE "n" - NUMBER.

MAXVALUE n:

- IT SHOWS THE MAXIMUM VALUE IN THE SEQUENCE.

HERE "n" - NUMBER.

EX:

```
        START WITH 1
        MINVALUE 1
        INCREMENT BY 1
        MAXVALUE 3;
```

OUTPUT:

```
1
2
3
```

NO CYCLE:

- IT IS DEFAULT PARAMETER OF SEQUENCE.
- WHEN WE CREATED A SEQUENCE OBJECT WITH "NO CYCLE" THEN THE SET

SEQUENCE NUMBERS ARE NOT REPEATED AGAIN AND AGAIN.

CYCLE:

- WHEN WE CREATED A SEQUENCE OBJECT WITH "CYCLE" THEN THE SET SEQUENCE NUMBERS ARE REPEATED AGAIN AND AGAIN.

EX:

```
START WITH 1
MINVALUE 1
INCREMENT BY 1
MAXVALUE 3;
CYCLE;
```

OUTPUT:

```
1
2
3
1
2
3
1
2
3
1
2
3
```

NO CACHE:

- IT IS A DEFAULT PARAMETER OF A SEQUENCE.
- WHEN WE CREATED A SEQUENCE OBJECT WITH "NO CACHE" THEN THE SET OF SEQUENCE NUMBERS ARE SAVED IN DB DIRECTLY.SO THAT WHENEVER A USER WANT TO ACCESS DATA FROM A TABLE BASED ON SEQUENCE NUMBERS THEN EACH AND EVERY REQUEST WILL GO TO DB SERVER SO THAT THE NO.OF REQUESTS WILL INCRESE THE BURDON ON DATABASE AND REDUCE THE PERFORMANCE OF DATABASE.

CACHE n:

- WHEN WE CREATED A SEQUENCE OBJECT WITH "CACHE" THEN THE SET OF SEQUENCE NUMBERS ARE SAVED IN DB AND ALSO COPY OF SEQUENCE DATA IS SAVED IN CACHE MEMORY.SO THAT WHENEVER A USER WANT TO ACCESS DATA FROM A TABLE BASED ON SEQUENCE NUMBERS THEN EACH AND EVERY REQUEST WILL GO TO CACHE MEMORY INSTEAD OF DB MEMORY SO THAT THE NO.OF REQUESTS WILL REDUCE TO DATABASE AND IMPROVE THE PERFORMANCE OF DATABASE.
- HERE "n" IS REPRESENT THE SIZE OF CACHE MEMORY, THE MINIMUM SIZE OF CACHE IS 2KB AND MAXIMUM SIZE IS 20 KB.

NOTE:

- WHEN WE WORK ON SEQUENCE OBJECTS WE NEED THE FOLLOWING TWO TYPES OF PSEUDO COLUMNS ARE,

I) NEXTVAL: TO GENERATE NEXT BY NEXT SEQUENCE NUMBER.

II) CURRVAL: IT SHOWS THE CURRENT SEQUENCE NUMBER.

EX:

```
1
2
3
4
5 ------- NEXTVAL: 6, CURRVAL: 5
```

EX1:

```
SQL> CREATE SEQUENCE SQ1
  2  START WITH 1
  3  MINVALUE 1
  4  INCREMENT BY 1
  5  MAXVALUE 3;
Sequence created.
```

TESTING:

```
SQL> CREATE TABLE TEST21(SNO INT,NAME VARCHAR2(10));
Table created.

SQL> INSERT INTO TEST21 VALUES(SQ1.NEXTVAL,'&NAME');
Enter value for name: A
/
Enter value for name: B
/
Enter value for name: C
/
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 <SEQEUCNE NAME> <PARAMETER NAME> n;

EX:

```
SQL> ALTER SEQUENCE SQ1 MAXVALUE 5;
Sequence altered.

SQL> INSERT INTO TEST21 VALUES(SQ1.NEXTVAL,'&NAME');
Enter value for name: D
/
Enter value for name: E
```

EX2:

```
SQL> CREATE SEQUENCE SQ2
  2  START WITH 1
  3  MINVALUE 1
  4  INCREMENT BY 1
  5  MAXVALUE 3
  6  CYCLE
  7  CACHE 2;
Sequence created.
```

TESTING:

```
SQL> CREATE TABLE TEST22(SNO INT,NAME VARCHAR2(10));
Table created.

SQL> INSERT INTO TEST22 VALUES(SQ2.NEXTVAL,'&NAME');
Enter value for name: A
/
..................................
/
..................................
```

EX3:

```
SQL> CREATE SEQUENCE SQ3
  2  START WITH 3
  3  MINVALUE 1
  4  INCREMENT BY 1
  5  MAXVALUE 5
  6  CYCLE
  7  CACHE 2;
Sequence created.
```

TESTING:

```
SQL> CREATE TABLE TEST23(SNO INT,NAME VARCHAR2(10));
Table created.

SQL> INSERT INTO TEST23 VALUES(SQ3.NEXTVAL,'&NAME');
Enter value for name: Q
/
......................................
```

NOTE:

- TO VIEW ALL SEQUENCE OBJECTS IN ORACLE DB THEN USE A DATADICTIONARY IS "USER_SEQUENCES".

EX:

```
SQL> DESC USER_SEQUENCES;
SQL> SELECT SEQUENCE_NAME FROM USER_SEQUENCES;
```

HOW TO DROP A SEQUENCE OBJECT:

SYNTAX:

DROP SEQUENCE <SEQUENCE NAME>;

EX:

```
DROP SEQUENCE SQ1;
```

LOCKS

- ALL DATABASES ARE MAINTAINING THE FOLLOWING TWO TYPES OF LOCKING MECHANISMS ARE,

1. ROW LEVEL LOCKING
2. TABLE LEVEL LOCING

1. ROW LEVEL LOCKING:

I) SINLGE ROW LOCKING
II) MULTIPLE ROWS LOCKING

I) SINLGE ROW LOCKING:

- IN THIS LOCKING DATABASE CAN LOCK A SINGLE ROW IN A TABLE.

```
        USER-1                              USER-2
        =======                             =======
SQL> CONN SYSTEM/TIGER                SQL> CONN MYDB4PM/MYDB4PM

SQL> UPDATE MYDB4PM.EMP SET           SQL> UPDATE EMP SET SAL=2200
SAL=1100 WHERE EMPNO=7369;            WHERE EMPNO=7369;
[ ROW IS LOCKED ]                     [ WE CANNOT PERFORM UPDATE ]

SQL> COMMIT / ROLLBACK;               1 row updated
[ FOR RELEASING LOCK ]
```

II) MULTIPLE ROWS LOCKING:

- When we want to lock multiple rows in a table by database then we should use"for update" clause in select statement.

```
        USER-1                              USER-2
        =======                             =======
SQL> CONN SYSTEM/TIGER                SQL> CONN MYDB4PM/MYDB4PM

SQL> SELECT * FROM MYDB4PM.EMP        SQL> UPDATE EMP SET SAL=3300
WHERE DEPTNO=10 FOR UPDATE;           WHERE DEPTNO=10;
[ DEPTNO 10 ROWS ARE LOCKED ]         [ WE CANNOT PERFORM UPDATE ]

SQL> COMMIT / ROLLBACK;               1 row updated
[ FOR RELEASING LOCK ]
```

DEAD LOCK:

- BOTH USERS ARE WAITING FOR THEIR RESOURCES.

EX:

```
        USER-1                              USER-2
        =======                             =======
SQL> CONN SYSTEM/TIGER                SQL> CONN MYDB4PM/MYDB4PM

SQL> UPDATE MYDB4PM.EMP SET           SQL> UPDATE EMP SET SAL=5500
SAL=4400 WHERE EMPNO=7788;            WHERE EMPNO=7900;
[ ROW WAS LOCKED ]                    [ ROW WAS LOCKED ]

SQL> UPDATE MYDB4PM.EMP               SQL> UPDATE EMP SET SAL=7700
SET SAL=6600 WHERE EMPNO=7900;        WHERE EMPNO=7788;
[ WE CANNOT PERFORM UPDATE ]          [ WE CANNOT PERFORM UPDATE ]
```

ERROR at line 1:
ORA-00060: deadlock detected while waiting for resource

SQL> COMMIT / ROLLBACK;
[FOR RELEASING LOCK]

2. TABLE LEVEL LOCKING:

- THE ENTIRE TABLE(ALL ROWS) CAN LOCK.
    - I) SHARE LOCK
    - II) EXCLUSIVE LOCK

I) SHARE LOCK:

- HERE MULTIPLE USERS CAN LOCK A TABLE.

SYNTAX:
LOCK TABLE <TABLE NAME> IN SHARE MODE;

EX:

```
         USER-1                                 USER-2
         =======                                =======
SQL> CONN SYSTEM/TIGER                 SQL> CONN MYDB4PM/MYDB4PM

SQL> LOCK TABLE MYDB4PM.EMP            SQL> LOCK TABLE EMP IN SHARE MODE.
IN SHARE MODE;
Table(s) Locked.                                 Table(s) Locked.

SQL> COMMIT / ROLLBACK;                SQL> COMMIT / ROLLBACK;
[ FOR RELEASING LOCK ]                 [ FOR RELEASING LOCK ]
```

II) EXCLUSIVE LOCK:

- HERE ANY ONE USER CAN LOCK A TABLE.

SYNTAX:
LOCK TABLE <TABLE NAME> IN EXCLUSIVE MODE;

EX:

```
         USER-1                                 USER-2
         =======                                =======
SQL> CONN SYSTEM/TIGER                 SQL> CONN MYDB4PM/MYDB4PM

SQL> LOCK TABLE MYDB4PM.EMP            SQL> LOCK TABLE EMP IN EXCLUSIVE MODE.
IN EXCLUSIVE MODE;                     [ WE CANNOT LOCK A TABLE ]
Table(s) Locked.

SQL> COMMIT / ROLLBACK;                Table(s) Locked.
[ FOR RELEASING LOCK ]

SQL> COMMIT / ROLLBACK;                1 row updated
[ FOR RELEASING LOCK ]
```

PARTITION TABLE

- Generally partitions are created on large scale data tables for dividing into small units and each unit is called as a "partition".
- Oracle supporting the following 3 types of partitions are,
  1. Range partition
  2. List partition
  3. Hash partition

1. RANGE PARTITION:

- Created a partition table based on a particular range value.

SYNTAX:

```
CREATE TABLE <TN>(<COLUMN NAME1> <DATATYPE>[SIZE],.........................................)
PARTITION BY RANGE(KEY COLUMN NAME)
(
PARTITION <PARTITION NAME1> VALUES LESS THAN(VALUE),
PARTITION <PARTITION NAME2> VALUES LESS THAN(VALUE),
.........................................................................,
);
```

Ex:

```
SQL> CREATE TABLE TEST1(EID INT,ENAME VARCHAR2(10),
  2  SAL NUMBER(10)) PARTITION BY RANGE(SAL)
  3  (PARTITION P1 VALUES LESS THAN(500),
  4  PARTITION P2 VALUES LESS THAN(2000),
  5  PARTITION P3 VALUES LESS THAN(3000)
  6  );
Table created.
```

TESTING:

```
SQL> INSERT INTO TEST1 VALUES(1,'SMITH',2300);
SQL> INSERT INTO TEST1 VALUES(2,'ALLEN',350);
.........................................................................;
```

HOW TO CALL A PARTICULAR PARTITION:

SYNTAX:

```
SELECT * FROM <TN> PARTITION(PARTITION NAME);
```

EX:

```
SELECT * FROM TEST1 PARTITION(P1);
```

2) LIST PARTITION:

- Created a partition table based on the list of values.

SYNTAX:

```
CREATE TABLE <TN>(<COLUMN NAME1> <DATATYPE>[SIZE],....................................)
PARTITION BY LIST(KEY COLUMN NAME)
(
PARTITION <PARTITION NAME1> VALUES (V1,V2,............),
PARTITION <PARTITION NAME2> VALUES (V1,V2,............),
........................................................................,
...........................................................................,
partition others values(default)
);
```

Ex:

```
SQL> CREATE TABLE COURSES(CID INT,CNAME VARCHAR2(10))
  2  PARTITION BY LIST(CNAME)
  3  (PARTITION P1 VALUES('C','C++'),
  4  PARTITION P2 VALUES('ORACLE','MYSQL','MSSQL'),
  5  PARTITION OTHERS VALUES(DEFAULT)
  6  );
Table created.
```

TESTING:

```
SQL> INSERT INTO COURSES VALUES(1,'ORACLE');
SQL> INSERT INTO COURSES VALUES(2,'C');
SQL> INSERT INTO COURSES VALUES(3,'JAVA');
```

CALLING A PARTICULAR PARTITION:

```
SELECT * FROM COURSES PARTITIONS(P2);
```

3) HASH PARTITION:

- Created a partition table by system automatically as per user request.

SYNTAX:

```
CREATE TABLE <TN>(<COLUMN NAME1> <DATATYPE>[SZIE],.................)
PARTITION BY HASH(KEY COLUMN NAME) PARTITIONS <NUMBER>;
```

EX:

```
SQL> CREATE TABLE TEST2(ENAME VARCHAR2(10),SAL NUMBER(10))
  2  PARTITION BY HASH(SAL) PARTITIONS 5;
```

NOTE:

- IF WE WANT TO VIEW ALL PARTITIONS OF A TABLE IN ORACLE DB THEN USE A DATADICTIONARY IS "USER_TAB_PARTITIONS".

EX:

```
SQL> DESC USER_TAB_PARTITIONS;
SQL> SELECT PARTITION_NAME FROM USER_TAB_PARTITIONS
  2  WHERE TABLE_NAME='TEST2';
```

NOTE:

- TO KNOW A TABLE IS PARTITIONED OR NOT THEN USE A DATADICTIONARY IS "USER_TABLES".

EX:

```
SQL> DESC USER_TABLES;
SQL> SELECT PARTITIONED FROM USER_TABLES WHERE TABLE_NAME='DEPT';
```

```
PAR
---
NO
```

SQL> SELECT PARTITIONED FROM USER_TABLES WHERE TABLE_NAME='TEST1';

```
PAR
---
YES
```

HOW TO ADD A NEW PARTITION TO AN EXISTING TABLE:
SYNTAX:
ALTER TABLE <TN> ADD PARTITION <PARTITION NAME> VALUES LESS THAN(VALUE);

EX:
SQL> ALTER TABLE TEST1 ADD PARTITION P4 VALUES LESS THAN(4000);

HOW TO DROP A PARTITION FROM AN EXISTING TABLE:
SYNTAX:
ALTER TABLE <TN> DROP PARTITION <PARTITION NAME>;

EX:
SQL> ALTER TABLE TEST1 DROP PARTITION P1;

INDEX

- IT IS A DB OBJECT WHICH IS USED TO RETRIEVE A PARTICULAR ROW FROM A TABLE FASTLY.
- HERE, DB INDEX IS WORKING JUST LIKE A BOOK INDEX PAGE IN TEXTBOOK. BY USING BOOK INDEX PAGE HOW WE RETRIEVE THE REQUIRED TOPIC FROM A TEXT BOOK FASTLY SAME AS BY USING DB INDEX WE CAN RETRIEVE A SPECIFIC ROW FROM A TABLE FASTLY.
- BY USING DB INDEX WE CAN SAVE SEARCHING TIME AND IMPROVE THE PERFORMANCE OF DATABASE.
- DB INDEX OBJECT IS CREATED ON A PARTICULAR COLUMN / COLUMNS AND THOSE COLUMN / COLUMNS ARE CALLED AS "INDEXED KEY COLUMN".
- BASED ON AN INDEXED KEY COLUMN ONLY WE CAN RETRIEVE THE REQUIRED DATA FROM A TABLE AND WHEN WE RETRIEVE DATA FROM A TABLE WE SHOULD USE AN INDEXED KEY COLUMN UNDER "WHERE" CLAUSE CONDITION OTHERWISE INDEXES CANNOT ACTIVATE.
- ALL DATABASES ARE SUPPORTING THE FOLLOWING TWO TYPES OF SEARCHING MECHANISMS:

  1. TABLE SCAN (DEFAULT)
  2. INDEX SCAN

1. TABLE SCAN:

- IN THIS SCAN ORACLE DB SERVER SCANNING THE ENTIRE TABLE FOR REQUIRED DATA.

EX:
SQL> SELECT * FROM EMP WHERE SAL=3000;------COMPARING 14 ROWS IN A TABLE

```
 SAL
----------
 800
1600
1250
2975
1250
2850
2450          ----------------------------------> WHERE SAL=3000;
3000
5000
1500
1100
 950
3000
1300
```

2. INDEX SCAN:

- IN THIS SCAN ORACLE DB SERVER IS SCANNING BASED ON AN INDEXED COLUMN FOR REQUIRED DATA.
- ALL DATABASES ARE SUPPORTING THE FOLLOWING TWO METHODS ARE,

- IMPLICIT INDEXES (AUTOMATICALLY)
- EXPLICIT INDEXES (MANNUALLY)

1. IMPLICIT INDEXES:

- WHEN WE CREATED A TABLE ALONG WITH "PRIMARY KEY / UNIQUE " CONSTRAINT THEN INTERNALLY
SYSTEM IS CREATED AN INDEX OBJECT ON A COLUMN AUTOMATICALLY.

EX:
SQL> CREATE TABLE TEST1(EID INT PRIMARY KEY,ENAME VARCHAR2(10));
SQL> CREATE TABLE TEST2(SNO INT UNIQUE,ENAME VARCHAR2(10));

NOTE:
IF WE WANT TO VIEW COLUMN NAME ALONG WITH INDEX NAME OF A PARTICULAR TABLE IN ORACLE DB THEN USE A DATADICTIONARY IS "USER_IND_COLUMNS".

EX:
SQL> DESC USER_IND_COLUMNS;
SQL> SELECT COLUMN_NAME,INDEX_NAME FROM USER_IND_COLUMNS
2 WHERE TABLE_NAME='TEST1';

```
COLUMN_NAME                                        INDEX_NAME
--------------------------------------------------  --------------------------------------------------------------------
EID                                                SYS_C007932
```

SQL> SELECT COLUMN_NAME,INDEX_NAME FROM USER_IND_COLUMNS
2 WHERE TABLE_NAME='TEST2';

```
COLUMN_NAME                                    INDEX_NAME
---------------------------------------------  -------------------------------------------------------------------
SNO                                            SYS_C007933
```

__2) EXPLICIT INDEXES:__

- THESE INDEXES ARE CREATED BY USER IN TWO TYPES,
    1. B-TREE INDEXES
    2. BITMAP INDEXES

1. B-TREE INDEXES:

- IT IS A DEFAULT INDEX OF DATABASE.
    - SIMPLE INDEX
    - COMPOSITE INDEX
    - UNIQUE INDEX
    - FUNCTIONAL BASED INDEX

i)SIMPLE INDEX:

- WHEN WE CREATED AN INDEX OBJECT BASED ON A SINGLE COLUMN IN TABLE.

SYNTAX:
CREATE INDEX <INDEX NAME> ON <TN>(COLUMN NAME);

EX:
SQL> CREATE INDEX I1 ON EMP(SAL);
Index created.

TESTING:
SQL> SELECT * FROM EMP WHERE SAL=3000;------ 3 TIMES COMPARING.

```
                                        B-TREE INDEX
                                             |
                                  (LP) < | 3000 | >=(RP) --------------root node
                                             |
                   (LP)<|2975|>=(RP)                      (LP) < | 5000 | >= (RP) --------------parent node
                            |                                          |
                                                   3000| *,* -------------child node  (HERE " * " = ROWID)
2850| * , 2450| *, 1600| *,
1500| * , 1300| * , 1250| *,*,
1100| * , 950| * , 800| *
```

ii) COMPOSITE INDEX:

- WHEN WE CREATED AN INDEX OBJECT BASED ON MULTIPLE COLUMNS.

SYNTAX:
CREATE INDEX <INDEX NAME> ON <TN>(<COLUMN NAME1>,<COLUMN NAME2>,...........);

EX:
SQL> CREATE INDEX I2 ON EMP(DEPTNO,JOB);

TESTING:
SQL> SELECT * FROM EMP WHERE JOB='CLERK';----- TABLE SCAN

```
SQL> SELECT * FROM EMP WHERE DEPTNO=10;------ INDEX SCAN
SQL> SELECT * FROM EMP WHERE DEPTNO=10 AND JOB='CLERK';------- INDEX SCAN
```

NOTE:
- IN COMPOSITE INDEX MECHANISM INDEXES ARE ACTIVATED BASED ON LEADING COLUMN ONLY.SO THAT WE SHOULD USE LEADING COLUMN UNDER "WHERE" CLAUSE CONDITION OTHERWISE
ORACLE SERVER WILL FOLLOW TABLE SCAN MECHANISM.

iii) UNIQUE INDEX:
- WHEN WE CREATED AN INDEX OBJECT BASED ON "UNIQUE CONSTRAINT".
- BY USING UNIQUE CONSTRAINT WE CAN RESTRICTED DUPLICATE VALUES IN A COLUMN SO THAT DB SEARCHING TIME WILL REDUCED AND IMPROVE THE PERFORMANCE OF DATABASE.

SYNTAX:
CREATE UNIQUE INDEX <INDEX NAME> ON <TN>(COLUMN NAME);

EX:
```
SQL> CREATE UNIQUE INDEX UI ON DEPT(DNAME);
Index created.
```

TESTING:
```
SQL> INSERT INTO DEPT VALUES(50,'SALES','HYD');
ERROR at line 1:
ORA-00001: unique constraint (MYDB4PM.UI) violated
```

iv) FUNCTIONAL BASED INDEX:
- WHEN WE CREATED AN INDEX OBJECT BASED ON A FUNCTION NAME.

SYNTAX:
CREATE INDEX <INDEX NAME> ON <TN>(<FUNCTION NAME>(COLUMN NAME));

EX:
```
SQL> CREATE INDEX I3 ON EMP(UPPER(ENAME));
Index created.
```

TESTING:
```
SQL> SELECT * FROM EMP WHERE UPPER(ENAME)='smith'; ----- NOT ALLOWED
SQL> SELECT * FROM EMP WHERE UPPER(ENAME)='SMITH'; ----- ALLOWED
```

BITMAP INDEX:
- THESE INDEXES ARE CREATED ON "LOW CARDINALITY OF COLUMNS".

CARDINALITY :
- IT REFER UNIQUENESS OF DATA IN A COLUMN.

SYNTAX:
CARDINALITY OF COLUMN = NO.OF DISTINCT VALUES IN A COLUMN
==============================
NO.OF ROWS IN A TABLE

EX:

CARDINALITY OF EMPNO = 14 / 14 -------------------> 1 (HIGH)
CARDINALITY OF JOB = 5 / 14 ---------------------------> 0.35 (LOW)

SYNTAX TO CREATE BITMAP INDEX:
CREATE BITMAP INDEX <INDEX NAME> ON <TN>(COLUMN NAME);

EX:
```
SQL> CREATE BITMAP INDEX BIT1 ON EMP(JOB);
Index created.
```

- WHENEVER WE ARE CREATING A BITMAP INDEX OBJECT ON A SPECIFIC COLUMN IN A TABLE INTERNALLY SYSTEM WILL CREATE A BITMAP INDEXED TABLE WITH BIT NUMBERS “1 “AND “0".HERE “1” IS REPRESENT CONDITION IS "TRUE" AND “0 " IS REPRESENT CONDITION IS "FALSE".

EX:
```
SQL> SELECT * FROM EMP WHERE JOB='PRESIDENT';
```

BITMAP INDEXED TABLE
|

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
PRESIDENT	0	0	0	0	0	0	0	0	1	0	0	0	0	0

NOTE:
HERE "1" IS REPRESENT ROWID IN A TABLE.

NOTE:
TO VIEW INDEX NAME AND TYPE OF INDEX OF A TABLE THEN USE A DATADICTIONARY IS "USER_INDEXES".

EX:
```
SQL> DESC USER_INDEXES;
SQL> SELECT INDEX_NAME,INDEX_TYPE FROM USER_INDEXES
  2  WHERE TABLE_NAME='EMP';

INDEX_NAME                              INDEX_TYPE
--------------------------------------  ----------------------------------------------
I1                                      NORMAL(B-TREE)
I2                                      NORMAL(B-TREE)
I3                                      FUNCTION-BASED NORMAL(B-TREE)
BIT1                                    BITMAP
```

HOW TO DROP AN INDEX OBJECT:
SYNTAX:
DROP INDEX <INDEX NAME>;

EX:

```
DROP INDEX I1;
DROP INDEX BIT1;
```

CLUSTER

- IT IS A GROUP OF TABLES TOGETHER SAVE AND SHARE SAME MEMEORY IS CALLED AS "CLUSTER".
- CLUSTER CAN BE CREATED TO IMPROVE THE PERFORMANCE JOINS.
- WHEN WE WANT CREATE CLUSTER TABLES WE SHOULD HAVE A COMMON COLUMN NAME OTHERWISE WE CANNOT CREATE CLUSTER TABLES IN DB.
- CLUSTER CAB CREATED AT THE TIME OF CREATING TABLES ONLY.
- TO CREATE A CLUSTER MEMORY THEN WE FOLLOW THE FOLLOWING 3 STEPS ARE,

STEP1: CREATE A CLUSTER MEMORY:

SYNTAX:

```
CREATE CLUSTER <CLUSTER NAME>(<COMMON COLUMN NAME> <DATATYPE>[SIZE]);
```

STEP2: CREATE AN INDEX ON CLUSTER:

SYNTAX:

```
CREATE INDEX <INDEX NAME> ON CLUSTER <CLUSTER NAME>;
```

STEP3: CREATE CLUSTER TABLES:

SYNTAX:

```
CREATE TABLE <TN>(<COLUMN NAME1> <DATATYPE>[SIZE],.................................................)
CLUSTER <CLUSTER NAME>(COMMON COLUMN NAME);
```

EX:

```
SQL> CREATE CLUSTER EMP_DEPT(DEPTNO INT);
Cluster created.

SQL> CREATE INDEX CI ON CLUSTER EMP_DEPT;
Index created.

SQL> CREATE TABLE EMP33(EID INT,ENAME VARCHAR2(10),DEPTNO INT)
        CLUSTER EMP_DEPT(DEPTNO);
Table created.

SQL> CREATE TABLE DEPT33(DEPTNO INT,DNAME VARCHAR2(10))
        CLUSTER EMP_DEPT(DEPTNO);
Table created.

SQL> INSERT INTO EMP33 VALUES(1,'SMITH',10);
SQL> INSERT INTO EMP33 VALUES(2,'JONES',20);
SQL> COMMIT;

SQL> INSERT INTO DEPT33 VALUES(10,'D1');
SQL> INSERT INTO DEPT33 VALUES(20,'D2');
SQL> COMMIT;
```

- TO CHECK THE ABOVE TABLES ARE IN CLUSTER OR NOT THEN WE SEE ROWID's OF THE TABLES
IF BOTH TABLES ROWID's ARE SAME THEN WE CALLED AS "CLUSTER TABLES" OTHERWISE WE CALLED AS
"NON-CLUSTER TABLES".

SQL> SELECT ROWID FROM EMP33;

```
ROWID
------------------
AAATVXAAHAAAAJTAAA
AAATVXAAHAAAAJWAAA
```

SQL> SELECT ROWID FROM DEPT33;

```
ROWID
------------------
AAATVXAAHAAAAJTAAA
AAATVXAAHAAAAJWAAA
```

NOTE:
- TO VIEW ALL CLUSTER OBJECTS IN ORACLE DATABASE THEN USE A DATADICTIONARY IS "USER_CLUSTERS".

EX:
SQL> DESC USER_CLUSTERS;
SQL> SELECT CLUSTER_NAME FROM USER_CLUSTERS;

```
CLUSTER_NAME
--------------------------------------------------------------------------------------------------------------------------------
EMP_DEPT
```

NOTE:
- TO VIEW CLUSTER TABLES THEN USE A DATADICTIONARY IS "USER_TABLES".

EX:
SQL> DESC USER_TABLES;
SQL> SELECT TABLE_NAME FROM USER_TABLES WHERE CLUSTER_NAME='EMP_DEPT';

```
TABLE_NAME
--------------------------------------------------------------------------------------------------------------------------------
EMP33
DEPT33
```

HOW TO DROP A CLUSTER:
SYNTAX:
DROP CLUSTER <CLUSTER NAME> INCLUDING TABLES;

EX:
SQL> DROP CLUSTER EMP_DEPT INCLUDING TABLES;
Cluster dropped.

USER - DEFINE DATATYPES

USER DEFINE DATATYPES ARE INTRODUCED IN ORACLE 8.0 VERSION.WHEN PRE-DEFINE DATATYPES ARE NOT REACHING TO OUR REQUIREMENTS THEN WE CREATE OUR OWN DATATYPES ARE CALLED AS USER DEFINE DATATYPES.
THE ADVANTAGE OF USER DEFINE DATATYPES ARE REUSABILITY THAT MEANS WE CAN CREATE DATATYPE AND REUSE IN MULTIPLE TABLES.ORACLE SUPPORTS THE FOLLOWING THREE TYPES OF USERS DEFINE DATATYPES.

1. OBJECT TYPE (OR) COMPOSITE TYPE
2. VARRAY
3. NESTED TABLE.

1. OBJECT TYPE (OR) COMPOSITE TYPE:

IT ALLOWS GROUP OF VALUES /ELEMENTS OF DIFFERENT DATATYPES.
SYNATAX:

CREATE TYPE <TYPE NAME> AS OBJECT(<COL1> DATATYPE[SIZE], <COL2> DATATYPE[SIZE], ...);
/

EX:

CREATE TYPE COURSE_TYPE AS OBJECT (CID NUMBER (4), CNAME VARCHAR2(10), FEE NUMBER (10));
/

TESTING:

CREATE TABLE STUDENTS (SID NUMBER (4), SNAME VARCHAR2(10), COURSE COURSE_TYPE);
TABLE CREATED.

SQL> INSERT INTO STUDENTS VALUES (101,'SAI', COURSE_TYPE (1021,'ORACLE',1200));
SQL> INSERT INTO STUDENTS VALUES (102,'WARD', COURSE_TYPE (1022,'C',500));

TO SELECT:

SQL> SELECT S.SID, S. SNAME, S.COURSE.CID, S. COURSE.CNAME, S.COURSE.FEE FROM STUDENTS S;
(OR)
SQL> SELECT S.SID, S. SNAME, S.COURSE.CID CID, S. COURSE.CNAME CNAME, S.COURSE.FEE FEE FROM STUDENTS S;

TO UPDATE:

SQL> UPDATE STUDENTS S SET S.COURSE.FEE=2000 WHERE S.SID=101;

TO DELETE:

SQL> DELETE FROM STUDENTS S WHERE S.COURSE.CID=1022;

2. VARRAY:

IT ALLOWS GROUP OF VALUES /ELEMENTS OF SAME DATATYPES.VARRAY SHOULD DECLARE WITH SIZE.
SYNTAX:

CREATE TYPE <TYPE NAME> IS VARRAY(SIZE) OF DATATYPE[SIZE];
/

EX:

```
CREATE TYPE MBNO_ARRAY1 IS VARRAY (3) OF NUMBER (10);
/
```

TESTING:

```
SQL> CREATE TABLE EMPLOYEE (EMPNO NUMBER (4), MBNO MBNO_ARRAY1);
TABLE CREATED.

SQL> INSERT INTO EMPLOYEE VALUES (1021,
MBNO_ARRAY1(9703542749,8502045789));
SQL> INSERT INTO EMPLOYEE VALUES (1022,
MBNO_ARRAY1(9632587412,8523691478,7412356896));
```

3.NESTED TABLE:

> A TABLE WITHIN ANOTHER TABLE IS CALLED AS NESTED TABLE.
> NESTED TABLE ALSO ALLOW GROUP OF VALUES /ELEMENTS OF DIFF. DATATYPES.
> NESTED TABLE IS NOT DECLARE WITH SIZE.

STEPS TO CREATE NESTED TABLE:

STEP1: CREATE AN OBJECT TYPE:

SYNATAX:

```
CREATE TYPE <TYPE NAME> AS OBJECT(<COL1>
DATATYPE[SIZE], <COL2> DATATYPE[SIZE], ……..);
/
```

STEP2: CREATE NESTED TABLE TYPE:

SYNTAX:

```
CREATE TYPE <TYPE NAME> AS TABLE OF <OBJECT TYPE NAME>;
/
```

STEP3: CREATE A TABLE:

SYNTAX:

```
CREATE TABLE <TN>(<COL1> <DATATYPE>[SIZE], ……, <COL N>
<NESTED TABLE TYPE NAME>)
NESTED TABLE <COL N NAME> STORE AS <ANY NAME>;
```

EX:

STEP1:

```
CREATE TYPE ADDR_TYPE AS OBJECT (HNO NUMBER (4),
STREET VARCHAR2(10), CITY VARCHAR2(10));
/
```

STEP2:

```
CREATE TYPE ADDR_ARRAY AS TABLE OF ADDR_TYPE;
/
```

STEP3:

```
CREATE TABLE CUSTOMER (CID NUMBER (4), CNAME
VARCHAR2(10), CADDRESS ADDR_ARRAY) NESTED TABLE
CADDRESS STORE AS CUST_ADDR;
```

TESTING:

```
SQL> INSERT INTO CUSTOMER VALUES (1,'SAI’, ADDR_ARRAY
(ADDR_TYPE (1122,'GANDHI','HYD')));
SQL> INSERT INTO CUSTOMER VALUES (2,'WARD’, ADDR_ARRAY
(ADDR_TYPE (1123,'ASHOK','CHE'), ADDR_TYPE
(1124,'VASATI','MUM')));
```

NOTE:

WE CAN ALSO SELECT, UPDATE, DELETE, INSERT DATA WITHIN NESTED TABLE BY USING THE FOLLOWING SYNTAX,

SYNTAX:

```
SELECT / UPDATE / DELETE / INSERT (SELECT <NESTED TABLE
TYPE COLUMN NAME> FROM <TN>);
```

EX:

```
SQL> SELECT * FROM TABLE (SELECT CADDRESS FROM CUSTOMER
WHERE CID=1);
SQL> UPDATE TABLE (SELECT CADDRESS FROM CUSTOMER
WHERE CID=2) SET HNO=1024 WHERE HNO=1124;

SQL> DELETE FROM TABLE (SELECT CADDRESS FROM CUSTOMER
WHERE CID=2) WHERE CITY='MUM';

SQL> INSERT INTO TABLE (SELECT CADDRESS FROM CUSTOMER
WHERE CID=1) VALUES (1124,'YUVIN','HYD');
```

NOTE:

IN ORACLE WE WANT TO VIEW USER TYPES THEN FOLLOW THE FOLLWING DATADICTIONARY IS "USER_TYPES".

EX:

```
SQL> DESC USER_TYPES;
SQL> SELECT TYPE_NAME FROM USER_TYPES;
```

SYNTAX TO DROP TYPE:

```
SQL> DROP TYPE <TYPE NAME> FORCE;
```

EX:

```
DROP TYPE MBNO_ARRAY1 FORCE;
```