ORACLE

Oracle content: (2 months)

- Topic-1 : DBMS
- Topic-2 : ORACLE
- Topic-3 : SQL
  - Introduction to SQL
  - Sub - Languages of SQL
  - Datatypes in oracle sql
  - Operators in oracle sql
  - Functions in oracle sql
  - Clauses in oracle sql
  - Joins
  - Constraints
  - Subqueries
  - Views
  - Sequence
  - Indexes----------Interview level
- Topic-4 : Normalization
  - What is Normalization
  - Where we want to use Normalization
  - Why we need Normalization
  - Types of Normalization
    - > First normal form
    - > Second normal form
    - > Third normal form
    - > BCNF (Boyce-codd normal form)
    - > Fourth normal form
    - > Fifth normal form
- Topic-5 : PL/SQL
  - Introduction to PL/SQL
  - Difference between SQL and PL/SQL
  - Conditional & Looping statements
  - Cursors
  - Exception Handling
  - Stored procedures
  - Stored functions
  - Triggers----------Interview level

==========

Topic-1 : DBMS

==========

- In IT field user can interact with two types of applications.those are,
    1. Front end application
    2. Back end application

1. Front end application:

==========

- FEA is an application where the end-users are interacting directly.
    Ex: Register form,Login form,Viewprofile form,Home page,.....etc

Design & Develop:

==========

- UI technologies.(html,css,javascript,angularJS,reactJS,.....etc)

2. Back end application:

==========

- BEA is an application where we store the end-users data/information.
    Ex: Database(Tables).

Design & Develop:

==========

- DB technologies.(oracle,sqlserver,mysql,postgresql,.....etc)

NOTE:

=====

- To establish connection between front end application and back end application then we must use a "serverside technology" are JAVA,.NET,PYTHON.

What is Data?

==========

- it is a rawfact.(i.e characters,numbers,special characters and symbols)
    Ex:
        10021 is data SMITH is data
        10022 is data ALLEN is data
        10023 is data JONES is data

- data never give meaningfull statements.

What is Information?

==========

- processing data is called information.
- information is always provide meaningfull statements.

Ex:

Employee_ID	Employee_NAME
10021	SMITH
10022	ALLEN
10023	JONES

What is Database?
==========

- it is a location where we store collection of inter-related information of a particular business organization.

Ex:

SBI_Organization
- group of branches ----------> group of customers
  - group of departments
    - group of employees

Ex:

No department = No employees
No employees = No department

No products = No customers
No customers = No products

Types of Databases:
==========

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

1. OLTP:
========

- these databases are used for storing "day-to-day" transactional information.
  Ex: oracle,sqlserver,mysql,postgresql,db2,....etc

2. OLAP:
=======

- these databases are used for storing "historical" information(i.e bigdata)
  Ex: datawarehouse(DWH)

What is DBMS?
==========

- it is a s/w which is used to manage and maintaine data/information with in the database.
- by using dbms s/w we will perform the following operations are,
    > creating database
    > create tables
    > inserting data
    > updating data
    > seleting data
    > deleting data
- DBMS s/w will act as an interface between user and database.

User<------> DBMS s/w <----------> Database
(interface)

Models of DBMS:
==========

- there are three types of models in DBMS.
    1. Hierarchical database management system(HDBMS)
       ex: IMS s/w (information management system)
    2. Network database management system(NDBMS)
       ex: IDBMS s/w(integrated database management system)

Note:
=====

- HDBMS,NDBMS models are outdated in real time.

3. Relational Database Management System(RDBMS):
==========

- there are two modules in RDBMS.
    i) Object Relational DBMS(ORDBMS):
    ==========
        - data can be stored in the form of "TABLE" format.
            > a table = group of rows and columns.
            > a row is also called as "record / tuple".
            > a column is also called as "attribute / field".
        - these databases are depends on "SQL" so that these are called as "SQLDatabase" in real world.
        Ex: oracle,sqlserver,mysql,.....etc.
    ii) Object Oriented DBMS(OODBMS):
    ==========
        - data can be stored in the form of "OBJECT" format.
        - these databases are depends on "OOPS" connect but not "SQL".so that these are called as "NoSQLDatabase" in real world.
    Ex: MongoDB,Cassandra,.....etc.

==========

Topic-2 : ORACLE

==========

Introduction to Oracle:

==========

- it is DB software / DB tool / Back end tool / RDBMS(ordbms) product.
- it was introduced by "oracle corporation" in 1979.
- it is used to store data/information permanently along with security.
- it can be deployed in any platform like windows,linux,unix,mac,solaries,....etc.
- it is a platform independent an rdbms product.

What is platform:

==========

- it is a combination operating system and micro-processor.
- there are two types of platforms in real time.

i) Platform dependent:

==========

- supporting only one operating system with the combination of any micro-processor.

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

ii) Platform independent:

==========

- supporting any operating system with the combination of
- any micro-processor.

Ex: Oracle,Java,.Net core,Python,Mysql,.....etc

Types of oracle s/w editions:

==========

- oracle s/w is available in two editions.

i) oracle express edition:

==========

- supporting oracle features partially.

Ex: recyclebin,flashback,purge,partition table,.....etc are not allowed.

ii) oracle enterprise edition:

==========

- supporting all features of oracle.

Ex: recyclebin,flashback,purge,partition table,.....etc are allowed.

Versions of oracle:
==========

- the first version of oracle s/w is "oracle 1.0" in 1979.
    - > oracle 1.0
    - > oracle 2.0
    - > oracle 3.0
    - > oracle 4.0
    - > oracle 5.0
    - > oracle 6.0
    - > oracle 7.0
    - > oracle 8.0
    - > oracle 8i (internet)
    - > oracle 9i
    - > oracle 10g (grid technology)
    - > oracle 11g
    - > oracle 12c (cloud technology)
    - > oracle 18c
    - > oracle 19c
    - > oracle 21c (latest version)
    - > oracle 23c (Beta version)

How to download oracle21c enterprise edition s/w:
==========
https://www.oracle.com/in/database/technologies/oracle21c-windows-downloads.html

How to install oracle21c enterprise edition s/w:
==========

- Installation video

Working with Oracle:
==========

- Once we are installing oracel s/w internally there are two components are installed in the system.
    1) Client component
    2) Server component

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

- by using client tool we will perform the following three operations are,

    step1: User can connect to oracle server.
        Enter username : system (default username)

Enter password : lion (created at the time of installation)
connected.

step2: User can send request to oracle server.
Request : SQL query / SQL command

step3: User will get responese from oracle server.
Response : Output / Result

Ex: SQLPlus,SQLDeveloper,Toad.

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

- server component is having two more sub-components.those are,
  i) Instance
  ii) Database

i) Instance :
==========

- it is a temporary memory which will allocate from RAM.
- data can be stored temporarly.

ii) Database:
===========

- it is a permanent memory which will allocate from Harddisk.
- data can be stored permanently.

NOTE:
=====

- when we want to work on oracle database server we need to follow the following two steps procedure.
  step1: connect
  step2: communicate

step1: connect:
=============

- when user want to connect to oracle server then we required a db tool is known as "SQLPLUS".

step2: communicate:
==================

- when user want to communicate with database then we need a db language is called as "SQL".

How to connect to oracle server:
==========
> go to all programs
> go to open oracle-oradb21home-1 folder
> click on SQLPLUS icon
Enter user-name: SYSTEM
Enter password: LION
connected.

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

How to create a new username and password:
==========
syntax:
======
create user <username> identified by <password>;

EX:
SQL> CONN
Enter user-name: SYSTEM/LION
Connected.

SQL> CREATE USER MYDB2PM IDENTIFIED BY 123;
ERROR at line 1:
ORA-65096: invalid common user or role name

Solution:
=======
SQL> ALTER SESSION SET "_ORACLE_SCRIPT"=TRUE;
Session altered.

SQL> CREATE USER MYDB2PM IDENTIFIED BY 123;
User created.
(OR)
SQL> CREATE USER C##SUDHAKAR IDENTIFIED BY 12345;
User created.

SQL> CONN / CONNECT
Enter user-name: MYDB2PM/123
ERROR:
ORA-01045: user MYDB2PM lacks CREATE SESSION privilege; logon denied

Granting all permissions to user:
==========
syntax:
=======
grant dba to <username>;

EX:
SQL> CONN
Enter user-name: SYSTEM/LION
Connected.

SQL> GRANT DBA TO MYDB2PM;
Grant succeeded.

SQL> CONN
Enter user-name: MYDB2PM/123
Connected.

How to change password for User:
==========
syntax:
=======
password;

EX:
SQL> CONN
Enter username: MYDB2PM/123
connected.

SQL> PASSWORD;
Changing password for MYDB2PM
Old password: 123
New password: ABC
Retype new password: ABC
Password changed

SQL> CONN
Enter username: MYDB2PM/ABC
connected.

How to re-create a new password for USER if we forgot it:
==========
syntax:
=======
alter user <username> identified by <new password>;

EX:
SQL> CONN
Enter user-name: SYSTEM/LION
Connected.

SQL> ALTER USER MYDB2PM IDENTIFIED BY MYDB2PM;
User altered.

SQL> CONN
Enter user-name: MYDB2PM/MYDB2PM
Connected.

How to re-create a new password for SYSTEM if we forgot it:
==========
EX:
SQL> CONN
Enter user-name: \sys as sysdba
Enter password: sys
Connected.

SQL> ALTER USER SYSTEM IDENTIFIED BY TIGER;
User altered.

SQL> CONN
Enter user-name: SYSTEM/TIGER
Connected.

How to view usernames in oracle database if we forgot it:
==========
syntax:
======
select username from all_users;

EX:
SQL> CONN
Enter user-name: SYSTEM/TIGER
Connected.

SQL> SELECT USERNAME FROM ALL_USERS;

How to drop a user from oracle:
==========
syntax:
=======
DROP USER <USERNAME>;

EX:
SQL> CONN
Enter user-name: SYSTEM/TIGER
Connected.

SQL> DROP USER MYDB2PM;
ERROR at line 1:
ORA-28014: cannot drop administrative user or role

Solution:
========
SQL> ALTER SESSION SET "_ORACLE_SCRIPT"=TRUE;
Session altered.

SQL> DROP USER MYDB2PM;
User dropped.

How to clear the screen :
==========
syntax:
=======
CL SCR;

How to disconnect from oracle:
==========
syntax:
=======
EXIT;

==========

Topic-3 : SQL

==========

Introduction to SQL:

==========

- SQL is database language which was introduced by "IBM".
- SQL is used to communicate with any database in real time.
  Ex: oracle,mysql,sqlserver,db2,sybase,postgresql,.....etc
- Initially SQL is called as "SEQUEL" language and later renamed as "SQL".
- SQL is not a case-sensitive language i.e user can write SQL queries in either upper / lower / combination of upper and lower case characters.
  Ex:
  SELECT * FROM EMP;--------executed
  select * from emp ;----------executed
  SelecT * From Emp;----------executed
- In oracle storage of data is a case-sensitive.
- Every sql query should ends with " ; ".

Sub-Languages of SQL:

==========

1) Data Definition Language(DDL):

==========

- CREATE
- ALTER
  > ALTER - MODIFY
  > ALTER - ADD
  > ALTER - RENAME
  > ALTER - DROP
- RENAME
- TRUNCATE
- DROP

New Features in oracle10g enterprise edition:

==========

- RECYCLEBIN
- FLASHBACK
- PURGE

2) Data Manipulate Language(DML):

==========

- INSERT
- UPDATE
- DELETE

3) Data Query / Retrieval Language(DQL/DRL):
==========
- SELECT(read only)

4) Transaction Control Language(TCL):
==========
- COMMIT
- ROLLBACK
- SAVEPOINT

5) Data Control Language(DCL):
==========
- GRANT
- REVOKE

==========
1) Data Definition Language(DDL):
==========
CREATE:
==========
- to create a new database object such as Tables,Views,Synonyms,Procedure,Function,Trigger,....etc

How to create a new table in oracle:
==========
syntax:
==========
create table <table name>(<column name1> <datatype>[size],<column name2> <datatype>[size],.....);

DATATYPES IN ORACLE:
==========
- it is an attribute which is used to store the type of data into a column in the table.
- oracle supports the following datatypes are,
  - > Numeric datatypes
  - > Character / String datatypes
  - > Long datatype
  - > Date datatypes
  - > Raw & Long Raw datatypes
  - > Lob datatypes

Numeric datatypes:
==========
NUMBER datatype:
==========

i) number(p) : it will store integer format data.
ii) number(p,s) : it will store integer and also float values.

Precision(p):
==========

- counting all digits from the given expression.
- maximum size of precision is 38 digits.

Ex:

i) 1256	i) 56.23
precision =4	precision = 4
ii) 687345	ii) 6874.12
precision = 6	precision = 6

Scale(s):
=======

- counting the right side digits of a decimal point from the given expression.
- there is no maximum size of scale because scale is a part of precision.

Ex:

i) 56.23
precision = 4
scale = 2

ii) 68745.346
precision = 8
scale = 3

EX:

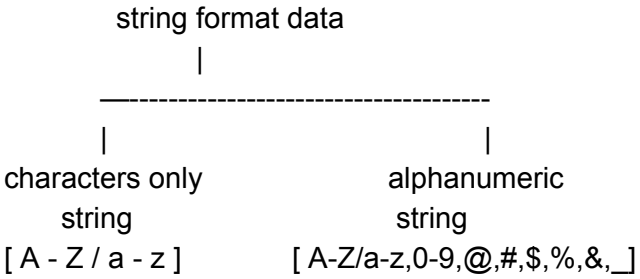
SNO number(5)	PRICE number(8,2)
==========	==========
0	0.0
1	25.34
2	
99999	999999.99
100000----------error	1000000(1000000.00)-----error

Character / String datatypes:
==========

- these datatypes are used for storing string format data only.
- in database string can be represent with '<string>'.

EX: Empname char(10)
==========

smith -------> error
'smith' -------> smith
123 -------> error
'123' -------> 123
23.12 -------> error
'23.12' -------> 23.12



Ex: 'SMITH','smith',.....etc
Ex: 'smith123@gmail.com',Password,PAN,HTNO,...etc

Types of character / string datatypes:
==========

- there are two types of character datatypes.those are,
  1. Non-unicode datatypes:
     ==========
     - these datatypes are used for storing "localized data".(i.e English Language only)
       i) char(size)
       ii) varchar2(size)

  2. Unicode datatypes:
     ==========
     - these datatypes are used for storing "globalized data".(i.e All National Language only)
       i) Nchar(size)
       ii) Nvarchar2(size)
     - "N" stands for "national language".

i) char(size):
==========

- it is a fixed length datatype(i.e static)
- it will store non-unicode characters in the form of 1 char=1 byte.
- the maximum size is 2000 bytes.

Disadvantage:
==========

- memory wasted.

ii) varchar2(size):
==========

- it is a variable length datatype(i.e dynamic)
- it will store non-unicode characters in the form of 1 char=1 byte.
- the maximum size is 4000 bytes.

Advantage:
==========

- memory saved.

i) Nchar(size):
==========

- it is a fixed length datatype(i.e static)
- it will store unicode characters in the form of 1 char=1 byte.
- the maximum size is 2000 bytes.

Disadvantage:
==========

- memory wasted.

ii) Nvarchar2(size):
==========

- it is a variable length datatype(i.e dynamic)
- it will store unicode characters in the form of 1 char=1 byte.
- the maximum size is 4000 bytes.

Advantage:
==========

- memory saved.

LONG datatype:
==========

- it is a variable length datatype(i.e dynamic)
- it will store non-unicode & unicode characters in the form of 1 char=1 byte.
- the maximum size is 2gb.
- a table is having only one long datatype column.

DATE datatypes:
==========

- to store date and time information of a particular day / transaction.
- rang of date datatypes are from '01-JAN-4712' BC to '31-DEC-9999' AD.
    i) DATE
    ii) TIMESTAMP

i) DATE:
=======

- it will store date and time information but time is optional.
- if user does not give time information then oracle server will take time is ' 00 : 00 : 00 ' by default.
- default date format of oracle database is :
    ' DD-MON-YY/YYYY HH:MI:SS'
    ' 30-JAN-25/2025   14:55:xx '
      1   1   2   1   1   1 ----------> 7 bytes (fixed memory)

ii) TIMESTAMP:
============

- it will store date and time information including milliseconds.
- default format of timestamp is :
    ' DD-MON-YY/YYYY HH:MI:SS.MS'
    ' 30-JAN-25/2025   14:55:xx.xxxx '
      1   1   2   1   1   1   4----------> 11 bytes (fixed memory)

Raw & Long Raw datatypes:
==========

- these are datatypes are used for storing image / audio / video file in the form of 010100101001 binary format.
    > Raw   - static datatype   - 2000bytes
    > Long raw   - dynamic datatype   - 2gb

LOB datatypes:
==========

- LOB stands for large objects.
    > CLOB
    > NCLOB

> BLOB

CLOB:
=====
- it stands for character large object.
- it is a dynamic datatype.
- it will store non-unicode characters in the form of 1 char= 1byte.
- maximum size is 4gb.

NCLOB:
======
- it stands for national character large object.
- it is a dynamic datatype.
- it will store unicode characters in the form of 1 char= 1byte.
- maximum size is 4gb.

BLOB:
=====
- it stands for binary large object.
- it is a dynamic datatype.
- it will store image / audio / video file in the form of 010010010101 binary format.
- maximum size is 4gb.

Non-unicode characters:
======================
- char(size) - 2000 bytes
- varchar2(size) - 4000 bytes
- long - 2gb
- clob - 4gb

Unicode characters:
======================
- Nchar(size) - 2000 bytes
- Nvarchar2(size) - 4000 bytes
- long - 2gb
- Nclob - 4gb

Binary data format:
===============
- Raw - 2000 bytes
- Long Raw - 2gb
- Blob - 4gb

==========

How to create a new table in oracle:
==========
syntax:
======
create table <table name>(<column name1> <datatype>[size],<column name2> <datatype>[size],.....);

EX:
SQL> CREATE TABLE STUDENT(ROLL_NO NUMBER(3),CNAME CHAR(10),CFEE NUMBER(6,2));

To view list of tables in oracle database:
==========
syntax:
======
select * from tab; (tab is predefined table)

To view the structure of a table:
==========
syntax:
======
desc <table name>; (describe command)

EX:
SQL> DESC STUDENT;

ALTER command:
==========
- to change / modify the structure of a table.
- it contains the four sub-commands are,
    i) ALTER - MODIFY
    ii) ALTER - ADD
    iii) ALTER - RENAME
    iv) ALTER - DROP

i) ALTER - MODIFY:
==========
- to change datatype and also the size of datatype of a particular column.

syntax:
======
alter table <table name> modify <column name> <new datatype>[new size];

EX:
SQL> ALTER TABLE STUDENT MODIFY CNAME VARCHAR2(20);

ii) ALTER - ADD:
==========
- to add a new column to an existing table.

syntax:
======
alter table <table name> add <new column name> <datatype>[size];

EX:
SQL> ALTER TABLE STUDENT ADD SMBNO NUMBER(10);

iii) ALTER - RENAME:
==========
- to change a column name.

syntax:
=======
alter table <table name> rename <column> <old column name> to <new column name>;

EX:
SQL> ALTER TABLE STUDENT RENAME COLUMN CNAME TO COURSE_NAME;

iv) ALTER - DROP:
==========
- to drop / delete a column from a table permanently.

syntax:
======
alter table <table name> drop <column> <column name>;

EX:
SQL> ALTER TABLE STUDENT DROP COLUMN CFEE;

RENAME command:
==========
- to change a table name.

syntax:
======
rename <old table name> to <new table name>;

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

TRUNCATE command:
==========

- to delete all rows but not columns from a table.
- deleting rows from a table permanently.
- we cannot delete a specific row from a table because truncate command is not supports "WHERE" clause condition.

syntax:
======
truncate table <table name>;

EX:
SQL> TRUNCATE TABLE DEPT WHERE DEPTNO=30;
ERROR at line 1:
ORA-03291: Invalid truncate option - missing STORAGE keyword

EX:
SQL> TRUNCATE TABLE DEPT;

DROP command:
==========

- to drop / delete the entire table (i.e rows & columns) from database.

syntax:
======
drop table <table name>;

EX:
SQL> DROP TABLE DEPT;

NOTE:
=====

- Before oracle10g enterprise edition once we drop a table then it was deleted from database permanently whereas from oracle10g enterprise edition once we drop a table from database then it was temporary.

New Features in oracle10g enterprise edition:
==========
RECYCLEBIN:
==========

- it is a system defined table which will store the information about deleted tables.

EX:
SQL> DESC RECYCLEBIN;

SQL> SELECT OBJECT_NAME,ORIGINAL_NAME FROM RECYCLEBIN;

OBJECT_NAME	ORIGINAL_NAME
BIN$gsk0lGtQQpahD9QRJazXFA==$0	DEPT

FLASHBACK:
==========
- it is a DDL command which is used to restore a deleted table from recyclebin to database.

syntax:
=======
flashback table <table name> to befor drop;

EX:
SQL> FLASHBACK TABLE DEPT TO BEFORE DROP;

PURGE:
=======
- it is a DDL command which is used to delete / drop a table from database permanently.

syntax:
=======
drop table <table name> purge;

EX:
DROP TABLE DEPT PURGE;

==========

Data Manipulation Language(DML):
==========
INSERT command:
==========
- to insert a new row data into a table.
- there are two types to inserting data into a table.

Method-1: To insert all columns values:
==========
syntax:
======
insert into <table name> values(value1,value2,.....);

EX:
SQL> INSERT INTO STUDENT VALUES(101,'C',9874563214);

Method-2: To insert values for required columns:
==========
syntax:
======
insert into <table name>(required column names)values(value1,value2,.....);

EX:
SQL> INSERT INTO STUDENT(ROLL_NO)VALUES(102);
SQL> INSERT INTO STUDENT(ROLL_NO,SMBNO)VALUES(103,7894561236);
SQL> INSERT INTO
STUDENT(COURSE_NAME,SMBNO,ROLL_NO)VALUES('ORACLE',8224561236,104);

How to insert multiple rows into a table dynamically:
==========
Method-1:
========
insert into <table name> values(&<column name1>,&<column name2>,.....);

Ex:
SQL> INSERT INTO STUDENT VALUES(&ROLL_NO,'&COURSE_NAME',&SMBNO);
Enter value for roll_no: 105
Enter value for course_name: JAVA
Enter value for smbno: 6392587415

SQL> / (To re-execute the lastly executed sql query in sqlplus editor)
Enter value for roll_no: 106
Enter value for course_name: .NET
Enter value for smbno: 7845123693

SQL> /
.....
.....
.....

Method-2:
========
syntax:
======
insert into <table name>(required column names)values(&<column name1>,.....);

EX:
SQL> INSERT INTO STUDENT(ROLL_NO)VALUES(&ROLL_NO);

Enter value for roll_no: 108

SQL> /
.....

UPDATE command:
==========
- to update all rows data in a table at a time.
(or)
- to update a specific row data in a table by using "WHERE" clause condition.

syntax:
=======
update <table name> set <column name1>=<value1>,<column name2>=<value2>,.....[where <condition>];

EX:
SQL> UPDATE STUDENT SET COURSE_NAME='SAP',SMBNO=9856214578 WHERE ROLL_NO=106;
SQL> UPDATE STUDENT SET ROLL_NO=NULL,COURSE_NAME=NULL,SMBNO=NULL WHERE COURSE_NAME='C++';
SQL> UPDATE STUDENT SET ROLL_NO=102,COURSE_NAME='.NET',SMBNO=7412589635 WHERE ROLL_NO IS NULL;

EX:
SQL> UPDATE STUDENT SET SMBNO=NULL;
SQL> UPDATE STUDENT SET SMBNO=9874563214;
SQL> UPDATE STUDENT SET SMBNO=9874563214 WHERE COURSE_NAME='.NET' OR COURSE_NAME='SAP';

DELETE command:
==========
- to delete all rows from a table at a time.
(or)
- to delete a specific row from a table by using "WHERE" clause condition.

syntax:
=======
delete from <table name> [where <condition>];

EX:
SQL> DELETE FROM STUDENT WHERE ROLL_NO=102;
SQL> DELETE FROM STUDENT WHERE COURSE_NAME='JAVA' OR COURSE_NAME='PYTHON';
SQL> DELETE FROM STUDENT WHERE COURSE_NAME IS NULL;

SQL> DELETE FROM STUDENT;

DELETE vs TRUNCATE:
==========

DELETE	TRUNCATE
1. it is a DML operation.	1. it is a DDL operation.
2. deleting a specific row.	2. cannot delete a specific row.
3. supporting "WHERE" clause.	3. does not support "WHERE" clause.
4. deleting data temporarly.	4. deleting data permanently.
5. we can restore deleted data by using "ROLLBACK".	5. we cannot restore deleted data by using "ROLLBACK".
6. deleting rows in one-by-one manner.	6. deleting rows as a page wise.
7. execution speed is slow.	7. execution speed is fast.

3) Data Retrieve / Query Language(DRL/DQL):
==========
SELECT command:
==========
- to retrieve all rows from a table at a time.
(or)
- to retrieve a specific row from a table by using " WHERE" clause condition.

syntax:
==========
select * from <table name> [where <condition>];

EX:
SQL> SELECT * FROM EMP;
SQL> SELECT * FROM EMP WHERE JOB='MANAGER';
SQL> SELECT ENAME,HIREDATE,DEPTNO FROM EMP WHERE DEPTNO=20;

EX:
SQL> SELECT * FROM EMP WHERE COMM IS NULL;
SQL> SELECT * FROM EMP WHERE COMM IS NOT NULL;

ALIAS NAME:
==========
- it is a temporary name for column and table.
- we can create alias name at two levels.
1) column level alias:
==========
- in this level we are creating alias name for columns.
2) table level alias:
==========
- in this level we are creating alias name for table.

syntax:
======
select <column name1> [as] <column alias name1>,<column name2> [as] <column alias name2>,
.....from <table name> <table alias name>;

EX:
SQL> SELECT ENAME,SAL AS BASIC_SALARY,SAL*12 AS ANNUAL_SALARY FROM EMP E;

(or)

SQL> SELECT ENAME,SAL BASIC_SALARY,SAL*12 ANNUAL_SALARY FROM EMP E;

CONCATENATION OPERATOR(||):
==========
- to add two or more than two expressions.

syntax:
======
<expression1> || <expression> || <expression3> ||.....;

EX:
SQL> SELECT 'THE EMPLOYEE'||' '||ENAME||' '||'IS WORKING AS A'||' '||JOB FROM EMP;

DISTINCT keyword:
==========
- to eliminate duplicate values from a column temporarly.

syntax:
======
distinct <column name>

EX:
SQL> SELECT DISTINCT JOB FROM EMP;
SQL> SELECT DISTINCT DEPTNO FROM EMP [ORDER BY DEPTNO];

NOTE:
=====
- if we want to display the large scale data tables in proper systematicall order in sqlplus then we must use the following two properties.
    i) PAGESIZE n
    ii) LINES n

i) PAGESIZE n:
==============
- by default a page can display up to 14 rows only.
- it is used to display no.of rows in a single page.Here "n" is no.of rows.
- the maximum size of a page is 50000 rows.

syntax:
=======
set pagesize n;

Ex:
SQL> SET PAGESIZE 100;

ii) LINES n:
==========
- by default a line can display up to 80 characters.
- this property is used to print no.of characters in a single line.Here "n" no.of characters.
- the maximum size is 32767 bytes / character.

syntax:
=======
set lines n;

EX:
SQL> SET LINES 160;

==========

OPERATORS IN ORACLE SQL:
========================
- to perform some operations on the given operand values.
- oracle supports the following operators are,

- Assignment operator => =
- Arithematic operators => + , - , * , /
- Relational operators => < , > , <= , >= , != (or) < >
- Logical operators => AND,OR,NOT
- Set operators => UNION,UNION ALL,INTERSECT,MINUS

- Special operators

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

Assignment operator:
==========
- to assign a value to variable / to a attribute.

syntax:
=======
<column name> <assignment operator> <value>;

EX:
SQL> UPDATE EMP SET SAL=50000;
SQL> UPDATE EMP SET JOB='HR';

Arithematic operators:
==========
- to perform addition,subtraction,multiple and division.

syntax:
=======
<column name> <arithematic operator> <value>

Ex:
waq to display all employees salary details after adding 2000/-?
SQL> SELECT ENAME,SAL AS OLD_SALARY,SAL+2000 AS NEW_SALARY FROM EMP;

Ex:
waq to display EMPNO,ENAME,BASIC SALARY and ANNUAL SALARY of the employees who are working under deptno is 20?
SQL> SELECT EMPNO,ENAME,SAL AS BASIC_SALARY,SAL*12 AS ANNUAL_SALARY
FROM EMP WHERE DEPTNO=20;

Ex:
waq to display all employees salaries details after increment of 5%?
SQL> SELECT ENAME,SAL AS BEFORE_INCREMENT,SAL+SAL*5/100 AS AFTER_INCREMENT FROM EMP;

Ex:
waq to display EMPNO,ENAME,JOB,BASIC SALARY,5% of HRA,10% of DA,5% of PF,5% of ESI,

GROSS SALARY and also NET SALARY from emp table who are working as a "MANAGER"?

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

EX:

waq to display all employees salaries after decrement of 10%?

```
SQL> SELECT ENAME,SAL BEFORE_DECREMENT,SAL-SAL*0.1 AS AFTER_DECREMENT
FROM EMP;
```

Relational operators:

==========

- comparing a specifc column values with user defined condition in the query.

syntax:

=======

where <column name> <relational operator> <value>;

Ex:

waq to display list of employees who are joined after 1981?

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

Ex:

waq to display list of employees who are joined before 1981?

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

Logical operators:

==========

- to check more than one condition in the query.
- AND,OR,NOT.

AND operator:

==========

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

Cond1	Cond2	
T	T	===> T
T	F	===> F
F	T	===> F
F	F	===> F

syntax:
======

where <condition1> and <condition2>

EX:
waq to fetch employees details whose name is SMITH and working as a "CLERK"?
SQL> SELECT * FROM EMP WHERE ENAME='SMITH' AND JOB='CLERK';

OR operator:
============

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

Cond1	Cond2	
T	T	===> T
T	F	===> T
F	T	===> T
F	F	===> F

syntax:
======

where <condition1> or <condition2>

Ex:
waq to fetch employees details who are working as a "MANAGER","PRESIDENT"?
SQL> SELECT * FROM EMP WHERE JOB='MANAGER' OR JOB='PRESIDENT';

NOT operator:
============

- it return all values except the given conditional values in the query.

syntax:
======

where not <condition1> and not <condition2>

Ex:
waq to display employees who are not working as a "MANAGER","PRESIDENT"?
SQL> SELECT * FROM EMP WHERE NOT JOB='MANAGER' AND NOT JOB='PRESIDENT';

Set operators:
============

- set operators are used to combined the results of two select queries.

syntax:
======

<select query1> <set operator> <select query2>;

Ex:

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

UNION:
=======

- to combined two sets values without duplicates.

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

UNION ALL:
==========

- to combined two sets values with duplicates.

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

INTERSECT:
==========

- it return common values from both sets.

A I B = { 30 }

MINUS:
=======

- it return uncommon values from the left set but not right set.

A - B = {10,20}

B - A = {40,50}

DEMO_TABLES:
=============

SQL> SELECT * FROM HYD_BRANCH;

EID	ENAME	SAL
1	SMITH	85000
2	ALLEN	68000
3	JONES	34000

SQL> SELECT * FROM PUNE_BRANCH;

EID	ENAME	SAL
1	SMITH	85000
4	MILLER	55000

EX:
waq to display the list of employees details who are working in HYD branch but not in PUNE branch?
SQL> SELECT * FROM HYD_BRANCH MINUS SELECT * FROM PUNE_BRANCH;

EX:
waq to display employees NAMES who are working in both branches?
SQL> SELECT ENAME FROM HYD_BRANCH INTERSECT SELECT ENAME FROM PUNE_BRANCH;

EX:
waq to display all employees details who are working in the organization?
SQL> SELECT * FROM HYD_BRANCH UNION ALL SELECT * FROM PUNE_BRANCH;(including duplicate rows)
SQL> SELECT * FROM HYD_BRANCH UNION SELECT * FROM PUNE_BRANCH;(excluding duplicate rows)

BASIC RULES:
==========
1. no.of columns and order of the columns should be same in both select queries.
2. those datatypes of columns must be match in both select queries.

Special operators:
==========
IN operator:
==========
- comparing the list of values with same datatype.

syntax:
=======
where <column name> IN (v1,v2,v3,.....);
where <column name> NOT IN (v1,v2,v3,.....);

EX:
waq to list out the employees who are working as a "CLERK","SALESMAN","MANAGER"?
SQL> SELECT * FROM EMP WHERE JOB IN('CLERK','SALESMAN','MANAGER');

EX:
waq to list out the employees who are not working as a "CLERK","SALESMAN","MANAGER"?
SQL> SELECT * FROM EMP WHERE JOB NOT IN('CLERK','SALESMAN','MANAGER');

BETWEEN operator:
==========
- comparing a particular range value.

syntax:
=======
where <column name> BETWEEN <low value> AND <high value>;
where <column name> NOT BETWEEN <low value> AND <high value>;

NOTE:
======
- BETWEEN operator is always return all values including source and destination value from the given range value.

EX:
waq to display list employees details who are joined in 1981?
SQL> SELECT * FROM EMP WHERE HIREDATE BETWEEN '01-JAN-1981' AND '31-DEC-1981';

EX:
waq to display list employees details who are not joined in 1981?
SQL> SELECT * FROM EMP WHERE HIREDATE NOT BETWEEN '01-JAN-1981' AND '31-DEC-1981';

IS NULL operator:
==========
- comparing NULLS in a table.

syntax:
=======
where <column name> IS NULL;
where <column name> IS NOT NULL;

Ex:
waq to display employees whose commission is empty / null / undefined?
SQL> SELECT * FROM EMP WHERE COMM IS NULL;

Ex:
waq to display employees whose commission is not empty / not null / defined?
SQL> SELECT * FROM EMP WHERE COMM IS NOT NULL;

What is NULL?
==========
- it is an empty / a unknown value / a undefined value in database.
- NULL != 0 and NULL != space.

FUNCTIONS IN ORACLE SQL:
==========
DATE FUNCTIONS:
==========
SYSDATE:
========
- it return the current date information of the system.

syntax:
======
sysdate

EX:
SQL> SELECT SYSDATE FROM DUAL;
17-FEB-25

SQL> SELECT SYSDATE AS CUR_DATE,SYSDATE+5 AS NEW_DATE FROM DUAL;

CUR_DATE NEW_DATE
---------- ----------
17-FEB-25 22-FEB-25

SQL> SELECT SYSDATE AS CUR_DATE,SYSDATE-5 AS PRV_DATE FROM DUAL;

CUR_DATE PRV_DATE
---------- ----------
17-FEB-25 12-FEB-25

ADD_MONTHS():
==========
- to add / subtract no.of months from / to the given date expression.

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

Ex:
SQL> SELECT ADD_MONTHS(SYSDATE,3) FROM DUAL;
17-MAY-25

SQL> SELECT ADD_MONTHS(SYSDATE,-3) FROM DUAL;
17-NOV-24

Ex:
SQL> CREATE TABLE PRODUCT(PCODE NUMBER(4),PNAME VARCHAR2(10),MFG_DATE DATE,EXP_DATE DATE);

SQL> INSERT INTO PRODUCT(PCODE,PNAME,MFG_DATE)VALUES(1001,'P1','12-JUN-2017');
SQL> INSERT INTO PRODUCT(PCODE,PNAME,MFG_DATE)VALUES(1002,'P2','03-MAY-2022');
SQL> COMMIT;

SQL> UPDATE PRODUCT SET EXP_DATE=ADD_MONTHS(MFG_DATE,24);
SQL> SELECT * FROM PRODUCT;

LAST_DAY():
==========
- it return the last day from the month in the date expression.

syntax:
=======
last_day(date)

Ex:
SQL> SELECT LAST_DAY(SYSDATE) FROM DUAL;
28-FEB-25

SQL> SELECT LAST_DAY('29-SEP-2024') FROM DUAL;
30-SEP-24

MONTHS_BETWEEN():
==========
- it return no.of months from the given two dates.

syntax:
=======
months_between(date1,date2)

NOTE:
======
- Here date1 is always greater than to date2 otherwise it returns (-ve) sign value.

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

MULTIPLE ROW FUNCTIONS:
==========

- these functions are also called as "grouping functions / aggregative functions" in database.

SUM():
======

- it return total value.

syntax:
=======

sum(column name)

EX:
SQL> SELECT SUM(SAL) AS TOTAL_SALARY FROM EMP;
SQL> SELECT SUM(SAL) AS TOTAL_SALARY FROM EMP WHERE DEPTNO=30;

AVG():
=====

- it return average value.

syntax:
=======

avg(column name)

EX:
SQL> SELECT AVG(SAL) AS AVG_SALARY FROM EMP;
SQL> SELECT AVG(SAL) AS AVG_SALARY FROM EMP WHERE DEPTNO=30;

MIN():
=====

- it return minimum value.

syntax:
=======

min(column name)

EX:
SQL> SELECT MIN(SAL) FROM EMP;
SQL> SELECT MIN(SAL) FROM EMP WHERE JOB='MANAGER';
SQL> SELECT MIN(HIREDATE) FROM EMP;

MAX():
=====

- it return maximum value.

syntax:
=======
max(column name)

EX:
SQL> SELECT MAX(SAL) FROM EMP;
SQL> SELECT MAX(SAL) FROM EMP WHERE JOB='SALESMAN';

COUNT():
========
- it again three formats.
i) count(*)
ii) count(column name)
iii) count(distinct <column name>)

i) count(*):
=========
- counting all rows including duplicate and nulls from a table.
syntax:
=======
count(*)

EX:
SQL> SELECT COUNT(*) FROM EMP;----------> 14
SQL> SELECT COUNT(*) FROM EMP WHERE JOB='MANAGER';----------> 3

EX:
waq to find out no.of employees from emp table who joined in 1982?
SQL> SELECT COUNT(*) FROM EMP WHERE TO_CHAR(HIREDATE,'YYYY')='1982';------2

Ex:
waq to find out no.of employees from emp table who are joined on weekends?
SQL> SELECT COUNT(*) FROM EMP WHERE TO_CHAR(HIREDATE,'DY')
IN('SAT','SUN');------2

ii) count(column name):
=====================
- counting duplicate values but not nulls from a specific column.
syntax:
======
count(column name)

EX:
SQL> SELECT COUNT(JOB) FROM EMP;----------> 14
SQL> SELECT COUNT(COMM) FROM EMP;----------> 4
SQL> SELECT COUNT(MGR) FROM EMP;----------> 13

iii) count(distinct <column name>):
==========
- counting unique values only.(i.e no duplicates & no nulls)

syntax:
=======
count(distinct <column name>):

EX:
SQL> SELECT COUNT(DISTINCT JOB) FROM EMP;----------> 5
SQL> SELECT COUNT(DISTINCT MGR) FROM EMP;----------> 6
SQL> SELECT COUNT(DISTINCT JOB) FROM EMP WHERE JOB='MANAGER';--------> 1
==========
CLAUSES:
========
- it is a statement which is used to add to sql query for providing some additional facilities are
"filtering rows,sorting values,grouping data" based on column / columns automatically.
- oracle supports the following clauses are,
> WHERE
> ORDER BY
> GROUP BY
> HAVING
syntax:
======
<sql query>+<clause statement>;

WHERE:
=======
- filtering rows before grouping data in a table.
- it is used in "SELECT,UPDATE,DELETE" commands only.
syntax:
=======
where <filtering condition>;

EX:
SQL> SELECT * FROM EMP WHERE EMPNO=7788;
SQL> UPDATE EMP SET SAL=5500 WHERE JOB='MANAGER';

SQL> DELETE FROM EMP WHERE DEPTNO=10;

ORDER BY:
==========
- to arrange a specific column values in ascending or descending order.
- by default ORDER BY clause will arrange the values in ascendning order only.
- it is used in "SELECT" command only.

syntax:
=======
select * from <table name> order by <column name1> <asc/desc>,<column name2> <asc/desc>,.....;

EX:
waq to arrange employees salaries in ascending order?
SQL> SELECT * FROM EMP ORDER BY SAL;
(OR)
SQL> SELECT * FROM EMP ORDER BY SAL ASC;

EX:
waq to display employees who are working under 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 the employees deptno's in ascending order and their salaries in descending order
from emp table?
SQL> SELECT * FROM EMP ORDER BY DEPTNO ASC,SAL DESC;

ORDER BY with NULL clauses:
==========
- there are two null clauses in database.
i) NULLS FIRST
ii) NULLS LAST

i) NULLS FIRST:
==========
- by default order by clause on NULLS in ascending order,
Ex:
SQL> SELECT * FROM EMP ORDER BY COMM;
First : Values
Later : Nulls

- To overcome the above order and make nulls in first in ascending order then we should use

"NULLS FIRST" clause in the query like below,

Solution:

========

SQL> SELECT * FROM EMP ORDER BY COMM NULLS FIRST.

First : Nulls

Later : Values

ii) NULLS LAST:

==========

- by default order by clause on NULLS in descending order,

Ex:

SQL> SELECT * FROM EMP ORDER BY COMM DESC;

First : Nulls

Later : Values

- To overcome the above order and make nulls in last in descending order then we should use

"NULLS LAST" clause in the query like below,

Solution:

========

SQL> SELECT * FROM EMP ORDER BY COMM DESC NULLS LAST.

First : Values

Later : Nulls

GROUP BY:

==========

- to make groups based on column / columns from a table.

- when we use "GROUP BY" clause we must use "grouping functions / aggregative functions"

to get final result.

- this clause is used in "SELECT" command only.

syntax:

=======

select <column name1>,<column name2>,.....,<grouping function name1>,.....
from <table name> group by <column name1>,<column name2>,.....;

Ex:

waq to find out no.of employees are working in the organization?

SQL> SELECT COUNT(*) AS NO_OF_EMPLOYEES FROM EMP;

NO_OF_EMPLOYEES

Ex:
waq to find out no.of employees are working under deptno is 20?
SQL> SELECT COUNT(*) AS NO_OF_EMPLOYEES FROM EMP WHERE DEPTNO=20;

```
NO_OF_EMPLOYEES
------------------------------
             5
```

Ex:
waq to find out no.of employees are working under each deptno wise?
SQL> SELECT DEPTNO,COUNT(*) AS NO_OF_EMPLOYEES FROM EMP GROUP BY DEPTNO ORDER BY DEPTNO;

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

Ex:
waq to display average and total expendature on each deptno wise?
SQL> SELECT DEPTNO,AVG(SAL) AS AVG_EXP,SUM(SAL) AS TOT_EXP FROM EMP
GROUP BY DEPTNO ORDER BY DEPTNO;

EX:
waq to display minimum and maximum expendature from each job wise?
SQL> SELECT JOB,MIN(SAL) AS MIN_EXP,MAX(SAL) AS MAX_EXP FROM EMP GROUP BY JOB;

EX:
waq to find out no.of employees are working under each deptno along with their job?
SQL> SELECT DEPTNO,JOB,COUNT(*) AS NO_OF_EMPLOYEES FROM EMP
GROUP BY DEPTNO,JOB ORDER BY DEPTNO;

EX:
waq to find out no.of employees from each year wise?
SQL> SELECT TO_CHAR(HIREDATE,'YYYY') AS YEARS,COUNT(*) AS NO_OF_EMPLOYEES FROM EMP
GROUP BY TO_CHAR(HIREDATE,'YYYY');

HAVING:
=======
- filtering rows after grouping data in a table.
- it can be used after "group by" clause only.

syntax:
=======
select <column name1>,<column name2>,.....,<grouping function name1>,.....
from <table name> group by <column name1>,<column name2>,.....having<filtering condition>;

EX:
waq to display deptno's from emp table in which deptno the no.of employees are less than 5?
SQL> SELECT DEPTNO,COUNT(*) FROM EMP GROUP BY DEPTNO HAVING COUNT(*)<5;

DEPTNO	COUNT(*)
10	3

EX:
waq to display total expendature of years from emp table in which year the total expendature is more than 15000?
SQL> SELECT TO_CHAR(HIREDATE,'YYYY') AS YEARS,SUM(SAL) AS TOT_EXP FROM EMP
2 GROUP BY TO_CHAR(HIREDATE,'YYYY') HAVING SUM(SAL)>15000;

YEAR	TOT_EXP
1981	22825

WHERE vs HAVING:
================

WHERE	HAVING
=======	========
1. filtering rows before grouping data in the table.	1. filtering rows after grouping data in the table.
2. WHERE condition will work on each individual row wise.	2. HAVING condition will work on each group of rows wise.
3. it does not supports "aggregative functions".	3. it supports "aggregative functions".

4. it will use before "group by" clause.	4. it will use after "group by" clause.
5. without "group by" clause WHERE Clause can be worked.	5. without "group by" clause HAVING clause can not be worked.

Using all clauses in a single SELECT statement:
==========
syntax:
=======
select <column name1>,<column name2>,.....,<grouping function name1>,.....
from <table name> [where <filtering condition>
group by <column name1>,<column name2>,.....
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
```

==========
JOINS:
======
- In RDBMS data can be organized in multiple tables.from those multiple tables if we want
to retrieve the required data/information then we use a technique is known as "JOINS".
- Joins are used to retrieve data from multiple tables at a time.
- Oracle supports the following types of joins those are,
    1. Inner joins
        - Equi join
        - Non-equi join
        - Self join
    2. Outer joins
        - Left outer join

- Right outer join
- Full outer join

3. Cross join / Cartisean join
4. Natural join

- we can write join statements in two formats in database.
  i) NON-ANSI format
  ii) ANSI format

i) NON-ANSI format:
==========

- these statements are not a portability statements.
  i.e we cannot move a join statement from one platform to another platform.
- here join conditions are preparing with "WHERE" clause.

syntax:
======
select * from <table name1>,<table name2> where <join condition>;

ii) ANSI format:
==========

- these statements are portability statements.
  i.e we can move a join statement from one platform to another platform.
- here join conditions are preparing with "ON" clause.

syntax:
======
select * from <table name1> <join key> <table name2> on <join condition>;

1. Inner joins:
==========

Equi join:
========

- retrieving data from multiple tables based on an " = " operator condition is known as EQUI JOIN.
- when we use equi join we will maintain at least one common column name(optional) in both tables and also those columns datatypes must be match (mandatory).
- relationship between tables are optional for joins.
- it always retrieving matching rows only.

syntax for NON-ANSI:
==========

where <table name1>.<common column name> = <table name2>.<common column name>;

syntax for ANSI:
==========
on <table name1>.<common column name> = <table name2>.<common column name>;

DEMO_TABLES:
==========
SQL> SELECT * FROM BRANCH;

```
    BCODE BNAME
     ---------- ----------
     1021   CSE
     1022   EEE
     1023   IT
```

SQL> SELECT * FROM STUDENT;

```
     STID        SNAME         BCODE
    ----------      ----------      ----------
       1          SMITH         1021
       2           ALLEN        1022
       3          JONES         1022
       4          SCOTT
```

EX:
waq to display students and their confirmed branch details from the given tables?
NON_ANSI:
==========
SQL> SELECT * FROM STUDENT,BRANCH WHERE STUDENT.BCODE=BRANCH.BCODE;
(OR)
SQL> SELECT STID,SNAME,BNAME FROM STUDENT S,BRANCH B WHERE
S.BCODE=B.BCODE;

ANSI:
=====
SQL> SELECT * FROM STUDENT JOIN BRANCH ON STUDENT.BCODE=BRANCH.BCODE;
(OR)
SQL> SELECT STID,SNAME,BNAME FROM STUDENT S INNER JOIN BRANCH B ON
S.BCODE=B.BCODE;

RULE FOR JOINS:
==========

- a row in a table comparing with multiple rows of another table.

EX:
waq to display students,branch details from the given tables who are selected "EEE" branch?
SQL> SELECT STID,SNAME,BNAME FROM STUDENT S INNER JOIN BRANCH B ON S.BCODE=B.BCODE WHERE BNAME='EEE';

(OR)

SQL> SELECT STID,SNAME,BNAME FROM STUDENT S INNER JOIN BRANCH B ON S.BCODE=B.BCODE AND BNAME='EEE';

EX:
waq to display employees and their working location from emp,dept tables who are working in "DALLAS"?
SQL> SELECT ENAME,LOC FROM EMP E JOIN DEPT D ON E.DEPTNO=D.DEPTNO WHERE LOC='DALLAS';

EX:
waq to display the total expendature of each department names wise from emp,dept tables?
SQL> SELECT DNAME,SUM(SAL) AS TOT_EXP FROM EMP E JOIN DEPT D
2 ON E.DEPTNO=D.DEPTNO GROUP BY DNAME;

EX:
waq to display the total expendature of each department names wise from emp,dept tables along
with deptno?
SQL> SELECT D.DEPTNO,DNAME,SUM(SAL) AS TOT_EXP FROM EMP E JOIN DEPT D
2 ON E.DEPTNO=D.DEPTNO GROUP BY D.DEPTNO,DNAME ORDER BY DEPTNO;

EX:
waq to display total expendature of department names from emp,dept tables in which department
total expendatures are more than 10000?
SQL> SELECT DNAME,SUM(SAL) AS TOT_EXP FROM EMP E JOIN DEPT D
2 ON E.DEPTNO=D.DEPTNO GROUP BY DNAME HAVING SUM(SAL)>10000;

Non-equi join:
==========

- retrieving data from multiple tables based on any condition except an " = " operator condition.
- in this join we will use the following operators are
<,>,<=,>=,!=,Between,And,Or,.....etc

DEMO_TABLES:
==========
SQL> SELECT * FROM TEST11;

```
      SNO       NAME
    ----------  ----------
       1        ALLEN
       2        JONES
```

SQL> SELECT * FROM TEST12;

```
      SNO     SAL
      ------  ----------
       1      23000
       3      45000
```

EX:
NON-ANSI:
==========
SQL> SELECT * FROM TEST11 T1,TEST12 T2 WHERE T1.SNO<T2.SNO;

ANSI:
=====
SQL> SELECT * FROM TEST11 T1 JOIN TEST12 T2 ON T1.SNO<T2.SNO;
SQL> SELECT * FROM TEST11 T1 JOIN TEST12 T2 ON T1.SNO<=T2.SNO;
SQL> SELECT * FROM TEST11 T1 JOIN TEST12 T2 ON T1.SNO>T2.SNO;
SQL> SELECT * FROM TEST11 T1 JOIN TEST12 T2 ON T1.SNO>=T2.SNO;
SQL> SELECT * FROM TEST11 T1 JOIN TEST12 T2 ON T1.SNO!=T2.SNO;

EX:
waq to display employees whose salary is between low salary and high salary from emp,salgrade tables?
SQL> SELECT ENAME,SAL,LOSAL,HISAL,GRADE FROM EMP JOIN SALGRADE
       ON SAL BETWEEN LOSAL AND HISAL;
                (OR)
SQL> SELECT ENAME,SAL,LOSAL,HISAL,GRADE FROM EMP JOIN SALGRADE
  2 ON (SAL>=LOSAL) AND (SAL<=HISAL);

EX:
waq to display employees who are comes under grade 3 from emp,salgrade tables?
SQL> SELECT ENAME,SAL,LOSAL,HISAL,GRADE FROM EMP JOIN SALGRADE
  2   ON SAL BETWEEN LOSAL AND HISAL WHERE GRADE=3;

2. Outer joins:
==========

- retrieving matching rows and also unmatching rows from multiple tables.
- it again classified into 3 types.

Left outer join:
==========

- retrieving matching rows from both tables and unmatching rows from the left side table only.

EX:
ANSI:
=====

```
SQL> SELECT * FROM STUDENT S LEFT OUTER JOIN BRANCH B ON
B.BCODE=S.BCODE;
```

NON-ANSI:
=========

- when we want to implement outer joins in non-ansi format then we must use a join operator is (+).

```
SQL> SELECT * FROM STUDENT S,BRANCH B WHERE S.BCODE=B.BCODE(+);
```

- when we apply join operator (+) on a table then that table will show matching rows but not unmatching rows.

Right outer join:
==========

- retrieving matching rows from both tables and unmatching rows from the right side table only.

EX:
ANSI:
=====

```
SQL> SELECT * FROM STUDENT S RIGHT OUTER JOIN BRANCH B ON
B.BCODE=S.BCODE;
```

NON-ANSI:
=========

```
SQL> SELECT * FROM STUDENT S,BRANCH B WHERE S.BCODE(+)=B.BCODE;
```

Full outer join:
==========

- it is a combination of left outer join and right outer join.
- we are retrieving all matching rows and also all unmatching rows from multiple tables.

ANSI:
=====

```
SQL> SELECT * FROM STUDENT S FULL OUTER JOIN BRANCH B ON
B.BCODE=S.BCODE;
```

NON-ANSI:
=========

- to implement full outer join in non-ansi format then we should use a set operator is "UNION" in between left outer join and right outer join non-ansi statements.

Solution:
========

```
SQL> SELECT * FROM STUDENT S,BRANCH B WHERE S.BCODE=B.BCODE(+)
  2  UNION
  3  SELECT * FROM STUDENT S,BRANCH B WHERE S.BCODE(+)=B.BCODE;
```

CROSS JOIN / CARTESIEAN JOIN:
============================

- joining two or more than two tables without any condition.
- in cross join mechanism each row of a table will join with each row of another table. for example a table is having (m) no.of rows and another table is having (n) no.of rows then the result is (mXn) rows.

EX:

ANSI:

```
SQL> SELECT * FROM STUDENT CROSS JOIN BRANCH;
```

NON-ANSI:

```
SQL> SELECT * FROM STUDENT,BRANCH;
```

EX:

```
SQL> CREATE TABLE LIST1(SNO NUMBER(3),INAME VARCHAR2(10),PRICE
NUMBER(6,2));

SQL> INSERT INTO LIST1 VALUES(1,'BURGER',85);
SQL> INSERT INTO LIST1 VALUES(2,'PIZZA',150);
SQL> COMMIT;

SQL> CREATE TABLE LIST2(SRNO NUMBER(3),INAME VARCHAR2(10),PRICE
NUMBER(6,2));
```

```
SQL> INSERT INTO LIST2 VALUES(101,'PEPSI',15);
SQL> INSERT INTO LIST2 VALUES(102,'COKE',25);
SQL> COMMIT;

SQL> SELECT L1.INAME,L1.PRICE,L2.INAME,L2.PRICE,
       L1.PRICE+L2.PRICE AS TOTAL_BILL_AMOUNT
       FROM LIST1 L1 CROSS JOIN LIST2 L2;

INAME          PRICE   INAME   PRICE  TOTAL_BILL_AMOUNT
----------     ----------  ----------  ---------- -----------------------------------
BURGER           85      PEPSI    15          100
```

NATURAL JOIN:
==========

- it is a similar to equi join for retrieving matching rows from multiple tables.

NATURAL JOIN vs EQUI JOIN:
==========

NATURAL JOIN	EQUI JOIN
1. preparing a join condition by implicitly.	1. preparing a join condition by explicitly.
2. common column name is mandatory.	2. common column name is optional.
3. to avoid duplicate columns from the result set(output) automatically.	3. it does not avoid duplicate columns from the result set automatically.
4. preparing join condition based on "USING" clause.	4. preparing join condition based on " WHERE / ON " clause.

EX:

```
SQL> SELECT * FROM STUDENT S NATURAL JOIN BRANCH B;
```

SELF JOIN:
==========

- joining a table by itself is known as SELF JOIN.

(or)

- comparing a table data by itself is called as SELF JOIN.
- it can be implemented on a single table only.
- when we use self join we must create alias names on a table otherwise we cannot implement self join.

- when we create alias name on a table internally oracle server is preparing a virtual table
on each alias name under buffer memory.
- we can create any no.of alias names on a single table but each alias name should be different.

Why we need SELF JOIN?
==========
- there are two cases to use self join technique in database.
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.

Examples on comparing a single column values by itself with in the table:
==========
EX:
waq to display employees who are working in the same location where the employee "ALLEN" is also
working?

DEMO_TABLE:
==========
SQL> SELECT * FROM TEST;

ENAME LOC
---------- ----------
ALLEN HYD
JONES PUNE
ADAMS CHENNAI
MILLER HYD

Solution:
==========
SQL> SELECT T1.ENAME,T1.LOC FROM TEST T1 JOIN TEST T2 ON T1.LOC=T2.LOC AND T2.ENAME='ALLEN';

ENAME LOC
---------- ----------
ALLEN HYD
MILLER HYD
EX:
waq to display employees whose salary is same as the employee MARTIN salary?
SQL> SELECT E1.ENAME,E1.SAL FROM EMP E1 JOIN EMP E2 ON
2 E1.SAL=E2.SAL AND E2.ENAME='MARTIN';

Examples on comparing two different columns values to each other with in the table:
==========
Ex:
waq to display MANAGERS and their EMPLOYEES from emp table?

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

EX:
waq to display employees who are working under KING manager?

```
SQL> SELECT M.ENAME AS MANAGERS,E.ENAME AS EMPLOYEES FROM EMP E JOIN
EMP M
  2  ON M.EMPNO=E.MGR WHERE M.ENAME='KING';
```

EX:
waq to display manager of the employess JONES?

```
SQL> SELECT M.ENAME AS MANAGERS,E.ENAME AS EMPLOYEES FROM EMP E JOIN
EMP M
  2  ON M.EMPNO=E.MGR WHERE E.ENAME='JONES';
```

EX:
waq to display employees who are joined before their MANAGER from emp table?

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

EX:
waq to display employees whose salary is more than to their manager salary?

```
SQL> SELECT E.ENAME AS EMPLOYEE,E.SAL AS EMP_SAL
  2  ,M.ENAME AS MANAGER,M.SAL AS MGR_SAL FROM
  3  EMP E JOIN EMP M ON M.EMPNO=E.MGR AND E.SAL>M.SAL;
```

==========
How to join more than two tables:
==========
syntax for NON-ANSI format:
==========

```
select * from <table name1>,<table name2>,<table name3>,.....
where <join condition1> and <join condition2> and <join condition3> and.....;
```

syntax for ANSI format:
==========
select * from <table name1> <join key> <table name2> on <join condition1>
<join key> <table name3> on <join condition2>
<join key> <table name4> on <join condition3>
.....
.....
<join key> <table name N> on <join condition N-1>;

DEMO_TABLES:
==========
SQL> SELECT * FROM COURSE;

CID	CNAME
1	ORACLE
2	JAVA

SQL> SELECT * FROM STUDENTS;

SNAME	CID
SCOTT	1
MILLER	

SQL> SELECT * FROM REGISTER;

REGDATE	CID
23-FEB-25	1
24-FEB-25	

EQUI JOIN:
==========
NON-ANSI:
SQL> SELECT * FROM STUDENTS S,COURSE C,REGISTER R WHERE S.CID=C.CID AND C.CID=R.CID;

ANSI:
SQL> SELECT * FROM STUDENTS S JOIN COURSE C ON S.CID=C.CID JOIN REGISTER R ON C.CID=R.CID;
==========

CONSTRAINTS:
==========

- constraints are used to enforce / restrict unwanted data into columns.
- oracle supports the following six types of constraints.
    i) UNIQUE
    ii) NOT NULL
    iii) CHECK
    iv) PRIMARY KEY
    v) FOREIGN KEY
    vi) DEFAULT
- constraint can be defined at two levels.
    1. column level:
    ==========
        - In this level constraint can be defined on each individual column wise.
    syntax:
    =======
    create table <table name>(<column name1> <datatype>[size] <constraint type>,
    <column name2> <datatype>[size] <constraint
type>,.....);

    2. table level:
    ==========
        - In this level constraint can be defined after all columns definitions i.e the end of the table definition.
    syntax:
    =======
    create table <table name>(<column name1> <datatype>[size],
    <column name2> <datatype>[size],.....,<constraint type>(<col1>,<col2>,.....));

i) UNIQUE:
=========

- to restricted duplicates but allowed NULLs.

EX:
column level:
==========
SQL> CREATE TABLE TEST1(SNO NUMBER(3) UNIQUE,NAME VARCHAR2(10) UNIQUE);

TESTING:
SQL> INSERT INTO TEST VALUES(1,'A'); ----------ALLOWED
SQL> INSERT INTO TEST VALUES(1,'A'); ----------NOT ALLOWED
SQL> INSERT INTO TEST VALUES(NULL,NULL);---ALLOWED

table level:
=========
SQL> CREATE TABLE MATCH_TABLE(TEAM1 VARCHAR2(10),TEAM2 VARCHAR2(10),UNIQUE(TEAM1,TEAM2));

TESTING:
SQL> INSERT INTO MATCH_TABLE VALUES('IND','AUS');------ALLOWED
SQL> INSERT INTO MATCH_TABLE VALUES('IND','AUS');-----NOT ALLOWED
SQL> INSERT INTO MATCH_TABLE VALUES('IND','SA');-------ALLOWED
SQL> INSERT INTO MATCH_TABLE VALUES(NULL,NULL);----ALLOWED

NOT NULL:
==========
- to restricted NULLs but allowed duplicates.
- it can not be defined at table level.

EX:
column level:
===========
SQL> CREATE TABLE TEST2(NAME VARCHAR2(10)NOT NULL,LOC VARCHAR2(10) NOT NULL);

TESTING:
SQL> INSERT INTO TEST2 VALUES('SMITH','HYD');------ALLOWED
SQL> INSERT INTO TEST2 VALUES('SMITH','HYD');-----ALLOWED
SQL> INSERT INTO TEST2 VALUES(NULL,NULL);------NOT ALLOWED

CHECK:
=======
- to check the values with user defined condition before accepting values into column.

EX:
column level:
============
```
SQL> CREATE TABLE REGISTER_FORM
  2  (
  3  REG_NO NUMBER(5) UNIQUE NOT NULL,
  4  CANDIDATE_NAME VARCHAR2(10) NOT NULL,
  5  ENTRY_FEE NUMBER(6,2) NOT NULL CHECK(ENTRY_FEE=500),
  6  AGE_LIMIT NUMBER(3) NOT NULL CHECK(AGE_LIMIT BETWEEN 18 AND 30),
  7  LOC VARCHAR2(10) NOT NULL CHECK(LOC IN('HYD','MUMBAI','DELHI'))
  8  );
```

TESTING:
SQL> INSERT INTO REGISTER_FORM
VALUES(10001,'SCOTT',499,17,'HYDERABAD');----NOT ALLOWED
SQL> INSERT INTO REGISTER_FORM
VALUES(10001,'SCOTT',500,18,'HYD');----------ALLOWED

table level:
==========
SQL> CREATE TABLE TEST3(SNAME VARCHAR2(10),SFEE NUMBER(8,2),
CHECK(SNAME=LOWER(SNAME) AND SFEE=5000));

TESTING:
SQL> INSERT INTO TEST3 VALUES('JAMES',8000);----NOT ALLOWED
SQL> INSERT INTO TEST3 VALUES('james',5000);------ALLOWED

PRIMARY KEY:
=============
- it is a combination of UNIQUE and NOT NULL constraint.
- by using primary key we can restrict duplicates and nulls at a time.
- a table is having only one primary key.
- a primary key is also called as "CANDIDATE KEY" in database.

EX:
column level:
============
SQL> CREATE TABLE PRODUCTS(PCODE NUMBER(4) PRIMARY KEY,
PNAME VARCHAR2(10) UNIQUE NOT NULL);

TESTING:
SQL> INSERT INTO PRODUCTS VALUES(1021,'P1');----------ALLOWED
SQL> INSERT INTO PRODUCTS VALUES(1021,'P1');--------NOT ALLOWED
SQL> INSERT INTO PRODUCTS VALUES(NULL,NULL);----NOT ALLOWED

COMPOSITE PRIMARY KEY(table level):
==================================
- when we apply a primary key constraint on multiple combination of columns in the table is
called as "composite primary key".
- in composite primary key mechanism each individual column is accepting duplicate values
but the combination of columns are never accept duplicate values.

EX:
SQL> CREATE TABLE BANK_DETAILS(BCODE NUMBER(4),BNAME VARCHAR2(10),BLOC
VARCHAR2(10), PRIMARY KEY(BCODE,BNAME));

TESTING:
SQL> INSERT INTO BANK_DETAILS(1021,'SBI','AMEERPET');-----ALLOWED
SQL> INSERT INTO BANK_DETAILS(1021,'SBI','MADHAPUR');----NOT ALLOWED
SQL> INSERT INTO BANK_DETAILS(1022,'SBI','MADHAPUR');----ALLOWED
SQL> INSERT INTO BANK_DETAILS(1022,'HDFC','MADHAPUR');----ALLOWED

FOREIGN KEY:
==========

- this constraint is used to establish relationship between tables for taking an identity (i.e referencial data) from one table to another table.

BASIC RULES:
==========
1. To maintain common column name(optional) in both tables.
2. Those columns datatypes must be match.
3. One table should have "primary key" and another table should have "foreign key".
4. A primary key table is called as "parent table" and a foreign key table is called s "child table" in the relationship.
5. A foreign key column is allowed the values which should be there in primary key column.
6. By default a foreign key column is allowed duplicates and nulls.

syntax:
=======
<common column name of child table> <datatype>[size] references
<parent table name>(primary key column)

EX:
SQL> CREATE TABLE DEPARTMENTS(DNO NUMBER(2) PRIMARY KEY,DNAME VARCHAR2(10));----> parent table

SQL> INSERT INTO DEPARTMENTS VALUES(10,'ORACLE');
SQL> INSERT INTO DEPARTMENTS VALUES(20,'JAVA');

SQL> CREATE TABLE EMPLOYEES(EID NUMBER(4) UNIQUE NOT NULL,ENAME VARCHAR2(10),
DNO NUMBER(2) REFERENCES DEPARTMENTS(DNO));-------child table

SQL> INSERT INTO EMPLOYEES VALUES(1021,'JAMES',10);
SQL> INSERT INTO EMPLOYEES VALUES(1022,'SCOTT',10);
SQL> INSERT INTO EMPLOYEES VALUES(1023,'ADAMS',20);
SQL> INSERT INTO EMPLOYEES VALUES(1024,'MILLER',NULL);

Updating NULLs in child table with referenced values of parent table primary key column:
==========
EX:
SQL> UPDATE EMPLOYEES SET DNO=10 WHERE EID=1024;
(or)
SQL> UPDATE EMPLOYEES SET DNO=20 WHERE EID=1024;

NOTE:
=====
- Once we established relationship between tables there are two rules are come into picture.

Rule-1(INSERTION):
==========
- we cannot insert the values into child table foreign key column which are not existing in primary key column of parent table.

i.e NO PARENT = NO CHILD

EX:
SQL> INSERT INTO EMPLOYEES VALUES(1025,'JONES',30);
ERROR at line 1:
ORA-02291: integrity constraint (MYDB2PM.SYS_C008405) violated - parent key not found

Rule-2(DELETION):
==========
- we cannot delete a row from parent table when those parent rows are having the corresponding child rows in child table without addressing to child.

EX:
SQL> DELETE FROM DEPARTMENTS WHERE DNO=10;
ERROR at line 1:
ORA-02292: integrity constraint (MYDB2PM.SYS_C008405) violated - child record found

How to address to child table:
==========
- therer are two cascade rules for addressing to child table.
  i) ON DELETE CASCADE
  ii) ON DELETE SET NULL

i) ON DELETE CASCADE:
==========

- when we delete a row from the parent table then the corresponding child rows are deleted
from child table automatically.

EX:
SQL> CREATE TABLE DEPARTMENTS1(DNO NUMBER(2) PRIMARY KEY,DNAME VARCHAR2(10));----> parent table

SQL> INSERT INTO DEPARTMENTS1 VALUES(10,'ORACLE');
SQL> INSERT INTO DEPARTMENTS1 VALUES(20,'JAVA');

SQL> CREATE TABLE EMPLOYEES1(EID NUMBER(4) UNIQUE NOT NULL,ENAME VARCHAR2(10),
DNO NUMBER(2) REFERENCES DEPARTMENTS1(DNO) ON DELETE CASCADE);-------child table

SQL> INSERT INTO EMPLOYEES1 VALUES(1021,'JAMES',10);
SQL> INSERT INTO EMPLOYEES1 VALUES(1022,'SCOTT',20);

TESTING:
SQL> DELETE FROM DEPARTMENTS1 WHERE DNO=10;-----ALLOWED

i) ON DELETE SET NULL:
==========

- when we delete a row from the parent table then the corresponding child rows foreign key
column values are conerting into NULL in child table automatically.

EX:
SQL> CREATE TABLE DEPARTMENTS2(DNO NUMBER(2) PRIMARY KEY,DNAME VARCHAR2(10));----> parent table

SQL> INSERT INTO DEPARTMENTS2 VALUES(10,'ORACLE');
SQL> INSERT INTO DEPARTMENTS2 VALUES(20,'JAVA');

SQL> CREATE TABLE EMPLOYEES2(EID NUMBER(4) UNIQUE NOT NULL,ENAME VARCHAR2(10),
DNO NUMBER(2) REFERENCES DEPARTMENTS2(DNO) ON DELETE SET NULL);-------child table

SQL> INSERT INTO EMPLOYEES2 VALUES(1021,'JAMES',10);
SQL> INSERT INTO EMPLOYEES2 VALUES(1022,'SCOTT',20);

TESTING:
SQL> DELETE FROM DEPARTMENTS2 WHERE DNO=10;-----ALLOWED

EX:
create a relationship between CUSTOMERS,PRODUCTS and ORDERS tables?

i) tbl_customer:
==========
- CID
- CNAME
- CMBNO
- CADDRESS

ii) tbl_product:
==========
- PCODE
- PNAME
- PRICE

iii) tbl_orders:
==========
- ORID
- ORDATE
- QUANTITY
- CID
- PCODE

Case-1:
=======
- if both customer and product is available then take order.

Case-2:
======
- if customer is available but not product then dont take order.

Case-3:
=======
- if product is available but not customer then dont take order.

DATADICTIONARIES / READ ONLY TABLES:
==========
- whenever we are installing oracle s/w internally system is preparing some pre-defined

tables are called as "datadictionaries".

- these datadictionaries are used for storing the information about Tables,Constraints,Views, Sequences,Indexes,Procedures,Functions,Triggers,.....etc

- we cannot perform DML operations on datadictionaries but we can perform "DESC","SELECT" commands.so that datadictionaries are also called as "READ ONLY TABLES" in oracle.

- to view all datadictionaries in oracle database then we follow the following

SYNTAX:
========
SELECT * FROM DICT; (Dictionary)

Pre-defined constraint names:
==========

- when we apply constraint on a particular column internally system is created a unique identity number for constraints for identifying constraint type.

EX:
SQL> CREATE TABLE TEST1(PCODE NUMBER(4) PRIMARY KEY,PNAME VARCHAR2(10) UNIQUE);

NOTE:
=====

- if we want to view constraint name along with column name of a specific table in oracle database then we use a datadictionary is "USER_CONS_COLUMNS".

EX:
SQL> DESC USER_CONS_COLUMNS;
SQL> SELECT COLUMN_NAME,CONSTRAINT_NAME FROM USER_CONS_COLUMNS
WHERE TABLE_NAME='TEST1';

COLUMN_NAME	CONSTRAINT_NAME
PCODE	SYS_C008349
PNAME	SYS_C008350

How to create user defined constraint name:
==========
syntax:
=======
<column name> <datatype>[size] <constraint> <user defined constraint name> <constraint type>

EX:
SQL> CREATE TABLE TEST2(PCODE NUMBER(4) CONSTRAINT PCODE_PK PRIMARY KEY,
PNAME VARCHAR2(10) CONSTRAINT PNAME_UQ UNIQUE);

SQL> SELECT COLUMN_NAME,CONSTRAINT_NAME FROM USER_CONS_COLUMNS
WHERE TABLE_NAME='TEST2';

COLUMN_NAME	CONSTRAINT_NAME
PCODE	PCODE_PK
PNAME	PNAME_UQ

How to add constraint to an existing table columns:
==========
syntax:
=======
alter table <table name> add <constraint> <constraint name> <constraint type>(column name);

EX:
SQL> CREATE TABLE PARENT(STID NUMBER(3),SNAME VARCHAR2(10),SFEE NUMBER(8,2));

I) ADDING PRIMARY KEY:
==========
SQL> ALTER TABLE PARENT ADD CONSTRAINT PK_STID PRIMARY KEY(STID);

II) ADDING UNIQUE,CHECK constraint:
==========
SQL> ALTER TABLE PARENT ADD CONSTRAINT UQ_SNAME UNIQUE(SNAME);
SQL> ALTER TABLE PARENT ADD CONSTRAINT CHK_SFEE CHECK(SFEE>=20000);

NOTE:
======
- if we want to view check constraint conditional values of a column in the table then we use a datadictionary is "USER_CONSTRAINTS".

EX:
SQL> DESC USER_CONSTRAINTS;
SQL> SELECT CONSTRAINT_NAME,SEARCH_CONDITION FROM USER_CONSTRAINTS WHERE TABLE_NAME='PARENT';

```
CONSTRAINT_NAME                SEARCH_CONDITION
------------------------------ --------------------------------------------
CHK_SFEE                       SFEE>=20000
```

III) ADDING "NOT NULL" constraint:
==========
syntax:
=======
alter table <table name> modify <column name> <constraint> <constraint name> NOT NULL;

EX:
SQL> ALTER TABLE PARENT MODIFY SNAME CONSTRAINT NN_SNAME NOT NULL;

IV) ADDING "FOREIGN KEY" references:
==========
syntax:
=======
alter table <table name> add <constraint> <constraint name> foreign key(common column of child table)
references <parent table name>(primary key column name) on delete cascade / on delete set null;

EX:
SQL> CREATE TABLE CHILD(BNAME VARCHAR2(10),STID NUMBER(4));

SQL> ALTER TABLE CHILD ADD CONSTRAINT FK_STID FOREIGN KEY(STID)
REFERENCES PARENT(STID) ON DELETE CASCADE;

How to remove constraint from an existing table columns:
==========
syntax:
=======
alter table <table name> drop <constraint> <constraint name>;

i) To drop / remove a primary key:
==========
case-1: WITHOUT RELATIONSHIP:
==========
SQL> ALTER TABLE PARENT DROP CONSTRAINT PK_STID;

case-2: WITH RELATIONSHIP:
==========
Method-1:

=========
SQL> ALTER TABLE CHILD DROP CONSTRAINT FK_STID; ----------> FIRST
SQL> ALTER TABLE PARENT DROP CONSTRAINT PK_STID;--------> LATER

Method-2:
========
SQL> ALTER TABLE PARENT DROP CONSTRAINT PK_STID CASCADE;

ii) To drop / remove UNIQUE,CHECK,NOT NULL constraint:
==
SQL> ALTER TABLE PARENT DROP CONSTRAINT UQ_SNAME;
SQL> ALTER TABLE PARENT DROP CONSTRAINT CHK_SFEE;
SQL> ALTER TABLE PARENT DROP CONSTRAINT NN_SNAME;

How to rename a constraint name:
==============================
syntax:
=======
alter table <table name> rename <constraint> <old constraint name> to <new constraint name>;

EX:
SQL> CREATE TABLE TEST4(SNO NUMBER(4) PRIMARY KEY);

SQL> SELECT COLUMN_NAME,CONSTRAINT_NAME FROM USER_CONS_COLUMNS
WHERE TABLE_NAME='TEST4';

COLUMN_NAME	CONSTRAINT_NAME
SNO	SYS_C008335

SQL> ALTER TABLE TEST4 RENAME CONSTRAINT SYS_C008335 TO SNO_PK;

COLUMN_NAME	CONSTRAINT_NAME
SNO	SNO_PK

DEFAULT constraint:
==================
- to assign a user defined default value to a column.

EX:
SQL> CREATE TABLE TEST5(NAME VARCHAR2(10),CITY VARCHAR2(10) DEFAULT 'HYD');

TESTING:
SQL> INSERT INTO TEST5(NAME,CITY)VALUES('SMITH','PUNE');
SQL> INSERT INTO TEST5(NAME)VALUES('MILLER');
SQL> SELECT * FROM TEST5;

How to add a default value to an existing table column:
==========
syntax:
=======
alter table <table name> modify <column name> default <value>;

EX:
SQL> CREATE TABLE TEST6(CNAME VARCHAR2(10),CFEE NUMBER(6,2));

TESTING:
SQL> ALTER TABLE TEST6 MODIFY CFEE DEFAULT 5000;

NOTE:
======
- if we want to view default value along with column name of a specific table in oracle then we 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='TEST6';

COLUMN_NAME	DATA_DEFAULT
CFEE	5000

How to remove a default value from an existing table column:
==========
SQL> ALTER TABLE TEST6 MODIFY CFEE DEFAULT NULL;

COLUMN_NAME	DATA_DEFAULT
CFEE	NULL

==========

TRANSACTION CONTROL LANGUAGE(TCL):
==========

What is Transaction?
==========

- to perform some operations over database.
- to control these transactions on database / database tables then we use

TCL commands are:

1) COMMIT
2) ROLLBACK
3) SAVEPOINT

1) COMMIT:
==========

- to make a transaction is permanent.
- there are two types of commit transactions.

i) Implicit commit:
==========

- these transactions are committed by system.

Ex: DDL commands.

ii) Explicit commit:
==========

- these transactios are committed by user.

Ex: DML commands.

syntax:
======

commit;

EX:

```
SQL> CREATE TABLE TEST10(PCODE NUMBER(4),PNAME VARCHAR2(10),PRICE
NUMBER(6,2));
```

TESTING:

```
SQL> INSERT INTO TEST10 VALUES(1021,'ORACLE',3500);
SQL> COMMIT;

SQL> UPDATE TEST10 SET PRICE=3000 WHERE PCODE=1021;
SQL> COMMIT;

SQL> DELETE FROM TEST10 WHERE PCODE=1021;
SQL> COMMIT;
```

(OR)

```
SQL> INSERT INTO TEST10 VALUES(1021,'ORACLE',3500);
SQL> UPDATE TEST10 SET PRICE=3000 WHERE PCODE=1021;
SQL> DELETE FROM TEST10 WHERE PCODE=1021;
SQL> COMMIT;
```

2) ROLLBACK:

==========

- to cancel a transaction.
- once we commit a transaction then we cannot rollback.

syntax:

=======

```
rollback;
```

EX:

```
SQL> INSERT INTO TEST10 VALUES(1021,'ORACLE',3500);
SQL> ROLLBACK;

SQL> UPDATE TEST10 SET PRICE=3000 WHERE PCODE=1021;
SQL> ROLLBACK;

SQL> DELETE FROM TEST10 WHERE PCODE=1021;
SQL> ROLLBACK;
```

(OR)

```
SQL> INSERT INTO TEST10 VALUES(1021,'ORACLE',3500);
SQL> UPDATE TEST10 SET PRICE=3000 WHERE PCODE=1021;
SQL> DELETE FROM TEST10 WHERE PCODE=1021;
SQL> ROLLBACK;
```

3) SAVEPOINT:

==========

- whenever we create a savepoint internally oracle server is allocating a special memory for a savepoint for storing the information about a row/rows which we want to rollback(i.e cancel) in the future.

syntax to create a savepoint:

==========

```
SAVEPOINT <pointer name>;
```

syntax to rollback a savepoint:

==========

```
ROLLBACK TO <savepoint name>:
```

DEMO_TABLE:

==========
SQL> SELECT * FROM TEST;

```
       EID ENAME
---------- ----------
         1 SMITH
         2 ALLEN
         3 JONES
         4 ADAMS
         5 MILLER
```

EX1:
SQL> DELETE FROM TEST WHERE EID=1;
SQL> DELETE FROM TEST WHERE EID=5;

SQL> SAVEPOINT P1;
Savepoint created.
SQL> DELETE FROM TEST WHERE EID=3;

CASE-1:
=======
SQL> ROLLBACK TO P1; (we cancelled 3rd row)

CASE-2:
=======
SQL> ROLLBACK / COMMIT (for remaining 1,5 rows)
==========
==========
EX2:
SQL> DELETE FROM TEST WHERE EID=1;

SQL> SAVEPOINT P1;
Savepoint created.
SQL> DELETE FROM TEST WHERE EID IN(3,5);

CASE-1:
=======
SQL> ROLLBACK TO P1; (we cancelled 3rd,5th row)

CASE-2:
=======
SQL> ROLLBACK / COMMIT (for remaining 1st row)
==========

ACID properties:
==========

- by default all relational databases are having ACID properties for manage and maintain accurate and consistency data / information in database.

> A - Automicity
> C - Consistency
> I - Isolation
> D - Durability

Automicity:
==========

- All operations in a transaction will treat as a single transaction in database.(Atomic=Single)

Ex:

Withdraw Transaction:
==========

ATM
====

step1: Insert ATM card
step2: Select Language
step3: Click on Banking option
step4: Click on Withdraw option
step5: Enter Amount : 5000
step6: Select Saving / Current account
step7: Enter Pin no : XXXX
step8: Yes(commit) / No(rollback)

Consistency:
==========

- To maintain accurate information after / before a transaction.

EX:

X-customer		Y-customer	
main bal :	7000 (before transaction)	main bal :	2000 (before transaction)
transfer :	(-) 4000 (debit amount)	recieved :	(+) 4000 (credit amount)
new bal :	3000	new bal :	6000
debit amt :	(+) 4000	credit amt :	(-)4000
main bal :	7000 (after transaction)	main bal :	2000 (after transaction)

Isolation:

========

- Every transaction is independent transaction i.e one transaction will never involved in an other transaction.

EX:

Withdraw Transaction:

==========

WD-1 : 500 ----------> Trid: XXXXXXXXX986 (unique)

WD-2 : 1000 --------> Trid: XXXXXXXXX987 (unique)

WD-3 : 400 ----------> Trid: XXXXXXXXX988 (unique)

Durability:

=========

- once we committed a transaction then we cannot rollback at any level.

==========

SUBQUERY / NESTED QUERY:

==========

- a query inside another query is known as "subquery / nested query".

syntax:

=======

select * from <table name> where <condition>(select * from.....(select * from .....));

- a subquery statement is having two more queries those are,
  - i) Inner query / Subquery / Child query
  - ii) Outer query / Main query / Parent query
- as per the execution process of subquery it again classified into two types.
  - 1. NON-CORELATED SUBQUERY:

    ==========

    - In non-corelated subquery,first inner query is executed and later outer query will execute to get the final result.
  - 2. CO-RELATED SUBQUERY:

    ==========

    - In co-related subquery,first outer query is executed and later inner query will execute to get the final result.

1. NON-CORELATED SUBQUERY:

==========

- i) Single row subquery
- ii) Multiple row subquery
- iii) Multiple column subquery
- iv) Inline view

i) Single row subquery:

==========

- when a subquery return a single value is known as SRSQ.
- in SRSQ we will use the following operators are " = , < , <= , > , >= , != (or) <> " .

EX:
waq to display employees details who are earning the first highest salary in the organization?

==========
subquery statement = outer query + inner query
==========

step1: Inner query:
==========
SQL> SELECT MAX(SAL) FROM EMP;

```
 MAX(SAL)
----------
      5000
```

step2: Outer query:
==========
SQL> SELECT * FROM EMP WHERE < return value of inner query column name>=(inner query);

step3: Subquery statement=(outer query+inner query):
==========
SQL> SELECT * FROM EMP WHERE SAL=(SELECT MAX(SAL) FROM EMP);

EX:
waq to display the senior most employees details from emp table?
SQL> SELECT * FROM EMP WHERE HIREDATE=(SELECT MIN(HIREDATE) FROM EMP);

EX:
waq to find out the second maximum salary from emp table?
SQL> SELECT MAX(SAL) FROM EMP WHERE SAL<(SELECT MAX(SAL) FROM EMP);

```
 MAX(SAL)
----------
      3000
```

EX:
waq to display employees details who are getting the second highest salary in the organization?
SQL> SELECT * FROM EMP WHERE SAL=(SELECT MAX(SAL) FROM EMP WHERE SAL<(SELECT MAX(SAL) FROM EMP));

EX:
waq to display employees whose salary is more than to the maximum salary of MANAGER?
SQL> SELECT * FROM EMP WHERE SAL>(SELECT MAX(SAL) FROM EMP WHERE JOB='MANAGER');

EX:
waq to find out 3rd highest salary from emp table?
SQL> SELECT MAX(SAL) FROM EMP WHERE SAL<
(SELECT MAX(SAL) FROM EMP WHERE SAL<
(SELECT MAX(SAL) FROM EMP));

EX:
waq to display employees details who are getting the 3rd highest salary in organization?
SQL> SELECT * FROM EMP WHERE SAL=
(SELECT MAX(SAL) FROM EMP WHERE SAL<
(SELECT MAX(SAL) FROM EMP WHERE SAL<
(SELECT MAX(SAL) FROM EMP)));

Nth	N+1
===	====
1st	2Q
2nd	3Q
3rd	4Q
30th	31Q
150th	151Q ----------> How to overcome ?

ii) Multiple row subquery:
==========
- when a subquery return more than one value is known as MRSQ.
- in this subquery we will use the following operators are "IN,ANY,ALL".

Ex:
waq to display list of employees details whose job is same as the employee "SMITH","MARTIN" jobs?
SQL> SELECT * FROM EMP WHERE JOB IN(SELECT JOB FROM EMP WHERE ENAME='SMITH' OR ENAME='MARTIN');

(OR)

SQL> SELECT * FROM EMP WHERE JOB IN(SELECT JOB FROM EMP WHERE ENAME IN('SMITH','MARTIN'));

EX:
waq to display employees details who are earning minimum,maximum salaries from emp table?
SQL> SELECT * FROM EMP WHERE SAL IN((SELECT MIN(SAL) FROM EMP),(SELECT MAX(SAL) FROM EMP));

(OR)

SQL> SELECT * FROM EMP WHERE SAL IN (SELECT MIN(SAL) FROM EMP UNION SELECT MAX(SAL) FROM EMP);

EX:
waq to display employees details who are getting the maximum salary from each job wise?
SQL> SELECT * FROM EMP WHERE SAL IN(SELECT MAX(SAL) FROM EMP GROUP BY JOB);

EX:
waq to display the senior most employees details from each deptno wise?
SQL> SELECT * FROM EMP WHERE HIREDATE IN(SELECT MIN(HIREDATE) FROM EMP GROUP BY DEPTNO);

ANY operator:
==========

- it return TRUE if any one value was satisfied with the given condition value from the list.

Ex:

If X(40) >ANY(10,20,30)

X = 09 ===> FALSE
X = 25 ===> TRUE
X = 40 ===> TRUE

ALL operator:
==========

- it return TRUE if all values are satisfied with the given condition value from the list.

Ex:

If X(40) >ALL(10,20,30)

X = 09 ===> FALSE
X = 25 ===> FALSE
X = 40 ===> TRUE

Ex:

waq to display the list of employees whose salary is more than to all SALESMAN salaries?
SQL> SELECT * FROM EMP WHERE SAL>ALL(SELECT SAL FROM EMP WHERE JOB='SALESMAN');

Ex:
waq to display the list of employees whose salary is more than to any SALESMAN salary?
SQL> SELECT * FROM EMP WHERE SAL>ANY(SELECT SAL FROM EMP WHERE JOB='SALESMAN');

ANY operator	ALL operator
X>ANY(list of values)	X>ALL(list of values)
X>=ANY(list of values)	X>=ALL(list of values)
X<ANY(list of values)	X<ALL(list of values)
X<=ANY(list of values)	X<=ALL(list of values)
X=ANY(list of values)	X=ALL(list of values)
X!=ANY(list of values)	X!=ALL(list of values)

iii) Multiple column subquery:
==========

- when we are comparing group of values by using multiple row subquery mechanism there is chance to get wrong results.
- to overcome this problem we should use a technique is known as "Multiple column subquery".
- in multiple column subquery mechanism multiple columns values of inner query is comparing with multiple columns values outer query is known as "MCSQ".

syntax:
======
select * from <table name> where(<column name1>,<column name2>,....) IN(select <column name1>,
<column name2>,.....from <table name>);

Ex:
waq to display employees details who are getting the maximum salary from each job wise?
SQL> UPDATE EMP SET SAL=1300 WHERE EMPNO=7902;
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

Solution:
=======
SQL> SELECT * FROM EMP WHERE(JOB,SAL) IN(SELECT JOB,MAX(SAL) FROM EMP GROUP BY JOB);

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

EX:
waq to display employees details whose MGR,SALARY are same as the employee MARTIN mgr,salary?
SQL> SELECT * FROM EMP WHERE(MGR,SAL) IN(SELECT MGR,SAL FROM EMP WHERE ENAME='MARTIN');

PSEUDO COLUMNS:
=================
- these columns are working just like normal columns in the table.
    i) ROWID
    ii) ROWNUM

i) ROWID:
========
- when we insert a new row into a table internally system is creating a unique row identity for each row wise on a table automatically.
- these rowid's are saved in database permanently.

EX:
SQL> SELECT ENAME,ROWID FROM EMP;

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

EX:
SQL> SELECT MIN(ROWID) FROM EMP;

MIN(ROWID)
------------------
AAASt0AAHAAAAHLAAA

SQL> SELECT MAX(ROWID) FROM EMP;

MAX(ROWID)
------------------
AAASt0AAHAAAAHLAAN
```

How to delete multiple duplicate rows except one duplicate row from a table:

==========

```
SQL> SELECT * FROM TEST;

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

Solution:

========

```
SQL> DELETE FROM TEST WHERE ROWID NOT IN(SELECT MAX(ROWID) FROM TEST
GROUP BY SNO);
SQL> SELECT * FROM TEST;
```

ii) ROWNUM:

==========

- to generate row numbers to each row wise / to each group of rows wise on a table automatically.
- these row numbers are not saved in database.so that these are temporary numbers.
- to perform "Nth","Top n" operations on table.

EX:
```
SQL> SELECT ROWNUM,EMP.* FROM EMP;
SQL> SELECT ROWNUM,ENAME,DEPTNO FROM EMP WHERE DEPTNO=10;
```

EX:
waq to fetch the 1st row from emp table by using ROWNUM?
```
SQL> SELECT * FROM EMP WHERE ROWNUM=1;
```

EX:
waq to fetch the 2nd row from emp table by using ROWNUM?
```
SQL> SELECT * FROM EMP WHERE ROWNUM=2;
no rows selected
```

NOTE:
=====
- Generally rownum is always starts with 1 for every selected row from a table.To overcome
this problem we must use the following < , <= along with MINUS operator.

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

EX:
waq to fetch the 9th row from emp table by using ROWNUM?
```
SQL> SELECT * FROM EMP WHERE ROWNUM<=9
  2  MINUS
  3  SELECT * FROM EMP WHERE ROWNUM<=8;
```

EX:
waq to fetch the rows from 11th row to 14th row from emp table by using ROWNUM?
```
SQL> SELECT * FROM EMP WHERE ROWNUM<=14
  2  MINUS
  3  SELECT * FROM EMP WHERE ROWNUM<11;
```

EX:

waq to fetch the last two rows from a table by using ROWNUM?

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

EX:

waq to fetch top 10 rows from emp table?

```
SQL> SELECT * FROM EMp WHERE ROWNUM<=10;
```

INLINE VIEW:

==========

- providing a subquery in place of table name in SELECT statement is known as "INLINE VIEW".

(or)

- providing a subquery under FROM clause in SELECT statement is known as "INLINE VIEW".

syntax:

======

```
SELECT * FROM (SUQBUERY);
```

NOTE:

=====

- In inline view mechanism,the result of inner query will act as a table for outer query.

Why we need INLINE VIEW:

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

- there are two reasons to use INLINE VIEW subquery.
    1. Generally column alias names are not allowed under WHERE clause condition.
    2. Generally order by clause is not allowed under SUBQUERY statement.

EX:

waq to display employees annual salary from a table. whose employee annual salary is more than 30000?

```
SQL> SELECT EMPNO,ENAME,SAL,SAL*12 AS ANNA_SAL FROM EMP WHERE
ANNA_SAL>30000;
ERROR at line 1:
ORA-00904: "ANNA_SAL": invalid identifier
```

Using Inline view:

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

SQL> SELECT * FROM(SELECT EMPNO,ENAME,SAL,SAL*12 AS ANNA_SAL FROM EMP) WHERE ANNA_SAL>30000;

EX:
waq to fetch top 5 highest salaries rows from emp table by using ORDER BY clause along with INLINE VIEW?
SQL> SELECT * FROM(SELECT * FROM EMP ORDER BY SAL DESC) WHERE ROWNUM<=5;

EX:
waq to fetch 5th highest salary row from emp table by using ORDER BY clause along with INLINE VIEW?
SQL> SELECT * FROM(SELECT * FROM EMP ORDER BY SAL DESC) WHERE ROWNUM<=5
  2 MINUS
  3 SELECT * FROM(SELECT * FROM EMP ORDER BY SAL DESC) WHERE ROWNUM<=4;

2. CO-RELATED SUBQUERY:
==========
- In co-related subquery,first outer query is executed and later inner query will execute to get the final result.

How to find out "Nth" high / low salary:
==========
syntax:
=======
select * from <table name> <table alias name1> where N-1=(select count(distinct <column name>)
from <table name> <table alias name2> where <table alias name2>.<column name> < / > <table alias name1>.<column name>);

Here,
< - for lowest salary
> - for highest salary

EX:
waq to find out the 1st highest salary employees details from emp table?

Solution:
========
If N=1 ===> N-1 ===> 1-1 ===> 0

SQL> SELECT * FROM EMP E1 WHERE 0=(SELECT COUNT(DISTINCT SAL) FROM EMP E2 WHERE E2.SAL>E1.SAL);

EX:
waq to find out the 3rd highest salary employees details from emp table?

Solution:
========
If N=3 ===> N-1 ===> 3-1 ===> 2

SQL> SELECT * FROM EMP E1 WHERE 2=(SELECT COUNT(DISTINCT SAL) FROM EMP E2 WHERE E2.SAL>E1.SAL);

EX:
waq to find out the 1st lowest salary employees details from emp table?

Solution:
========
If N=1 ===> N-1 ===> 1-1 ===> 0

SQL> SELECT * FROM EMP E1 WHERE 0=(SELECT COUNT(DISTINCT SAL) FROM EMP E2 WHERE E2.SAL<E1.SAL);

How to display "TOP n" high / low salaries:
==========
syntax:
=======
select * from <table name> <table alias name1> where N>(select count(distinct <column name>)
from <table name> <table alias name2> where <table alias name2>.<column name> < / > <table alias name1>.<column name>);

Here,
< - for lowest salary
> - for highest salary

EX:
waq to display top 3 highest salaries employees details from emp table?
Solution:
========
If N=3 ====> N> ===> 3>
SQL> SELECT * FROM EMP E1 WHERE 3>(SELECT COUNT(DISTINCT SAL) FROM EMP E2 WHERE E2.SAL>E1.SAL);

EX:

waq to display top 3 lowest salaries employees details from emp table?
Solution:
========
If N=3 ====> N> ===> 3>

SQL> SELECT * FROM EMP E1 WHERE 3>(SELECT COUNT(DISTINCT SAL) FROM EMP E2 WHERE E2.SAL<E1.SAL);

EXISTS operator:
==============
- it is a special operator which is used in co-related subquery only.
- it is used to check the required row is existing in a table or not.
    > if a row is exists then it returns TURE.
    > if a row is not exists then it return FALSE.

syntax:
=======
where exists(inner query);

EX:
waq to display department details in which department the employees are working?
SQL> SELECT * FROM DEPT D WHERE EXISTS(SELECT DEPTNO FROM EMP E WHERE E.DEPTNO=D.DEPTNO);

EX:
waq to display department details in which department the employees are not working?
SQL> SELECT * FROM DEPT D WHERE NOT EXISTS(SELECT DEPTNO FROM EMP E WHERE E.DEPTNO=D.DEPTNO);

VIEWS:
======
- it is a subset / virtual table of base table.
- it does not store data / information but store "SELECT query".
- the main advantage of view is :
    > providing security for data.
        i) column level:
        =============
            - In this level we are hiding the required columns from user.
        ii) row level:
        ==========
            - in this level we are hiding the required rows from users.
    > to convert complex queries into simple query.

Types of views:

==========

i) Simple views

ii) Complex views

i) Simple views:

==========

- when we created view based a single base table is known as simple view.
- by default simple views are allowed all DML operations on base table.

syntax:

=======

create view <view name> as <select query>;

EX:

create a view to access tha data from DEPT table?

SQL> SELECT * FROM DEPT;----------> Base table

SQL> CREATE VIEW V1 AS SELECT * FROM DEPT;

TESTING:

SQL> INSERT INTO V1 VALUES(50,'ORACLE','HYD');

SQL> UPDATE V1 SET LOC='PUNE' WHERE DEPTNO=50;

SQL> DELETE FROM V1 WHERE DEPTNO=50;

SQL> SELECT * FROM V1;

EX:

create a view to access EMPNO,ENAME,SAL from emp table? (Hiding columns)

SQL> CREATE VIEW V2 AS SELECT EMPNO,ENAME,SAL FROM EMP;

TESTING:

SQL> INSERT INTO V2 VALUES(1122,'YUVIN',5000);

EX:

create a view to access employees details who are working under deptno is 20? (Hiding rows)

SQL> CREATE VIEW V3 AS SELECT * FROM EMP WHERE DEPTNO=20;

SQL> SELECT * FROM V3;

EX:

create a view to access ENAME,HIREDATE and JOB details from emp table who are working as a "MANAGER"?(Hiding rows & columns)

SQL> CREATE VIEW V4 AS SELECT ENAME,HIREDATE,JOB FROM EMP WHERE JOB='MANAGER';

SQL> SELECT * FROM V4;

VIEW OPTIONS:

==========

- view can be created with two options in oracle database.
    - i) WITH CHECK OPTION
    - ii) WITH READ ONLY

i) WITH CHECK OPTION:
==========

- to restricted data on a base table through a view object.

EX:
create a view to display and accept the employees details whose salary is 3000?
SQL> CREATE VIEW V5 AS SELECT * FROM EMP WHERE SAL=3000 WITH CHECK OPTION;

TESTING:
SQL> INSERT INTO V5 VALUES(1122,'YUVIN','HR',NULL,NULL,8000,NULL,20);-------NOT ALLOWED
SQL> INSERT INTO V5 VALUES(1122,'YUVIN','HR',NULL,NULL,2000,NULL,20);------NOT ALLOWED
SQL> INSERT INTO V5
VALUES(1122,'YUVIN','HR',NULL,NULL,3000,NULL,20);-------ALLOWED

ii) WITH READ ONLY:
==========

- to restricted DML operations on base table through a view object.

EX:
create a view to restrict DML operations on DEPT table?
SQL> CREATE VIEW V6 AS SELECT * FROM DEPT WITH READ ONLY;

TESTING:
SQL> INSERT INTO V6 VALUES(50,'ORACLE','HYD');----------> NOT ALLOWED
SQL> UPDATE V6 SET LOC='PUNE' WHERE DEPTNO=10;----------> NOT ALLOWED
SQL> DELETE FROM V6 WHERE DEPTNO=30;----------> NOT ALLOWED
SQL> SELECT * FROM V6;----------> ALLOWED

ii) Complex views:
==========

- when we create a view based on :
    - > multiple tables
    - > by using group by
    - > by using having
    - > by using distinct
    - > by using set operators

> by using joins
> by using subquery
> by using grouping functions.

- by default complex views are not allowed DML operations on base table.

syntax:
========
create view <view name> as <select query>;

EX:
create a view to display sum of salaries of each deptno wise from emp table?
SQL> CREATE VIEW V7 AS SELECT DEPTNO,SUM(SAL) AS SUM_OF_SALARY FROM EMP GROUP BY DEPTNO;

TESTING:
SQL> SELECT * FROM V7;(before updating data in emp table)
SQL> UPDATE EMP SET SAL=SAL+1000 WHERE DEPTNO=10;
SQL> SELECT * FROM V7;(after update data in emp table)

- WE CANNOT PERFORM DML OPERATIONS ON BASE TABLE THROUGH COMPLEX VIEW.

NOTE:
======
- if we want to see all views in oracle database then we use "USER_VIEWS" datadictionary.

EX:
SQL> DESC USER_VIEWS;
SQL> SELECT VIEW_NAME FROM USER_VIEWS;

To see "SELECT query" in a view:
============================
SQL> SELECT TEXT FROM USER_VIEWS WHERE VIEW_NAME='V7';

How to drop a view:
================
syntax:
======
DROP VIEW <VIEWNAME>;

EX:
SQL> DROP VIEW V1;

SQL> DROP VIEW V7;

==========

SEQUENCE:

==========

- it is a database object which is used to generate sequence numbers on a specific column
in the table automatically.
- it will provide a facility on a table is called as "Auto Incremental Value".

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:

==========

- to specify starting value of sequence.here "n" is a number.

minvalue n:

=========

- to show minimum value in the sequence.here "n" is a number.

increment by n:

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

- to specify incremental value in between seqeucne numbers.here "n" is a number.

maxvalue n:

==========

- to show maximum value from sequence.here "n" is a number.

no cycle:

========

- it a default attribute of sequence object.
- when we created a sequecne object with "NO CYCLE" then the set of sequence numbers are not repeat again and again.

cycle:

========

- when we created a sequecne object with "CYCLE" then the set of sequence numbers are repeat again and again.

no cache:
=========
- it is a default attribute of sequence object.
- cache is a temporary memory.
- when we created a sequence object with "NO CACHE" then the set of sequence numbers are saved in database memory directly.so that every user request will go to database and retrieving the required data from database and send to client application.
by this reason the burdon on database will increse and degrade the performance of database.

cache n:
========
- when we created a sequence object with "CACHE" then the set of sequence numbers are saved in database memory and also the copy of data is saved in cache memory. so that now every user request will go to cache instead of database.and retrieving the required data
from cache memory and send to client application.so that we reduce the burdon on database and improve the performance of database.here "n" is the size of cache file and will create with 2kb.

NOTE:
=====
- when we generate sequence numbers on a column then we must use a pseudo column is "NEXTVAL".

NEXTVAL:
=========
- it is a pseudo column of a sequecne.
- it is used to generate next by next number.

syntax:
=======
<sequence name>.NEXTVAL

EX1:

```
SQL> CREATE SEQUENCE SQ1
  2  START WITH 1
  3  MINVALUE 1
  4  INCREMENT BY 1
  5  MAXVALUE 3;
```

TESTING:

```
SQL> CREATE TABLE TEST55(SNO NUMBER(3),NAME VARCHAR2(10));
```

```
SQL> INSERT INTO TEST55 VALUES(SQ1.NEXTVAL,'&NAME');
Enter value for name: A

SQL> /
Enter value for name: B

SQL> /
Enter value for name: C

SQL> /
Enter value for name: D
ERROR at line 1:
ORA-08004: sequence SQ1.NEXTVAL exceeds MAXVALUE and cannot be instantiated
```

ALTERING A SEQUENCE:
==========
syntax:
======
alter sequence <sequence name> <attribute name> n;

EX:

```
SQL> ALTER SEQUENCE SQ1 MAXVALUE 5;
Sequence altered.

SQL> INSERT INTO TEST55 VALUES(SQ1.NEXTVAL,'&NAME');
Enter value for name: D

SQL> /
Enter value for name: E

SQL> SELECT * FROM TEST55;
```

EX2:

```
SQL> CREATE SEQUENCE SQ2
  2  START WITH 3
  3  MINVALUE 1
  4  INCREMENT BY 1
  5  MAXVALUE 5
  6  CYCLE
  7  CACHE 2;
```

TESTING:

```
SQL> CREATE TABLE TEST56(SNO NUMBER(2),NAME VARCHAR2(10));

SQL> INSERT INTO TEST56 VALUES(SQ2.NEXTVAL,'&NAME');
Enter value for name: Q

SQL> /
.....

SQL> SELECT * FROM TEST56;
```

NOTE:
=====

- Once we created a sequence object then we can apply on multiple tables i.e reusability.

NOTE:
=====

- If we want to view all sequence objects in oracle database then use a datadictionary is "USER_SEQUENCES".

```
EX:
SQL> DESC USER_SEQUENCES;
SQL> SELECT SEQUENCE_NAME FROM USER_SEQUENCES;
```

How to drop sequence object from a table:
==========

syntax:
=======

```
drop sequence <sequence name>;
EX:
SQL> DROP SEQUENCE SQ1;
```

==========

INDEXES:
========

- it is a database object which is used to retrieve the required row/rows from a table fastly.
- database index is similar to book index page in a text book.by using book index page how we are retrieving the required topic from a text book fastly same as by using database index
we are retrieving the required row/rows from a table fastly.
- database index can be created on a particular column in the table and this column is called as "INDEXED KEY COLUMN".
- whenever we want to retrive the required row/rows from a table then we must use INDEXED KEY COLUMN under WHERE clause condition then only indexes are working.

- all databases are supporting the following two types of searching mechanisms.
    1. Table scan (default scan)
    2. Index scan

1. Table scan:
==========

- In this scan,oracle server is searching the entire table for required data.so that it takes much time to give the required data to users.

EX:
SQL> SELECT * FROM EMP WHERE SAL=3000;

```
                          SAL
                          ----------
                          800
                          1600
                          1250
                          2975
                          1250
  WHERE SAL=3000;         2850
                          2450
                          3000
                          5000
                          1500
                          1100
                          950
                          3000
                          1300
```

2. Index scan:
==========

- In this scan oracle server is searching the required row based on an indexed column wise.
- there are two types of indexes in oracle.
    i) B-tree index
    ii) Bitmap index

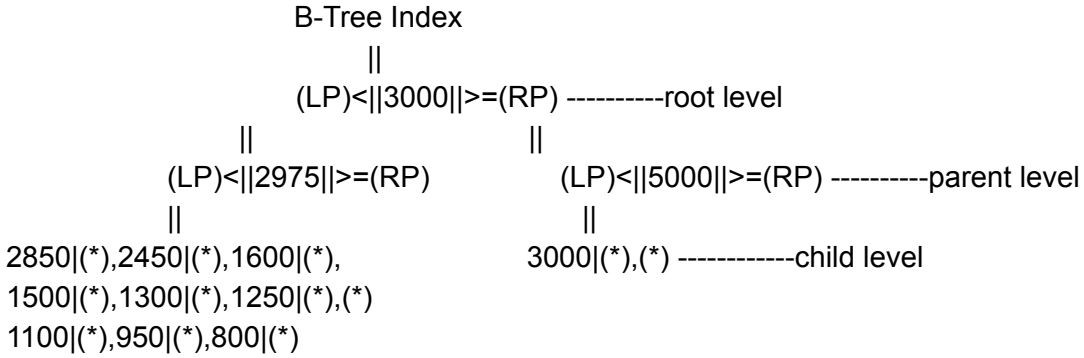
i) B-tree index:
==========

- in this mechanism data can be organized in the form "Tree" structure by the system.

syntax:
=======

create index <index name> on <table name>(column name);

EX:
SQL> CREATE INDEX I1 ON EMP(SAL);
SQL> SELECT * FROM EMP WHERE SAL=3000;



Here,
LP - left pointer
RP - right pointer
* - rowid / rowaddress

ii) Bitmap index:
==========
- in this mechanism data can be organized in the form "Table" format by the system based on
bit numbers are 0,1.
Here,
0 is represent when condition is false
1 is represent when condition is true.

syntax:
=======
create bitmap index <index name> on <table name>(column name);

EX:
SQL> CREATE BITMAP INDEX BIT1 ON EMP(JOB);
SQL> SELECT * FROM EMP WHERE JOB='CLERK';

EX:

Bitmap Indexed Table Format
==========

JOB|| 1 || 2 || 3 || 4 || 5 || 6 || 7 || 8 || 9 || 10 || 11 || 12 || 13 || 14
==========
CLERK|| 1 || 0 || 0 || 0 || 0 || 0 || 0 || 0 || 0 || 0 || 1 || 1 || 0 || 1
==========
(*) (*) (*) (*) (*)

Here, " * " is represent ROWID

To view all indexes in oracle:
==========
SQL> DESC USER_IND_COLUMNS;
SQL> SELECT COLUMN_NAME,INDEX_NAME FROM USER_IND_COLUMNS WHERE TABLE_NAME='EMP';

To view type of index in oracle:
==========
SQL> DESC USER_INDEXES;
SQL> SELECT INDEX_NAME,INDEX_TYPE FROM USER_INDEXES WHERE TABLE_NAME='EMP';

INDEX_NAME	INDEX_TYPE
I1	NORMAL(B-TREE INDEX)
B1	BITMAP

How to drop index:
==========
syntax:
======
DROP INDEX <INDEX NAME>;

EX:
SQL> DROP INDEX I1;
SQL> DROP INDEX B1;
==========
NORMALIZATION:
==========
What is Normalization?
==========
- Normalization is a technique which is used to decompose a table data into multiple tables.

Where we want to use Normalization?
==========
- DB designing level

Why we need Normalization?
==========
EX:

Branch_Student_Details

STID	SNAME	BRANCH	HOD	OFFICE_NUMBER
1021	smith	cse	Mr.x	040-22334455
1022	allen	cse	Mr.y	040-22334455
1023	ward	cse	Mr.x	040-22334455
1024	jones	cse	Mr.y	040-22334455
1025	scott	cse	Mr.x	040-22334455------> new row

Disadvantages:
==========
- Data Redundancy problem.(i.e duplicate data)
- Memory wasted
- Data Inconsistency problem.(i.e irregular data)
- Insertion problem
- Updation problem
- Delete problem

- To overcome the above problems on table then we must used a technique is known as "Normalization".

Using Normalization:
==========

|----------Relationship----------|

(PK) Branch_Details

Bcode	Branch	Hod	Offiec_number
1	cse	Mr.y	040-22334455

Student_Details (FK)

Stid	Sname	Bcode
1021	smith	1
1022	allen	1
1023	ward	1
1024	jones	1
1025	scott	1
1026	miller	1

Advantages:
==========

- To avoid data redundancy problem.(i.e duplicate data)
- Memory saved
- To avoid data inconsistency problem.(i.e irregular data)
- To avoid Insertion problem
- To avoid Updation problem
- To avoid Deletion problem

Types of Normalization Forms:
==========

1. First normal form(1NF)
2. Second normal form(2NF)
3. Third normal form(3NF)
4. Boyce-Codd normal form(BCNF)
5. Fourth normal form(4NF)
6. Fifth normal form(5NF)

1. First normal form(1NF):
==========

- For a table to be in the First Normal Form, it should follow the following 4 rules:
  1. Each column should contain atomic value (atomic = single value).
  2. A column should contain values that are same datatype.
  3. All the columns in a table should have unique names.
  4. The order in which data is stored, does not matter.

Ex: (pk)

STID	SNAME	REGDATE	BRANCH
1	smith	12-may-2024	IT
2	clark	23-mar-2024	CSE
3	allen	02-jan-2025	EEE
4	blake	29-sep-2023	EC

2. Second normal form(2NF):
==========

- For a table to be in the Second Normal Form, it must satisfy two conditions:
  1. The table should be in the First Normal Form.
  2. There should be no Partial Dependency.

WHAT IS DEPENDENCY:
==========

- IN A TABLE IF NON-KEY COLUMNS (NON PRIMARY KEY) ARE DEPENDS ON KEY COLUMN (PRIMARY KEY) THEN IT IS CALLED AS FULLY DEPENDENCY / FUNCTIONAL
DEPENDENCY.

(PK)
EX: STID SNAME BRANCH ADDRESS

- Here, “STID “IS A KEY COLUMN and “SNAME”,”BRANCH”,”ADDRESS” ARE NON-KEY COLUMNS.
These non-key columns are linked with key column is STID.so that in this table there is no partial dependency columns.

WHAT IS PARTIAL DEPENDENCY:
==========
- IN A TABLE IF NON-KEY COLUMN DEPENDS ON PART OF THE KEY COLUMN, THEN IT IS CALLED AS PARTIAL DEPENDENCY.

<PRIMARY KEY (stu_id,sub_id) / COMPOSITE PRIMARY KEY>
EX: STU_ID SUB_ID STU_MARKS TEACHER

- Here, “STU_ID and SUB_ID “ IS A KEY COLUMNS - ” STU_MARKS”,” TEACHER” ARE
NON-KEY COLUMNS. THEN “TEACHER” DEPENDS ON “SUB_ID” BUT NOT “STU_ID” COLUMN.
- Here we found a partial dependency column is "TEACHER" so that we need to do decompose a table like below,

Solution:
=========
Ex:
Subject_Table Student_table
========== ==========
(pk) (pk) (fk)
SUB_ID SUB_NAME TEACHER STU_ID STU_MARKS SUB_ID

3. Third normal form(3NF):
==========
- For a table to be in the third normal form there is two conditions.
1. It should be in the Second Normal form.
2. And it should not have Transitive Dependency.

TRANSITIVE DEPENDENCY:

==========

- IN TABLE IF NON-KEY COLUMN DEPENDS ON ANOTHER NON-KEY COLUMN, THEN IT IS CALLED AS TRANSITIVE DEPENDENCY.

EX:

|--------CPK----------|

STUDENT_ID	SUBJECT_ID	STU_MARKS	EXAM_NAME	TOTAL_MARKS

- Here, “STU_ID and SUB_ID “ ARE KEY COLUMNS . ” EXAM_NAME”,” TOTAL_MARKS”
ARE NON-KEY COLUMNS. THEN “TOTAL_MARKS” DEPENDS ON “EXAM_NAME” BUT NOT “STU_ID and SUB_ID” COLUMNS.

- Here we found transitive dependency columns are "EXAM_NAME" and "TOTAL_MARKS"
so that we need to do decompose the above table into multiple tables.

Solution:

==========

(pk) Exam_Table (cpk) Score_Table (fk)

EXAM_ID	EXAM_NAME	TOTAL_MARKS	STUDENT_ID	SUBJECT_ID	STU_MARKS	EXAM_ID

4. Boyce-Codd normal form(BCNF):

==========

- For a table to satisfy the Boyce - Codd Normal Form, it should satisfy the following two conditions:
    1. It should be in the Third Normal Form.
    2. And, for any dependency A → B, A should be a super key.

SUPER KEY:

==========

- A COLUMN (OR) COMBNATION OF COLUMNS WHICH ARE UNIQUELY IDENTIFYING
A ROW IN A TABLE IS CALLED AS SUPER KEY.

CANDIDATE KEY:

==========

- A MINIMAL SUPER KEY WHICH IS UNIQUELY IDENTIFYING A ROW IN A TABLE IS CALLED AS CANDIDATE KEY.

(OR)

- A SUPER KEY WHICH IS SUBSET OF ANOTHER SUPER KEY, BUT THE

COMBINATION
OF SUPER KEYS ARE NOT A CANDIDATE KEY.

EX:

STUDENT TABLE

STUDENT_ID	NAME	BRANCH	MAILID	REG_NUMBER

Super key columns:

student_id		student_id + mailid	
mailid		mailid + reg_number	student_id + mailid + reg_number
reg_number		reg_number + student_id	

Candidate key columns:

student_id
mailid
reg_number

EX: Professor Table

|----------cpk ----------|

PROFESSOR_ID	SUBJECT(B)	PROFESSOR(A)
1	java	p.java
2	java	p.java

- Here, PROFESSOR column depends on SUBJECT so that PROFESSOR should be super key but is not a super key.
- Now to make a PROFESSOR column is a super key and SUBJECT is non-super key column in the table then we do some changes in a table like below.

Solution:

Professor Table

|----------cpk----------|

PROFESSOR_ID	PROFESSOR(A)	SUBJECT(B)
1	p.java	java
2	p.java	java

5. Fourth normal form(4NF):

==========

- For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:

1. It should be in the Boyce-Codd Normal Form.
2. A table does not contain more than one independent multi valued attribute / Multi Valued Dependency.

Multi valued Dependency:
==========

- In a table one column same value mapping with multiple values of another column is called as multi valued dependency.

EX:

COLLEGE ENROLLMENT TABLE (5NF)

STUDENT_ID	COURSE	HOBBY
1	ORACLE	Cricket
1	JAVA	Reading
1	HTML	Hockey

Mapping with multiple values of columns after decomposing a table:
==========

Course_details (4NF)

STUDENT_ID	COURSE
1	oracle
1	java
1	html

Hobbies_details(4NF)

STUDENT_ID	HOBBY
1	cricket
1	reading
1	hockey

Fifth Normal Form (5NF):
==========

- If a table is having multi valued attributes and also that table cannot decomposed into multiple tables are called as fifth normal form.

EX:

COLLEGE ENROLLMENT TABLE (5NF)

STUDENT_ID	COURSE	HOBBY
1	ORACLE	Cricket
1	JAVA	Reading
1	HTML	Hockey

==========

PL/SQL

EX:
write a pl/sql program to print "WELCOME TO PL/SQL PROGRAMS"?

```
SQL> BEGIN
  2  DBMS_OUTPUT.PUT_LINE('WELCOME TO PL/SQL PROGRAMS');
  3  END;
  4  /
PL/SQL procedure successfully completed.
```

To view the output of pl/sql program:

syntax:

```
SET SERVEROUTPUT OFF / ON;
```

Here,

OFF - oracle server not display output of a program.(default)
ON - oracle server display output of a program.

EX:

```
SQL> SET SERVEROUTPUT ON;

SQL> /
WELCOME TO PL/SQL PROGRAMS
```

EX:
write a pl/sql program to print variables values?

```
SQL> DECLARE
  2  X NUMBER(10):=10;
  3  Y NUMBER(10):=20;
  4  BEGIN
  5  DBMS_OUTPUT.PUT_LINE(X||','||Y);
  6  END;
  7  /
```

OUTPUT:

10,20

EX:
write a pl/sql program of sum of two numbers at runtime?

```
SQL> DECLARE
  2  A NUMBER(3);
  3  B NUMBER(3);
  4  BEGIN
  5  A:=&A;
  6  B:=&B;
  7  DBMS_OUTPUT.PUT_LINE('SUM OF TWO NUMBERS:-'||(A+B));
  8  END;
  9  /
Enter value for a: 10
old   5: A:=&A;
new 5: A:=10;
Enter value for b: 20
old   6: B:=&B;
new 6: B:=20;
SUM OF TWO NUMBERS:-30
```

How to disable :OLD,:NEW bind variables statements:
==========
syntax:
======
SET VERIFY ON / OFF;

Here,

ON - to enable :old,:new bind variables statements.
OFF - to disable :old,:new bind variables statements.

Ex:

```
SQL> SET VERIFY OFF;
SQL> /
Enter value for a: 55
Enter value for b: 22
SUM OF TWO NUMBERS:-77
```

HOW TO STORE COLUMNS VALUES INTO VARIABLES:
==========
syntax:
=======
select <column name1>,<column name2>,.....into <variable name1>,<variable name2>,..... from <table name> [where <condition>];

EX:

waq to retrieve ENAME,SALARY details from emp table.whose EMPNO is 7788?
SQL> SELECT ENAME,SAL FROM EMP WHERE EMPNO=7788;

OUTPUT:
==========
ENAME SAL
---------- ----------
SCOTT 3000

EX:
write a pl/sql program to retrieve ENAME,SALARY details from emp table whose EMPNO is 7788?
SQL> DECLARE
2 v_ENAME VARCHAR2(10);
3 v_SAL NUMBER(8,2);
4 BEGIN
5 SELECT ENAME,SAL INTO v_ENAME,v_SAL FROM EMP WHERE EMPNO=7788;
6 DBMS_OUTPUT.PUT_LINE(v_ENAME||','||v_SAL);
7 END;
8 /

OUTPUT:
=========
SCOTT,3000

EX:
```
DECLARE
   v_ENAME EMP.ENAME%TYPE;
   v_SAL EMP.SAL%TYPE;
   BEGIN
   SELECT ENAME,SAL INTO v_ENAME,v_SAL FROM EMP WHERE EMPNO=7788;
   DBMS_OUTPUT.PUT_LINE(v_ENAME||','||v_SAL);
   END;
   /
```
OUTPUT:
========
SCOTT,3000

Rowtype with specific columns:
==========
```
DECLARE
   i EMP%ROWTYPE;
   BEGIN
```

```
   SELECT ENAME,SAL INTO i.ENAME,i.SAL FROM EMP WHERE EMPNO=7788;
   DBMS_OUTPUT.PUT_LINE(i.ENAME||','||i.SAL);
   END;
   /
```

OUTPUT:

```
========
SCOTT,3000
```

Rowtype with all columns:

```
============================
DECLARE
   i EMP%ROWTYPE;
   BEGIN
   SELECT * INTO i FROM EMP WHERE EMPNO=7788;
   DBMS_OUTPUT.PUT_LINE(i.EMPNO||','||i.ENAME||','||i.SAL||','||i.DEPTNO);
   END;
   /
```

OUTPUT:

```
========
7788,SCOTT,3000,20
```

```
=========================================================================
```

CURSOR:

```
========
```

- it is a temporary memory / SQL private area / workspace.
- there two types of cursor in pl/sql.
    1. Explicit cursor
    2. Implicit cursor

1. Explicit cursor :

```
===============
```

- these cursor are created by users for fetching multiple rows from a database table.
- to create an explicit cursor then we follow the following 4 steps are,

step1: Declare cursor variable:

```
=========================
```

syntax:

```
======
```

declare cursor <cursor name> is <select query>;

step2: Open cursor connection:

```
=========================
```

syntax:

```
======
```

open <cursor name>;

step3: Fetching rows from cursor memory table:
==========
syntax:
======
fetch <cursor name> into <variables>;

step4: Close cursor connection:
==========
syntax:
======
close <cursor name>;

Attributes of an explicit cursor:
==========
- to check the status of cursor.

syntax:
======
<cursor name>%<attribute name>;

i) %isopen:
==========
- it is a default attribute of cursor.
- it return TRUE when cursor connection successfully open otherwise return FALSE.

ii) %notfound:
==========
- it return TRUE when cursor is not having data otherwise return FALSE.

iii) %found:
==========
- it return TRUE when cursor is having data otherwise return FALSE.

iv) %rowcount:
==========
- it return how many no.of rows are fetched from cursor.

EX:
write a cursor program to fetch a single row from a table?

```
SQL> DECLARE CURSOR C1 IS SELECT ENAME,SAL FROM EMP;
   v_ENAME VARCHAR2(10);
```

```
   v_SAL NUMBER(8,2);
   BEGIN
   OPEN C1;
   FETCH C1 INTO v_ENAME,v_SAL;
   DBMS_OUTPUT.PUT_LINE(v_ENAME||','||v_SAL);
   CLOSE C1;
   END;
   /
```

OUTPUT:

```
============
SMITH,800
```

EX:

write a cursor program to fetch multiple rows from a table?

```
DECLARE CURSOR C1 IS SELECT ENAME,SAL FROM EMP;
   v_ENAME VARCHAR2(10);
   v_SAL NUMBER(8,2);
   BEGIN
   OPEN C1;
   FETCH C1 INTO v_ENAME,v_SAL;
   DBMS_OUTPUT.PUT_LINE(v_ENAME||','||v_SAL);
   FETCH C1 INTO v_ENAME,v_SAL;
   DBMS_OUTPUT.PUT_LINE(v_ENAME||','||v_SAL);
   CLOSE C1;
   END;
 /
```

OUTPUT:

```
========
SMITH,800
ALLEN,1600
```

- In the above example we are using multiple fetch statements for fetching multiple rows from a table.To avoid multiple fetch statements then we should use "LOOPING STATEMENTS".

i) By using "Simple Loop":

```
======================
SQL> DECLARE CURSOR C1 IS SELECT ENAME,SAL FROM EMP;
  2  v_ENAME VARCHAR2(10);
  3  v_SAL NUMBER(8,2);
  4  BEGIN
  5  OPEN C1;
  6  LOOP
```

```
  7  FETCH C1 INTO v_ENAME,v_SAL;
  8  EXIT WHEN C1%NOTFOUND;
  9  DBMS_OUTPUT.PUT_LINE(v_ENAME||','||v_SAL);
 10  END LOOP;
 11  CLOSE C1;
 12  END;
 13  /
```

OUTPUT:

```
=========
SMITH,800
ALLEN,1600
WARD,1250
JONES,2975
MARTIN,1250
BLAKE,2850
CLARK,2450
SCOTT,3000
KING,5000
TURNER,1500
ADAMS,1100
JAMES,950
FORD,3000
MILLER,1300
```

i) By using "While Loop":

```
======================
SQL> DECLARE CURSOR C1 IS SELECT ENAME,SAL FROM EMP;
  2  v_ENAME VARCHAR2(10);
  3  v_SAL NUMBER(8,2);
  4  BEGIN
  5  OPEN C1;
  6  FETCH C1 INTO v_ENAME,v_SAL;----------------------------> fetch starts from 1st row
  7  WHILE(C1%FOUND)
  8  LOOP
  9  DBMS_OUTPUT.PUT_LINE(v_ENAME||','||v_SAL);
 10  FETCH C1 INTO v_ENAME,v_SAL;----------------------------> fetch will continue upto last row
 11  END LOOP;
 12  CLOSE C1;
 13  END;
 14  /
```

OUTPUT:

=========
SMITH,800
ALLEN,1600
WARD,1250
JONES,2975
MARTIN,1250
BLAKE,2850
CLARK,2450
SCOTT,3000
KING,5000
TURNER,1500
ADAMS,1100
JAMES,950
FORD,3000
MILLER,1300

iii) By using "For loop":
==========
```
SQL> DECLARE CURSOR C1 IS SELECT ENAME,SAL FROM EMP;
  2  BEGIN
  3  FOR i IN C1
  4  LOOP
  5  DBMS_OUTPUT.PUT_LINE(i.ENAME||','||i.SAL);
  6  END LOOP;
  7  END;
  8  /
```

OUTPUT:
==========
SMITH,800
ALLEN,1600
WARD,1250
JONES,2975
MARTIN,1250
BLAKE,2850
CLARK,2450
SCOTT,3000
KING,5000
TURNER,1500
ADAMS,1100
JAMES,950
FORD,3000
MILLER,1300

2. Implicit cursor:
==========

- these cursor are created by the system by default when we perform DML operations on table.
- implicit cursor is used to store the status of DML query is executed successfully or not.

EX:

```
SQL> INSERT INTO DEPT VALUES(50,'DBA','HYD');
1 row created.

SQL> UPDATE DEPT SET LOC='PUNE' WHERE DEPTNO=20;
1 row updated.

SQL> DELETE FROM DEPT WHERE DEPTNO=10;
1 row deleted.
```

==========

EXCEPTION HANDLING:
==========

What is an Exception ?
==========

- it is a runtime error / execution error.

What is Exception Handling?
==========

- it is used to handle abnormal terimination of a program execution process.

- PL/SQL supports the following two types of exceptions.
  1. Pre-defined exceptions
  2. User-defined exceptions

1. Pre-defined exceptions:
==========

- these are defined by oracle server by default.

Ex: no_data_found,too_many_rows,zero_divide,value_error,.....etc

EX:

write a pl/sql program to input EMPNO and display that employee name from a table?

```
SQL> DECLARE
  2  v_ENAME VARCHAR2(10);
  3  BEGIN
  4  SELECT ENAME INTO v_ENAME FROM EMP WHERE EMPNO=&EMPNO;
  5  DBMS_OUTPUT.PUT_LINE(v_ENAME);
```

```
  6  END;
  7  /
Enter value for empno: 7788
SCOTT
SQL> /
Enter value for empno: 1122
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 4
```

- To handle the above exception oracle server is providing a predefined exception name is known as "no_data_found".

Handling Exception by using "no_data_found":

```
====================================
SQL> DECLARE
  2  v_ENAME VARCHAR2(10);
  3  BEGIN
  4  SELECT ENAME INTO v_ENAME FROM EMP WHERE EMPNO=&EMPNO;
  5  DBMS_OUTPUT.PUT_LINE(v_ENAME);
  6  EXCEPTION
  7  WHEN NO_DATA_FOUND THEN
  8  DBMS_OUTPUT.PUT_LINE('Sorry,Record is not found.Try Again!!!');
  9  END;
 10  /
Enter value for empno: 7900
JAMES
SQL> /
Enter value for empno: 1122
Sorry,Record is not found.Try Again!!!
```

EX:
write a pl/sql program to fetch employee salary from a table?

DEMO_TABLE:

```
============
SQL> SELECT * FROM TEST;

ENAME          SAL
----------     ----------
SMITH          25000
ADAMS           43000

SQL> DECLARE
```

```
  2  v_SAL NUMBER(8,2);
  3  BEGIN
  4  SELECT SAL INTO v_SAL FROM TEST;
  5  DBMS_OUTPUT.PUT_LINE(v_SAL);
  6  END;
  7  /
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 4
```

- Generally by using "select.....into....." statement we will fetch a single row / a single value from a table but in our TEST table we have more than one row so that oracle server return an exception is "exact fetch returns more than requested number of rows".

- To handle the above exception then oracle server provide a predefined exception name is known as "too_many_rows".

Handling Exception by using "too_many_rows":

```
==========
SQL> DECLARE
  2  v_SAL NUMBER(8,2);
  3  BEGIN
  4  SELECT SAL INTO v_SAL FROM TEST;
  5  DBMS_OUTPUT.PUT_LINE(v_SAL);
  6  EXCEPTION
  7  WHEN TOO_MANY_ROWS THEN
  8  DBMS_OUTPUT.PUT_LINE('Your table is having more than one row.Plz check it!!!');
  9  END;
 10  /
Your table is having more than one row.Plz check it!!!
```

EX:
write a pl/sql program to division of two numbers?

```
SQL> DECLARE
  2  X NUMBER(3);
  3  Y NUMBER(3);
  4  Z NUMBER(4);
  5  BEGIN
  6  X:=&X;
  7  Y:=&Y;
  8  Z:=X/Y;
  9  DBMS_OUTPUT.PUT_LINE('DIV IS:-'||Z);
 10  END;
 11  /
```

```
Enter value for x: 10
Enter value for y: 2
DIV IS:-5

SQL> /
Enter value for x: 10
Enter value for y: 0
ERROR at line 1:
ORA-01476: divisor is equal to zero
ORA-06512: at line 8
```

- To handle the above exception oracle server provide a pre-defined exception name is ZERO_DIVIDE.

Handling exception by using "zero_divide":

```
==================================
SQL> DECLARE
  2  X NUMBER(3);
  3  Y NUMBER(3);
  4  Z NUMBER(4);
  5  BEGIN
  6  X:=&X;
  7  Y:=&Y;
  8  Z:=X/Y;
  9  DBMS_OUTPUT.PUT_LINE('DIV IS:-'||Z);
 10  EXCEPTION
 11  WHEN ZERO_DIVIDE THEN
 12  DBMS_OUTPUT.PUT_LINE('Second Number Should Not Be Zero.Plz Try Again!!!');
 13  END;
 14  /
Enter value for x: 10
Enter value for y: 5
DIV IS:-2

SQL> /
Enter value for x: 10
Enter value for y: 0
Second Number Should Not Be Zero.Plz Try Again!!!
```

2. User-defined exceptions:

```
=======================
```

- when we created our own exception name to handle exceptions in a pl/sql program then we
called as user defined exceptions.
- to create a user defined exception name then we follow the following three steps are,

Step1: Declare user defined exception name:
==========
syntax:
=======
Declare
<UD exception name> Exception;

Step2: Raise an exception with user defined exception name:
==========
Method-1: RAISE statement:
==========
- to raise and handle an exception.
syntax:
=======
raise <UD exception name>;

Method-2: RAISE_APPLICATION_ERROR(number,message):
==========
- to raise an exception but not handle an exception in a program.
- this method is having two arguments are,
Number : it return user defined error number.it must be -20001 to -20999.
Message : it return user defined error message.
syntax:
=======
raise_application_error(user defined error number,user defined error message);

Step3: Handling exception with user defined exception name:
==========
syntax:
=======
Exception
when <exception name> then
< user defined statements>;
end;
/

EX:
i) By using "RAISE" statement:

```
==========================
SQL> DECLARE
  2  X NUMBER(4);
  3  Y NUMBER(4);
  4  Z NUMBER(5);
  5  EX EXCEPTION;----------> (1)
  6  BEGIN
  7  X:=&X;
  8  Y:=&Y;
  9  IF Y=0 THEN
 10  RAISE EX;--------------------------> (2)
 11  ELSE
 12  Z:=X/Y;
 13  DBMS_OUTPUT.PUT_LINE('DIVISION IS:-'||Z);
 14  END IF;
 15  EXCEPTION
 16  WHEN EX THEN---------------------> (3)
 17  DBMS_OUTPUT.PUT_LINE('SECOND NUMBER NOT BE ZERO!!!');
 18  END;
 19  /
Enter value for x: 10
Enter value for y: 2
DIVISION IS:-5

SQL> /
Enter value for x: 10
Enter value for y: 0
SECOND NUMBER NOT BE ZERO!!!

ii) By using RAISE_APPLICATION_ERROR(number,message):
==================================================
SQL> DECLARE
  2  X NUMBER(4);
  3  Y NUMBER(4);
  4  Z NUMBER(5);
  5  EX EXCEPTION;
  6  BEGIN
  7  X:=&X;
  8  Y:=&Y;
  9  IF Y=0 THEN
 10  RAISE EX;
 11  ELSE
 12  Z:=X/Y;
```

```
 13  DBMS_OUTPUT.PUT_LINE('DIVISION IS:-'||Z);
 14  END IF;
 15  EXCEPTION
 16  WHEN EX THEN
 17  RAISE_APPLICATION_ERROR(-20478,'Sorry,Your Exception Is Not Handle!!!');
 18  END;
 19  /
Enter value for x: 10
Enter value for y: 2
DIVISION IS:-5

SQL> /
Enter value for x: 10
Enter value for y: 0
ERROR at line 1:
ORA-20478: Sorry,Your Exception Is Not Handle!!!
ORA-06512: at line 17
```

==========

SUB BLOCK:

==========

- it is a named block which will store in database automatically.
- oracle pl/sql supports the following types of sub block objects.
    1. stored procedure
    2. stored function
    3. triggers

1. stored procedure:

==========

- it is database object which contains "pre-compiled code / query".

Purpose of stored procedure:

==========

- to avoid unnecessary compilation of code.
- to improve the performance database.
- code reusability & code security.

syntax:

=======

```
create [or replace] procedure <pname>(<parameter name1> [mode type] <datatype>,.....)
is
<declare variables>;
begin
<procedure body / statements>;
```

```
end;
/
```

How to call a stored procedure:
==========
1. By using "anonymous block":
==========
syntax:
=======
```
begin
<pname>(values);
end;
/
```

2. By using "executed" command:
==========
syntax:
======
```
execute <pname>(values);
```
(or)
```
exec <pname>(values);
```

Types of parameters modes:
==========
- In pl/sql stored procedures are supporting the following three types of parameters modes those are,

i) IN mode:
==========
- these are default parameters of stored procedure.
- these parameters are used for storing input values which was given by user at runtime / execution time.

ii) OUT mode:
==========
- these parameters are used for return a value(output) to user.

iii) IN OUT mode:
==========
- these parameters are used for storing and also returning a value to / from the user.

Examples on "IN" parameter mode:
==========

EX:
create a stored procedure to find out sum of two numbers?

```
SQL> CREATE OR REPLACE PROCEDURE SP1(X IN NUMBER,Y IN NUMBER)
  2  IS
  3  BEGIN
  4  DBMS_OUTPUT.PUT_LINE('SUM OF TWO NUMBERS:-'||(X+Y));
  5  END;
  6  /
```

Procedure created.

Calling stored procedure:
==========

i)

```
SQL> BEGIN
  2  SP1(10,20);
  3  END;
  4  /
```

SUM OF TWO NUMBERS:-30

ii)

```
SQL> EXECUTE SP1(10,20);
        (or)
   SQL> EXEC SP1(20,30);
```

SUM OF TWO NUMBERS:-50

NOTE:
=====
- if we want to view all sub block objects(SP/SF/Triggers) in oracle database then use a datadictionary is "USER_OBJECTS".

EX:

```
SQL> DESC USER_OBJECTS;
SQL> SELECT OBJECT_NAME FROM USER_OBJECTS WHERE
OBJECT_TYPE='PROCEDURE';
```

NOTE:
======
- if we want to view the source code of a particular sub block object then use a datadictionary is "USER_SOURCE".

EX:

```
SQL> DESC USER_SOURCE;
SQL> SELECT TEXT FROM USER_SOURCE WHERE NAME='SP1';
```

EX:
create a procedure to input EMPNO and display that employee NAME,SALARY details from emp table?

```
SQL> CREATE OR REPLACE PROCEDURE SP2(p_EMPNO IN NUMBER)
  2  IS
  3  v_ENAME VARCHAR2(10);
  4  v_SAL NUMBER(8,2);
  5  BEGIN
  6  SELECT ENAME,SAL INTO v_ENAME,v_SAL FROM EMP WHERE EMPNO=p_EMPNO;
  7  DBMS_OUTPUT.PUT_LINE(v_ENAME||','||v_SAL);
  8  END;
  9  /
```

OUTPUT:
=========

```
SQL> EXECUTE SP2(7788);
SCOTT,3000
```

EX:
create a stored procedure to find out the location of the given department name?

```
SQL> CREATE OR REPLACE PROCEDURE SP3(p_DNAME IN VARCHAR2)
  2  IS
  3  v_LOC VARCHAR2(10);
  4  BEGIN
  5  SELECT LOC INTO v_LOC FROM DEPT WHERE DNAME=p_DNAME;
  6  DBMS_OUTPUT.PUT_LINE('Location is:-'|| v_LOC);
  7  END;
  8  /
```

OUTPUT:
========

```
SQL> EXECUTE SP3('SALES');
Location is:- CHICAGO
```

Examples on "OUT" parameter mode:
==========

EX:
create a stored procedure to return the cube of the given number?

```
SQL> CREATE OR REPLACE PROCEDURE SP4(X IN NUMBER,Y OUT NUMBER)
  2  AS
  3  BEGIN
  4  Y:=X*X*X;
  5  END;
```

```
 6  /
```

OUTPUT:

```
SQL> EXECUTE SP4(5);
ERROR at line 1:
PLS-00306: wrong number or types of arguments in call to 'SP4'
```

- To overcome the above problem we need to follow the following 3 steps are,

step1: Declare a bind / referenced variables for "OUT" parameters of SP:

==========

syntax:

======

var[iable] <bind/referenced variable name> <datatype>[size];

step2: Adding this bind / referenced variables to a SP:

==========

syntax:

======

execute <pname>(value1,value2,.....,:<bind variables name1>,.....);

step3: Print bind / referenced variables :

==========

syntax:

======

print < bind / referenced variable name>;

OUTPUT

=======

```
SQL> VAR r NUMBER;
SQL> EXECUTE SP4(5,:r);
SQL> PRINT r;

         R
----------
       125
```

EX:

create a stored procedure to input EMPNO and retrun that employee Provident Fund,Professional Tax
at 5%,10% on basic salary by using "OUT" parameters?

```
SQL> CREATE OR REPLACE PROCEDURE SP5(p_EMPNO IN NUMBER,PF OUT
```

```
NUMBER,PT OUT NUMBER)
  2  IS
  3  v_BSAL NUMBER(8,2);
  4  BEGIN
  5  SELECT SAL INTO v_BSAL FROM EMP WHERE EMPNO=p_EMPNO;
  6  PF:=v_BSAL*0.05;
  7  PT:=v_BSAL*0.1;
  8  END;
  9  /
```

OUTPUT:

```
SQL> VAR rPT NUMBER;
SQL> VAR rPF NUMBER;
SQL> EXECUTE SP5(7788,:rPF,:rPT);
SQL> PRINT rPF  rPT;

       RPF
----------
       150

       RPT
----------
       300
```

Examples on "IN OUT" parameters:

==========

EX:

create a stored procedure to return the square of the given number by using IN OUT parameter?

```
SQL> CREATE OR REPLACE PROCEDURE SP6(X IN OUT NUMBER)
  2  IS
  3  BEGIN
  4  X:=X*X;
  5  END;
  6  /
```

OUTPUT:

```
SQL> EXECUTE SP6(10);
ERROR at line 1:
PLS-00363: expression '10' cannot be used as an assignment target
```

- To overcome the above problem we need to follow the following 4 steps are,

step1: Declare a bind / referenced variables for "OUT" parameters of SP:
==========
syntax:
======
var[iable] <bind/referenced variable name> <datatype>[size];

step2: To assign value to referenced / bind variable:
==========
syntax:
=======
execute :<ref/bind variable name> := <value>;

step3: Adding this bind / referenced variables to a SP:
==========
syntax:
======
execute <pname>(:<bind variables name1>,.....);

step4: Print bind / referenced variables :
==========
syntax:
======
print < bind / referenced variable name>;

OUTPUT:
========
```
SQL> VAR r NUMBER;
SQL> EXECUTE :r:=10;
SQL> EXECUTE SP6(:r);
SQL> PRINT r;

         R
----------
       100
```

EX:
create a stored procedure to input EMPNO and return that employee SALARY from a table by using
IN OUT parameter?
```
SQL> CREATE OR REPLACE PROCEDURE SP7(X IN OUT NUMBER)
  2  IS
  3  BEGIN
  4  SELECT SAL INTO X FROM EMP WHERE EMPNO=X;
```

```
 5  END;
 6  /
```

OUTPUT:

```
SQL> VAR r NUMBER;
SQL> EXECUTE :r:=7900;
SQL> EXECUTE SP7(:r);
SQL> PRINT r;

         R
----------
       950
```

How to drop stored procedure:

==========

syntax:

======

drop procedure <pname>;

EX:

SQL> DROP PROCEDURE SP1;

==========

STORED FUNCTION:

==========

- it is a block of code to perform some task and must return a value.
- these functions are also called as "user defined functions".

syntax:

======

create [or replace] function <fname>(<parameter name> <datatype>,.....)
return <return variable TYPE>
as
<declare variables>;
begin
<function body>;
return <return variable NAME>;
end;
/

How to call a stored function:

==========

syntax:

======

select <fname>(values) from dual;

EX:
create a stored function to input EMPNO and return that ENAME from a table?

```
SQL> CREATE OR REPLACE FUNCTION SF1(p_EMPNO NUMBER)
  2  RETURN VARCHAR2
  3  AS
  4  v_ENAME VARCHAR2(10);
  5  BEGIN
  6  SELECT ENAME INTO v_ENAME FROM EMP WHERE EMPNO=p_EMPNO;
  7  RETURN v_ENAME;
  8  END;
  9  /
```

OUTPUT:

```
SQL> SELECT SF1(7369) FROM DUAL;

SF1(7369)
--------------------
SMITH
```

EX:
create a stored function to input EMPNO and return that employee GROSS SALARY based on the
following conditions are :

i) HRA -------- 5%
ii) DA ---------10%
iii) PF ---------2% on basic salary?

```
SQL> CREATE OR REPLACE FUNCTION SF2(p_EMPNO NUMBER)
  2  RETURN NUMBER
  3  AS
  4  v_BSAL NUMBER(8,2);
  5  v_HRA NUMBER(8,2);
  6  v_DA NUMBER(8,2);
  7  v_PF NUMBER(8,2);
  8  v_GROSS NUMBER(8,2);
  9  BEGIN
 10  SELECT SAL INTO v_BSAL FROM EMP WHERE EMPNO=p_EMPNO;
 11  v_HRA:=v_BSAL*0.05;
 12  v_DA:=v_BSAL*0.1;
 13  v_PF:=v_BSAL*0.02;
 14  v_GROSS:=v_BSAL+v_HRA+v_DA+v_PF;
```

```
 15  RETURN v_GROSS;
 16  END;
 17  /
```

OUTPUT:
==========

```
SQL> SELECT SF2(7788) FROM DUAL;

 SF2(7788)
----------
      3510
```

How to view all stored functions in oracle:
==========

```
SQL> SELECT OBJECT_NAME FROM USER_OBJECTS WHERE
OBJECT_TYPE='FUNCTION';
```

How to view the source code of stored function:
==========

```
SQL> SELECT TEXT FROM USER_SOURCE WHERE NAME='SF1';
```

How to drop stored function:
==========

syntax:
======

```
drop function <fname>;
```

EX:

```
SQL> DROP FUNCTION SF1;
```

==========

TRIGGER:
==========

- it is a named block which will execute by the system automatically when user perform DML,DDL operations on a table / on a database.

Types of Triggers:
==========

1. DML triggers
2. DDL triggers(DB triggers)

1. DML triggers:
==========

- when we created a trigger object based on DML commands(insert,update,delete) are called as DML triggers.
- these triggers are executing by the system when we perform DML operations on specific table.

syntax:
======
create [or replace] trigger <trigger name>
before / after insert or update or delete on <table name>
[for each row]
begin
<trigger body>;
end;
/

Trigger events:
==========
- trigger can be created with two events.
    i) Before :
    =========
        First : trigger body(logic) is executed.
        Later : DML command will execute.
    ii) After:
    =======
        First: DML command is executed.
        Later : trigger body(logic) will execute.

NOTE:
======
- Both events of trigger will provide same result.

Bind variables:
==========
- trigger supports the following two types of bind variables.
    i) :NEW
    ii) :OLD
- these bind variables are working just like normal variables for storing data/values while inserting,updating and deleting values from a table.

:NEW:
=====
- it will store new values when we insert.

syntax:
======

:NEW.<column name>;

:OLD:
=====
- it will store old values when we delete.
syntax:
======
:OLD.<column name>;

EX:
create a trigger to raise a security alert on DML operations?
```
SQL> CREATE OR REPLACE TRIGGER TRDML1
  2  AFTER INSERT OR UPDATE OR DELETE ON EMP
  3  BEGIN
  4  RAISE_APPLICATION_ERROR(-20478,'Alert!!!You cannot perform DML operations on
EMP table.');
  5  END;
  6  /
```

TESTING:
```
SQL> INSERT INTO EMP VALUES(1,'SMITH',23000);
ERROR at line 1:
ORA-20478: Alert!!!You cannot perform DML operations on EMP table.
```

EX:
create a trigger to restricted all DML operations on EMP table on every weekends(SAT,SUN)?
```
SQL> CREATE OR REPLACE TRIGGER TRDAY
  2  AFTER INSERT OR UPDATE OR DELETE ON EMP
  3  BEGIN
  4  IF TO_CHAR(SYSDATE,'DY')IN('SAT','SUN') THEN
  5  RAISE_APPLICATION_ERROR(-20547,'We cannot perform DML operations on EMP table
on weekends.');
  6  END IF;
  7  END;
  8  /
```
TESTING:
```
SQL> UPDATE EMP SET SAL=18000 WHERE EID=2;
ERROR at line 1:
ORA-20547: We cannot perform DML operations on EMP table on weekends.
```

EX:
create a trigger to validate insert operation on a table if new salary is less than to 25000?

```
SQL> CREATE OR REPLACE TRIGGER TRIN
  2  BEFORE INSERT ON EMP
  3  FOR EACH ROW
  4  BEGIN
  5  IF :NEW.SAL<25000 THEN
  6  RAISE_APPLICATION_ERROR(-20478,'NEW SALARY SHOULD NOT BE LESS THAN TO
25000');
  7  END IF;
  8  END;
  9  /
```

TESTING:

```
SQL> INSERT INTO EMP VALUES(4,'JONES',24000);-----NOT ALLOWED
SQL> INSERT INTO EMP VALUES(4,'JONES',26000);-----ALLOWED
```

2. DDL triggers:

==========

- when we created a trigger object based on DDL commands(create,alter,rename,drop) are called as DDL triggers.
- these triggers are executing by the system when we perform DDL operations on specific database.so that these triggers are also called as "DB triggers".
- DDL triggers are handling by DBA only.

syntax:

======

```
create [or replace] trigger <trigger name>
before / after create or alter or rename or drop on <DB name/ User name>.schema
[ for each row ]
begin
<trigger body>;
end;
/
```

EX:

create a trigger to raise security alert on CREATE operation ON mydb11am database?

```
SQL> CREATE OR REPLACE TRIGGER TRDDL
  2  AFTER CREATE ON MYDB11AM.SCHEMA
  3  BEGIN
  4  RAISE_APPLICATION_ERROR(-20456,'Alert!!!You cannot perform CREATE operation in
MYDB11AM database');
  5  END;
  6  /
```

TESTING:

SQL> CREATE TABLE T1(SNO NUMBER(2));
ERROR:
ORA-20456: Alert!!!You cannot perform CREATE operation in MYDB11AM database

To view all triggers in oracle:
==========
SQL> SELECT OBJECT_NAME FROM USER_OBJECTS WHERE
OBJECT_TYPE='TRIGGER';

To view the source code of a trigger:
==========
SQL> SELECT TEXT FROM USER_SOURCE WHERE NAME='TRDDL';

How to drop a trigger:
==========
syntax:
======
drop trigger <trigger name>;

EX:
SQL> DROP TRIGGER TRDDL;

==========THE END==========