ORACLE

Database: It is collection of meaningful data.

Ex:

SNO	SNAME	MARKS
101	MALLI	99
102	SREE	98
103	KRISH	97
104	VISH	96

Management System: It is a software it helps to manage the database management system should able to perform the following activities very easily.

1. Inserting the new data.
2. Updating the exiting data.
3. Deleting unnecessary data.
4. Retrieving the require data.

A database along with the software which helps to manage. The database is called database management system (DBMS).
A DBMS which is based on relational theory is called as relational database management system.

Examples of RDBMS:

1. ORACLE
2. SQL SERVER
3. DB2
4. MYSQL
5. SYBASE
6. TERA DATA
7. MS ACCESS

SQL:

Structured query language pronounced as (SEQUEL). This language is used to communicate to oracle database.

Features of SQL:

1. It is a command based language.
2. It is not case sensitive.
3. Every command should end with ';'.

4. Every command starts with "verb".
5. It is similar to English. This language is developed in the year 1972. Mr.CODD, by IBM developed by "IBM".

SUB LANGUAGE OF SQL:

1. DDL (Data Definition Language)
2. DML (Data Manipulation Language)
3. DRL/DQL (Retrieval/query)
4. TCL (Transaction Control Language)
5. DCL (Data Control Language)

*__DDL__: This language is used to manage database objects. It is collection of five commands.
CREATE, ALTER, DROP, TRUNCATE, RENAME

*__DML__: This language is used to manipulate the data you have stored. It is collection of four commands.
INSERT, UPDATE, DELETE, MERGE

*__DRL__: This language is used to retrieve the data from the database. It is collection of only one command.
SELECT

*__TCL__: It is used to maintain the transaction of Oracle database. It is collection of three commands.
COMMIT, ROLLBACK, SAVEPOINT

*__DCL__: This language is used to control the axis of the data to the users it is collection of two commands.
GRANT, REVOKE

TABLE: Table is an object which is used to store some data. In general it is collection of Rows and Columns.

Create command: This command is used to create a table.
Syntax:
CREATE TABLE<TABLE_NAME>(COL_NAME1 DATATYPE(SIZE),COL_NAME2 DATATYPE(SIZE), COL_NAME3 DATATYPE(SIZE));

Ex:

Create table student (SNO number (3), SNAME varchar2(20), MARKS number(3));

SQL * PLUS: it is a client environment.
SQL>Conn: it is called connect.
Default username: Scott
Password: tiger

SQL*PLUS: It is a client environment where the developer will write the queries and submit the queries is execution.

***TO ESTABLISH CONNECTIO**:

SQL>CONN
USER NAME: SCOTT
PASSWORD: TIGER
CONNECTED

***INSERT COMMENT**:

Syntax:

INSERT INTO<TABLE_NAME>VALUES(VAL1,VAL2,VAL3,…………VALn);

EX:

Insert into student values(101,Arun,60);
Error: Arun
Correct: 'Arun'
Insert into student values(101,'Arun',60);

Note: for varchar to value we need enclose with single(' ') codes.

Insert into student values(102,'kiran',86);
Insert into student values(103,'Ajay',50);
Insert into student values(104,'vijay');//this statement is wrong
Insert into student values(105,'vijay',null);

Note: Null is a keyword which is used to represent unavailable or undefined symbol.

Insert into student values(106,null,null);

***SELECT**: This commend is used to return the data from the table.

Syntax1: SELECT * FROM <TABLE_NAME>;
Ex: select * from student; // * represent ALL

Note: where we use * all the information (ALL the columns and the rows are display).

Syntax2: for insert command:
INSERT INTO <TABLE_NAME> (COL1,COL2,…..COLn) VALUES (VAL1,VAL2,…….VALn);
Ex: insert into student (SNO,SNAME) values (106,'Amit');

*__insert the values at runtime using '&' operator:__
Ex: INSERT INTO STUDENT VALUES (&SNO,'&SNAME',&MARKS);

*__SELECTING SPECIFIC COLUMNS__:
Ex:

```
Select SNO, MARKS from student;         Select MARKS, SNO from student;
    SNO          MARKS                        MARKS       SNO
    101          60                           60          101
    102          85                           85          102
    ----         ----                         ----        ----
    ----         ----                         ----        ----
    108          98                           98          ----
```

Select SNO, SNO, SNAME from student;
Select * from student;
Select * From student;   // not case sensitive

Create table employee(ENO number(2),ENAME varchar2(20),SALARY number(5),JOB varchar2(20));

Insert into employee values(101,'vijay',10000,'HR');
Insert into employee values(102,'ABC',33000,'GM');

*write the one column to be display in the student?
Select SNAME from student;
Select EMPNO,ENAME,SAL,JOB FROM EMP;

We can perform some calculation using Arithmetic operations to desired output.

Ex:

```
Select EMPNO,ENAME,SAL,SAL*12,DEPTNO from EMP;
```

```
EMPNO        ENAME        SAL       SAL*12        DEPTNO
101          Mallikarjuna 50000     600000          GM
102           ----         ----       ----          ----
103           ----         ----       ----          ----
```

***Columns ALIAS**: Column ALIAS user defined column heading given to the required column.

Ex:

```
Select EMPNO,ENAME,SAL,SAL*12 ANNUAL_SAL,DEPTNO from EMP;
Select EMPNO ROLL_NO,ENAME,SAL from EMP;
```

Note:

1. Column Alias can be provides for any column.
2. Column Aliases are temporary.
3. The scope of column Alias is to respect to the same query.

***DATA TYPES**:

1. **CHAR**: The data type is used to store alpha numeric values.
It can be also store special symbols like -,:,/,* and etc.
This data type of fixed size. The Maximum size is 255 bytes.

Note: Memory is not used efficiently.

2.**VARCHAR2**: This data type it can store alphanumeric values special symbols like -,:,_ and etc. This data type is of variable size. Maximum size 4000 bytes.

Note: Memory is efficiently used.

3.**NUMBER(P,S)**: p-precision/s-scale
This data type used to store numeric values maximum size 38 digits.

4.**DATE**: This data type is used to store date values it will not have size.
Range 01-JAN-4712 BC TO 31-DEC-9999 AD

Default Date format is DD-MM-YY.

Ex:

```
Create table student1(SNO number(2),
                      SNAME varchar2(20),
                                        MARKS number(3),
                                           DOJ   Date);
```

Insert into student1 values(101,’Ravi’,99,SYSDATE);

5.**LONG**: Similar to varchar2 data type. Maximum size 2 GB.

6.**RAW**: used to store Images,logos,digital signatures etc. Maximum size 255 bytes.

7.**LONG RAW**: Similar to RAW data type. Maximum size 2 GB.

8.**CLOB**:(character large object) used to store characters. Maximum size 4 GB.

9.**BLO**:(binary large object) used to store binary data. Maximum size 4 GB.

10.**BFILE**:(binary file)

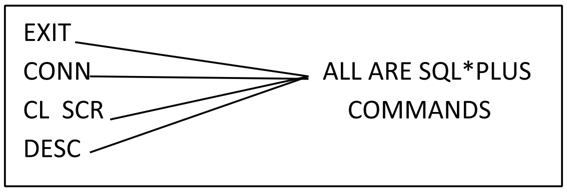
***Describe command**: This command is used to see the structure of the table.
Ex:

```
   DESCRIBE Student //(not use ;)
NAME              NULL        TYPE
SNO                           NUMBER(2)
SNAME                         VARCHAR2(20)
MARKS                         NUMBER(3)
```

NOTE:

It is SQL*PLUS command

Short form is DESC



*<u>**where clause**</u>: class is used to select specific rows basing on a condition.

<u>**Ex**</u>:

```
    Select * from emp where sal>2000;
                  SAL
                  2975
                  2875
                  3000
   Select * from emp where deptno = 10;
   Select * from student where sno < = 103;
   Select eno,ename,sal,deptno from emp where dept = 20;
```

<u>**Note**</u>: Data which is stored the inside the table is case sensitive.

<u>**Ex**</u>:

```
    Select * from emp where job = 'clerk';
    Select * from student where comm is NULL;
```

<u>**Note**</u>: is NULL is operator which is use to retrieve the rows based on NULL values.

<u>***Ex***</u>:

```
  Select * from student where marks is NULL;
  Select * from student where sname is NULL;
```

<u>**Distinct Keyword**</u>: it is display distinct values (unique values).

Duplicates of suppressed.

<u>**Ex**</u>: select distinct deptno from emp;

```
    DEPTNO
       30
       20
       10
Select distinct job from emp;
```

*<u>**update**</u>: The command is used to change data present in the table.

<u>**Syntax**</u>:

```
        Update<TABLE_NAME> set <COL_NAME> = <VALUE> where <CONDITION>;
```

<u>**Ex**</u>:

```
   Update student set marks = 30 where sno = 102;
   Update emp set sal = 1000 where ename = 'scott';
   Update emp set ename = 'Arun' where eno = 7369;
```

Update emp set comm = 500 where eno = 7698;

NOTE: when where clunse is not use, all the rows are updated.

Ex:

Update emp set sal = 5000;

Update emp set sal = 3500 where comm is NULL;
Update student set marks = NULL where sname = 'kiran';
Update emp set sal = NULL where comm is NULL;
Update emp set job = 'HR', sal = 5000 where ename = 'Alen';

***DELETE**: This command is used to remove the complete row from the table.
SYSTAX: Delete from <table_name> where <condition>;

*write a
Ex:
query delete command?

Delete from student where sno = 101;
Update student set sname = null, marks = null where sno = 101;

Note: when where clause is not use all the rows delete.
Ex:

Delete from student1;

***LOGICAL OPERATORS**: There are three logical operators. They are

1. AND
2. OR
3. NOT

1. **AND(T AND T = T)**:
   Systax: select * from EMP where SAL > 2000 AND JOB = 'MANAGER';
   Note: AND operator will return the rows when all the conditions are satisfied.

   Select * from emp where SAL >2000 AND JOB = 'MANAGER' AND deptno = 30;
   Select * from emp where DEPTNO = 10 OR DEPTNO = 20;
   Select * from emp where SAL >2000 AND SAL < 4000;

Write a query to delete first and last rows?

Ex:

Select * from empno = 7499 OR 7521 OR 7566; ✗

Select * from empno =7499 OR empno = 7521 OR empno = 7566;

Note: query to display the rows who are not managers?

Select * from emp where NOT job = 'MANAGER';
Select * from emp where NOT deptno = 30;

BETWEEN: between operator is used to display the rows which is following in the range of values.

Select * from EMP where SAL BETWEEN 2000 AND 3000;

Note: Extreme values are included
Always specify first lower limit first and then higher limit.

*__IN OPERATOR__:
1. IN operator will be return the rows when the values are matching in the list.
2. IN operator can be use as a replacement of OR operator.

Ex: Select * from EMP where EMPNO IN(7369,7499,7521);
Select * from EMP where DEPTNO IN(10,20);
Select * from EMP where DEPTNO IN(10,20,30);

*__PATTERN MATCHING OPERATOR__: They are two pattern matching operator.
1. Percentage (%)
2. Under score (_)

1. **Percentage (%)**: This command is used to select the letters.
   Ex: Select * from emp where ename like 'S%'; //starts with letter 'S'
   Select * from emp where ename like 'R%'; //ends with letter 'R'
   Select * from emp where ename like 'J%S'; // first letter J and last letter S
   Select * from emp where ename like '%A%'; //middle letter A
   Select * from emp where NOT ename like '%A%'; //not middle letter A

2. **Under Score**: This command is also select the letters.
   Ex: Select * from emp where ename like '_A%';
   Select * from emp where ename like '__A%';

   Can we display the rows who's emp name ends with last position 'E'.

   Select * from emp where ename like '%E_';
   Select * from emp where ename like '____';
   Note: Like keyword is used with pattern matching operator.

   Select * from emp where deptno NOT IN(10,20);

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

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

2) **ALTER**: By using ALTER command we can perform the following task.
   - 2.1. ADDING A NEW COLUMN
   - 2.2. DROPING AN EXISTING COLUMN
   - 2.3. MODIFYING A COLUMN
   - 2.4. RENAMING A COLUMN

i. **ADDING A NEW COLUMN**:
   Syntax: ALTER TABLE <TABLE_NAME> ADD (COL1_NAME DATA TYPE(SIZE),
   (COL2_NAME DATA TYPE(SIZE));
   Ex: ALTER table student ADD(city varchar2(10));
   ALTER table student ADD(state varchar2(10), pincode number(6));

Note: New column can be added only at last.
The new column will have null values.
Update student set city = 'Hyderabad' where sno = 101;
Update student set state = 'Andra pradesh' where sno = 101;
Update student set pincode = 500082 where sno = 101;

ii. **DROPING AN EXISTING COLUMN**:
   Syntax: ALTER TABLE <TABLE_NAME> DROP(COL1_NAME,COL2_NAME);

Ex: ALTER table student drop(state);
ALTER table student drop(city,pincode);

iii. **MODIFYING A COLUMN**: (increasing/decreasing the size of columns)
Syntax: ALTER TABLE <TABLE_NAME> MODIFY(COL1_NAME DATA TYPE(SIZE));
Ex:
ALTER table student modify(Sname varchar2(10));
ALTER table student modify(Sname varchar2(8));
Note: .we can increase and decrease as well as the size of the column.
.We can decrease the column size only when existing column values can fit into new size.
.By using modify keyword we can change the data type of a column.
.Column should be empty to change it's data type.

iv. **RENAMING A COLUMN**:
Syntax: ALTER TABLE <TABLE_NAME> RENAME COLUMN<OLD_COL_NAME> TO <NEW_COL_NAME>;
Ex:
ALTER table emp rename column SAL TO WAGES;

3) **DROP**: This command is used to rename the table from the database.
Syntax: DROP TABLE <TABLE_NAME>;
Ex:
DROP table student;
DROP table emp;

4) TRUNCATE: This command is used to remove all the rows from the table.
Syntax: TRUNCATE TABLE <TABLE_NAME>;
Ex: TRUNCATE table student;
Select * from TAB; // ALL TABLE NAME'S ARE DISPLAYED

*Difference between TRUNCATE and DELETE?

DELETE	**TRUNCATE**
• We can Roll Back the data.	* We cannot Roll Back the data.
• Rows are deleting temporally.	* Rows are deleting permanently.
• Where clause can be used.	* Where clause cannot be used.

- Delete is sub language DML. * Delete is sub language DDL.

5) **Rename**: This command is used to change the table name.
   Syntax: RENAME <OLD_TABLE_NAME> TO <NEW_TABLE_NAME>;
   Ex: Rename student To student1;
   Rename SALGRADE To GRADE;

NOTE: When we use a Truncate command the table gets dropped and re-created. As the structure is effected is called as a DDL command.
All DDL command are permanent.
All DML command are Temporary.

*To see list of all the tables of a user: Select * from TAB;

***Creating duplicate tables or backup tables**: By using the combination of create and select. We can create copy of a table.
Ex:

```
Create table emp1 AS select * from emp; //total table copy
Create table emp2 AS select * from emp where deptno = 30; // only deptno = 30
Create table emp3 AS select * from emp where 10; // only deptno =10 is copy
Create table emp4 AS select empno, ename, wages, deptno from emp where
deptno = 20; //empno,ename,wages,deptno is coped by emp4 table
Create table emp5 AS select *from emp where 1=2; //This mean total table copy

Select * from emp where 1 = 1;
Select * from 'malli' from emp;
```

***Right click $\longrightarrow$ properties $\longrightarrow$ options and select quick edit modifier and ok.

/ $\longrightarrow$ Run the same query
ED $\longrightarrow$ Open the Buffer command
SET NUM 5
SCOTT is a new user

*****Creating a new user**:
Connect in the database AS DBA.
*user_name: /AS SYSDBA
*create user: create user malli Identified by malli123; //user and password created

*giving permissions to the user;
*GRANT CONNECT, RESOURCE TO malli; //Grant Succeeded
*SHOW user $\longrightarrow$ To connect the current user.

**__FUNCTIONS__: Functions will manuplate the data items and gives the result. They are two types of functions.

1. Group functions or multiple row functions
2. Scalar functions or single row function

1) Group functions or multiple row functions: This functions act on group of rows.
   i. **__AVG__**: select AVG(sal) from emp;
   ii. **__SUM__**: select SUM(sal) from emp;
   iii. **__MAX__**: select MAX(sal) from emp;
   iv. **__MIN__**: select MIN(sal) from emp;
   v. **__COUNT(*)__**: select COUNT(*) from emp; //Return total no.of rows in the table
   vi. **__COUNT(EXPR)__**: Return no.of values present in the column.
   __Ex__: Select COUNT(sal) from emp;
   Select COUNT(empno) from emp;
   Select COUNT(comm) from emp;

__Dual table__: It is a dummy table which is generally used to perform some calculation is seeing to the system date and etc. Dual table is collection of one row and one column with 'X' in it.
__Ex__: Select SYSDATE from dual;
Select 10+20 from dual;
Select 20+40, 50+60 from dual;

2) **__Scalar functions or single row functions__**: Scalar function are decided into four types. They are given that.
   i. Character functions
   ii. Number functions
   iii. Data functions
   iv. Conversion functions

i. **__Character functions__**:
   a. **__Upper__**: converts into lower case to upper case.
   __Ex__: Select upper('oracle') from dual; //ORACLE

   b. **__Lower__**: This function is convert to the upper to lower case.
   __Ex__: Select lower('ORACLE') from dual; //oracle

Select ENO, lower('ENAME'), SAL from emp;

c. **INITCAP**: First letter is capital and reaming letters are small letters.

Ex: Select INITCAP('oracle training') from dual; //Oracle

Select INITCAP('ORACLE TRAINING') from dual; //Oracle

d. **LENGTH**: Returns length of the string.

Ex: Select LENGTH('oracle') from dual; //length 6

Select LENGTH('MALLIKHARJUNA') from dual; //length 13

Select * from emp where length(ename) = 4;

e. **LPAD**: pads the character towards the left side.

Ex: select LPAD('oracle',10,'z') from dual; //zzzzoracle

f. **RPAD**: Rpad the character towards the right side.

Ex: Select RPAD('ORACLE',10,'X') from dual; //ORACLEzzzz

Select LPAD(RPAD('ORACLE',8,'Z',10,'Z') from dual; //Nesting function

g. **LTRIM**:

Ex: Select LTRIM('zzoracle','z') from dual;

h. **RTRIM**:

Ex: Select RTRIM('ZORACLEZZZ','Z') from dual; //ZORACLE

Select RTRIM('oracle') from dual; // oracle

Select RTRIM('ORACLE   ') from dual; // ORACLE

Select LENGTH(RTRIM('ORACLE   ')) from dual; // length is 6

i. **TRIM**: Removes the specified characters from both sides.

Ex: Select TRIM('z' from 'zzoraclezz') from dual;

Select TRIM('   ORACLE   ') from dual;

j. **INSTR**: Returns the passion of the string

Ex: Select INSTR('ORACLE','A') from dual; //3

Select INSTR('oracle','H') from dual;

Select INSTR('DATABASE','A') from dual; //2

k. **SUBSTR**: Returns the part of the string.

Ex: Select SUBSTR('ORACLE',2,3) from dual; //RAC

Select SUBSTR('ORACLE',2,4) from dual; //RACLE

Select SUBSTR('ORACLE',2,1) from dual; //R

Select SUBSTR('ORACLE',3,2) from dual; //AC

Select SUBSTR('ORACLE',3,10) from dual; //ACLE
Select SUBSTR('ORACLE',3) from dual; //ACLE

l. **CONCAT**: To join the two words. It will accept only two character.
Ex: Select concat('MAGA','STAR') from dual; //MAGASTAR
Select concat(concat('MAGA','STAR'),'CHIRU') from dual;

```
Select * from test1;
SNO     SNAME     MARKS
101     ARUN      80
102     ARUN      80
103     VIJAY     80
```

Select * from test1 where SNAME = 'ARUN'; //101 ARUN 80
Select SNO, SNAME, LENGTH(SNAME) from test1;

```
SNO     SNAME     LENGTH(SNAME)
101     ARUN      4
102     ARUN      5
103     VIJAY     5
```

Select * from test1 where RTRIM(SNAME) = 'ARUN';
Select * from test1 where TRIM(SNAME) = 'ARUN';
Update test1 set SNAME = TRIM(SNAME);
Select SNO, SNAME, LENGTH(SNAME) from test1;

ii. **Number functions**:

a. **ABS**: Returns absolute values
Ex: Select ABS(-40) from dual; // 40
Select ABS(40) from dual; //40

b. **SQRT**: Returns the squawroot values.
Ex: Select SQRT(25) from dual; // 5
Select SQRT(26) from dual; //5.09901951

c. **MOD(A,B)**: Returns the MOD vaues.
Ex: select MOD(10,3) from dual; // 1

d. **POWER(A,B)**:
   Ex: Select POWER(2,5) from dual; // 32

e. **CEIL**:
   Ex: Select CEIL(40.9) from dual; //41
   Select CEIL(40.2) from dual; //41
   Select CEIL(40.5) from dual; //41

f. **FLOOR**:
   Ex: Select FLOOR(40.9) from dual; //40
   Select FLOOR(40.2) from dual; //40
   Select FLOOR(40.5) from dual; //40

g. **TRUNC**:(TRUNCATE) Remove the decimal points.
   Ex: Select TRUNC(40.9) from dual; // 40
   Select TRUNC(40.2) from dual; // 40
   Select TRUNC(40.5) from dual; // 40
   Select TRUNC(40.1234,2) from dual; // 40.12
   Select TRUNC(685.195364,3) from dual; // 685.195
   Select TRUNC(6854,-1) from dual; // 6850
   Select TRUNC(6854,-2) from dual; // 6800
   Select TRUNC(7777,-3) from dual; // 7000

h. **ROUND**: Rounds of the nearest value.
   Ex: Select ROUND(40.9) from dual; //41
   Select ROUND(40.2) from dual; //40
   Select ROUND(40.5) from dual; //41
   Select ROUND(123.863,2) from dual; //123.86
   Select ROUND(123.868,2) from dual; //123.87
   Select ROUND(856.81766,3) from dual; //856.818
   Select ROUND(123,-1) from dual; //120
   Select ROUND(140,-2) from dual; //100
   Select ROUND(127,-3) from dual; //130
   Select ROUND(17763,-3) from dual; //18000

i. **GREATEST**:
   Ex: Select GREATEST(100,200,300) from dual; // 300

j. **LEAST**:
   Ex: Select LEAST(100,200,300) from dual; // 100

iii. **Datafunctions**: They are four data functions.
   A. ADD_MONTHS
   B. MONTHS_BETWEEN
   C. NEXT_DAY
   D. LAST_DAY

A.**ADD_MONTHS**: ADD_MONTHS of months to the given date.
   Ex: Select ADD_MONTHS(SYSDATE,12) from dual; // 16-JUN-13
   Select ADD_MONTHS('11-APR-05',3) from dual; // 11-APR-05

B.**MONTH_BETWEEN**: Returns number of months b/w given the two months.
   Ex: Select MONTHS_BETWEEN('11-JAN-05','11-JAN-04') from dual; // 12
   Select MONTHS_BETWEEN('11-JAN-04','11-JAN-05') from dual; // -12
   Select EMPNO, ENAME, SAL, HIREDATE from emp; //display emp table
   Select EMPNO, ENAME, SAL, MONTHS_BETWEEN(SYSDATE,HIREDATE) from emp;
   Select EMPNO, ENAME, SAL, MONTHS_BETWEEN(SYSDATE,HIREDATE)/12 from emp;
   Select EMPNO, ENAME, SAL, ROUND(MONTHS_BETWEEN(SYSDATE, HIREDATE)/12) from emp;
   Select EMPNO, ENAME, SAL, ROUND(MONTHS_BETWEEN(SYSDATE, HIREDATE)/12) EXP from emp;

C.**NEXT_DAY**: Returns date of the specified date.
   Ex: Select NEXT_DAY(SYSDATE,'MONDAY') from dual; // 18-JUN-12
   Select NEXT_DAY('11-JAN-05','MONDAY') from dual; // 17-JAN-05

D. **LAST_DAY**: Returns the last day of the month.
   Ex: Select LAST_DAY(SYSDATE) from dual; // 30-JUN-12
   Select LAST_DAY('11-FEB-05') from dual; // 28-FEB-05

iv. **Conversion functions**:
   Conversion functions are one data type to another data type conversion.
   They are three conversion functions.

- TO_CHAR
- TO_NUMBER
- TO_DATE

1.**TO_CHAR:** This functions is having two functionalities.

a. **Number to_char:** This function is used only $ or u-rows and number is used 9.

<u>**Ex**</u>: Select eno,ename, TO_CHAR(sal,'9,999') from emp;
Select eno,ename, TO_CHAR(sal,'99,99') from emp;
Select eno,ename, TO_CHAR(sal,'$9,999') from emp;
Select eno,ename, TO_CHAR(sal,'8,888') from emp; // invalid number format

b.**Date to_char:**

<u>**Ex**</u>: Select eno,ename,hiredate from emp;
Select eno,ename, TO_CHAR(HIREDATE,'DD-MM-YY') from emp;
Select eno,ename, TO_CHAR(HIREDATE,'DD-MM-YYYY') from emp;
Select SYSDATE from dual; // 18-JUN-12
Select TO_CHAR(SYSDATE,'DD-MONTH-YY') from dual; // 18-JUN-12
Select TO_CHAR(SYSDATE,'DAY') from dual; // Monday
Select TO_CHAR(SYSDATE,'YYYY') from dual; // 2012
Select TO_CHAR(SYSDATE,'MM') from dual; // 06
Select TO_CHAR(SYSDATE,'DDD') from dual; // 170
Select TO_CHAR(SYSDATE,'DD') from dual; // 18
Select TO_CHAR(SYSDATE,'MON') from dual; // MONDAY
Select TO_CHAR(SYSDATE,'DY') from dual; // mon
Select TO_CHAR(SYSDATE,'DD-MM-YY HH:MI:SS') from dual; //18-06-12 12:40:44
Select * from emp where TO_CHAR(HIREDATE,'YYYY') = '1981';
Select * from emp where TO_CHAR(HIREDATE,'YYYY') = '1980';
Select * from emp where TO_CHAR(HIREDATE,'MON') = 'DEC';

2.**TO_NUMBER:**

Ex: Select TO_NUMBER(LTRIM('$1400','$')) + 10 from dual; // 1410

3.**TO_DATE:** This function is used to convert character values to data value.

Ex: ADD_MONTHS
Select ADD_MONTH('11-JAN-05',2) from dual;
Select ADD_MONTH('11-JANUARY-2005 11:45 A.M.','DD-MONTH-YYYY HH:MI A.M'),2) from dual;
Select ADD_MOONTHS(TO_DATE('11-01-2005 11:45 A.M.',

'DD-MM-YYYY HH:MI A.M.'),2) from dual; //11-MAR-2005

*__implicit convertion__:

Ex: Select * from emp where DEPTNO = '10';

Select sum(sal) from emp where deptno = '10'; //8750

Select sum(sal) from emp where deptno = '20'; //10875

Select sum(sal) from emp where deptno = '30'; // 9400

*__Group By clause__: Group By clause is used to divided rows into several group. So that we can apply group function on each group.

Ex: Select deptno, sum(sal) from emp Group By deptno;

Deptno	**Sal**
30	9400
20	10875
10	8750

Select deptno,min(sal),max(sal) from emp Group By deptno;

Deptno	**Min**	**Max**
30	950	2850
20	800	3000
10	1300	5000

Select job, avg(sal) from emp Group By job;

Select job,count(*) from emp Group By job;

Select job,count(job) from emp Group By job;

Select Deptno, sum(Sal), min(Sal), max(Sal), avg(Sal), count(*) from emp Group By deptno;

Not equal in oracle <>

We can use the combination of where clause and Group By clause.

First where clause is executed on the result of where clause Group By clause is apply.

Select deptno, sum(sal) from emp where deptno <> 10 Group By deptno;

Select deptno, job, sum(sal) from emp Group By deptno, job;

*__Rule of group by clause__: All the column in the select of list should use group functions or should by included in group by clause.

Select deptno, sum(sal), ename from emp Group By deptno;
Error: Not a Group By Expression

*__Having clause__: (to use Group By clause)
Having clause is used to filter the output from Group By clause.
__Ex__: Select deptno, sum(sal) from emp Group By deptno having sum(sal) > 9000;

Deptno	**Sum(sal)**
30	9400
20	10875

*__Order By clause__:
Order By clause is used to arrange the rows in the table.
By default order by clause ascending order.
Null values are arranged in the last.
__Ex__: Select * from emp Order By sal;
Select * from emp Order By ename;
Select * from emp Order By HIREDATE; //Chronological order 1980…..1985
Select * from emp Order By eno;
Select * from emp Order By job,sal;
Select * from emp Order By sal DESC; //descending order by depending the query

__Note__: Order by clause should be the last change of the query.
Select deptno, sum(sal) from emp Group By deptno Having sum(sal) > 9000
Order By sum(sal) DESC;
Select deptno, sum(sal) from emp where ename <> 'King' Group By deptno;
Select deptno, sum(sal) from emp where ename <> 'King' Group By deptno
Having sum(sal) > 9000;
Select deptno, sum(sal) from emp where ename <> 'King' Group By deptno
Having sum(sal) > 9000 Order By sum(sal) DESC;

__Interview questions__

1. What is SQL?
   SQL transfer Structured Query Language are also called as SEQUEL.
2. List out sub language of SQL?
   They are 5 sub languages DDL,DML,DRL/DQL,TCL,DCL .
3. What is different between char and varchar2?
   Char is a fixed size and varchar2 is a not a fixed size.

4. What is projection?
   Selecting specific columns is projection.
5. How can we filter the rows in the table by use the Where, Group BY, Having, Order By clause?
   Select deptno, sum(sal) from emp where ename <> 'KING' Group By deptno Having sum(sal) > 9000 Order By sum(sal) DESC;
6. What is column Alias?
   Providing the duplicate to column Name. This is not a permanent.
7. Can we perform a arithmetic operation by using dual?
   Select 10 + 20 Result from dual;
8. What is dual table?
   Dual table is dummy table to calculate the some problems. This is one column and one row. Specified to 'X'
9. Write a query to display current date along with HH:MI:SS?
   Select To_Char(sysdate,'DD-MON-YYYY HH:MI:SS') from dual?
10. Write a query to see the current date?
    Select sysdate from dual;
11. Which operator is used to accept the values from the user?
    INSERT, UPDATE,CREATE and etc.
12. How can you see all the table which are in the data base?
    Select * from TAB
13. Which command is used to remove the table from the data base?
    Drop command is used to the table.
14. What is different between delete and truncate command?
    Delete to the table and getting the roll back. Truncate is used not possible the table roll back.
15. Which operator is used to retrieve the rows based on null values?
    IS NULL
16. In how many ways we can rename all the rows from the table?
    They are two ways Delete and Truncate
17. How can we create copy of a table?
    Create table emp1 AS select * from emp;
    Create table emp2 AS select * from emp where deptno = 30;
18. Write a query to display no.of rows in the table?
    BY using count(*)
    Select count(*) from emp;
19. What is different between count(*) and count(Expr)?
    *is total table. Expr is only one column to count the table.

20. What is difference between group functions and scalar function?
    Group functions will act on total table and scalar functions will act on one row.
21. What is a use of Group By clause?
    Group By clause will decided into several groups.
22. How can we filter the rows of Group By clause?
    Having clause is used to filter the data.
23. Which clause is used to arrange the rows in the table?
    Order By clause is used to arrange.
24. Which clause should be the last clause of the query?
    Is order By caluse.
    ***Any operation performed on null will result to null values.
25. What is a TOAD?
    Tool for Oracle Application Development.

*__INTEGRITY CONSTRAINS__:
Constrains are rules which are applied on tables.
Constrains helps in improving the accurency and quality of the data base.
They are five types of constrains.

1. NOT NULL
2. UNIQUE
3. PRIMARY KEY
4. FOREIGN KEY or REFERENTIAL INTEGRITY CONSTRAINS
5. CHECK

Constrains can be created at two levels

a. Column level
b. Table level

1.__NOT NULL__: This constrains will not accept null values.
NOT NULL constrains can be created only at column level.
__Ex__:

```
*Create table student1(Sno number(3) NOT NULL,
                                        Sname varchar2(10),
                                                 Marks number(3));

Insert into student1 values(101,’Arun’,50);
Insert into student1 values(101,’Arun’,NULL);
Insert into student1 values(101,NULL,50);
Insert into student1 values(NULL,’Arun’,50); ✗
```

Error: cannot insert into null to scott, student1, sno.

```
*Create table student2(Sno number(3) NOT NULL,
                                        Sname varchar2(10),
                                                 Marks number(3) NOT NULL);

Insert into student2 values(101,’Arun’,50);
Insert into student2 values(101,’Arun’,NULL);  ×
Insert into student2 values(101,NULL,50);
Insert into student2 values(NULL,’Arun’,50);  ×
```

2.**<u>UNIQUE CONSTRAINS</u>**: This constrains will not accept duplicate values. This constrains can be created at column level as well as table level.

<u>Ex</u>: **<u>Creating unique constraint at column level</u>**.

```
     * Create table student3(Sno number(3) UNIQUE,
                                       Sname varchar2(10),
                                                 Marks number(3));

Insert into student3 values(101,’Arun’,50);
Insert into student3 values(102,’Arun’,NULL);
Insert into student3 values(101,NULL,50);   ×
Insert into student3 values(NULL,’Arun’,50);
Insert into student3 values(NULL,’Arun’,50);
```

<u>Note</u>: UNIQUE constraint will accept multiple will values.

```
*Create table student4(Sno number(3) UNIQUE,
                                        Sname varchar2(10),
                                                 Marks number(3) UNIQUE);

Insert into student4 values(101,’Arun’,50);
Insert into student4 values(102,’Arun’,50);  ×
```

<u>Creating unique constraint at table level</u>:

<u>Ex</u>:

```
     * Create table student5(Sno number(3),
                                        Sname varchar2(10),
                                        Marks number(3),
                                            UNIQUE(Sno));
```

```
Insert into student5 values(101,'Arun',50);
Insert into student5 values(102,'Arun',NULL);
Insert into student5 values(101,NULL,50);   ✗
Insert into student5 values(NULL,'Arun',50);
Insert into student5 values(NULL,'Arun',50);
```

<u>Note</u>: There is no different when a constrain at column level or table level.

3. **<u>PRIMARY KEY CONSTRIANS</u>:
A primary key constrains of a combination of NOT NULL and UNIQUE.
A primary key constrains will not accept null values as well as duplicate values.
Primary key column is used to uniquely every row in a table.
A table can have only one primary key.
Primary key constrains can be created at column level or table level.

<u>Create primary key constraint at column level</u>:
<u>Ex</u>:

```
    * Create table student6(Sno number(3) PRIMARY KEY,
                            Sname varchar2(10),
                                 Marks number(3));
Insert into student6 values(101,'Arun',50);
Insert into student6 values(102,'Arun',50);
Insert into student6 values(101,Arun,50);   ✗
Insert into student6 values(NULL,'Arun',50); ✗
Insert into student6 values(103,'Arun',50);
```

<u>Create Primary key constraint at table level</u>:
<u>Ex</u>:

```
    * Create table student7(Sno number(3),
                            Sname varchar2(10),
                                 Marks number(3)
                                     PRIMARY KEY(Sno));
```

***3.1.**<u>COMPOSITE PRIMARY KEY</u>**:
When primary key is applied a multiple columns it is called composite primary key.

Composite primary key can be applied only at table level.

***Creating composite primary key constraint at table level**:

Ex:

```
   Create table student9(Surname varchar2(10),
                                           Firstname varchar2(10),
                                                Marks number(3),
                                                     PRIMARY KEY(Surname,Firstname));
```

```
Insert into student9 values('xyz','Arun',40);
Insert into student9 values('xyz','Kiran',40);
Insert into student9 values('mno','Arun',40);
Insert into student9 values('xyz','Kiran',40); ✗
Insert into student9 values(NULL,'Arun',40); ✗
Insert into student9 values('abc',NULL,40); ✗
```

***4.**FOREIGN KEY CONSTRAINS or REFERENTIAL INTEGRITY**:

These constraints establish relationship between tables.

This relationship is called as parent and child relationship. It is also called master detail relationship.

A foreign key column in the child table will only accept values which are their the primary key column or unique column of parent table.

Creating the parent table:

```
      Create table school(sno number(3),
                                     Sname varchar2(10),
                                         Marks number(3),
                                             primary key(sno));
```

Insert Rows parent table:

```
      Insert into school values(101,'Arun',90);
      Insert into school values(102,'Kiran',92);
      Insert into school values(103,'Amit',45);
```

Creating the child table:

```
      Create table library(sno number(3) REFERENCES school(sno),
              Book_name varchar2(10));
```

Insert Rows child table:

```
Insert into library values(102,’java’);
Insert into library values(103,’c++’);
Insert into library values(103,’oracle’);
Insert into library values(108,’dotnet’); //error
Insert into library values(Null,’DBA’); //valid
```

Foreign key column name need not match with primary key column name or unique column name. But the data type should match.
To establish relationship it is mandatory that the parent table should have primary key constraint or at least unique constraints.

Delete from school where sno = 101;
1 row deleted
Delete from school where sno = 102; //error
Message: child record found
Note: we can not delete to the row from the parent table in the corresponding value existing child table.

***Using on delete cascade**:
When using on delete cascade. We can delete the rows from the parent table and the corresponding child table rows deleted automatically.

Create the parent table:

```
Create table school1(sno number(3),
                     Sname varchar2(10),
                         Marks number(3),
                             primary key(sno));
```

Insert Row parent table:

```
Insert into school1 values(101,’Arun’,90);
Insert into school1 values(102,’Kiran’,92);
Insert into school1 values(103,’Amit’,45);
```

Creating the child table:

```
Create table library1(sno number(3) REFERENCES school1(sno)
                 on delete cascade,
                     Book_name varchar2(10));
```

<u>Insert Rows child table</u>:

```
Insert into library1 values(102,’java’);
Insert into library1 values(103,’c++’);
Insert into library1 values(103,’oracle’);
Insert into library1 values(108,’dotnet’); //error
Insert into library1 values(Null,’DBA’); //valid
```

```
Delete from student1 where sno = 101; //valid
1 row deleted
Delete from student1 where sno = 102; //valid
1 row deleted
```

One row will be deleting from parent table.
One row will be deleting from child table automatically.

***<u>On delete set null</u>**: When delete the row from parent table. The corresponding value will be changed to null.

<u>Create the parent table</u>:

```
Create table school2(sno number(3),
                     Sname varchar2(10),
                        Marks number(3),
                           primary key(sno));
```

<u>Insert Row parent table</u>:

```
Insert into school2 values(101,’Arun’,90);
Insert into school2 values(102,’Kiran’,92);
Insert into school2 values(103,’Amit’,45);
```

<u>Creating the child table</u>:

```
Create table library2(sno number(3) REFERENCESschool2(sno)
                   on delete set null,
                      Book_name varchar2(10));
```

<u>Insert Rows child table</u>:

```
Insert into library2 values(102,’java’);
Insert into library2 values(103,’c++’);
Insert into library2 values(103,’oracle’);
Insert into library2 values(108,’dotnet’); //error
```

```
        Insert into library2 values(Null,'DBA'); //valid
```

Delete from school2 where sno = 102; //valid
One row from parent table is deleted.
Foreign key column value 102 is changed to null.

*<u>**Create foreign key constraint at table level**</u>:

```
        Create table school3(sno number(3),
                                    Sname varchar2(10),
                                         Marks number(3),
                                              primary key(sno));
```

<u>**Insert Row parent table**</u>:

```
        Insert into school3 values(101,'Arun',90);
        Insert into school3 values(102,'Kiran',92);
        Insert into school3 values(103,'Amit',45);
```

<u>**Creating the child table**</u>:

```
        Create table library3(Rollno number(3),
                              Book_name varchar2(10),
                                     Foreign key (Rollno)
                                    REFERENCES school3(sno) on delete cascade);
```

*<u>**Check constraint**</u>: Check constraint is used to define domain of a column. Domain of column means values a column can store.

```
Create table student4(sno number(3),
                           Sname varchar2(10),
                                 Marks number(3));
```

```
Insert into student4 values(101,'ARUN',66);
Insert into student4 values(102,'ARUN',80);
Insert into student4 values(103,'ARUN',166);
```

*<u>**Domain**</u>: Create table student5(sno number(3),

```
                                   Sname varchar2(10),
                                        Marks number(3) check(marks <=100));
```

```
Insert into student5 values(101,’ARUN’,60);
Insert into student5 values(102,’ARUN’,80);
Insert into student5 values(103,’ARUN’,160); //Error
Msg: check constraint violated
Insert into student5 values(101,’ARUN’,-160);

Create table student6(sno number(3),
                                        Sname varchar2(10),
                                         Marks number(3) check(marks between 0 AND 100));

Insert into student6 values(101,’ARUN’,60);
Insert into student6 values(101,’ARUN’,80);
Insert into student6 values(101,’ARUN’,160);//Error
Create table student7(sno number(3),
                                          Sname varchar2(10);
                                               Marks number(3),
            City varchar2(10) check(city IN(‘HYDERABAD’,’CHENNAI’,’DELHI’));

Insert into student7 values(101,’ARUN’,66,’HYDERABAD’);
Insert into student7 values(101,’ARUN’,66,’DELHI’);
Insert into student7 values(101,’ARUN’,66,’CHENNAI’);
Insert into student7 values(101,’ARUN’,66,’NELLORE’); //Error

Create table student8(sno number(3),
                                       Sname varchar2(10) check(Sname = upper(sname)),
                                               Marks number(3));

Insert into student8 values(101,’ARUN’,60);// Valid
Insert into student8 values(101,’ARuN’,60);// error
Insert into student8 values(101,’arun’,60);// error
```

*__Check constraint at table level__:

```
Create table student8(sno number(3),
                                       Sname varchar2(10),
                                              Marks number(3),
                           Check (sname = upper(Sname)));
```

Note: Every constraint will have a constraint name in the format of SYS_Cn(where N is number).
Ex: SYS_c004525, SYS_c004526

*****Data Dictionary tables**:
Select TABLE_NAME, CONSTRAINT_NAME,CONSTRINT_TYPE from USER_CONSTRAINT;
Set of predefined table which constraints meta data information are called as data dictionary table.
Ex: USER_CONSTRAINT
Query to find constraint_name and constraint_type of the table.

Select TABLE_NAME, CONSTRAINT_NAME, CONSTRAINT_TYPE from USER_CONSTRINT where TABLE_NAME = 'student5'; or 'student7';

We can also provide user defined constraint_name
Ex:
```
        Create table student9(sno number(3) CONSTRAINT MY_UN UNIQUE,
                                                              Sname varchar2(10),
                                                                        Marks number(10));
```

In the above query "MY_UN" is the constraint_name.

*****ALTER**:
*****Adding Constraints**: Alter command is used to add a constraint to an Existing table.
Ex:
```
       Create table Student10(sno number(3),
                                                       Sname varchar2(10),
                                                               Marks number(3));
```

```
Insert into student10 values(101,'Arun',60);
Insert into student10 values(102,'Arun',80);
Insert into student10 values(103,'Arun',90);
```

```
ALTER table student10 ADD(Primary key(Sno));
```

*****Dropping a constraint**:
Alter command is used to drop a constraint to an existing table.
Ex:

```
Alter table student10 DROP Primary key;
```

*__Unique Constraint__:

__Ex__:

```
Create table Student11(sno number(3),
                       Sname varchar2(10),
                          Marks number(3));

Insert into student11 values(101,'Arun',60);
Insert into student11 values(102,'Arun',80);
Insert into student11 values(103,'Arun',90);
ALTER table student11 ADD(Unique(sno));
ALTER table student11 DROP Unique(Sno);
```

***__JOINS__: joins are used to retrieve the data from multiple tables.

__Types of Joins__:

1. EQUI_JOIN
2. NON EQUI_JOIN
3. SELF JOIN
4. OUTER JOIN
   - 4.1 Right outer join
   - 4.2 Left outer join
   - 4.3 Full outer join

1.__EQUI_JOIN__: when tables are joined basing on a common column it is called EQUI_JOIN.

__Ex__:

```
select empno, ename, dname
        from emp, dept
              where emp.deptno = dept.deptno;
```

output:

```
EMPNO   ENAME     DNAME
7369    SMITH     RESEARCH
7499    ALLEN     SALES
7521    WARD      SALES
```

__Note__:

We need to mention join conditions in the where clause.

In EQUI_JOINS we along use to equal to operator in join condition.

__Ex__:

```
Selete empno, ename, sal, job, dname, loc
      from emp, dept
             where emp.deptno = dept.deptno;

Selete empno, ename, sal, deptno, dname, loc
      from emp, dept
             where emp.deptno = dept.deptno;// error

Selete empno, ename, sal, emp.deptno, dname, loc
      from emp, dept
             where emp.deptno = dept.deptno; //valid
```

__**Note**__: we need to mention table name dot column(emp.deptno) name for the common column to resolve the any table.
The common column can be retrieved from any of the table.
We can filter the data from the result of join.
__Ex__:

```
Select empno, ename, sal, emp.deptno, dname, loc
             from emp, dept
                   where emp.deptno = dept.deptno AND sal > 2000;
```

To improve the performance of the join we need mention table name dot column name for all the columns.
__Ex__:

```
 Select emp.empno, emp.ename, emp.sal,emp.deptno, dept.dname, dept.loc
             from emp,dept
                   where emp.deptno = dept.deptno AND sal > 2000;
```

*__**Table alias**__:
Table alias is an alternate name given to a table.
By using a table alias length of the table reduces and at the same time performance is maintains.
Table alias are create in same clause can be used in select clause as well as where clause.
Table alias is temporary once the query is executed the table alias are losed.
__Ex__:

```
Select E.Empno, E.Ename, E.sal, E.deptno, D.Dname, D.loc
             from emp E, Dept D
                  where E.deptno = D.deptno;
```

*<u>**Join the multiple tables(3 tables)**</u>:
Select * from Areas;

<u>**City**</u>	<u>**State**</u>
Newyork	AP
Dallas	Mh

<u>**Ex**</u>: Select E.empno, E.ename, E.sal,D.dname,A.state from emp E, dept D, Areas A
where E.deptno = D.deptno AND D.loc = A.city;

<u>**Note**</u>: To join 'n' tables we need n-1 conditions.

*<u>**NON EQUI JOIN**</u>: When we do not use NON EQUI JOIN to operator in the join condition is NON EQUI JOIN.

<u>**Ex**</u>:
Select * from SALGRADE;

<u>**GRADE**</u>	<u>**LOSAL**</u>	<u>**HISAL**</u>
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

Select e.empno, e.ename, e.sal, s.grade from emp e, salgrade s
where e.sal BETWEEN s.losal AND hisal;

<u>**EMPNO**</u>	<u>**ENAME**</u>	<u>**GRADE**</u>
7369	SMITH	1
7876	ADAMS	1
7900	JAMES	2

Select e.empno, e.ename, s.grade from emp e, salgrade s
where e.sal BETWEEN s.losal AND s.hisal AND s.grade = 4;

*<u>**SELF JOIN**</u>: When a table is joining to it self it is called self join. In self joins we need to create two table aliases for the same table.

Select empno, ename, job, mgr, from emp;
Select e.empno, e.ename, e.job, m.ename from emp e, emp m
where e.mgr = m.empno;

<u>**Empno**</u>	<u>**Ename**</u>	<u>**Job**</u>	<u>**Ename**</u>

```
7902                FORD          ANALYST      JONES
7869                SCOTT         CLERK        JONES
7900                JAMES         SALESMAN     BLAKE
```

*__CARTESIAN PRODUCT__:
When tables are joined without any join condition it is called Cartesian product. In the result we get all possible combination.

```
Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc
         from emp e, dept d; //14*4=56 rows are selected
```

*__ANSI JOINS__: They are the three types.
1. __Inner joins__: It is same as Equi join.
__Ex__:

```
     Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc
         from emp e INNER JOIN dept d ON(e.deptno = d.deptno);
```

2.__NATURAL JOIN__: It is same as Equi join.
__Ex__:

```
      Select empno, ename, sal, deptno, dname,loc from NATURAL JOIN dept;
```

3.__CROSS PRODUCT/CROSS JOIN__: It is same as Cartesian product.
__Ex__:

```
     Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc
         from emp e CROSS JOIN dept d; //14*4 = 56 rows are displayed.
```

*__DEFAULT__:
__Ex__:

```
    Create table stu1(sno number(3),
                            Sname varchar2(10),
                                 Marks number(3) default 100,
                                       Doj Date DEFAULT sysdate);

   Insert into stu1(sno, sname) values(101,'malli');
   Insert into stu1 values(102,'ARUN',40,'11-JAN-09');
   Insert into stu1 values (103,'KIRAN',NULL,'12-FEB-10');
```

SNO	SNAME	MARKS	DOJ
101	malli	100	26-JUN-12
102	ARUN	40	11-JAN-09

103 KIRAN 12-FEB-10

*<u>**SUPER KEY**</u>: Combination of columns which can be used unique key identify every row is called as super key.

Table	object
Column	Attributes
Row	Tuple/Record

*<u>**OUTER JOINS**</u>: It is extension of EQUI JOINS.

In outer joins we get match as well as non matching rows.

(+) This called as outer join operator.

1. <u>**RIGHT OUTER JOIN**</u>:

<u>**SQL Syntax**</u>:

```
Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc
        from emp e, dept d
              where e.deptno(+) = d.deptno; //14 + 1 = 15 rows
```

empno	**ename**	**sal**	**deptno**	**dname**	**loc**
7900	james	950	30	sales	chicago
8963	adams	1400	20	clerk	newyork
6798	adams	2000	10	sales	india
				anaylist	ap

*<u>**ANSI SYNTAX OF RIGHT OUTER JOIN**</u>:

ANSI SYSTAX:

```
Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc
        from emp e RIGHT OUTER JOIN dept d ON(e.deptno = d.deptno);
```

2. <u>**LEFT OUTER JOIN**</u>:

<u>**SQL Syntax**</u>:

```
Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc
    from emp e, dept d
           where e.deptno = d.deptno(+); //14+3 = 17 row displayed
```

<u>**ANSI SYNTAX OF LEFT OUTER JOIN**</u>:

ANSI SYNTAX:

```
Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc
        from emp e LEFT OUTER JOIN dept d ON(e.deptno = d.deptno);
```

3. **FULL OUTER JOIN**:
   ANSI SYNTAX:

```
Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc
    from emp e FULL OUTER JOIN dept d ON(e.deptno = d.deptno);
```

   //14 + 2 + 3 = 19 rows are displayed.

*__SET OPERATORS__: Set operators are used to retrieve the data from multiple tables. They are different types.

1. **UNION**:

   Select * form student10;

Sno	**sname**	**marks**
101	Arun	40
102	Arun	50
103	Arun	69

   Select * from student20;

Sno	**sname**	**marks**
103	Arun	90
104	Arun	60

   Union Syntax: (no duplicates)

```
Select sno from student 10
Union                              //0/p sno: 101 102 103 104
Select sno from student 20;
```

2. **UNION ALL**:
   Union All Syntax: (All rows)

```
Select sno from student 10
Union All                          // o/p sno: 101 102 103 103 104
Select sno from student 20;
```

3. **INSERT SECT**:
   Insert Sect Syntax: (common rows)

```
Select sno from student 10
Insert Sect                        // o/p sno: 103
Select sno from student 20;
```

4. **MINUS**:

```
Select sno from student 10
Minus                          //o/p sno: 101,102
Select sno from student 20;

Select sno from student 20
Minus                        // o/p sno: 104
Select sno from student10;
```

RULES OF SET OPERATORS:

1. Number of columns used in the query should match.
2. Column data type should match for the queries in set operators.

```
Select empno from emp
Union
Select sno from student10
Union
Select deptno from dept;        // valid
```

INTERVIES QUESTIONS

1. What is need for Integrity constraint?
   Constrains are rules which are applied on tables.

2. List out types of constraints?
   They are 5 types NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, and CHECK.

3. In how many level constraints can be created?
   Those are two levels i.e. column level and table level.

4. Which can constraint can be created?
   The constraint created not null.

5. Dose not null constraints accept duplicate values?
   Yes

6. Which constraint is used to unique for every row in the table?
   Primary key

7. What is composite primary key?
   When primary key is applied on multiple columns it is called composite primary key. Composite primary key can be applied only at table level.

8. Can a table name two primary key?
   It is not possible.

9. What is foreign key constraint explain?
   This foreign key is a established in parent table and child table relationship.

10. Can we establish a parent & child relationship without having constraint in the parent table?
    On

11. Con you explain change related to foreign key on delete cascade on delete set null constraint?
    Foreign key column in the child table will only accept values which are their the primary key column or unique column.
    We can delete the rows from the parent table and the corresponding child table rows deleted automatically.
    When we delete row from parent table. The corresponding values will be changed to null.

12. Does every constraint it have constraint name?

13. How can you know constraint name and combination type apply for a table?
    By using user constraint.

14. Is their any different when a constraint is a created at column level or table level?
    No difference.

15. Can you provide user defined constraint name?
    Yes

16. What are data dictionary tables?
    Predefined tables or user constraints

17. What is the need for join?
    To retrieve the multiple tables

18. What is EQUI join?
    When tables are joined basing on a common column it is called EQUI_JOIN.

19. How many conditions are required to join ‘n’ tables?
    We need to n-1 conditions.

20. How can be display matching as well as non matching rows?
    By using outer joins.

21. What is outer join operator?
    (+)

22. What is Cartesian product?
    All possible in the table matching

23. What is difference between union and union all?
    The union set operator display only original values and union all set operator is display all values. Duplicate values also.

****<u>TCL (Transaction Control Language</u>**): It is collection of three commands. They are

1. COMMIT
2. ROLLBACK
3. SAVE POINT

***<u>Commit</u>**: This command is used to make changes permanent to the data base.

<u>Syntax</u>: COMMIT;

<u>Ex</u>:

```
  Create table student(sno number(3),
                                        Name varchar2(10),
                                                  Marks number(3));

 Insert into student values(300,’Arun’,69);
 Insert into student values(301,’Kiran’,69);
 Insert into student values(302,’Naga’,69);
```

```
Select * from student300;
      Create table student1(sno number(3),
                                           Name varchar2(10),
                                                      Marks number(3));

      Insert into student1 values(300,'Arun',69);
      Insert into student1 values(301,'Kiran',69);
      Insert into student1 values(302,'Naga',69);

COMMIT;
```

In three ways we can make changes permanent to the data base.

1. By using command COMMIT
2. By using DDL command
3. By using the environment using EXIT command

*<u>**ROLLBACK**</u>: The rollback will undo the changes which are not permanent.

<u>**Syntax**</u>:

```
            ROLLBACK;
```

<u>**Ex**</u>:

```
      Create table student2(sno number(3),
                                           Name varchar2(10),
                                                      Marks number(3));

      Insert into student2 values(300,'Arun',69);
      Insert into student2 values(301,'Kiran',69);
      Insert into student2 values(302,'Naga',69);

COMMIT;

      Insert into student2 values(304,'Arun',69);
      Insert into student2 values(305,'Kiran',69);

Select * from student2;   //display 5 rows
ROLLBACK;
Select * from student2;  //display 3 rows there are permanently
```

*__SAVE POINT__: Save points are logical marking given for series of transaction. Instead of complete rollback we can rollback to a save point.

__Syntax__: SAVEPOINT<SAVEPOINT_NAME>;

__Ex__:

```
    Create table student3(sno number(3),
                            Name varchar2(10),
                                   Marks number(3));

    Insert into student3 values(300,'Arun',69);
    Insert into student3 values(301,'Kiran',69);
    Insert into student3 values(302,'Naga',69);
    Insert into student3 values(303,'Arun',69);
    Insert into student3 values(304,'Kiran',69);

SAVEPOINT A;

    Insert into student3 values(305,'Naga',69);
    Insert into student3 values(306,'Kiran',69);
    Insert into student3 values(307,'Naga',69);

Select * from student3;    //8 rows displayed

ROLLBACK;

Select * from student3;    //5 rows are displayed

Create table student4(sno number(3),
                            Name varchar2(10),
                                   Marks number(3));

    Insert into student4 values(300,'Arun',69);
    Insert into student4 values(301,'Kiran',69);

SAVEPOINT P;

   Insert into student4 values(302,'Naga',69);
   Insert into student4 values(303,'Naga',69);
```

SAVEPOINT Q;

```
     Insert into student4 values(304,'Naga',69);
     Insert into student4 values(305,'Naga',69);
```

```
Select * from student4;     // 6 rows
ROLLBACK;
Select * from student4;     //0 rows
```

Note: All the save points are lost when the DB is permanent.

***__**DCL (Data Control Language)**__:

They are two Data Control Languages.

1. GRANT
2. REVOKE

Schema: Schema is a memory location which is associated to a user.
It is collection of objects.

Privilege: privileges are permissions (rights given to a user)
They are two types of privileges.

1. System Privileges
2. Object Privileges

***System Privileges**: These privileges are given by DBA to user.

***Object Privileges**: These Privileges are given by one user to another user.

***GRANT**: Grant command is used to Grant the privileges to the user.

Syntax:

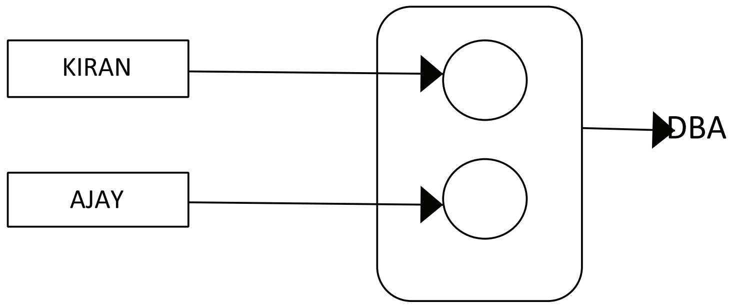
```
          GRANT <PRIVILEGE_NAME1>,<PRIVILEGE_NAME2> TO <USER_NAME>;
```

Ex:

```
    Create user kiran IDENTIFIED by kiran123;
    Create user naga IDENTIFIED by naga123;
```

```
DBA> GRANT CONNECT, RESOURCE to kiran;
```

***Object Privileges**: These privileges are given by one user to another user.



KIRAN> create table student(Sno number(3),
                                        Name varchar2(10),
                                              Marks number(3));

KIRAN> insert into student values(101,'Arun',99);
KIRAN> insert into student values(101,'Anil',97);
KIRAN> insert into student values(101,'Anitha',95);

COMMIT;

Select * from student;

AJAY> select * from student; //There is no response

KIRAN> GRANT select ON student TO AJAY; // KIRAN given privileges to AJAY

Examples of object privileges are select, insert, update, drop and alter etc.

KIRAN> GRANT insert, update ON student TO AJAY;
AJAY> insert into KIRAN.student values(104,'Nandini',89);
AJAY> update KIRAN.student set sno = 102 where ename = 'Arun';
KIRAN> GRANT ALL ON student TO AJAY,ANIL;
KIRAN> GRANT select ON student TO public;

****REVOKE**: This command is used to get back the privileges which are granted.
Syntax: REVOKE<privilege_name><privilege_name> ON <table_name> from <user_name>;
Ex:
KIRAN> REVOKE select, insert ON student from AJAY;

```
KIRAN> REVOKE ALL ON student from AJAY_ANIL;
KIRAN> REVOKE select ON student from public;
```

****DML** (data manipulation language)

MERGE: MERGE command is used as a combination of insert and update.

Select * from student10;   // 3 rows are displayed

SNO	**SNAME**	**MARKS**
101	Arun	30
102	Anil	40
103	Kiran	50

Select * from student20;   // 2 rows are selected

SNO	**SNAME**	**MARKS**
101	James	90
105	Smith	50

```
KIRAN> merge into student10 s1
2>        using student20 s2
3>        on(s1.sno = s2.sno)
4>        when matched
5>        then updated set sname = s2, sname, marks = s2, marks
6>        when not matched
7>        then insert(sno, sname, marks) values(s2.sno, s2.sname, s2.marks);
```

o/p: 2 rows merge

SNO	**SNAME**	**MARKS**
101	James	90
102	Anil	40
103	Kiran	50
105	Smith	50

Note: There will be no changes student20 table.

```
Delete from emp where ename = null;
Select * from emp where eno = 7369 or eno = 7499 or eno = 7521;
Select * from emp where eno IN(7369,7499,7521);
```

****PSEUDO COLUMNS**:

1. **ROWNUM**: It is ROWNUM is a pseudo column which starts with one and increment by 1. (1 and 1+1)
   Ex:
   ```
   Select Rownum, empno, ename, sal, deptno from emp;
   ```
   Rownum values are temporary.
   Rownum values are generated when query is executed.
   Rownum values generation always starts with one and increment by one.

*Query to display first three rows from emp table?
```
Select * from emp where Rownum <=3;
Select * from emp where Rownum <=10;
```

*write a query to display 5th row of emp table?
```
Select * from emp where Rownum <=5
Minus
Select * from emp where Rownum <=4;   //5th row is display
```

*write a query to display 3rd row to 7th row?
```
Select * from emp where Rownum <=7
Minus
Select * from emp where Rownum <=2; //3rd to 7th row display
```

<u>ROWID</u>:
ROWID is pseudo column which contains hexadecimal values.
ROWID value indicates the address where the row is store in the database.
ROWID values are permanent.
Ex:
```
Select ROWID, empno, ename, sal, deptno from emp;
```

*Difference between ROWNUM and ROWID?

ROWNUM	**ROWID**
1.Rownum values starts with 1 and increment by one.	1. Rowid's are hexadecimal values.
2.Rownum values are temporary.	2. Rowid values are permanent.
3.Rownum values are generated when query is exiecuted.	3. The Rowid values are generated when Row is created or inserted.

```
Create table student(Sno number(3),
                                        Sname varchar2(10),
                                                  Marks number(3));
Insert into student values(101,'Arun'60);
Insert into student values(101,'Arun'60);
Insert into student values(101,'Arun'60);
Insert into student values(102,'Arun'70);
Insert into student values(102,'Arun'70);

Select * from student;

Delete from student where Rownum IN(1,2,3); // this is not right query
Select Rowid, sno, sname, marks from student;
Select min(Rowid) from student;
Select min(Rowid) from student group by sno;
```

*__Subqueries__:

Subqueries are used to get the result based on unknown values. They are different type.

1. Single Row subquery
2. Multiple Row subquery
3. Multiple column subquery
4. Co-related subquery
5. Scalar subquery
6. Inline view

*__Single Row Subquery__:

When subquery returns one row (1 value). It is called Single RowSubquery.

__Ex__: write a query to display details are having salary > 'ALLENS' sal ?

Select * from emp where sal > (select sal from emp where ename = 'ALLEN');

o/p: 1600

__Note__:

Subqueries should always place in the inside.

Subqueries are executed first and then parent query is executed by using the result of sub query.

Level Two query**:**

Select * from emp where job = (select job from emp where ename = 'ALLEN')
AND job = (select job from emp where ename = 'BLAKE');

Level Three query:

```
Select * from emp where sal > (select sal from emp
                     Where ename = (select ename from emp
                                    Where empno = 7499));
```

Note: The above query is three level query.
Sub query can be nested upto 32 levels.

****Multiple Row Subquery**:
When subquery returns multiple rows. It is called multiple row salary.
Note: we should multiple row operators with multiple row subqueries. They are three multiple row operators.

1. IN
2. ANY
3. ALL

***ALL**: Select * from emp

```
                     Where sal > ALL(Select sal from emp
                                     Where deptno = 30);
```

Empno	**ename**	**job**	**sal**
7369	SMITH	salesman	2975
7860	ALLEAN	ANALYST	3000

Ex: Select * from emp where sal < ALL(1600,2500,1250,3000,950);

***ANY**: Select * from emp where sal > ANY(select sal from emp

```
                                                       where deptno = 30);
```

Select * from emp where sal < Any(select sal from emp where deptno = 30);

***IN**: Select * from emp where ename IN(‘ALLEN’, ‘KING’,’FORD’);
Select * from emp where sal IN(select sal from emp where deptno = 30);

***MULTIPLE COLUMN SUBQUERY**:
When subquery return more then one column. It is called multiple column subquery.
We should use in operator with multiple column subqueries.

<u>Ex</u>:

Select * from emp where(job,sal) IN(select job, sal from emp where deptno = 30);

```
o/p:    Job        sal
        salesman  1600
        manager   1250
        salesman  2850
```

<u>Note</u>: In the o/p we get the rows when both the values are matching.

<u>Delete some valuesu</u>:

Select * from student;

Select min(rowid) from student group by sno;

Select max(rowid) from student group by sno;

```
Delete from student
        where rowid not
                IN(select min(rowid) from
                        student group by sno);
```

*write a query to display the row from emp table who is having the highest salary?

Select * from emp where sal = (select max(sal) from emp);

*write a query to display all the rows who are having salary grater than AVG salary of emp table?

Select * from emp where sal >(select AVG(sal) from emp);

*write a query to display all deptno AVG salary?

Select deptno, AVG(sal) from emp group by deptno;

***<u>Co-RELATED SUBQUERY</u>**:

When subquery is executed in relation to parent query, it is called co-related subquery.

*write a query to display all the rows who are having salary grater than AVG salary his department?

Select AVG(sal) from emp;

```
Select * from emp where sal > (select AVG(sal) from emp group by deptno); //invalid
Select * from emp where sal > (select AVG(sal) from emp where deptno = 10);

***Select * from emp e
        where sal > (select AVG(sal) from emp where deptno = e.deptno);
```

o/p:

sal	**deptno**
1600	30
2975	20
2850	30
3000	20
5000	10
3000	20

The above example is a co-related subquery.
In co-related subquery, parent query is executed first and then subquery is executed in relation to result of parent query.

***SCALAR subquery**: when we use subquery in the select clause. It is called as Scalar subquery.
*write a query to display following output?

Deptno	**Dname**	**loc**	**sumsal**
10	Accounting	New York	8750
20	Research	Dallas	10875
30	Sales	Chicago	9400
40	Operations	Boston	------

```
Select deptno, dname, loc, (Select sum(sal) from emp where deptno = d.deptno)
                Sum_sal from dept d;
```

Scalar subquery are also called sub select.

***INLINE VIEW**:
When a subquery is used in from clause. It is called INLINE view.

```
Select Rownum, empno, ename, sal, deptno from emp;
Select * from emp where Rownum <=5;
Select * from emp;
Select * from emp ORDER BY sal desc;
```

*write a query to display details of employees who are having top 5 salaries?

```
Select * from emp where Rownum <=5 ORDER BY  sal desc;
Select * from (select * from emp ORDER BY sal desc) where rownum <=5;
```

*write a query to display details of 5th highest salary?

```
Select * from (select * from emp ORDER BY sal desc)
        where rownum <=5)
minus
Select * from (select * from emp ORDER BY sal desc)
                 where  rownum <=4;
```

clause		**subcaluse**
select	yes	scalar
from	yes	inline
where	yes	
group by	no	
having	yes	
order by	no	

INTERVIEW QUESTIONS

1. What are pseudo columns?
   It is rownum is a pseudo column which starts with one and increment by 1.

2. Write a query to display first n rows from the table?
   Select rownum, empno, ename, sal, deptno from emp;

3. What are different between rownum and rowid?

Rownum	**Rowid**
Rownum values starts with 1 and increment by one.	Rowid's are hexadecimal values.
Rownum values are temporary.	Rowid values are permanent.
Rownum values are generated when query is executed.	The Rowid values are generated when row is created or inserted.

4. Write query to delete duplicate rows from the table?
   Delete from student where Rowid Not IN(select min(Rowid)

from student group by sno);

5. write a query to display the first five of highest?
   Select * from(select * from emp ORDER BY sal desc) where rownum <=5)
   Minus
   Select * from(select * from emp ORDER BY sal desc) where rownum <=4);

6. Explain about correlated subquery?
   When subquery is executed in relation to parent query, it is called correlated subquery.

7. Whar are multiple row operators?
   IN
   ANY
   ALL

8. Explain scalar subquery?
   When we use sub query in the select clause it is called scalar subquery.

9. Explain inline view?
   When a sub query is used inform clause it is called inline view.

*__views__: view is a logical representation of data from one or more then one table.
They are different types
Simple views
Complex views
Read only views
With check option views
Materialized views

*__Single views__: when view is created using one base table it is called as Single view.
__Sybtax__:

```
Create view<view_name> as <select STMT>;
```

__Ex__:

```
Create view v1 as select empno, ename, sal from emp;
```

View does not contain any data.
View does not consume memory location.
When we write select statement on view, we get the data from the table.

Ex: Select * from v1;

Empno	**Ename**	**Sal**
7369	Smith	800
7499	Allen	1600
-------	------	------
-------	------	------

Tables which are used for creating the view are called as above tables.
Select * from TAB; will gives list of all the tables and view which are the data base tables.
Ex:
Create view emp_V10 AS select empno, ename, sal, deptno, from emp
where deptno = 10;
Create view emp_V10 AS select empno, ename, sal, deptno, from emp
where deptno = 20;
Create view emp_V10 AS select empno, ename, sal, deptno, from emp
where deptno = 30;

Select * from V10;
Select * from V20;
Select * from V30;

We can perform DML operations on simple views.
Any DML operation performed on simple view will be reflected on base table.
To see the structure of the table.
Ex:
DESC V1

Name	**Null**	**Type**
Empno	Notnull	number(4)
Ename		varchar2(10)
Sal		number(7,2)

*Query to see only view tables?
Select view_name from user_views;
Outputs: V1, emp_v10, emp_20, emp_30, V5

user_views is an example of data dictionary table.
Create view test_v6 as select empno, ename, sal, deptno from emp where deptno = 30;
Select * from Test_v6;   // 6 rows are selected
Insert into Test_v6 values(6868, ‘Anil’,2000,10);
Select * from Test_v6;   // 6 rows are selected
Select empno, ename, sal, deptno from emp where deptno = 30;

View_name	**Text**
Test_v6	select empno, ename, sal, deptno from emp where deptno = 30;

In user_view from the database we get list of all the view and corresponding select statement used for the view.
Select view_name, Text from user_views;

*__Complex view__:
When a view is created using multiple base tables it is called Complex view.
Ex:

```
        Create view Test_v7
          As select e.empno, e.ename, e.sal, e.deptno,
           d.dname, d.loc from emp e, dept d where e.deptno = d.deptno;
```

insert into Test_v7 values(7878,’ravi’,9000,40,’HR’,’HYD’);   // Error
msg: DML operations are not allowed in complex views.

```
Create view Test_v8
          As select empno, ename, sal, sal*12 from emp;   // Error
Create view Test_v8
          As select empno, ename, sal, sal*12 Annual_sal from emp;
Select * from Test_v8;
Insert into Test_v8 values(1212,’GMR’,1000,12000);        // Error
Create view Test_v9
          As select empno, ename, Lower(ename) Name from emp;
```

A view is called as complex view when we use arithmetic operations an functions are group by clauses.
Create view Test_v10

```
          As select deptno, sum(sal) sum_sal from emp group by deptno;
```

***Different between simple and complex views?

Simple view	Complex view
1. Created by using only one table.	1. Created by using multiple tables.
2. DML operations are allowed.	2. DML operations are not allowed.
3. Should not be created using arithmetic operations or functions or group by clauses.	3. Can be created using arithmetic operations or functions or group by clauses.

*<u>**Read Only View**</u>:

We can restrict DML operation views by creating read only view.

<u>**Ex**</u>:

```
     Create view v3
          As select empno, ename, sal, deptno from emp with read only;
Select * from v3;
Insert into v3 values(3131,'ABC',10000,60);  // Error
```

Create or replace clause is used to change definition of the view.

```
Create or replace view v4
          As select empno, ename, sal, deptno, job from emp;
```

*<u>**With Check Option View**</u>:

These views will allow DML operation only when where condition is satisfied.

```
Create view Test_V12
       As select empno, ename, sal, deptno from emp
             Where deptno = 30
                  With check option;

Insert into Test_V12 values(666,'AAA',4000,30);    // valid
Insert into Test_V12 values(888,'AAA',4000,10);    // error (invalid)

Create view Test_V13
          As select empno, ename, sal, deptno from emp
               Where sal > 2000
                     With check option;
Select * from Test_V13;
Insert into Test_V13 values(6969,'AAA',3100,10);  // valid
```

Insert into Test_V13 values(6955,’AAA’1510,10);   // Invalid

Update Test_V13 set sal = 4000 where empno = 7566;   //valid
Update Test_V13 set sal = 1100 where empno = 7902;   // invalid

*__materialized view__:
Materialized views will help improving the performance of select statement on view.
To create materialized view, the based table should have primary key.
Changes to base table will not reflect on materialized view.
Materialized view or previously called as snap short.
__Ex__:

```
 Create view Test_V14
      As select empno, ename, sal, deptno from emp;

Create view Test_V15
      As select e.empno, e.ename, e.sal, e.deptno,d.dname,d.loc
            from emp e, dept d where e.deptno = d.deptno;
```

select * from Test_V14;   //performance fast
select * from Test_V15;   //performance fast two tables

__Syntax__:

```
Create MATERIALIZED view <VIEW_NAME> AS <select STMT>;
```

__Ex__:

```
Create materialized view MV1
    As select empno, ename, sal, deptno from emp;
Select * from MV1;
```

*__To Refresh materialized View__:
Sql> exec DBMS_SNAPSHOT.REFRESH(‘MV1’);
Select * from MV1;

DBMS_SNAPSHOT-PACKAGE NAME
REFRESH ---procedures

Select view_name from user_views;   //   All view tables are display

*__To Drop a view__:
__Syntax__:

```
Drop view <view_name>;
```

__Ex__:

```
Drop view Test_V14;
```

When base tables are drop, the view becomes invalid.
When base tables are re_created view will become valid.

***__INDEX__:
Index is an object which is used to improve the performance of select statements.

__Types of Indexes__: They are two types of Indexes.

1. Simple Index
2. Composite Index

1.__Simple Index__:
When index is created on one column it is called as simple index.
__Syntax__:

```
CREATE INDEX <INDEX_NAME> ON <TABLE_NAME> (COL_NAME);
```

__Ex__:

```
Create index IDX1 on emp(sal);
```

Index should be created on columns which we regularly use in the where clause.
When a index is created a separate structure is created with first column is ROWID and second column as indexed column.
The Rows in the index will be arranged in the ascending order of indexed column.

IDX1

ROWID	SAL
	800
	950
	1100
	1250
	1600
	5000

Using algorithm is identifies the back of ROWID's Using which rows are displayed.

*__Composite Index__: when Index is created multiple columns it is called composite index.

__Ex__: create index IDX2 on emp(sal,job);

The above index IDX2 is used only when both the columns are used in the where clause.

__Disadvantages of index__:

Index will consume memory.

The performance of DML command will be decreased.

__Index can also be categorized two types__:

1. Unique index
2. Non-unique index

*__Unique Index__:

If the indexed column contains unique value it is called unique index.

A unique index is automatically created. When we create a table with primary key constraint or unique constraint.

*__Non-unique index__:

If an index column contains duplicated values they are called as non-unique index.

__Ex__:

    Create index IDX1 on emp(sal);

__See to index tables__:

Select index_name, from user_indexes;

*Query to see list of all the indexes.

Select index_name, table_name from user_indexes;

*Query to see list of all the indexes along with column name.

Select index_name, table_name, column_name from user_ind_columns;

Desc user_indexes

Desc user_ind_columns

*__function based index__:

When index is created by using functions it is called as function based index.

__Ex__:

Create index indexs on emp (lower(ename));

The index is used only when use the appropriate function in the where clause function.

Select * from emp where lower(ename) = ‘king’;

<u>To drop on index</u>:

<u>Ex</u>:

Drop INDEX IDX1;

*<u>**Sequences**</u>: sequence is an object which is used to generate numbers.

<u>Syn</u>:

Create sequence <SEQUENCE_NAME> start with <value> increment by <value>;

<u>Ex</u>:

Create sequence SE1 start with 1 increment by 1;

<u>Ex</u>:

Create sequence SE4 start with 1000 increment by 1 maxvalue 5000 cycle;

*<u>**Using the Sequence**</u>: There are two pseudo to sequence.

1. NEXTVAL
2. CURRVAL

*<u>**NEXTVAL**</u>: Nextval is used to generate a number.

*<u>**CURRVAL**</u>: Currval is used to know the latest number generated.

Create sequence SE5 start with 101 increment by 1;

Insert into student7 values(SE5.NEXTVAL,’Arun’,60);

Insert into student7 values(SE5.NEXTVAL,’Amit’61);

Sequence is a sharable object.

When sequence is shared we get gaps in numbers.

<u>Example of CURRVAL</u>: To know the latest value generated.

<u>Ex</u>:

Select SE5.CURRVAL from dual;

Sequence with cache option will help in generating the numbers faster.

<u>Ex</u>:

Create sequence SE6 start with 1000 increment by 1 maxvalue 10000 cycle cache 40;

Cache option will help in improving the performance of sequence.

*<u>**Synonym**</u>: it is an alternate name given to an object.

<u>**Syntax**</u>: create synonym <Synonym_name> for <Table_name);

<u>**Ex**</u>:

```
Create synonym E1 for emp;
```

Synonym helps in reducing the length of the query.

Synonym is used instead of table names for all the commands.

<u>**Ex**</u>:

```
Select * from E1;
```

<u>**Query to see all synonyms**</u>:

```
SQL> select synonym_name from user_synonyms;
```

<u>**To drop a synonym**</u>:

```
SQL> DROP SYNONYM E1;
```

<u>**Object of Oracle**</u>:

1. Table
2. View
3. Index
4. Sequences
5. Synonyms

<u>**Synonym Objects of PL/SQL**</u>:

1. PROCEDURE
2. FUNCTION
3. PACKAGE
4. TRIGGER

<u>**Normalization**</u>: Normalization is process of removing redundancy and improving accurency of the database. Normalization can be achieved by using normal forms.

*<u>**1st Normal form (1NF)**</u>: A database is in 1NF if it satisfies following rules.

Rule1: Each cell should have one value.

Rule2: Table should have primary key.

<u>**Ex**</u>:

Author ID	Author Name	Book Name	Book Cost
<u>A101</u>	IVAN	SQL	200/-
<u>A101</u>	IVAN	PLSQL	250/-
<u>A102</u>	RAGHU	JAVA	150/
<u>A102</u>	RAGHU	J2EE	250

PRIMARY KEY

Author ID

Author Name

*<u>**2nd Second Normal form (2NF)**</u>: A database is in 2NF if it satisfies following rules.
Rule1: Database should be in 1NF.
Rule2: There should be no partial dependency.

<u>**Partial dependency**</u>: When a non-key attribute is dependent on part of a primary key. Then these exists partial dependency. The following table is satisfying 2NF rules.

PRIMARY KEY

-

Part ID	Supplier ID	S Name	Price	Address
<u>65</u>	2	TATA	59	Bangalore
<u>73</u>	2	TATA	60	Bangalore
<u>65</u>	1	BIRLA	54	Hyderabad

Partial dependency

In the above table S Name (non key attribute) is depending on supplier ID (part of primary key). Hence these exists partial dependency. We can eliminate partial dependency by dividing the table into two different tables. The following tables are satisfying 2NF rules.

Primary key

Part ID	Supplier ID	Price
65	2	59
73	2	60
65	1	54

Primary key

Supplier ID	S Name	Address
2	TATA	Bangalore
1	BIRLA	Hyderabad

*<u>**3rd Normal form (3NF)**</u>: A database is in 3NF if it satisfies the following rules.
Rule1: Database should be in 2nd NF.
Rule2: These should be no transitive dependency.

<u>**Transitive dependency**</u>: When a non key attribute is dependent on another non key attribute then these Exists transitive dependency the following table is not satisfying 3rd NF rules.

PART NO	MANFNAME	MANFADDRESS
<u>1000</u>	TOYOTA	PARK AVENUE
<u>1001</u>	MISTUBUSHI	LOS ANGELS
<u>1002</u>	TOYOTA	PARK AVENUE

Primary key

In the above table manufacture address (non key) is dependent of manufacture name (non key). Hence there exists transitive dependency.
We can eliminate transitive dependencies by dividing the table into two different tables.

Primary key

Part no	MenufName
1000	Toyota
1001	Mistubushi
1002	Toyota

ManfName	ManfAddress
Toyota	Park Avenue
Mistubushi	Los Angels

<u>Inter view Questions</u>:

1. What is view?
   A view is a logical representation of data from one or more then one table.

2. List of different between single views and complex views?

3. In which cases a view is called complex view?
   When a view is created using multiple base tables it is called complex view.

4. How can we restrict DML operations on simple views?
   We can perform DML operations on simple views. Any DML operation performance on simple view will be reflected on base table.

5. What is with check option view?
   These views will allow DML operation only when where condition is satisfied.

6. Do you think view contains data?
   View does not contain any data.
   View does not consume memory location.
   When we write select statement on view, we get the data from the table.

7. What is a data dictionary table used to see the list of view?

8. What is a materialized view?
   Materialized views will help improving the performance of select statement on view.
9. What is the advantage of index?
   Index is an object which is used to improve the performance of select statements.

10. What is the disadvantage of index?
    Index will consume memory.
    The performance of DML command will be decreased.

11. What is unique index?
    If the indexed column contains unique value it is called unique index.
    A unique index is automatically created. When we create a table with primary key constraint or unique constraint.

12. What is sequence?
    Sequence is an object is used to generate numbers.

13. What is different between simple index and composite index?

14. What are pseudo columns related to sequence?

15. When can we have gaps in sequence?

16. What is a synonym?
    It is an alternate name given to an object.

17. What is different between table alias and synonym?

CODD RULES: These rules are developed by Mr.'s E F CODD.
If a DBMS satisfies at least 6 rules out of 12 then it is called as RDBMS.

Rule 1: The information rule:
All information in the data type is to be represented in one and only one way, in the form of rows and columns.

Rule 2: The guarantied access rule:
If you can insert, you should able to select.

Rule 3: Systematic treatment of null values:
The DBMS must allow each field to remain null (or empty) specifically, It must support a representation of missing information ad inapplicable information.

Rule 4: Achieve online catalog based on the relational model:
Users must be able to access the data base's structure (catalog) using the same query language that they use to access the data base's data.

Rule 5: The comprehensive data sub language rule:
Language should be divided into several sub language basing on activities the command will perform.

Rule 6: The view updating rule:
All views that are theoretically update must be update table by the system.

Rule 7: High-level insert, update, and delete:
This rule states that insert, update and delete operations should be supported for any retrievable set rather then just for a single row in a single table.

Rule 8: Physical data independence:
Changes to the physical level (How the data is stored, whether in arrays or linked lists etc.) must not require a change on application based on the structure.

Rule 9: Logical data independence:
Changes to the logical level (tables, columns, rows, and so on) must not require a change to an application based on the structure.

Rule 10: Integrity independence:
This rule says that, it must be possible to change such constrains as and when appropriate without unnecessarily a selecting existing application.

Rule 11: Distribution independences:
The distribution of portions of the database to various locations should be invisible to users of the database.

Rule 12: The non subversion rule:
If the system provides a low-level (record-at-a-time) interface, then that interface cannot the used to subvert (degrade) the system.
*__Script__: A script is a file which is collection of commands same with extension .SQL
To run the script @D:\malli\FIRST.SQL

<u>**PL/SQL**</u>

It is extension of SQL the following or advantages of PL/SQL.

1. We can use programming features like if statement loops etc.
2. PL/SQL helps in reducing network traffic.
3. We can have user defined error massages by using concept of exception handling.
4. We can perform related actions by using concept of Triggers.
5. We can save the source code permanently for repeated execution.

<u>PL/SQL Block</u>:

A PL/SQL programs called as PL/SQL block.

<u>PL/SQL Block</u>:

```
DECLARE
------------------------
------------------------   --DECLARE SECTION    --OPTIONAL
------------------------
BEGIN
------------------------
------------------------  --EXECUTABLE SECTION   --MANDATORY
------------------------
EXCEPTION
------------------------
------------------------  --EXCEPTION SECTION   --OPTIONAL
------------------------
END;
/
```

***<u>Declare</u>**: This section is used to declare local variables, cursors, Exceptions and etc. This section is optional.

*<u>**Executable Section**</u>: This section contains lines of code which is used to complete table. It is mandatory.

*__Exception Section__: This section contains lines of code which will be executed only when exception is raised. This section is optional.

*__Simplest PL/SQL Block__:

```
Begin
--------
--------
--------
END;
/
```

*write a PL/SQL Block which will display Hello World?

```
SET SERVEROUTPUT ON

Begin
DBMS_OUTPUT.PUT_LINE( 'HELLO WORLD' );
END;
/
```

```
o/p: Hello world
PL/SQL procedure successfully completed.
```

__Note__: To get the output of the program server output environment variable should be on.

__Command__: SET SERVEROUTPUT ON

DBMS_OUTPUT is name of the PACKAGE

.PUT_LINE is name of the PROCEDURE

'/' Slash is used to submit the program in DBS.

*Write PL/SQL block which will calculate some of two numbers and display the output?

```
DECLARE
A number(2);
B number(2);
C number(3);
BEGIN
```

```
A := 10;
B := 20;
C := A + B;
DBMS_OUTPUT.PUT_LINE(C);
DBMS_OUTPUT.PUT_LINE( 'sum of two numbers' || C);
END;
/
```

```
o/p: 30
o/p: sum of two numbers 30
PL/SQL procedure successfully completed.
```

--is a Single line comment.

/*   */ ----Multi-Line comment

<u>Ex</u>:

```
    Initialization in declare section
    A number(2) := 50;
    B number(2) := 25;
```

*Write a PL/SQL block which accepts two numbers from the user and display the sum?

```
DECLARE
A number(2);
B number(2);
C number(3);
BEGIN
A := &A;    or A := &malli;
B := &B;    or B := &iNetSlov
C := A + B;
DBMS_OUTPUT.PUT_LINE( 'sum of the two numbers' || C);
END;
/
```

To modify the data

Begin

Merge-----------------

Insert------------------

Update----------------

```
Commit---------------
Select---------------- // not execute
END;
/
```

DCL--------NO
DDL--------NO
DML------YES
DRL--------NO
TCL--------YES

*Write a PL/SQL block which accepts employee number and increment is salary by 1000?

```
DECLARE
A number(4);
A := &Empno;
Update emp set sal = sal + 1000 where Empno = A;
END;
/
```

*Write a PL/SQL block which empno and delete that row from the emp table?

```
DECLARE
A number(4);
BEGIN
A := &Empno;
Delete from emp where Empno = A;
END;
/
```

***<u>Control statement</u>**: If – Then – Else

```
Declare
A number := 5;
Begin
DBMS_OUTPUT.PUT_LINE( ‘welcome’ );
If A > 10 Then
```

```
DBMS_OUTPUT.PUT_LINE( 'HELLO1' );
Else
DBMS_OUTPUT.PUT_LINE( 'HELLO2' );
END If;
DBMS_OUTPUT.PUT_LINE( 'THANK YOU' );
END;
/
```

*__LOOPS__: There are three types
1. Simple loop
2. While loop
3. for loop

1. **Simple**:
   ```
   Declare
   A number(2) := 1;
   Begin
   DBMS_OUTPUT.PUT_LINE( 'welcome' );
   LOOP
   DBMS_OUTPUT.PUT_LINE( 'HELLO1' );
   DBMS_OUTPUT.PUT_LINE( 'HELLO2' );
   Exit when A = 4;
   A := A + 1;
   END LOOP;
   DBMS_OUTPUT.PUT_LINE( 'THANK YOU' );
   END;
   /
   ```

2. **While**:
   ```
   Declare
   A number(2) :=1;
   Begin
   DBMS_OUTPUT.PUT_LINE( 'WELCOME' );
   While A <=4 loop
   DBMS_OUTPUT.PUT_LINE( 'HELLO1' );
   DBMS_OUTPUT.PUT_LINE( 'HELLO2' );
   A := A + 1;
   END LOOP;
   ```

```
DBMS_OUTPUT.PUT_LINE( 'THANK YOU' );
END;
/
```

3. **FOR LOOP**:

```
Declare
A number;
Begin
DBMS_OUTPUT.PUT_LINE( 'WELCOME' );
FOR A IN 1 .. 4 LOOP
DBMS_OUTPUT.PUT_LINE( 'HELLO1' );
DBMS_OUTPUT.PUT_LINE( 'HELLO2' );
END LOOP;
DBMS_OUTPUT.PUT_LINE( 'THANK YOU' );
END;
/
```

Note: in for loop the loop variable is implicitly declare.

*write a select statement in PL/SQL block?
Every select statement in a PL/SQL block should have into clause.

***write a program to display employee name for the empno 7698?

```
Declare
A varchar2(15);       // Ename varchar2(15);
Begin
Select ename into A from emp where empno = 7698;
DBMS_OUTPUT.PUT_LINE( A );
END;
/
```

*write a program accepts deptno and display deptname and location?

```
Declare
L_Deptno number(2);
L_DName varchar2(15);
L_Loc varchar2(10);
Begin
L_Deptno := &Deptno;
Select DName, Loc into L_DName, L_Loc from dept where Deptno = L_Deptno;
```

DBMS_OUTPUT.PUT_LINE('L-DName');
DBMS_OUTPUT.PUT_LINE('L_Loc');
END;
/

***Using % type attribute**:

Percentage type attribute is used to declare local variable with respect to column of a table.

Syntax: <VAR_NAME><TABLE_NAME>.<COL_NAME> % TYPE;

Ex:

```
    L_Dname Dept.Dname%TYPE;
```

*Write a program which accepts empno to display ename, job and salary?

```
Declare
L_empno emp.empno % type;
L_ename emp.ename % type;
L_job emp.job % type;
L_salary emp.salary % type;
Begin
L_empno := &empno;
Select ename, job, salary into L_ename, L_job, L_salary from emp where empno =
L_empno;
DBMS_OUTPUT.PUT_LINE( 'L_ename' );
DBMS_OUTPUT.PUT_LINE( 'L_job' );
DBMS_OUTPUT.PUT_LINE( 'L_salary' );

                    (or)

DBMS_OUTPUT.PUT_LINE( L_ename||'…' L_job||'…'||L_salary);
END;
/
```

Note:

When a select statement does not return any row program will terminate abnormally. When select statement returns more then one row, program will terminate abnormally.

***Percentage (%) Row type attributes**:

This attribute to declare a local variable which can store complete Row of a table.

Syn:
<VARIABLE_NAME> <TABLE_NAME> % ROW TYPE;
Ex:

```
A EMP%ROWTYPE;
```

We cannot directly display a Row type variable.
We can access one value in Row type variable by using.
<VARIABLE_NAME>.<COL_NAME>
Ex:

```
Declare
A EMP%ROWTYPE;
BEGIN
DBMS_OUTPUT.PUT_LINE( 'WELCOME' );
Select * INTO A from emp where empno = 7782;
DBMS_OUTPUT.PUT_LINE(A.sal||A.job||A.Hiredate);
DBMS_OUTPUT.PUT_LINE( 'THANK YOU' );
END;
  /
```

***EXCEPTIONS**:
Runtime Errors are called as Exceptions. They are three types of Exceptions.

1. ORACLE Predefined Exception
2. ORACLE Non Predefined Exception
3. USER Defined Exception

***Oracle Predefined Exception**:
These Exceptions will have Exception name and Exception number. Examples of predefined Exceptions are:

1. NO_DATA_FOUND
2. TOO_MANY_ROWS
3. ZERO_DIVIDE
4. VALUE_ERROR
5. DUP_VAL_ON_INDEX

***NO_DATA_FOUND**: This Exception is raised when select statement does not return any Row.
Ex:

```
Declare
L_sal emp.sal%type;
```

```
Begin
DBMS_OUTPUT.PUT_LINE( ‘WELCOME’ );
Select sal INTO L_sal from emp where empno = &empno;
DBMS_OUTPUT.PUT_LINE(L_sal);
DBMS_OUTPUT.PUT_LINE( ‘THANK YOU’ );
EXCEPTION
when NO_DATA_FOUND then
DBMS_OUTPUT.PUT_LINE( ‘INVALID EMPNO’);
END;
  /
```

***TOO_MANY_ROWS**:
This Exception is raised when select statement more then one row.
<u>**Ex**</u>:

```
Declare
L_sal emp.sal%type;
Begin
DBMS_OUTPUT.PUT_LINE( ‘WELCOME’ );
Select sal INTO L_sal from emp where deptno = 30;
DBMS_OUTPUT.PUT_LINE(L_sal);
DBMS_OUTPUT.PUT_LINE( ‘THANK YOU’ );
EXCEPTION
when TOO_MANY_ROWS then
DBMS_OUTPUT.PUT_LINE( ‘MORE THEN ONE ROW RETURNED’);
END;
  /
```

***ZERO_DIVIDE**:
<u>**Ex**</u>:

```
Declare
A Number;
Begin
A := 5/0;
Exception
when ZERO_DIVIDE then
DBMS_OUTPUT.PUT_LINE( ‘DO NOT DIVIDE BY 0’ );
END;
 /
```

Note:

This Exception is raised when we try to divided by zero.

***VALUE_ERROR**: This Exception is raised when there is miss match with the value and data type of local variable or size of local variables.

Ex 1:

```
Declare
L_sal emp.sal%type;
Begin
DBMS_OUTPUT.PUT_LINE( ‘WELCOME’ );
Select ename INTO L_sal from emp where empno = 7521;
DBMS_OUTPUT.PUT_LINE(L_sal);
DBMS_OUTPUT.PUT_LINE( ‘THANK YOU’ );
EXCEPTION
when VALUE_ERROR then
DBMS_OUTPUT.PUT_LINE( ‘please check the local variables’);
END;
  /
```

Ex 2:

```
            Declare
A number(3);
Begin
A := 1234;
Exception
when VALUE_ERROR then
DBMS_OUTPUT.PUT_LINE( ‘PLEASE CHECK THE LOCAL VARIABLES’ );
END;
  /
```

***DUP_VAL_ON_INDEX**: (duplicate value on index)

This Exception is raised when we try to insert a duplicate value in primary key constraint.

Ex:

```
Begin
DBMS_OUTPUT.PUT_LINE( ‘welcome’ );
Insert into student values(104, ‘ARUN’,60);
DBMS_OUTPUT.PUT_LINE( ‘Thank you’ );
```

```
Exception
when DUP_VAL_ON_INDEX then
DBMS_OUTPUT.PUT_LINE(' Do not insert duplicates' );
END;
  /
```

The above program works on an assumption the table student for if having a primary key SNO column with value 104.

Exception Name	**Exception No**	**Exception Message**
1.NO_DATA_FOUND	ORA-1403	NO DATA FOUND
2.TOO_MANY_ROWS	ORA-1422	EXACT FETCHED REQUESTED MOTRE THEN ONE ROW
3.ZERO_DIVIDE	ORA-1476	DIVISON IS EQUAL TO ZERO
4.VALUE_ERROR	ORA-6502	NUMRIC OR VALUE ERROR
5.DUP_VAL_ON_INDEX	ORA-0001	UNIQUE CONSTRAINT VIOLATED

*<u>**WHEN OTHERS**</u>:
When others are a universal Exception angular this can catch all the Exceptions.

```
Declare
L_sal number(4);
A number;
Begin
DBMS_OUTPUT.PUT_LINE( 'Welcome' );
Select sal INTO L_SAL from emp where deptno = &deptno;
DBMS_OUTPUT.PUT_LINE('The sal is ….'||L_sal);
A :=10/0;
DBMS_OUTPUT.PUT_LINE( 'Thank you' );
Exception
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE( 'please check the code' );
END;
  /
```

*<u>**ERROR REPORTING FUNCTIONS**</u>: They are two Error Reporting functions.
1. SQLCODE
2. SQLERRM

These error reporting functions are used in when others clause to identified the exception which is raised.

1. **SQLCODE**: It returns ERRORCODE
2. **SQLERRM**: It returns Exception number and Exception message.

Note: for NO_DATA_FOUND Exception SQLCODE will return 100.

```
Declare
L_sal number(4);
A number;
Begin
DBMS_OUTPUT.PUT_LINE( 'Welcome' );
Select sal INTO L_SAL from emp where deptno = &deptno;
DBMS_OUTPUT.PUT_LINE('The sal is ….'||L_sal);
A :=15/0;
DBMS_OUTPUT.PUT_LINE( 'Thank you' );
Exception
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE( 'please check the code' );
DBMS_OUTPUT.PUT_LINE(SQLCODE);
DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
  /
```

*__NESTED BLOCK__:

```
Declare
A number := 10;
Begin
DBMS_OUTPUT.PUT_LINE('HELLO1');
Declare
B number := 20;
Begin
DBMS_OUTPUT.PUT_LINE('HELLO2');
DBMS_OUTPUT.PUT_LINE(B);
DBMS_OUTPUT.PUT_LINE(A);
END;
DBMS_OUTPUT.PUT_LINE('HELLO3');
DBMS_OUTPUT.PUT_LINE(B); --ERROR
END;
 /
```

Note: outer block variables can be accessed in nested block nested block variables can not be accessed in outer block.

*__EXCEPTION PROPAGATION__:

```
Begin
DBMS_OUTPUT.PUT_LINE('HELLO1');
L_SAL EMP.SAL%TYPE;
Begin
DBMS_OUTPUT.PUT_LINE('HELLO2');
Select sal INTO L_SAL from emp where empno = 1111;
DBMS_OUTPUT.PUT_LINE('HELLO3');
END;
DBMS_OUTPUT.PUT_LINE('HELLO4');
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('HELLO5');
END;
  /
```

*__ORALE NON PREDEFINED EXCEPTIONS__:

These Exceptions will have only Exception number. But does not have Exception name.

Steps to handle non predefined exceptions.

Syntax:

Step1: Declare the Exception

```
            <EXCEPTION_NAME> EXCEPTION;
```

Step2: Associate the Exception

```
            PRAGMA EXCEPTION_INIT(<EXCEPTION_NAME>,<EXCEPTION NO>);
```

Step3: Catch the Exception

```
            WHEN <EXCEPTION_NAME> THEN
            ------------------
            ------------------
            ------------------
            END;
```

```
/
```

ORA -2292 is an example of non predefined exception.
This exception is raised when we delete row from a parent table. If the corresponding value existing the child table.

```
Declare
MY_EX1 Exception;    --step1
PRAGMA EXCEPTION_INIT(MY_EX1, -2292);    --step2
Begin
DBMS_OUTPUT.PUT_LINE(‘Welcome’);
Select from student where eno = 102;
EXCEPTION
WHEN MY_EX1 THEN    --step3
DBMS_OUTPUT.PUT_LINE(‘do not delete pargma table’);
END;
 /
```

Pragma Exception_init is a compiler directive which is used to associated an Exception name to the predefined number.

*<u>**USER DEFINED EXCEPTION**</u>:
These Exceptions are defined and controlled by the user. These Exceptions neither have predefined name nor have predefined number. Steps to handle user defined Exceptions.
Step1: Declare the Exception
Step2: Raised the Exception
Step3: Catch the Exception

```
Declare
MY_EX1 EXCEPTION;     --Step1
L_SAL EMP.SAL%TYPE;
Begin
DBMS_OUTPUT.PUT_LINE(‘welcome’);
Select SAL INTO L_SAL from emp where empno = &empno;
IF L_SAL > 2000 THEN
RAISE MY_EX1;       --Step2
ENDIF;
DBMS_OUTPUT.PUT_LINE(‘The sal is … ‘||L_sal);
DBMS_OUTPUT.PUT_LINE('Thank you');
EXCEPTION
WHEN MY_EX1 THEN      --Step3
DBMS_OUTPUT.PUT_LINE('Sal is two high');
END;
 /
```

Note: When others should be the last handler of the exception section other wise we get a compiler ERROR.

*__RAISE_APPLICATION_ERROR:__ RAISE_APPLICATION_ERROR is a procedure which is used to throw one error number and error message to the calling environment.
It internally performance rolls back.
ERROR number should be range of -20000 to -20999. ERROR message should be displayed less then or equal to 512 characters.

```
Declare
L_sal emp.sal%TYPE;
Begin
DBMS_OUTPUT.PUT_LINE('Welcome');
Insert INTO dept values (08,'arun',70);
Select sal INTO L_sal from emp where empno = 7698;
IF L_sal > 2000 THEN
RAISE_APPLICATION_ERROR(-20150, 'SAL IS TOO HIGH');
END IF;
DBMS_OUTPUT.PUT_LINE('THE SAL IS…'||L_SAL);
END;
  /
```

*__CURSORS__: CURSOR is a memory location which is used to run SQL commands. They are two types of cursors.

1. Implicit cursor
2. Explicit cursor

*__Implicit cursor__: All the activities related to cursors like opening the cursor, processing the cursor and closing the cursor are done automatically. Hence these cursor are called as implicit cursor.

*__Implicit cursor attributes__: They are 4 implicit cursor attributes.

1. SQL%ISOPEN
2. SQL%FOUND
3. SQL%NOTFOUND
4. SQL%ROWCOUNT

*__SQL%ISOPEN__: It is a Boolean attribute. It always returns false.
*__SQL%FOUND__: it is a Boolean. It returns true. If SQL command is successful returns false if SQL command is fails.
*__SQL%NOTFOUND__: It is a Boolean attribute returns true. If SQL command is fails returns false if SQL command is successful.
__Note__:
SQL%NOTFOUND attribute is exactly opposite to SQL%FOUND.
*__SQL%ROWCOUNT__: It returns the number of rows affected by SQL command.

*__using SQL%FOUND__: It is used to know whether a specific command is effecting the table data or not.
__Ex__:

```
Begin
Update emp set sal = 2000 where empno = 1111;
IF SQL%FOUND THEN
DBMS_OUTPUT.PUT_LINE('UPDATE SUCCESSFUL');
ELSE
DBMS_OUTPUT.PUT_LINE('UPDATE FAILED');
END IF;
END;
  /
```

*__USING SQL%ROWCOUNT__:

```
Begin
Update emp set sal = 2000 where deptno =30;
DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT||'ROWS UPDATED');
END;
  /
```

*__SQL%ISOPEN__

```
Begin
Update emp set sal = 1000 where empno = 7654;
IF SQL%ISOPEN THEN
DBMS_OUTPUT.PUT_LINE('CURSOR IS OPEN');
ELSE
DBMS_OUTPUT.PUT_LINE('CURSOR IS CLOSED');
END IF;
END;
  /
```

Note: By the time we check whether cursor is open or not it is already closed by the oracle engine. Hence it is always returns false.

*write a PL/SQL block which accepts deptno and display all the employee names of the department?

```
Declare
L_ename emp.ename%TYPE;
Begin
Select ename into L_ename from emp where deptno = &deptno;
DBMS_OUTPUT.PUT_LINE(L_ename);        --ERROR
END;
  /
```

****EXPLICIT CURSORS**: All the activities related to cursor like.

1. Opening the cursor
2. Processing the cursor
3. Closing the cursor

Act should be done by the developer. Hence this cursor is called explicit cursors.
We should use explicit cursors to run a select statement. Which returns more then one row?

Steps to use explicit cursors:
Step 1: declare the cursor
Step 2: open the cursor
Step 3: fetch data from cursor to local variables
Step 4: close the cursor

Syntax:

Step 1: declare the cursor

```
CURSOR <CURSOR_NAME> IS <SELECT STMT>;
```

Step 2: open the cursor

```
OPEN <CURSOR_NAME>;
```

Step 3: fetch data from cursor to local variables

```
FETCH <CURSOR_NAME> INTO <VAR1>,<VAR2>,….,….,….<VARn>;
```

Step 4: close the cursor

```
CLOSE <CURSOR_NAME>;
```

*__EXPLICIT CURSOR ATTRIBUTES__: These are four explicit cursor attributes.

1. %ISOPEN
2. %FOUND
3. %NORFOUND
4. %ROWCOUNT

*__%ISOPEN__: It is a Boolean attribute which returns true when cursor is open. Returns false when cursor is closed.

*__%FOUND__: It is a Boolean attribute returns true when fetch statement is successful.

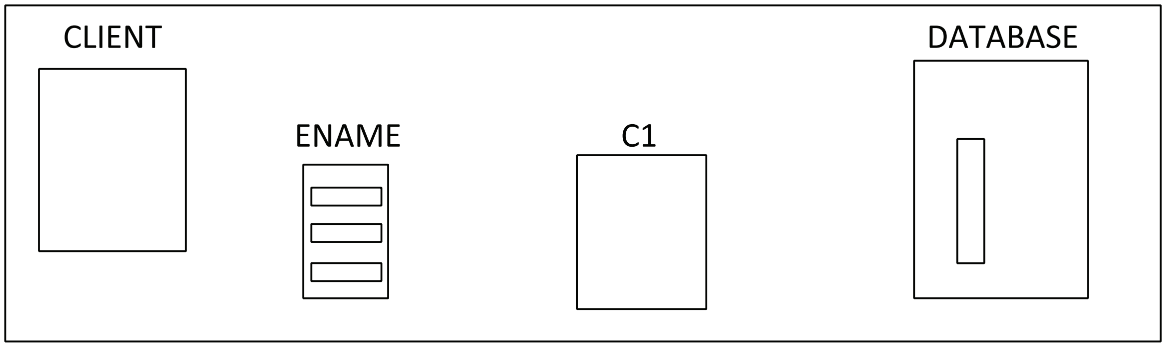
*__%NOTFOUND__: It is a Boolean attribute returns true. When fetch statement fails. Returns false when fetch statement successful.
Note: %NOTFOUND attributes is exactly opposite to % is found attribute.

*__ROWCOUNT__: Returns number of rows processed by fetches statement.

*write a program to display all the employee names working in dept number 10?

```
Declare
Cursor c1
IS select ename from emp where deptno = 10;
L_ename emp.ename%TYPE;
Begin
open c1
loop
FETCH c1 into L_ename;
EXIT WHEN c1%NOTFOUND;
```

```
DBMS_OUTPUT.PUT_LINE(L_ename);
END LOOP;
CLOSE c1;
END;
  /
```



ACTIVE DATA SET:

1. When we open a cursor the memory location is created.
2. Memory location will be loaded by data returned by select statement.
3. Cursor pointer points to 1st row of the memory location. This status is called active data set.

*write a PL/SQL block to display all the department names and locations from the department table?

```
Declare
Cursor c1
IS select dname, loc from dept;
L_dname dept.dname%TYPE;
L_loc dapt.loc%TYPE;
Begin
Open c1;
Loop
Fetch c1 into L_dname, L_loc;
EXIT WHEN c1%NOTFOUND;
DBMS_OUTPUT.PUT_LINE(L_dname||L_loc);
END loop;
Close c1;
END;
 /
```

*write a program to display employee names, job, salary who are working in a department number 30?

```
Declare
CURSOR c1
IS select ename,job,salary from emp where deptno = 30;
A   c1%ROWTYPE;
Begin
OPEN c1;
LOOP
FETCH c1 INTO A;
EXIT WHEN c1 %NOTFOUND
DBMS_OUTPUT.PUT_LINE(A.ename||'…'||A.job||'…'||A.salary);
END LOOP;
END;
  /
```

***CURSOR FOR LOOPS**:

It is a short cut way of writing an explicit cursor program. When we use cursor for loop following steps are not required.

1. Opening the cursor not required.
2. Closing the cursor not required.
3. Fetch statement not required.
4. Exit when condition not required.
5. Declaring local variable not required.

```
Declare
CURSOR c1
IS select ename, job, salary from emp where deptno = 30;
Begin
FOR A IN c1 LOOP
DBMS_OUTPUT.PUT_LINE(A.ename||'…'||A.job||'…'||A.salary);
END LOOP;
END;
 /
```

*Write a program to display the entire employee numbers their names and location?

```
Declare
```

```
CURSOR c1
IS select E.empno, E.ename, D.loc from emp E, Dept D where E.Deptno = D.Deptno;
Begin
FOR A IN c1 LOOP
DBMS_OUTPUT.PUT_LINE(A.empno||'…'||A.ename||'…'||A.loc);
END LOOP;
END;
  /
```

SNO	**SNAME**	**SUB1**	**SUB2**	**SUB3**
101	ARUN	60	70	80
102	VARUN	65	77	80
103	SREENU	60	91	80
104	AMIT	60	70	88
105	VEERA	40	50	60

*Write a program it will calculate AVG marks from the entire student table?

```
Declare
CURSOR c1
IS select * from student;
L_AVG number(5.2);
Begin
FOR A IN c1 LOOP
L_AVG := (A.sub1 + A.sub2 + A.sub3)/3;
DBMS_OUTPUT.PUT_LINE('AVERAGE OF' ||A.sno||'IS'||L_AVG);
END LOOP;
END;
 /
```

*<u>**Using%ISOPEN attribute**</u>:

```
Declare
Cursor c1
IS select empno, ename from emp where deptno = 20;
L_empno emp.empno%TYPE;
L_ename emp.ename%TYPE;
Begin
open c1;
IF c1%ISOPEN then
```

```
DBMS_OUTPUT.PUT_LINE(‘cursor is open’);
Else
DBMS_OUTPUT.PUT_LINE(‘cursor is closed’);
END IF;
END;
 /
```

*__Using%ROWCOUNT attribute__: This attribute is used to fetch limited number of rows from the local variables.

__Ex__:

```
Declare
Cursor c1
IS select empno, ename from emp where deptno = 20;
L_empno emp.empno%TYPE;
L_ename emp.ename%TYPE;
Begin
open c1;
Loop
FETCH c1 INTO L_empno, L_ename;
Exit when c1%ROWCOUNT = 3;
DBMS_OUTPUT.PUT_LINE(L_empno||’…..’||L_ename);
END Loop;
close(1);
END;
 /
```

*__Parameterized cursor__: A cursor which accepts a parameter from the user is called as parameterized cursor. Active dataset changes dynamically basing on the value passed to the cursor.

__Ex__:

```
Declare
Cursor c1(A number)
IS select empno, ename from emp where deptno = A;
L_empno emp.empno%TYPE;
L_ename emp.ename%TYPE;
Begin
open c1(&deptno);
Loop
```

```
FETCH c1 INTO L_empno,L_ename;
Exit when c1%NOTFOUND;
DBMS_OUTPUT.PUT_LINE(L_empno||'…..'||L_ename);
END Loop;
close c1;
END;
 /
```

***Procedures**: A procedure is a PL/SQL block which is compiled and permanently stored in the database for repeated execution.

Syntax:

```
create or replace procedure<procedure_Name>
IS
Begin
--------------
--------------
--------------
END;
 /
```

Ex:

```
create or replace procedure p1
IS
Begin
DBMS_OUTPUT.PUT_LINE('HELLO');
END;
 /
```

***To execute the procedure**:

SQL>EXEC P1

Procedure can have 3 types of parameters

1. Inparameter
2. OutParameter
3. In Out parameter

1. **InParameter**:
   InParameter's are used to accept values from the user.

Ex: create a procedure which accepts two numbers and display the num?

```
*Create or replace procedure ADD_NUM(A IN number, B IN number)
IS
C number;
Begin
C := A + B;
DBMS_OUTPUT.PUT_LINE('The sum is…'||C);
END;
 /
```

*create a procedure which accepts employee number and increment salary by 1000?

```
Create or replace procedure INC_SAL (A IN number)
IS
Begin
Update emp set sal = sal + 1000 where empno = A;
END;
 /
```

*create a procedure which accepts employee number and display salary?

```
Create or replace procedure display_sal (L_empno IN emp.empno%TYPE)
IS
L_sal emp.sal%TYPE;
Begin
Select sal INTO L_sal from emp where empno = L_empno;
DBMS_OUTPUT.PUT_LINE('The sal is ….'||L_sal);
END;
 /
```

*create a procedure which accepts deptno and display the name and location?

```
Create or replace procedure display_details(L_deptno IN dept.deptno%TYPE)
IS
L_Dname dept.Dname%TYPE;
L_loc dept.loc%TYPE;
Begin
Select Dname, loc INTO L_Dname, L_loc from dept where deptno = L_Deptno;
DBMS_OUTPUT.PUT_LINE('L_Dname||'….'||L_loc);
Exception
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('IN value NO');
```

```
END;
 /
```

Query to see all the list of procedures:

Select object_NAME from user_objects where object_TYPE = 'PROCEDURE';

To see the source code of a procedure:

Select text from user_source where NAME = 'ADD_NUM';

Spool command:

This command is used to extract the character which we can see in SQL*PLUSE environment to a text file.

Ex:

```
SQL>spool c:\sample\abc.txt
SQL>select * from emp;
SQL>select * from dept;
SQL>spool off
```

To edit the procedure:

```
SQL>spool c:\sample\xyz.txt
SQL>select text from user_SOURCE where name = 'p1';
SQL>spool off
```

Create a procedure which accepts a number and display its square.

Note: if there are any errors in the source code, then procedure will be created with compilation errors.

Show errors command is used to see the compilation errors.

We can execute multiple procedures act once by using a PL/SQL block.

Ex:

```
        Begin
        P1;
        ADD_NUM(20,15);
        DISPLAY_SQUARE(1);
        END;
         /
```

We can remove the procedure by using drop command.

Ex: drop procedure p1;

We can call a procedure using another procedure there exist depending between the procedures.

Ex:

```
Create or replace procedure test11
IS
Begin
DBMS_OUTPUT.PUT_LINE('This is from test11');
END;
 /
Create or replace procedure sample11
IS
Begin
DBMS_OUTPUT.PUT_LINE('This is from sample11');
END;
 /
```

Note:

When we drop test11 procedure, sample11 procedure will become invalid.

<ins>Out parameters</ins>: out parameters are used to return the value to the user.

Ex:

Create a procedure which accepts two numbers and return the sum?

```
Create or replace procedure ret_sum
                                (A IN number,B IN number, C OUT number)
IS
Begin
C := A + B;
END;
 /
```

***<ins>Steps to invoke procedures which are having out parameters</ins>**:

Step 1: create bind variable

```
SQL> variable N number
```

Step 2: execute the procedure

```
SQL> EXEC Ret_sum(10,20,:N)
```

Step 3: print bind variable

```
SQL>print N
```

Ex:

Create a procedure which accepts a number and return its square?

```
            Create or replace procedure ret_square(A IN number,B IN number)
            IS
            Begin
            B := A * A;
            END;
             /
SQL> variable N number
SQL> EXEC ret_square(7,:N)
SQL> print N
```

IN out parameters: These parameters are used to accept the value as well as to return the value to user.

Ex: create a procedure which accepts a number and return its square?

```
            Create of replace procedure ret_square1(A IN out number)
            IS
            Begin
            A := A * A;
            END;
             /
```

Step 1: SQL> variable m number

Step 2: initialize bind variable

```
            Begin
            :m = 5;
            END;
             /
```

Step 3: EXEC ret_square1(:m);

Step 4:print m;

Note:

The scope of the bind variable is with respect to one session.

Once the environment is closed all the bind variables are lost.

*__Functions__:

A function is a named PL/SQL block which must and should return a value.

Syntax: create or replace function<FUNCTION_NAME>(VAR1 datatype, var2 datatype,…….,varn datatype);

Return datatype

IS

Begin

```
-------
-------
-------
END;
 /
```

Ex: Create a function which accepts two numbers and returns the sum?

```
Create or replace function ADD_NUM_FUN(A number, B number)
Return number
IS
C number;
Begin
C := A + B;
Return C;
END;
 /
```

Output: function created

Calling:

```
Declare
N number;
Begin
N := ADD_NUM_FUN(10,20);
DBMS_OUTPUT.PUT_LINE(N);
END;
 /
```

We can invoke the function using ‘select’ statement.
Ex: select ADD_NUM_FUN(50,10) from dual;

Functions can be invoked from expression.
Ex: select 100 + ADD_NUM_FUN(50,10) from dual;
We can not have DML commands inside a function.
If a function has DML commands, we get the error when we invoke it.
Functions are preferred to perform calculations.

*create a function which accepts salary and returns tax value. Tax value is 10% of salary?

```
Create or replace function CAL_TAX(L_sal emp.sal%TYPE)
Return number
IS
Begin
L_TAX = L_sal * 10/100;
Return L_TAX;
END;
 /
```

Ex: select empno, ename, sal, cal_tax(sal) from emp;

Differences between procedure and functions:

Procedures	**Functions**
1. Procedures need not return any value can return one or more than one value.	1. Must and should return only one value.
2. DML commands are allowed.	2. DML commands are not allowed.
3. Can not be invoked from select statement.	3. Can be invoked from select statement and expressions.

*__Query to see list of all the functions__:

Select object_Name from user_objects where object_type = 'function';

Procedures and functions are called sub programs.

*__Packages__:

A package is collection of logically related sub programs.

Creation of package invokes two steps.

Step 1: creating package specification.

Step 2: creating package body.

Package specification: package specification contains declaration of sub programs.

Package body: package body contains definition of sub programs.

Example of PKS:

```
create or replace package test_PKG1
IS
Procedure p1;
Procedure display_sal(L_empno IN number);
Procedure display_details(L_deptno IN emp.deptno%TYPE);
```

```
Function cal_tax(L_sal emp.sal%TYPE)
Return number;
END test_PKG1;
 /
```

Example of PKB:

```
Create or replace package body test_PKG1
IS
Procedure p1
IS
Begin
DBMS_OUTPUT.PUT_LINE('HELLO WORLD');
END;
Procedure display_sal(L_empno IN number)
IS
L_sal emp.sal%TYPE;
Begin
Select sal into L_sal from emp where empno = L_empno;
DBMS_OUTPUT.PUT_LINE('The sal is…'||L_sal);
END;
Procedure display_details(L_deptno IN emp.deptno%TYPE)
IS
Cursor c1
IS select ename from emp where deptno = L_deptno;
Begin
For emp_rec IN c1 LOOP
DBMS_OUTPUT.PUT_LINE(emp_rec.ename);
END LOOP;
END;
Function cal_tax(L_sal emp.sal%TYPE)
Return number
IS
Begin
Return(L_sal*10/100);
END;
END test_PKG1;
 /
```

To invoke a sub program which is inside the package, we need to mention package name subprogram name.

Ex: Exec test_PKG1.p1

Select test_PKG1.cal_tax(2500) from dual;

Note:

Packages can not be parameterized.

Packages can not be nested.

Packages help in modularization.

Packages help in overloading subprograms.

Example of overloading sub programs:

```
Create or replace package test_pkg2
IS
Procedure p1(A number);
Procedure p1(A number, B number);
END test_pkg2;
 /
```

PKB:

```
Create or replace package body test_pkg2
IS
Procedure p1(A number)
IS
B number;
Begin
B := A * A;
DBMS_OUTPUT.PUT_LINE(‘The square is … ‘||B);
END;
Procedure p1(A number, B number)
IS
C number;
Begin
C := A + B;
DBMS_OUTPUT.PUT_LINE(‘The sum is … ‘||C);
END;
END test_pkg2;
 /
```

<u>Invoke the package</u>

```
Begin
Test_pkg2.p1(10);
Test_pkg2.p1(10,20);
END;
 /
```

Dropping the package involves two steps:
Step 1: drop package body
Step 2: drop package specification

<u>Ex</u>: drop package body test_pkg2;
            Package body dropped

<u>Ex</u>: drop package test_pkg2;
            Package dropped.

***<u>**Triggers**</u>:

A Trigger is a PL/SQL block which is executed automatically basing on an event.
Triggering events are inserted, update, delete.
Trigger timing can be before, after, instead of

<u>Syntax</u>:

```
Create or replace trigger <TRIGGER_NAME><TIMMING><EVENT> ON <OBJECT_NAME>
Begin
-----------
-----------
-----------
END;
 /
```

<u>Ex</u>:

```
create or replace trigger trg1
after insert on dept
begin
DBMS_OUTPUT.PUT_LINE('hello');
END;
 /
Trigger created
SQL> insert into dept values(65,'Admin','HYD');
```

Trigger can be created on multiple events

Ex:

```
Create or replace trigger trg1
After insert or update or delete on dept
Begin
If inserting then
DBMS_OUTPUT.PUT_LINE('thank you for inserting');
ElsIf updating then
DBMS_OUTPUT.PUT_LINE('thank you for updating');
Else
DBMS_OUTPUT.PUT_LINE('thank you for deleting');
End If;
END;
 /
```

```
SQL> insert into dept values(65,'arun','hyd');
Thank you for inserting
SQL> deleting from dept where deptno = 65;
Thank you for deleting
```

*Triggers are divided into two types
1. Statement level trigger
2. Row level trigger

***Statement level trigger**:
These triggers are executed only once irrespective of number of rows affected by the event. By default every trigger is a statement level.

***Row level trigger**:
These triggers are executed for every row which is affected by the event (multiple numbers of times). We can create a row level trigger by using "FOR EACH ROW" clause.
Ex:

```
Create or replace trigger trg2
After update on emp for each row
Begin
DBMS_OUTPUT.PUT_LINE('thank you for updating');
```

```
END;
  /
Trigger updated
SQL> update emp set sal = 2000 where deptno = 30;
```

Output:

```
Thank you for updating
Thank you for updating
Thank you for updating
Thank you for updating
Thank you for updating
Thank you for updating
6 rows updated
```

Triggers are used to enforce business rules.
We can enforce business rules by using :OLD and :NEW qualifiers.

*__Query to see list of all the triggers__:

Select object_Name from user_objects where object_TYPE = 'Trigger';

OR

Select Trigger_Name from user_Trigger;

*__To see the source code of a trigger__:

Select text from user_source where name = 'trg2';

*__To drop a trigger__:

Syntax: drop trigger <trigger_name>.

Ex: drop trigger trg2;

*__Using: new qualifier__:

*Create a trigger which will allow insert command only when salary is less than 5000. Trigger should reject the insert command by providing a meaningful message. If SAL > 5000?

```
Create or replace trigger TRG5 before insert on emp for each row
Begin
If :new.sal>5000 then
Raise_application_error(-20150,'sal cannot be more than 5000');
END IF;
```

END;
 /

<u>Ex</u>: insert into emp(empno, ename, sal, deptno) values(444,’BBB’,7500,10);
<u>Output</u>: ERROR ORA-20150 :sal cannot be more than 5000

Insert into emp(empno, ename, sal, deptno) values(111,’AAA’,2500,10);
<u>Output</u>: 1 row created

*create a trigger which will accept insert command only for deptno 10?
Create or replace trigger trg3 before insert on emp for each row
Begin
IF :new.deptno<>10 then
Raise_application_error(-20150,’deptno should be only 10’);
END IF;
END;
 /

***<u>Using :old qualifier</u>**:
*A trigger should restrict delete operation on president?
Create or replace trigger trg7 before delete on emp for each row
Begin
IF :old.JOB = ‘president’ then
Raise_Application_Error(-20150,’cannot delete president’);
END IF;
END;
  /

Delete from emp where empno = 7839;
<u>Error</u>: ORA -20152, cannot delete president
Delete from emp where empno = 7499;   --- valid

*Create a trigger which will restrict don’t operations on weekends?
Create or replace trigger trg8 before insert or update or delete on emp
Begin
IF TO_CHAR(sysdate,’DY’) IN (‘sal’,’sun’) then
Raise_Application_Error(-20160,’cannot perform dml operations on weekends’);
END IF;

```
END;
 /
```

*<u>**Instead of triggers**</u>:

Instead of triggers are created on complex view.

By using instead of trigger we can execute insert command on a complex view

<u>**Ex**</u>:

<u>**COMPLEX VIEW**</u>

```
Create view vv1
As select e.empno, e.ename, e.sal, e.deptno, d.deptno, d.loc from emp e1 dept d
where e.deptno = d.deptno;

View created
Select * from vv1;
```

*<u>**Instead of trigger example**</u>:

```
Create or replace trigger trg9 instead of insert on vv1 for each row
Begin
Insert into dept values (:new.deptno, :new.dname, :new.loc);
Insert into emp(empno, ename, sal, deptno) values
(:new.empno, :new.ename, :new.sal, :new.deptno);
END;
 /
```

<u>**Ex**</u>: insert into vv1 values (555,’ddd’,2000,60,’ADMIN’,’HYD’);   --- valid

<u>**Note**</u>:

:new and :old qualifiers can be used only in the row level trigger.

*<u>**Abstract datatypes**</u>:

Abstract data types are consists of one or more subtypes. Rather than being constrained to the standard oracle data types of number, date and varchar2 data types can more accurately describe your data.

<u>**Ex**</u>:

```
SQL>create type address_ty5 as object
                        (street varchar(20),
                          city varchar2(10),
                             state char(10),
                              pin number);
```

```
                                   /
Type created

SQL> create type person_ty5 as object
                       (name varchar2(20),
                        Address address_ty5);
                               /
Type created

SQL> create table customer5
                       (customer_ID number(3),
                        Person person_ty5);

SQL> insert into customer5 values
        (1,person_ty5('hari',address_ty5('#102 lokhanadala','mumbai','MH',10101)));

SQL> select customer_ID, c.person.name from customer c;
```

*__Nested table__:
Nested table is a collection of rows, represented as a column with in the main table. For each record with in the main table, the nested table may contain multiple rows. In one sense, it's a way of storing a one-to-many relationship with in one table.

```
SQL> create or replace type emp_ty5 as object
                       (desg varchar2(20),
                        dname varchar2(20),
                        doj date);
                          /
Type created

SQL> created type emp_nt5 as table of emp_ty5;
           /
Table created

SQL> created table emp data5
                       (ename varchar2(20),
                        details emp_nt5)
                        nested table details store as emp_nt_tab5);
```

Table created

```
SQL> set describe depth2
SQL> desc empdata5
```

Name	**Type**
Ename	Varchar2(20);
Details	Emp_nt
Desg	Varchar2(20)
Dname	Varchar2(20)
Doj	Date

```
SQL> insert into empdata5 values
          ('Raju', emp_nt5('clerk','sales','12-sep-05'),
                emp_ty5('Asst','mrket','15-oct-04'),
                emp_ty5('mngr','sales','13-aug-09')));

SQL> select * from emp data5;
```

*__VARRAYS__: varrays can also be used to create one-to-many relationship with in the table.

*__creating varray__:
__Ex__: create type dependent_brithdate_t5 as varray(10) of date;

__Using varray in table__:
```
Create table employee5( id number, name varchar2(20),
                        dependent ages dependent-brithdate-t5);
```

__inserting row into table__:
```
insert into employee5 values(42,'Arun', dependent_brithdate_t5
                              ('12-jan-1765','04-jul-1977','11-mar-2021');
```

*****Differences between nested tables and varrays*****

Nested tables **Varrays**

1. There is no restriction on size.	1. We need to define the maximum size.
2. Data is stored in special auxiliary tables called as store tables.	2. Data is stored inline to the rest of the table data.

*<u>**Execute immediate**</u>:

One can call DDL statement like create, drop, truncate and etc from PL/SQL by using the “Execute immediate” statement.

<u>**Ex**</u>:

```
Begin
Execute immediate ‘drop table dept’;
END;
 /

Begin
Execute immediate ‘Truncate table emp;
END;
 /
```

*<u>**BULK COLLECT**</u>:

Bulk collect feature helps in improving the performance of explicit cursor programs. Fetch statement can fetch all the rows from the cursor to the programs local variable at once thus helps in improving the performance.

<u>**Ex**</u>:

```
Declare
Type string_array is varray(20) of varchar2(20);
L_ename string_array;
Cursor c1
Is select ename from emp;
Begin
Open c1;
Fetch c1 bulk collect into L_ename;
Close c1;
For i in L_ename.first .. L_ename.last loop
DBMS_OUTPUT.PUT_LINE(L_ename(i));
END loop;
END;
 /
```

*__REF CURSOR__:

A ref cursor is basically a data type.

A variable created based on such a data type is generally called a cursor variable.

A ref cursor can be associated with more than one select statement at run time. Before associating a new select statement. we need to close the cursor.

__Ex__:

```
Declare
Type r_cursor is REF cursor;
C_emp r_cursor;
Type rec_emp is record{
name varchar2(20);
sal number(6);
      };
er rec_emp;
begin
open c_emp for select ename, sal from emp where deptno = 10;
DBMS_OUTPUT.PUT_LINE(‘department: 10’);
DBMS_OUTPUT.PUT_LINE(‘……………………’);
Loop
Fetch c_emp into er;
Exit when c_emp%notfound;
DBMS_OUTPUT.PUT_LINE(er.name||’…..’||er.sal);
End loop;
Close c_emp;
Open c_emp for select ename, sal from emp where deptno = 20;
DBMS_OUTPUT.PUT_LINE(‘department: 20’);
DBMS_OUTPUT.PUT_LINE(‘……………………..’);
Loop
Fetch c_emp into er;
Exit when c_emp%notfound;
DBMS_OUTPUT.PUT_LINE(er.name||’-‘||er.sal);
End loop;
Close c_emp;
END;
  /
```

__SQL * loader__:

SQL loader is a tool which is use to load the data from the file to the table.

This tool requires control file(.Ctrl).
Control file contains all the information about source and destination.
It is developer responsibility to create a control file.

<u>Syntax to create control file</u>:

```
LOAD data
Infile '<Data filepath>'
Insert into table <Table_name> fields Terminated by ','
(col1, col2,...., clon)
```

<u>Ex</u>:

```
            LOAD data
            INFILE 'E:\sunil\student_data.txt'
            Insert into table sutdent50 fields terminated by ','
            (sno, sname, marks)
```

<u>Steps to invoke the tool</u>:
Step 1: open the command prompt.
Step 2: >SQLLDR SCOTT/TIGER
            CONTROL: E:\SUNIL\Load.CTL

*__<u>Autonomous transactions</u>__:
In general a commit command used in a PL/SQL block will act globally and make all the changes permanent.
To restrict commit command to a specific program we need to make the PL/SQL block autonomous.
We can create autonomous PL/SQL block by using 'PRAGMA AUTONOMOUS_TRANSACTION' in the declare section.
PRAGMA autonomous_transaction is a compiler directive.

<u>Ex</u>:

```
Create table student20(sno number(3), sname varchar2(10), marks number(3));
Insert into student20 values(101,'Arun',40)
Insert into student20 values(102,'Arun',40)
Declare
Pragma autonomus_transactions;
Begin
Insert into student20 values(103,'Arun',40);
Insert into student20 values(104,'Arun',40);
commit;
END;
```

```
 /
```

<u>Where current of</u>:

Where current of clause is used in some update statements. The where current of clause is an update or delete statement states that the most recent row fetched from the table should be updated.

We must declare the cursor with ‘for update’ clause to use this feature.

<u>Ex</u>:

```
Declare
Cursor c1
IS
Select empno, ename, sal from emp
where comm is null
for update of comm.;
var_comm number(4);
begin
for cur_rec.sal < 2000 then
var_comm := 200;
elsif cur_rec.sal < 4000 then
var_comm := 400;
else
var_comm := 100;
end if;
update emp set comm = var_comm
where current of c1;
end loop;
end;
 /
```

***<u>Managing large objects</u>**:

***<u>Creating table with LDB columns</u>**:

Ex: create table airbus_desc5(airbusno char(5), airbus_det bfile, airbus_profile clob);

***<u>Insert values in lobs</u>**: To insert values in the bfile, the function bfilename is used.

It takes the os path of the directory and the name of the file.

<u>Ex</u>:

Insert into airbus_desc5 values('ABO1', bfilename('E:\sunil','esert.jpg'),
'the description the plane is as follows');

***displaying data from lobs**: data from lobs cannot be displayed, except for clob by using select statement.
Ex: select aitbusno, airbus_profile from airbus_desc5;

***Locks**:
As oracle is a multiuser environment there is always a chance that multiple users will perform DML operators on some table parallel in such case the data becomes inconsistent.
To maintain the consistency of data base a user can lock the table.
Select for update command is used for locking a table.
Ex:
Ajit> select * from student for update;
Table is locked by Ajit
ASHWIN> update Ajit.student set marks = 95 where sno = 101;
Client will be in waiting state.
Locks are released when a user executes commit command.

***Levels of locks**: There are three levels of locks.
1. Row level
2. Page level
3. Table level

***Row level**: when where clause in select for update command is evaluating to one row, row level lock is applied.
Ex: select * from student where sno =101 for update;

***Page level**: when where clause in select for update command is evaluating to multiple rows, page level lock is applied.
Ex: Select * from emp where deptno = 20 for update;

***Table level**:
When where clause is not used in select for update, table level lock is applied.
Ex: select * from emp for update;

***FLASHBACK and PURGE command**:

*What is Recycle bin?
Oracle has introduced “Recycle bin” feature oracle log to store all the dropped objects. A user drops a very important table accidentally. Oracle log Recycle bin feature the user can easily restore the dropped object.

***To enable the recycle bin**:

```
SQL> Alter system set recycle bin = on;
                                  or
SQL> Alter session set recycle bin = on;
```

To view the recycle bin use the follows command:

```
SQL> show recycle bin;
```

Ex:

```
Create table test_inet1(val number(2));
SQL> insert into test_inet1(val) values(10);
SQL>drop table test_inet1;
Print the recycle bin;
```

Restore the objects back to data base:

```
FLASHBACK table <<table_name>> to before drop;
```

Ex: Flashback tabe test_inet1 to before drop;

```
SQL> select * from test_RBIN;
```

***Clearing the recycle bin (RB)**:

To clear the recycle bin the following statement can be used.

```
SQL> purge table <<table_name>>;
SQL> purge table student6;
```