‘Exercise : 1

AIM : Execute DDL,DML,DCL and TCL commands on below given relational schema.
EMP(Empno,Ename,Job,Salary,Mgr,Comm,Hiredate,Deptno).

Description **:**

SQL(Structured Query Language) is a standard language for storing, manipulating and retrieving data in databases.

The SQL uses four different languages for the commands

They are:

1. DDL – Data Definition Language.
2. DML –Data Manipulation Language.
3. DCL- Data Control Language.
4. TCL - Transaction Control Language.

Data Definition Language (DDL)

Data definition language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in database

List of DDL commands :

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

1. CREATE :

This command is used to create the database or its objects like table,index,function,views,store procedure and triggers.

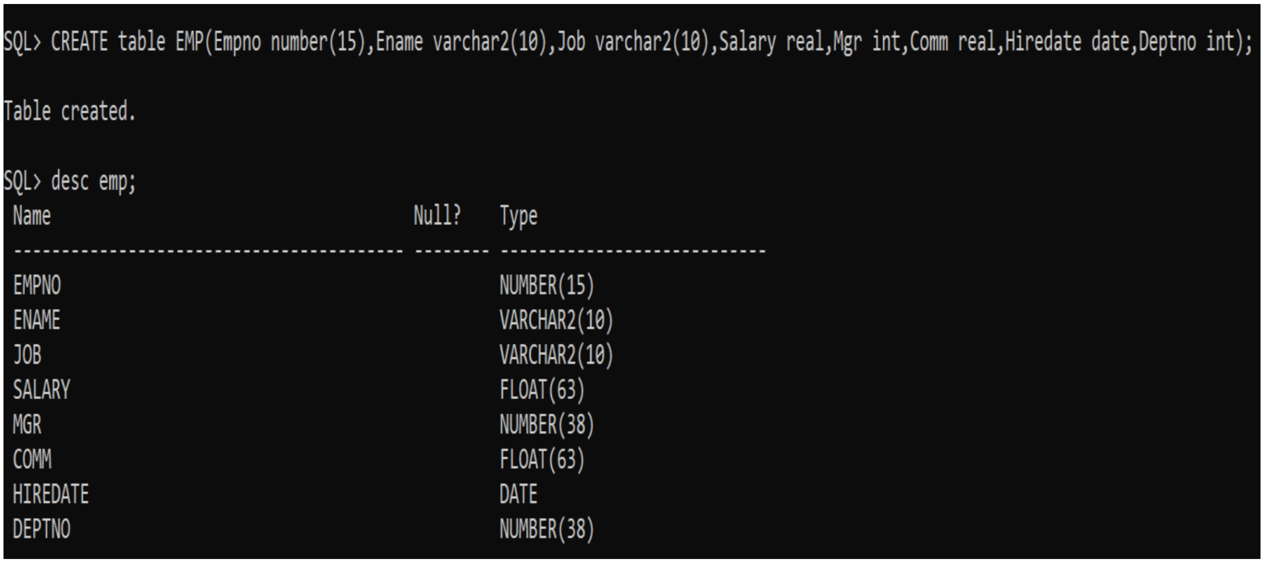
Syntax :

CREATE table table_name(column 1 domain type 1 , column 2 domain type 2 , …);

Example :

CREATE table EMP(Empno number(15),Ename varchar2(10),Job varchar2(10),Salary real,Mgr int,Comm real,Hiredate date,Deptno int);

desc emp;



```
SQL> CREATE table EMP(Empno number(15),Ename varchar2(10),Job varchar2(10),Salary real,Mgr int,Comm real,Hiredate date,Deptno int);

Table created.

SQL> desc emp;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                              NUMBER(15)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(10)
 SALARY                                             FLOAT(63)
 MGR                                                NUMBER(38)
 COMM                                               FLOAT(63)
 HIREDATE                                           DATE
 DEPTNO                                             NUMBER(38)
```

2. DROP :

This command is used to delete objects from database.

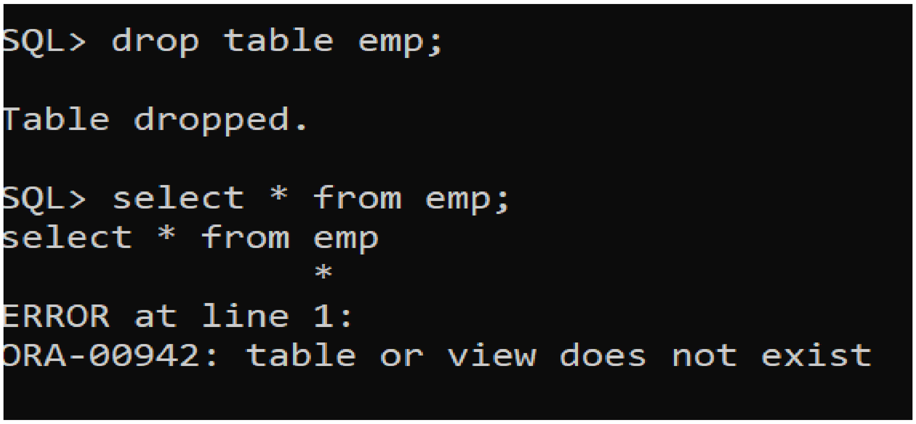
Syntax :

DROP table table_name;

Example :

DROP table EMP;

Select * from emp;



```
SQL> drop table emp;

Table dropped.

SQL> select * from emp;
select * from emp
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```

3. ALTER :

This is used to alter the structure of the database.

Syntax :

ALTER TABLE - ADD COLUMN :

ALTER table table_name add column_name domain type ;

ALTER TABLE – DROP COLUMN :

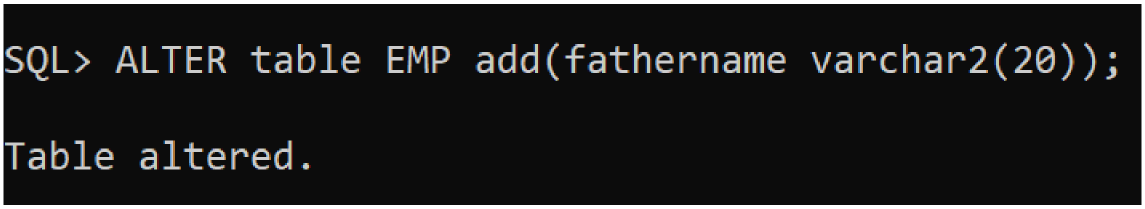
ALTER table table_name DROP column column_name;

ALTER TABLE – MODIFY COLUMN :

ALTER table table_name MODIFY column_name datatype ;

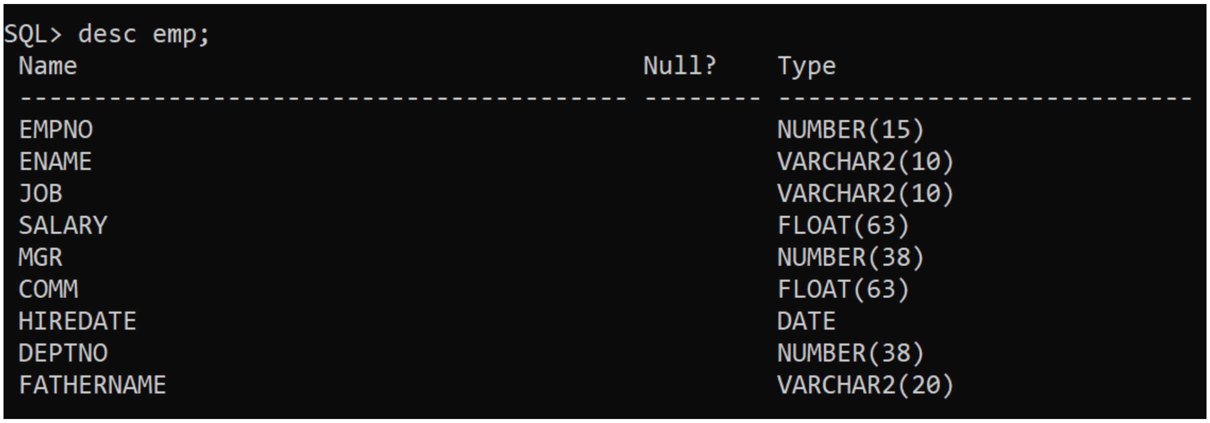
Example :

ALTER table EMP add(fathername varchar2(20)) ;



```
SQL> ALTER table EMP add(fathername varchar2(20));

Table altered.
```

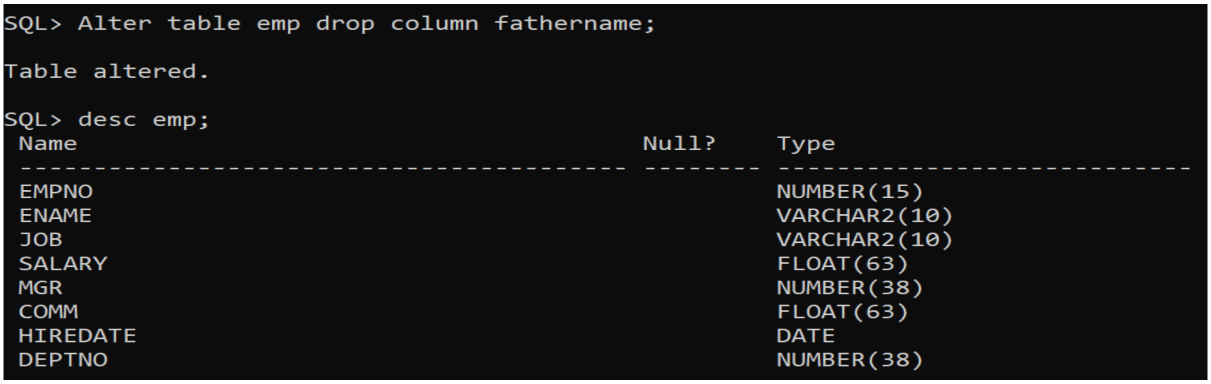
Desc emp;



```
SQL> desc emp;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                              NUMBER(15)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(10)
 SALARY                                             FLOAT(63)
 MGR                                                NUMBER(38)
 COMM                                               FLOAT(63)
 HIREDATE                                           DATE
 DEPTNO                                             NUMBER(38)
 FATHERNAME                                         VARCHAR2(20)
```

Alter table emp drop column fathername;

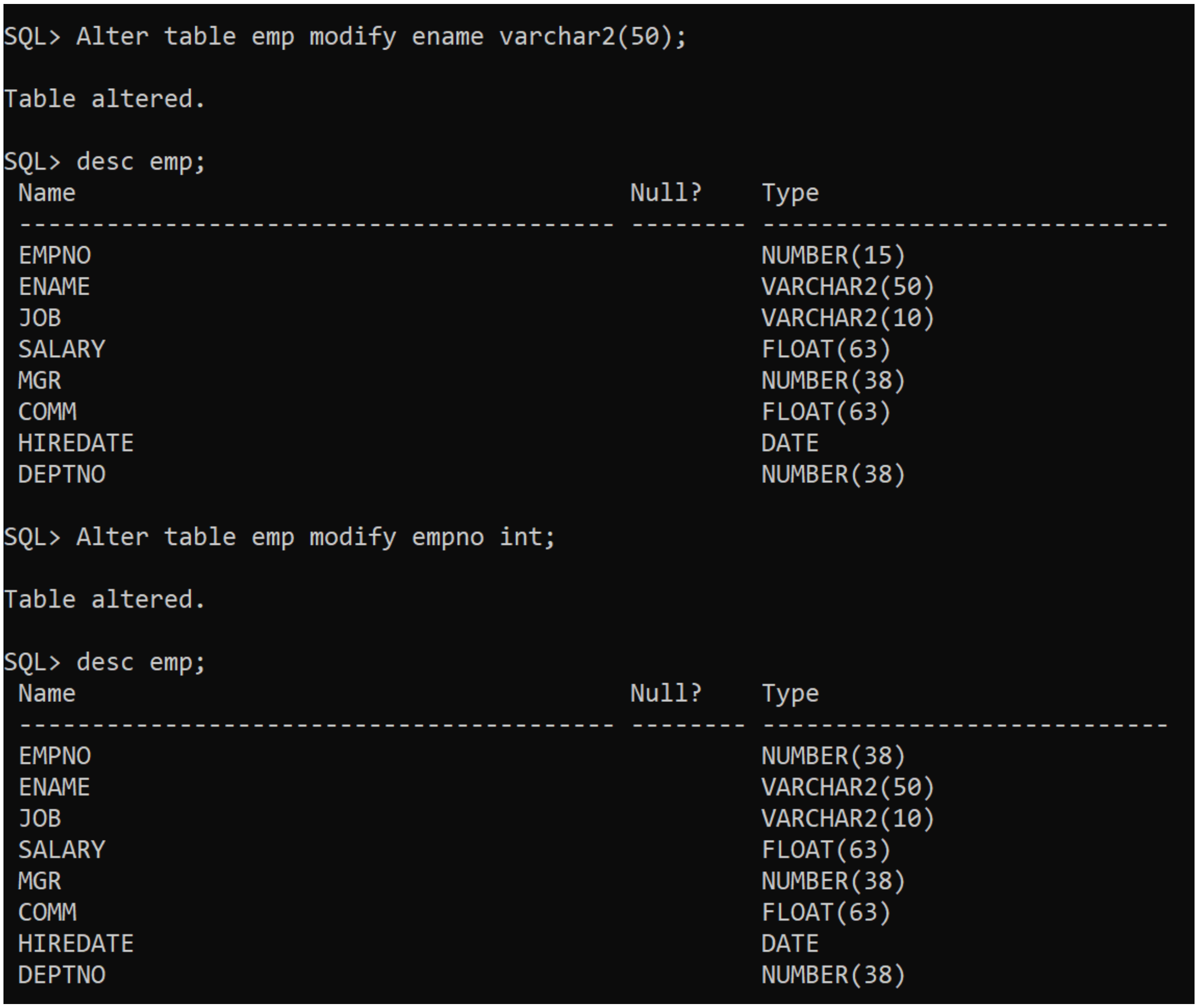
Desc emp;



```
SQL> Alter table emp drop column fathername;

Table altered.

SQL> desc emp;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                              NUMBER(15)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(10)
 SALARY                                             FLOAT(63)
 MGR                                                NUMBER(38)
 COMM                                               FLOAT(63)
 HIREDATE                                           DATE
 DEPTNO                                             NUMBER(38)
```

Alter table emp modify name varchar2(20);

Alter table emp modify empno int;



```
SQL> Alter table emp modify ename varchar2(50);

Table altered.

SQL> desc emp;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                              NUMBER(15)
 ENAME                                              VARCHAR2(50)
 JOB                                                VARCHAR2(10)
 SALARY                                             FLOAT(63)
 MGR                                                NUMBER(38)
 COMM                                               FLOAT(63)
 HIREDATE                                           DATE
 DEPTNO                                             NUMBER(38)

SQL> Alter table emp modify empno int;

Table altered.

SQL> desc emp;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                              NUMBER(38)
 ENAME                                              VARCHAR2(50)
 JOB                                                VARCHAR2(10)
 SALARY                                             FLOAT(63)
 MGR                                                NUMBER(38)
 COMM                                               FLOAT(63)
 HIREDATE                                           DATE
 DEPTNO                                             NUMBER(38)
```

4.TRUNCATE :

This is used to remove all records from a table , including all spaces allocated for the records are removed.

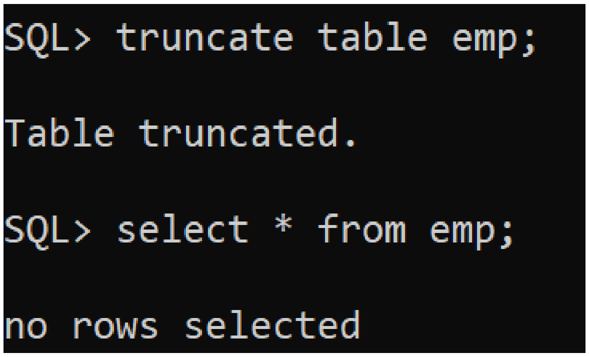
Syntax :

TRUNCATE table table_name

Example:

TRUNCATE table EMP;

Select * from emp;



```
SQL> truncate table emp;

Table truncated.

SQL> select * from emp;

no rows selected
```

5. RENAME

Rename will be in two situations.

1. To change the name of the table.
2. To change the name of the column.

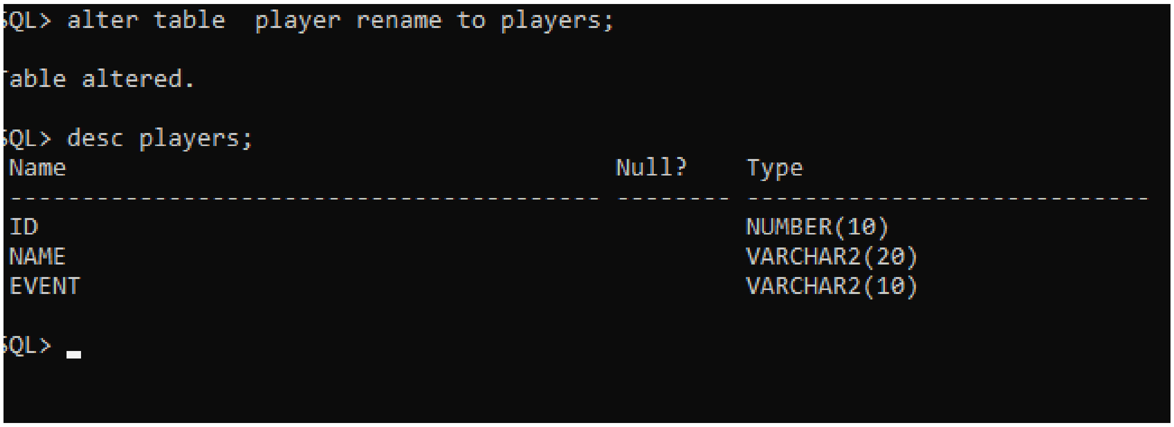
Syntax

i) alter table tablename rename to players.

Example

alter table player rename to players;
Table altered.
desc players;

Output



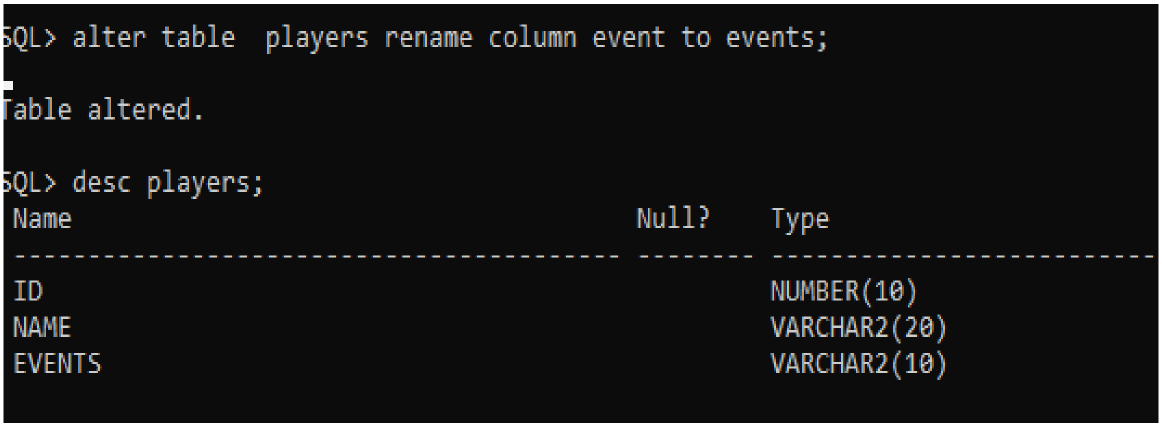
```
QL> alter table  player rename to players;

able altered.

QL> desc players;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                                 NUMBER(10)
 NAME                                               VARCHAR2(20)
 EVENT                                              VARCHAR2(10)

QL> _
```

ii) alter table tablename column<old-column> to <new-coloumn>

Example

alter table players rename column Event to Events;
table altered.
desc players;

Output



```
SQL> alter table  players rename column event to events;

Table altered.

SQL> desc players;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                                 NUMBER(10)
 NAME                                               VARCHAR2(20)
 EVENTS                                             VARCHAR2(10)
```

Data Manipulating Language (DML) :

The SQL commands that deals with the manipulation of data present in the data .

DML is the component of SQL statement that controls access to data and to the database.

List of DML commands :

1. SELECT – it is used to retrieve data from the database.
2. INSERT – it is used to insert data into a table.
3. UPDATE – it is used to update existing data within a table.
4. DELETE – it is used to delete records from a database table.

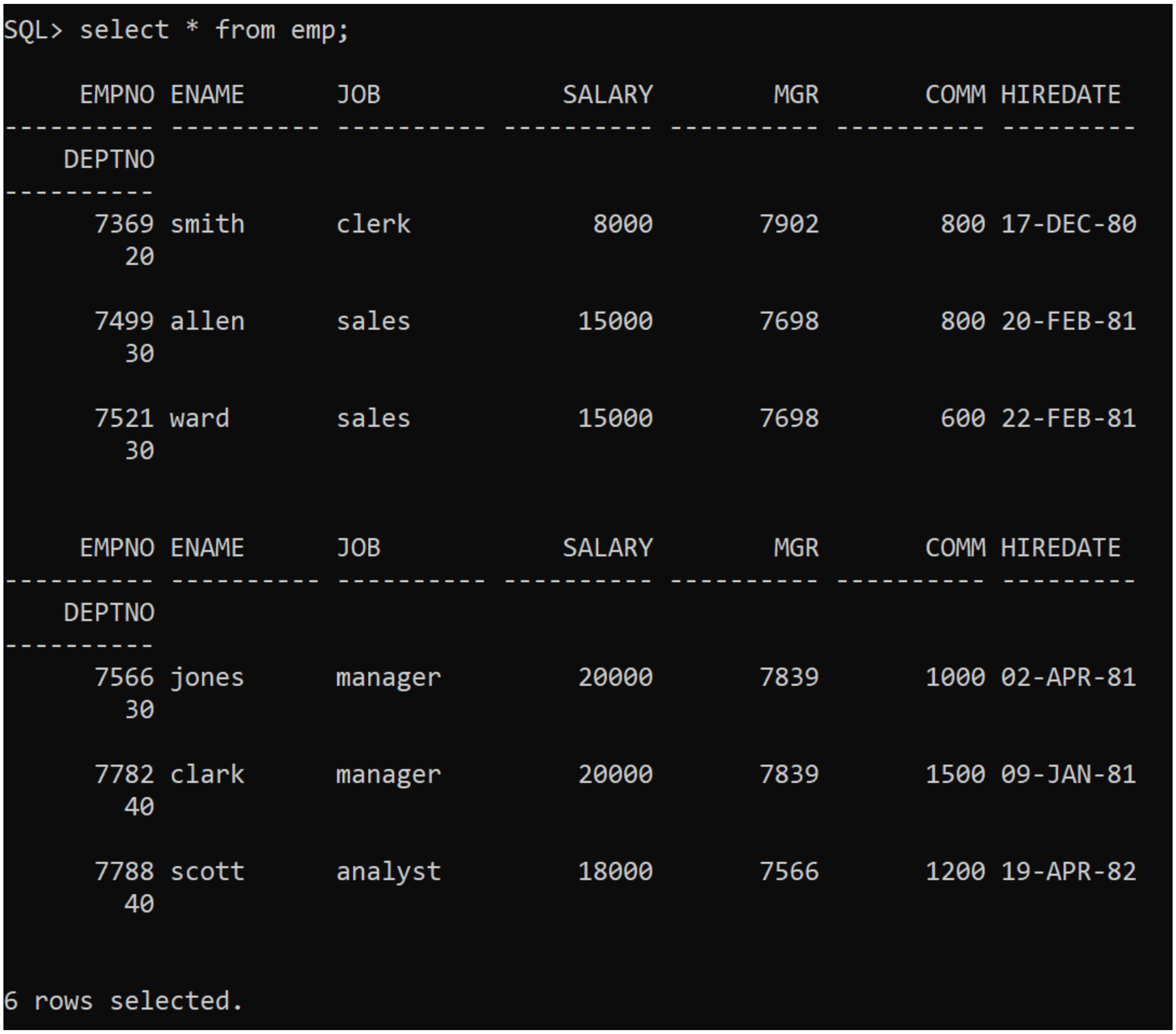
1. SELECT : It is used to select data from a database. The data returned is stored in a result table, called the result set.

Syntax :

SELECT * FROM table_name ;

Example :

Select * from EMP;



```
SQL> select * from emp;

     EMPNO ENAME      JOB            SALARY        MGR       COMM HIREDATE
---------- ---------- ---------- ---------- ---------- ---------- ---------
    DEPTNO
----------
      7369 smith      clerk            8000       7902        800 17-DEC-80
        20

      7499 allen      sales           15000       7698        800 20-FEB-81
        30

      7521 ward       sales           15000       7698        600 22-FEB-81
        30


     EMPNO ENAME      JOB            SALARY        MGR       COMM HIREDATE
---------- ---------- ---------- ---------- ---------- ---------- ---------
    DEPTNO
----------
      7566 jones      manager         20000       7839       1000 02-APR-81
        30

      7782 clark      manager         20000       7839       1500 09-JAN-81
        40

      7788 scott      analyst         18000       7566       1200 19-APR-82
        40


6 rows selected.
```

2. INSERT : It is an SQL command used to insert new rows in a table.

Syntax :

INSERT INTO table_name values(value1 , value 2 , …);

Example :

insert into emp values(7369,'smith','clerk',8000,7902,800,'17-dec-80',20);

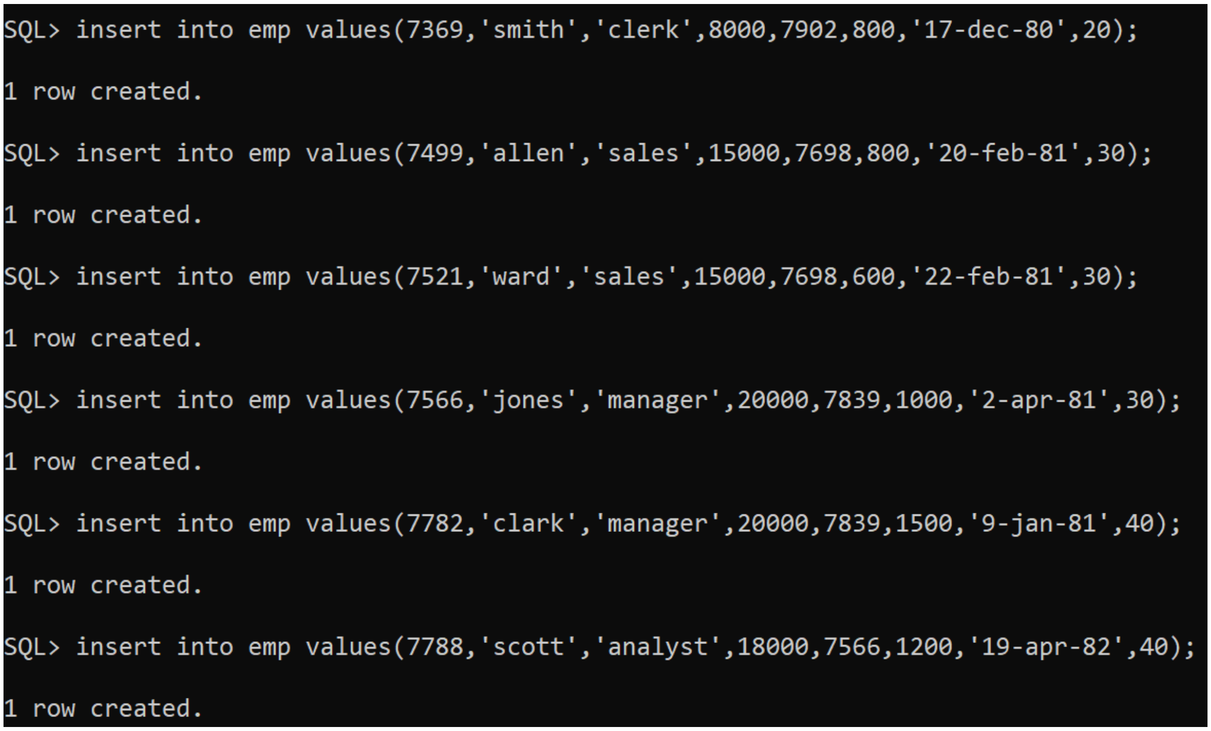
 insert into emp values(7499,'allen','sales',15000,7698,800,'20-feb-81',30);

 insert into emp values(7521,'ward','sales',15000,7698,600,'22-feb-81',30)

insert into emp values(7566,'jones','manager',20000,7839,1000,'2-apr-81',30);

 insert into emp values(7782,'clark','manager',20000,7839,1500,'9-jan-81',40);

 insert into emp values(7788,'scott','analyst',18000,7566,1200,'19-apr-82',40);



```
SQL> insert into emp values(7369,'smith','clerk',8000,7902,800,'17-dec-80',20);

1 row created.

SQL> insert into emp values(7499,'allen','sales',15000,7698,800,'20-feb-81',30);

1 row created.

SQL> insert into emp values(7521,'ward','sales',15000,7698,600,'22-feb-81',30);

1 row created.

SQL> insert into emp values(7566,'jones','manager',20000,7839,1000,'2-apr-81',30);

1 row created.

SQL> insert into emp values(7782,'clark','manager',20000,7839,1500,'9-jan-81',40);

1 row created.

SQL> insert into emp values(7788,'scott','analyst',18000,7566,1200,'19-apr-82',40);

1 row created.
```

3. UPDATE : It is an SQL command used to update existing rows in a table.

Syntax

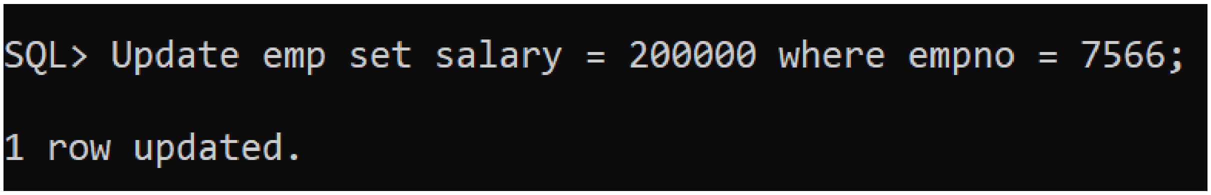
UPDATE table_name

SET attribute = value

WHERE condition;

Example :

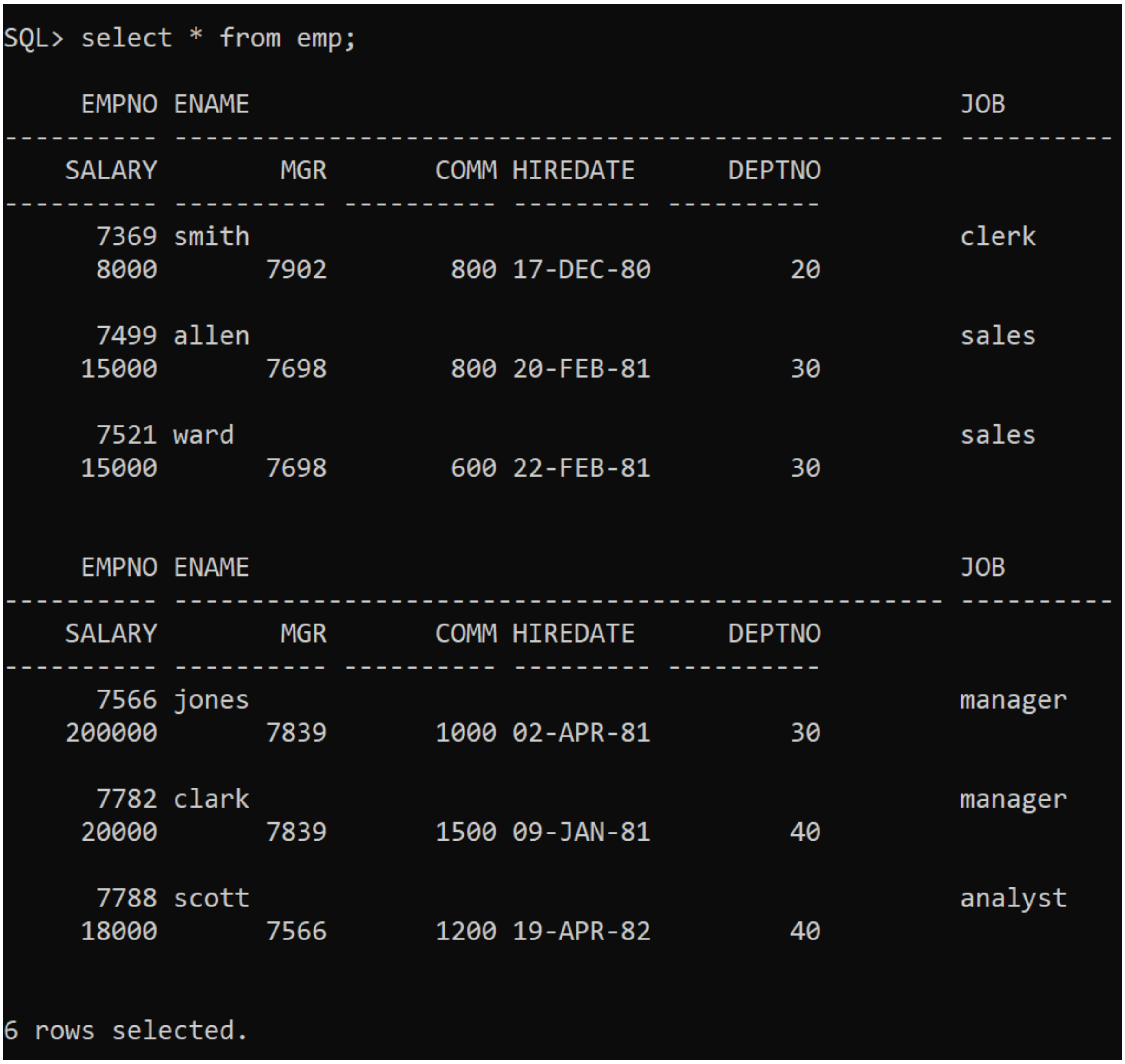
Update emp set salary = 200000 where empno = 7566;



```
SQL> Update emp set salary = 200000 where empno = 7566;

1 row updated.
```

...

select * from emp;



```
SQL> select * from emp;

     EMPNO ENAME                                              JOB
---------- -------------------------------------------------- ----------
    SALARY        MGR       COMM HIREDATE      DEPTNO
---------- ---------- ---------- --------- ----------
      7369 smith                                              clerk
      8000       7902        800 17-DEC-80         20

      7499 allen                                              sales
     15000       7698        800 20-FEB-81         30

      7521 ward                                               sales
     15000       7698        600 22-FEB-81         30


     EMPNO ENAME                                              JOB
---------- -------------------------------------------------- ----------
    SALARY        MGR       COMM HIREDATE      DEPTNO
---------- ---------- ---------- --------- ----------
      7566 jones                                              manager
    200000       7839       1000 02-APR-81         30

      7782 clark                                              manager
     20000       7839       1500 09-JAN-81         40

      7788 scott                                              analyst
     18000       7566       1200 19-APR-82         40


6 rows selected.
```

4. DELETE : The delete command is an SQL command used to delete existing records in a table.

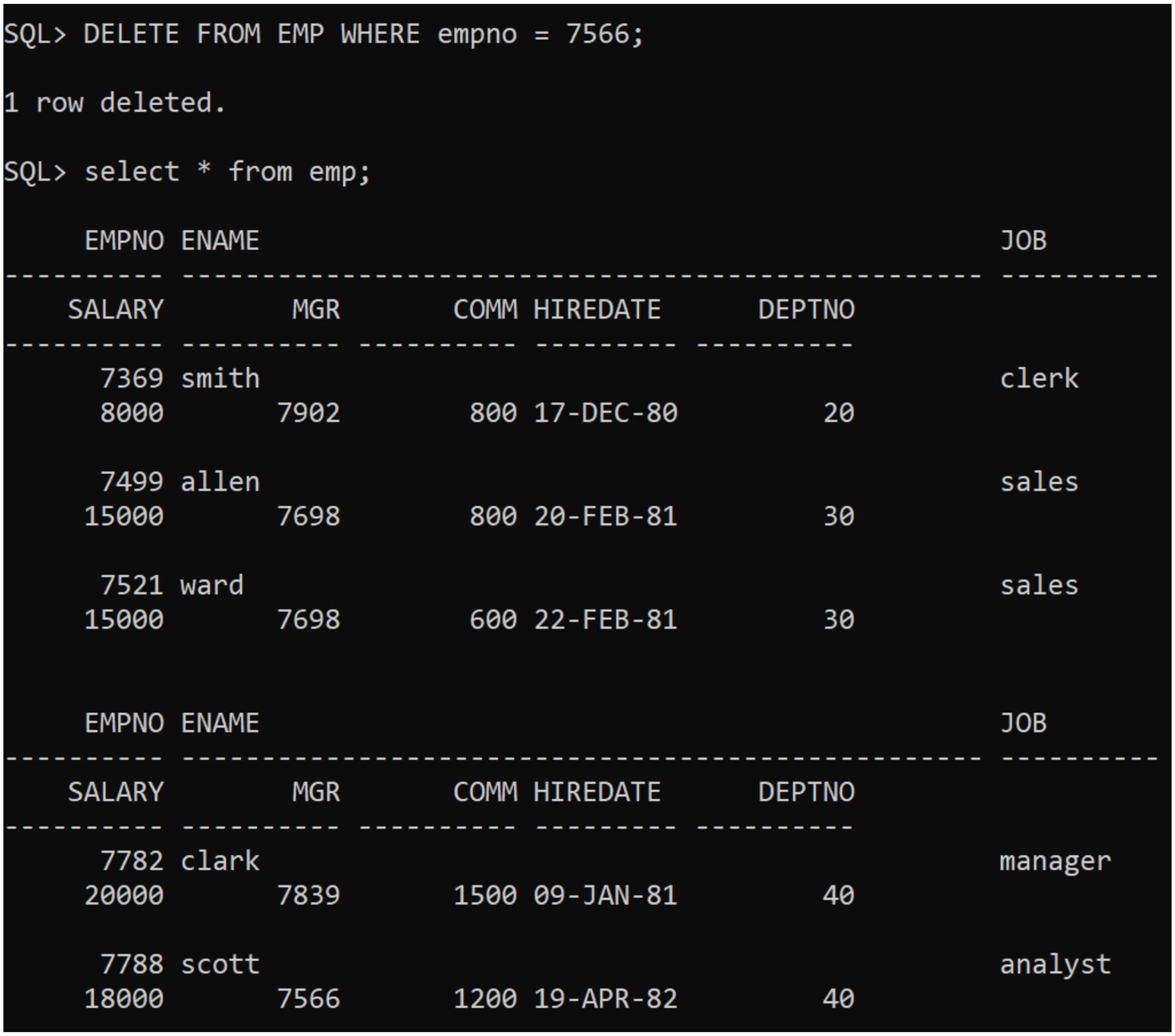
Syntax :

DELETE FROM table_name WHERE condition ;

Example :

DELETE FROM EMP WHERE empno = 7566;

Select * from emp;



```
SQL> DELETE FROM EMP WHERE empno = 7566;

1 row deleted.

SQL> select * from emp;

     EMPNO ENAME                                              JOB
---------- -------------------------------------------------- ----------
    SALARY        MGR       COMM HIREDATE      DEPTNO
---------- ---------- ---------- --------- ----------
      7369 smith                                              clerk
      8000       7902        800 17-DEC-80         20

      7499 allen                                              sales
     15000       7698        800 20-FEB-81         30

      7521 ward                                               sales
     15000       7698        600 22-FEB-81         30


     EMPNO ENAME                                              JOB
---------- -------------------------------------------------- ----------
    SALARY        MGR       COMM HIREDATE      DEPTNO
---------- ---------- ---------- --------- ----------
      7782 clark                                              manager
     20000       7839       1500 09-JAN-81         40

      7788 scott                                              analyst
     18000       7566       1200 19-APR-82         40
```

Data Control Language (DCL) :

DCL commands mainly deals with the rights , permissions , and other controls of the database system.

List of DCL commands :

1. GRANT – this command gives users access privileges to the database.
2. REVOKE – this command withdraws the users access privileges given by the GRANT command.

1. GRANT : SQL grant command is specifically used to provide privileges to database objects for a user. This command also allows users to grant permissions to other users too.

Syntax

Grant privilege_name on object_name to {usesr_name};

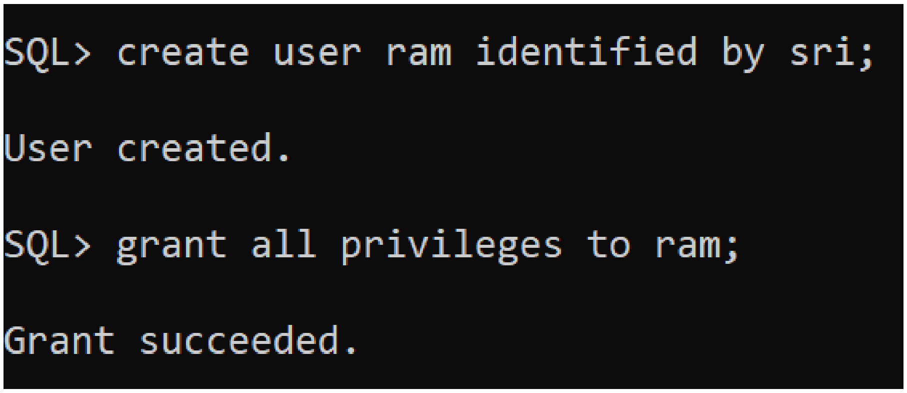
Example :

Create user ram identified by sri;

User created.

Grant all privileges to ram;

Grant succeded.



```
SQL> create user ram identified by sri;

User created.

SQL> grant all privileges to ram;

Grant succeeded.
```

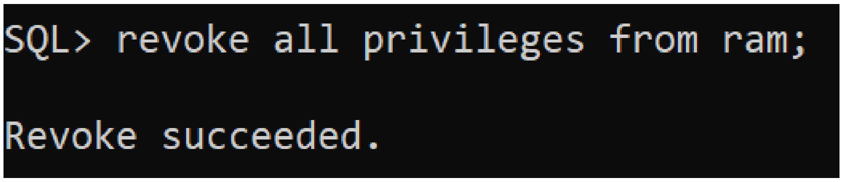
2. REVOKE : Revoke command withdraw user privileges on database objects if any granted.

Syntax :

Revoke privilege_name on onject_name from {user_name};

Example:

Revoke all privileges from ram;



```
SQL> revoke all privileges from ram;

Revoke succeeded.
```

Transaction Control Language(TCL) :

List of TCL commands :

1. COMMIT – commits a transaction.
2. ROLLBACK – rollbacks a transaction in case of any error occurs.
3. SAVEPOINT – sets a savepoint within a transaction.

1. COMMIT : Commit command is used to save all transactions to the database.

Syntax :

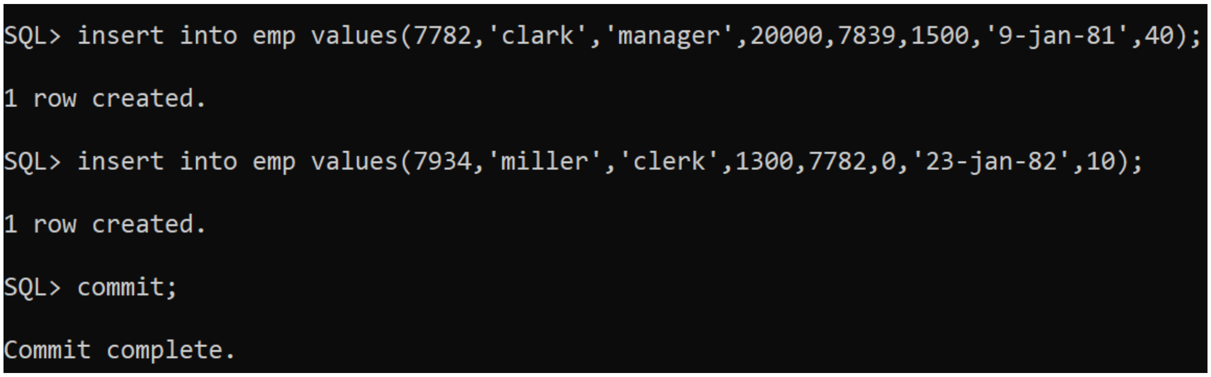
COMMIT;

Example :

insert into emp values(7782,'clark','manager',20000,7839,1500,'9-jan-81',40);

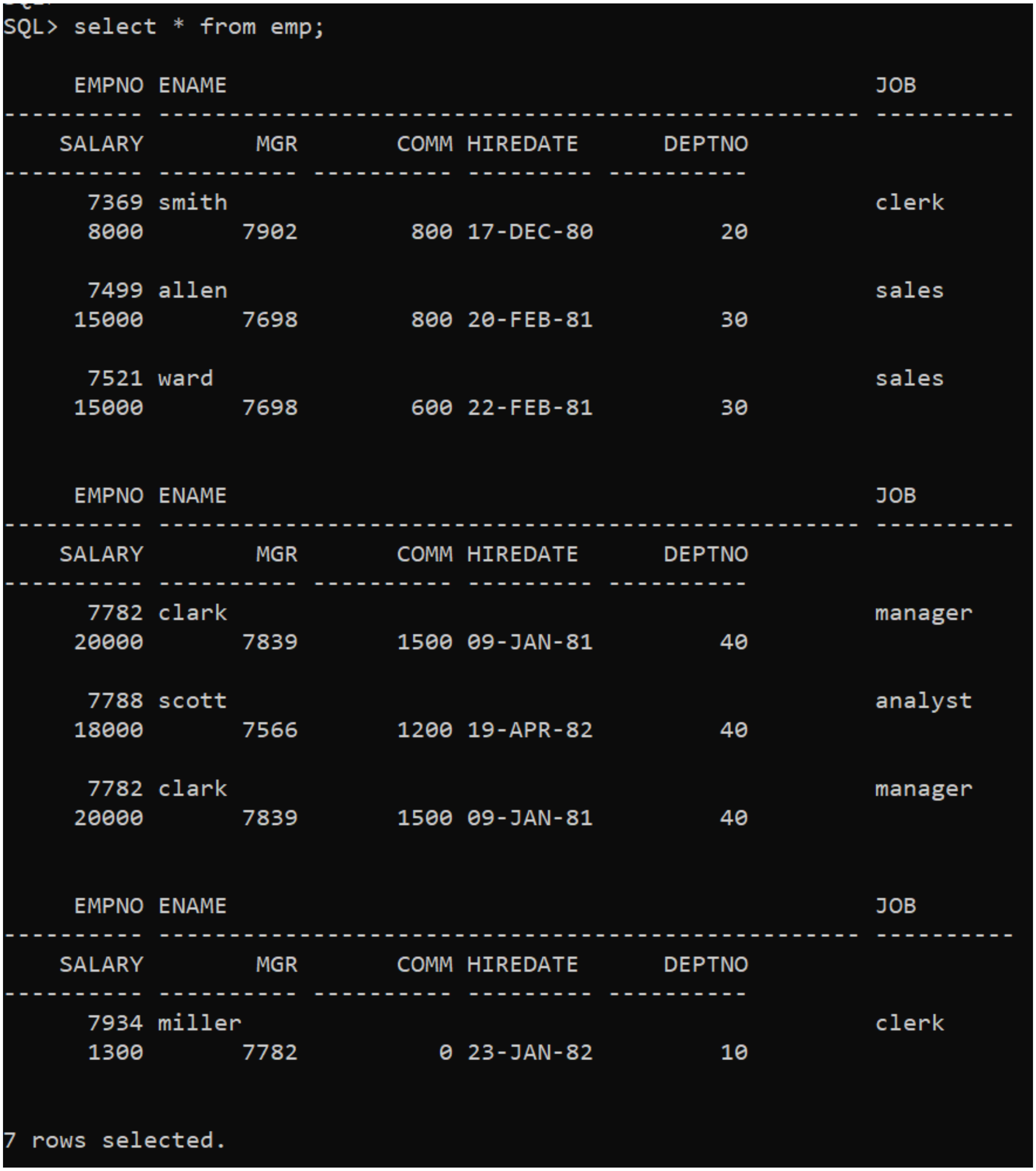
insert into emp values(7934,'miller','clerk',1300,7782,0,'23-jan-82',10);

Commit;



```
SQL> insert into emp values(7782,'clark','manager',20000,7839,1500,'9-jan-81',40);

1 row created.

SQL> insert into emp values(7934,'miller','clerk',1300,7782,0,'23-jan-82',10);

1 row created.

SQL> commit;

Commit complete.
```

Select * from emp;



```
SQL> select * from emp;

     EMPNO ENAME                                              JOB
---------- -------------------------------------------------- ----------
    SALARY        MGR       COMM HIREDATE      DEPTNO
---------- ---------- ---------- --------- ----------
      7369 smith                                              clerk
      8000       7902        800 17-DEC-80         20

      7499 allen                                              sales
     15000       7698        800 20-FEB-81         30

      7521 ward                                               sales
     15000       7698        600 22-FEB-81         30


     EMPNO ENAME                                              JOB
---------- -------------------------------------------------- ----------
    SALARY        MGR       COMM HIREDATE      DEPTNO
---------- ---------- ---------- --------- ----------
      7782 clark                                              manager
     20000       7839       1500 09-JAN-81         40

      7788 scott                                              analyst
     18000       7566       1200 19-APR-82         40

      7782 clark                                              manager
     20000       7839       1500 09-JAN-81         40


     EMPNO ENAME                                              JOB
---------- -------------------------------------------------- ----------
    SALARY        MGR       COMM HIREDATE      DEPTNO
---------- ---------- ---------- --------- ----------
      7934 miller                                             clerk
      1300       7782          0 23-JAN-82         10


7 rows selected.
```

2. ROLLBACK :

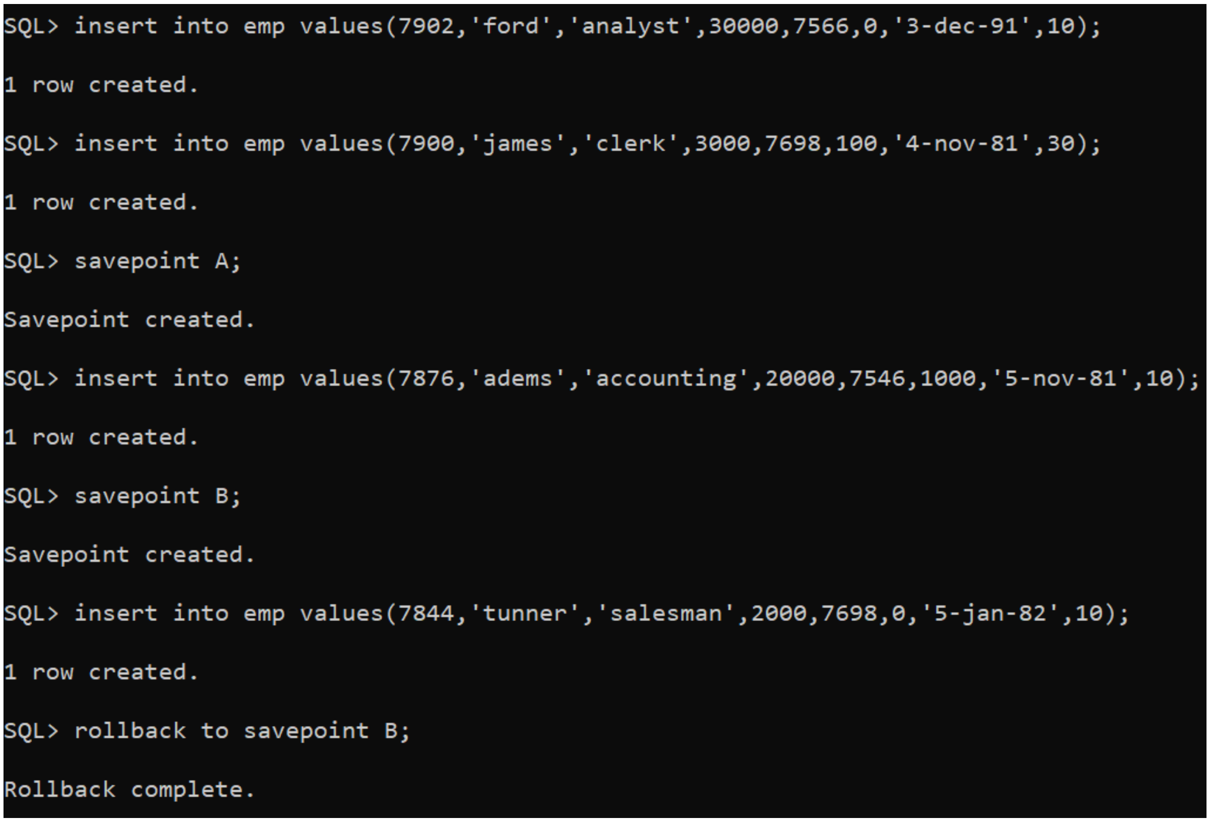
It is used to undo transactions that have not already been saved to the database.

Syntax :

ROLLBACK;

Example :

```
insert into emp values(7902,'ford','analyst',30000,7566,0,'3-dec-91',10);

1 row created.

insert into emp values(7900,'james','clerk',3000,7698,100,'4-nov-81',30);

1 row created.

 savepoint A;

Savepoint created.

insert into emp values(7876,'adems','accounting',20000,7546,1000,'5-nov-81',10);

1 row created.

savepoint B;

Savepoint created.

insert into emp values(7844,'tunner','salesman',2000,7698,0,'5-jan-82',10);

1 row created.

rollback to savepoint B;

Rollback complete.
```



```
SQL> insert into emp values(7902,'ford','analyst',30000,7566,0,'3-dec-91',10);

1 row created.

SQL> insert into emp values(7900,'james','clerk',3000,7698,100,'4-nov-81',30);

1 row created.

SQL> savepoint A;

Savepoint created.

SQL> insert into emp values(7876,'adems','accounting',20000,7546,1000,'5-nov-81',10);

1 row created.

SQL> savepoint B;

Savepoint created.

SQL> insert into emp values(7844,'tunner','salesman',2000,7698,0,'5-jan-82',10);

1 row created.

SQL> rollback to savepoint B;

Rollback complete.
```

3. SAVEPOINT : It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax :

SAVEPPOINT savepoint_name;

Example :

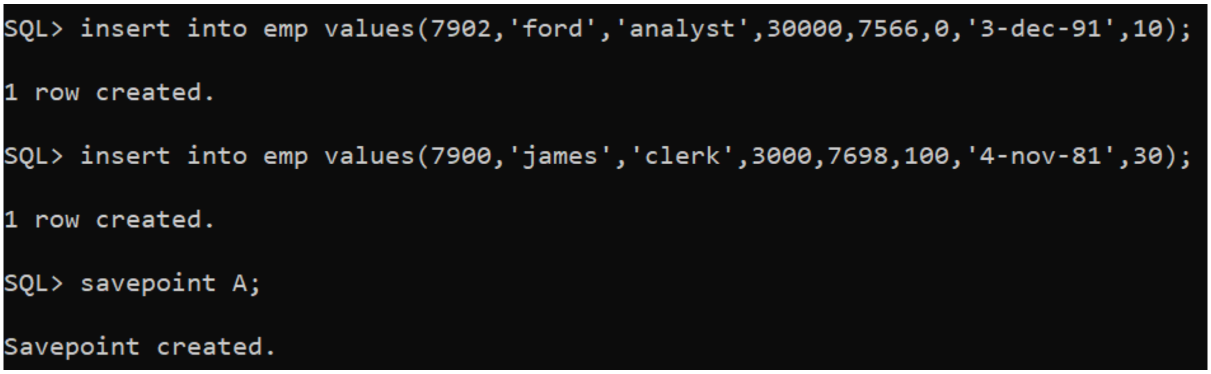
insert into emp values(7902,'ford','analyst',30000,7566,0,'3-dec-91',10);

1 row created.

insert into emp values(7900,'james','clerk',3000,7698,100,'4-nov-81',30);

1 row created.

savepoint A;



```
SQL> insert into emp values(7902,'ford','analyst',30000,7566,0,'3-dec-91',10);

1 row created.

SQL> insert into emp values(7900,'james','clerk',3000,7698,100,'4-nov-81',30);

1 row created.

SQL> savepoint A;

Savepoint created.
```

VIVA-VOCE QUESTIONS

1. List out DDL, DML, TCL and DCL commands.
2. Difference between Truncate and Drop.
3. Difference between Commit and Savepoint.
4. Creation of a table.

EXERCISE : 2

AIM : Implement the following integrity constraints on the following database EMP (Empno, Ename, Job, Salary, Mgr, Comm, Hiredate, Deptno) DEPT(Deptno, Dname, Location) a. Primary Key b. Foreign Key c. Unique d. Not NULL e. Check

Description :

Constraints

❖ **KEY CONSTRAINTS**

- **Super key :** set of one or more attributes that uniquely identifies a tuple in a relation is called as a super key.
- **Candidate key :** minimal set of attributes that uniquely identifies a tuple in a relation is called as a candidate key.
- **Primary key :** it is a key which uniquely identifies a tuple in a relation . the two properties of primary key are unique and not null.
- **Alternate key:** an alternate key is a key that can be work as a primary key .basically it is a candidate key that is not a primary key.
- **Foreign key:** ensure that referential integrity of the data in one table to match values in another table . ensure that the foregin key in the child table match with the primary key in the parent table.

❖ **INTEGRITY CONSTRAINTS**

- **Unique key** : unique key is a set of one or more fields/columns of a table that uniquely identify a record in database table .it is like primary key but it can accept only one null value and it cannot have duplicate values.
- **Check :** ensures that the value in a field meets a specified condition.
- **Not NULL :** indicates that a field cannot store a NULL value.

❖ **Constraints according to the aim :**

a) Primary Key constraint :

The primary key constraint uniquely identifies each record in atable. They must contain UNIQUE values and cannot contain NULL values. A table can have only ONE primary key and in the table, this primary key can consist of single or multiple columns/fields .

Syntax :

Create table table_name(attribute name domaintype primary key , ..);

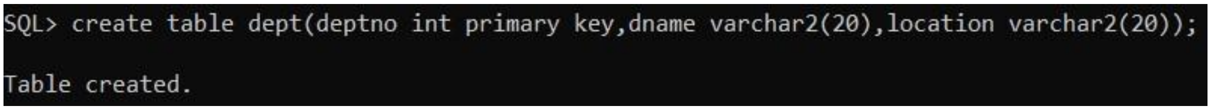
Or

By using alter :

Alter table table_name add constraint constraint_name primary key(attribute);

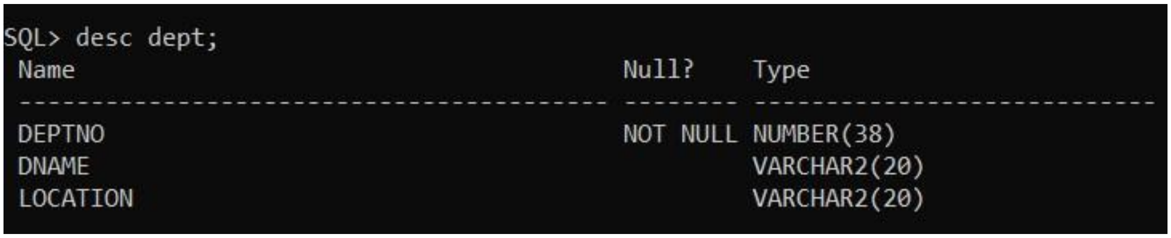
Example :

create table dept(deptno int primary key,dname varchar2(20),location varchar2(20));



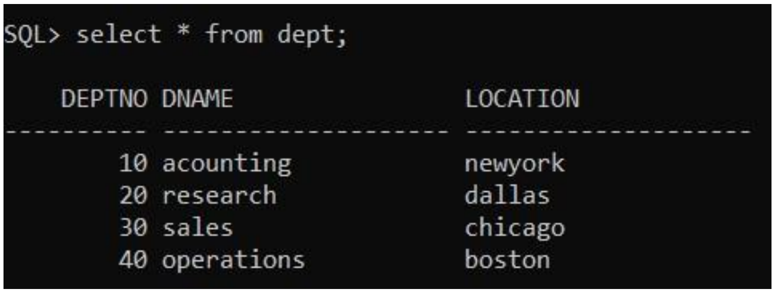
```
SQL> create table dept(deptno int primary key,dname varchar2(20),location varchar2(20));

Table created.
```

Desc dept;



```
SQL> desc dept;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPTNO                                    NOT NULL NUMBER(38)
 DNAME                                              VARCHAR2(20)
 LOCATION                                           VARCHAR2(20)
```

Select * from dept;



```
SQL> select * from dept;

    DEPTNO DNAME                LOCATION
---------- -------------------- --------------------
        10 acounting            newyork
        20 research             dallas
        30 sales                chicago
        40 operations           boston
```

b) Foreign Key Constraint:

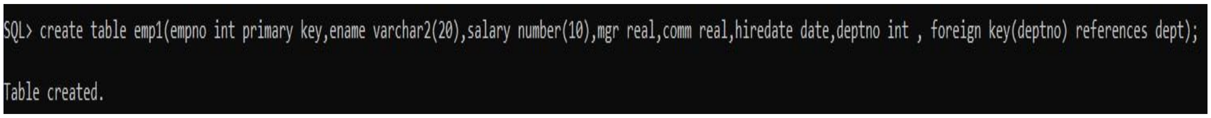
The foreign key constraint is used to prevent actions that would destroy links between tables. A foreign key is a field or a collection of fields in one table , that refers to the primary key in another table. The table with the foreign key is called the child table , and the table with the primary key is called the referenced or parent table.

Syntax :

Create table table_name(column domain type ,… , column n domain type n , foregin key(column) , references column in parent table);

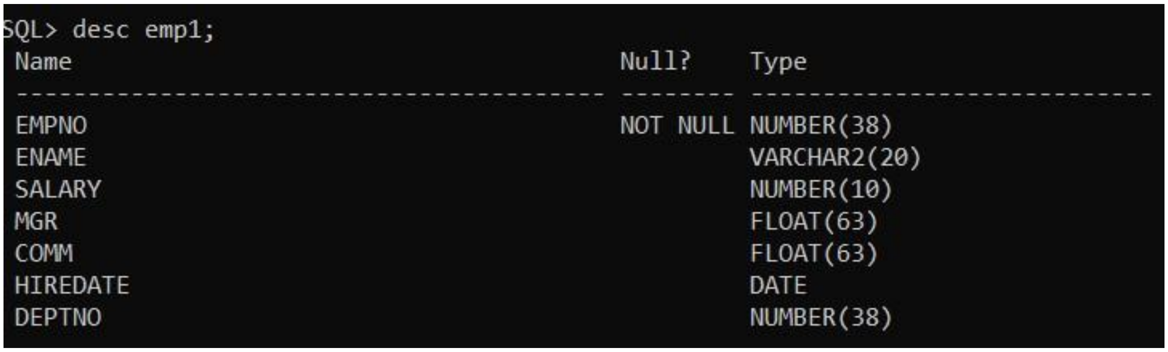
Example :

create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept);



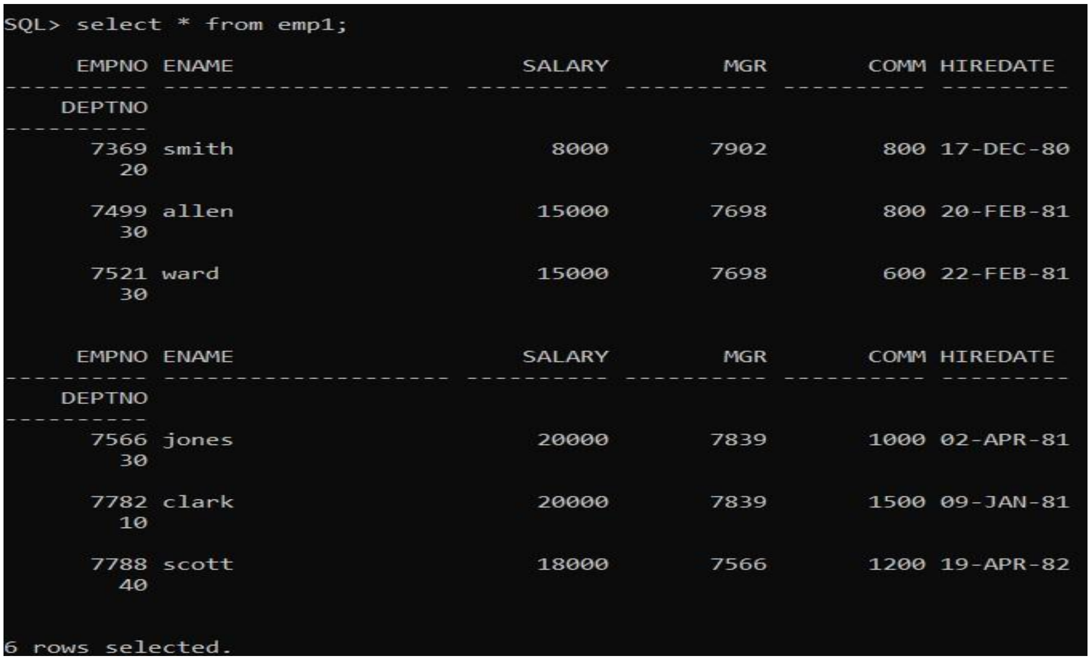
```
SQL> create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept);

Table created.
```

Desc emp;



```
SQL> desc emp1;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                     NOT NULL NUMBER(38)
 ENAME                                              VARCHAR2(20)
 SALARY                                             NUMBER(10)
 MGR                                                FLOAT(63)
 COMM                                               FLOAT(63)
 HIREDATE                                           DATE
 DEPTNO                                             NUMBER(38)
```

Select * from emp;



```
SQL> select * from emp1;

     EMPNO ENAME                    SALARY        MGR       COMM HIREDATE
---------- -------------------- ---------- ---------- ---------- ---------
    DEPTNO
----------
      7369 smith                      8000       7902        800 17-DEC-80
        20

      7499 allen                     15000       7698        800 20-FEB-81
        30

      7521 ward                      15000       7698        600 22-FEB-81
        30


     EMPNO ENAME                    SALARY        MGR       COMM HIREDATE
---------- -------------------- ---------- ---------- ---------- ---------
    DEPTNO
----------
      7566 jones                     20000       7839       1000 02-APR-81
        30

      7782 clark                     20000       7839       1500 09-JAN-81
        10

      7788 scott                     18000       7566       1200 19-APR-82
        40


6 rows selected.
```

It has two attributes :

1. ON DELETE CASCADE :

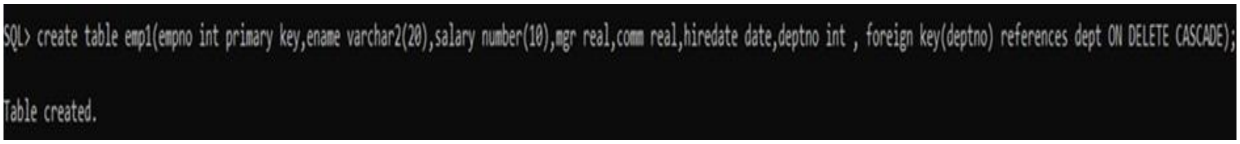
when a primary key is deleted in the parent table then corresponding data in the child table also gets deleted.

Syntax :

Create table table_name(attribute domain type , foregin key(attribute) references parent table ON DELETE CASCADE):

Example :

create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept ON DELETE CASCADE);



```
SQL> create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept ON DELETE CASCADE);

Table created.
```

2. ON DELETE SET NULL :

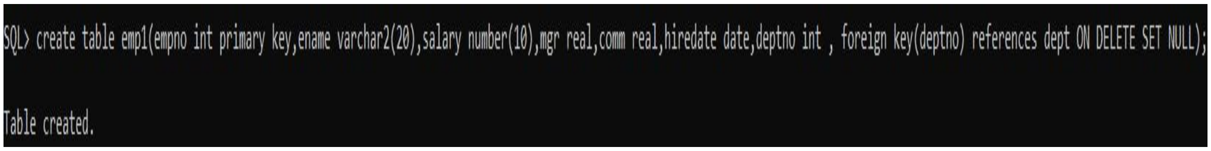
When a primary key and its corresponding tuples gets deleted in parent table and then corresponding records in the child table will have the foreign key field set to null but not get deleted.

Syntax :

Create table table_name(attribute domain type , foregin key(attribute) references parent table ON DELETE SET NULL):

Example :

create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept ON DELETE SET NULL);



```
SQL> create table emp1(empno int primary key,ename varchar2(20),salary number(10),mgr real,comm real,hiredate date,deptno int , foreign key(deptno) references dept ON DELETE SET NULL);

Table created.
```

3. Unique Key Constraint :

The unique constraint imposes that every value in a column or set of columns by unique. It means that no two rows of a table can have duplicate values in a specified column or set of columns.

Syntax:

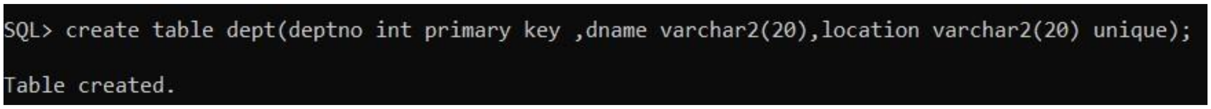
While creating table :

Syntax :

Create table table_name(coloumn domain type unique);

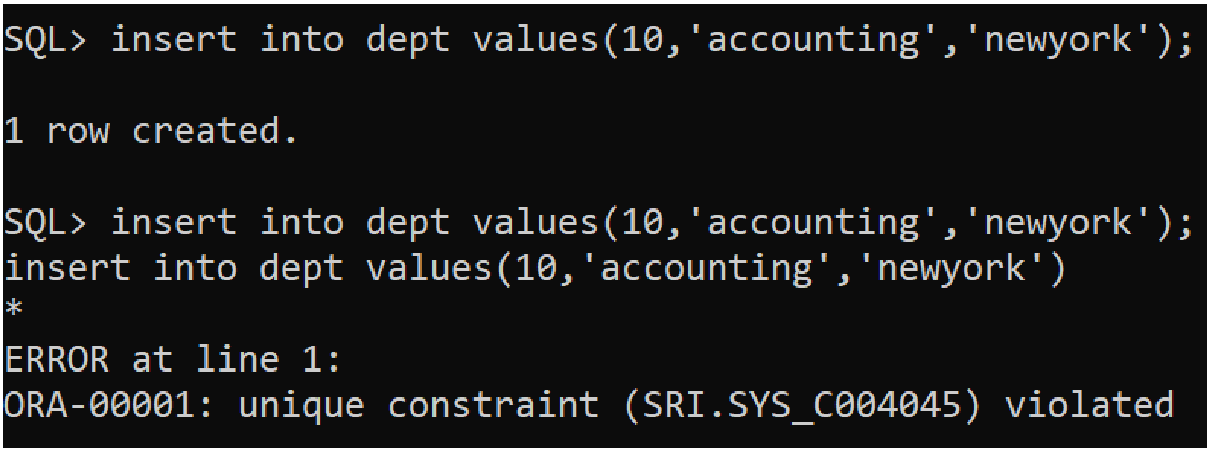
Example :

create table dept(deptno int primary key ,dname varchar2(20),location varchar2(20) unique);



```
SQL> create table dept(deptno int primary key ,dname varchar2(20),location varchar2(20) unique);

Table created.
```

Unique constraint violation :



```
SQL> insert into dept values(10,'accounting','newyork');

1 row created.

SQL> insert into dept values(10,'accounting','newyork');
insert into dept values(10,'accounting','newyork')
*
ERROR at line 1:
ORA-00001: unique constraint (SRI.SYS_C004045) violated
```

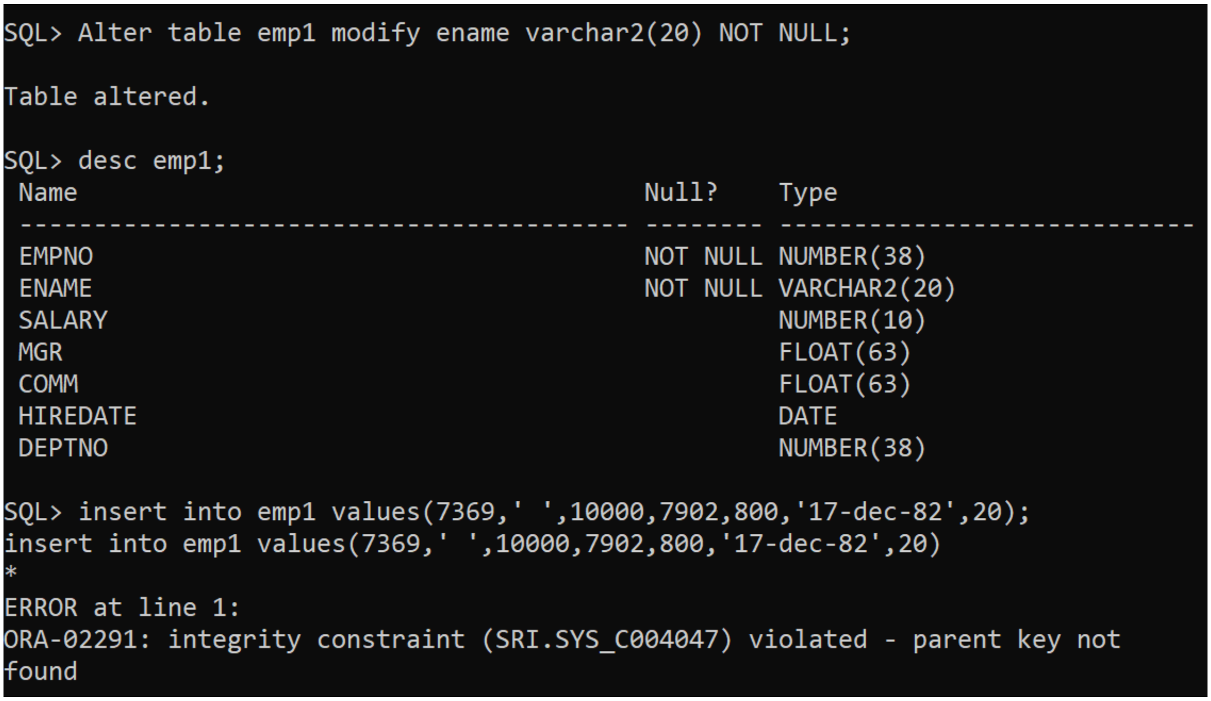
4. NOT NULL: Indicates that a column cannot store NULL value.

Syntax :

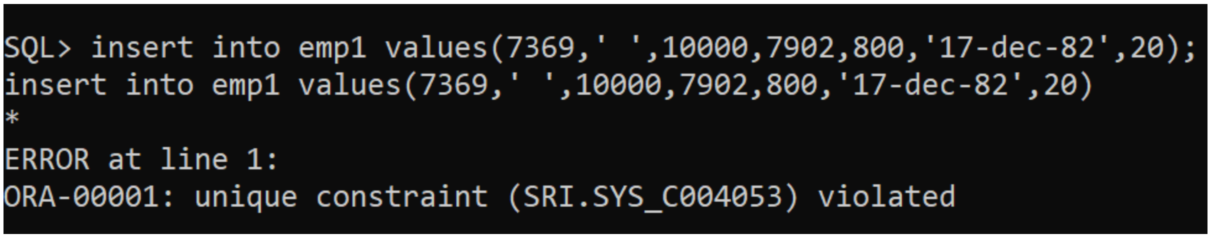
Alter table table_name modify attribute domain type NOT NULL;

Example :

Alter table emp modify attribute varchar2(20) NOT NULL;



```
SQL> Alter table emp1 modify ename varchar2(20) NOT NULL;

Table altered.

SQL> desc emp1;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                     NOT NULL NUMBER(38)
 ENAME                                     NOT NULL VARCHAR2(20)
 SALARY                                             NUMBER(10)
 MGR                                                FLOAT(63)
 COMM                                               FLOAT(63)
 HIREDATE                                           DATE
 DEPTNO                                             NUMBER(38)

SQL> insert into emp1 values(7369,' ',10000,7902,800,'17-dec-82',20);
insert into emp1 values(7369,' ',10000,7902,800,'17-dec-82',20)
*
ERROR at line 1:
ORA-02291: integrity constraint (SRI.SYS_C004047) violated - parent key not
found
```



```
SQL> insert into emp1 values(7369,' ',10000,7902,800,'17-dec-82',20);
insert into emp1 values(7369,' ',10000,7902,800,'17-dec-82',20)
*
ERROR at line 1:
ORA-00001: unique constraint (SRI.SYS_C004053) violated
```

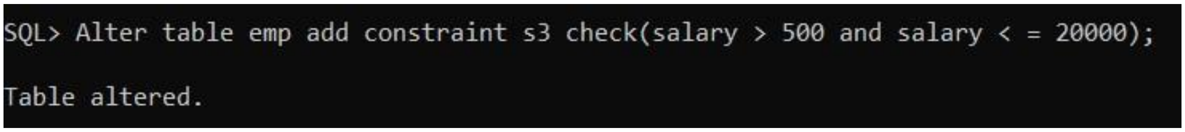
5. Check : Ensures that the value in a column meets a specific condition.

Syntax :

Alter table table_name add constraint constraint_name check(condition);

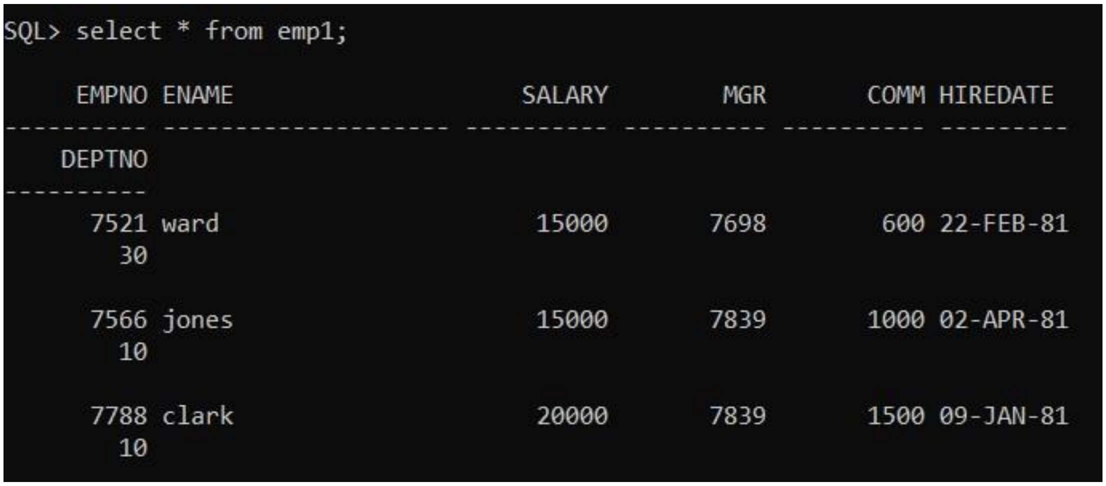
Example :

Alter table emp add constraint s3 check(salary > 500 and salary < = 20000);



```
SQL> Alter table emp add constraint s3 check(salary > 500 and salary < = 20000);

Table altered.
```

Select * from emp;



```
SQL> select * from emp1;

     EMPNO ENAME                    SALARY        MGR       COMM HIREDATE
---------- -------------------- ---------- ---------- ---------- ---------
    DEPTNO
----------
      7521 ward                      15000       7698        600 22-FEB-81
        30

      7566 jones                     15000       7839       1000 02-APR-81
        10

      7788 clark                     20000       7839       1500 09-JAN-81
        10
```

bEXERCISE :2a

AIM : Execute Set opersations on various Relations

• UNION • UNION ALL • INTERSECT

• MINUS.

Description :

Set operations in sql:

UNION :

Let R and S are two union compatible relations then, union operation returns the tuples that are present in R or s or both.

- **Two relational instances are said to be union compatible if the following conditions are hold.**

1. **They have the same number of columns.**
2. **Corresponding columns taken in order from left to right have same data type.**

1. **Find the names of sailors who have reserved red or green boat.99**

Query

```
select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'red'
UNION
select  s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color =
'green';
```

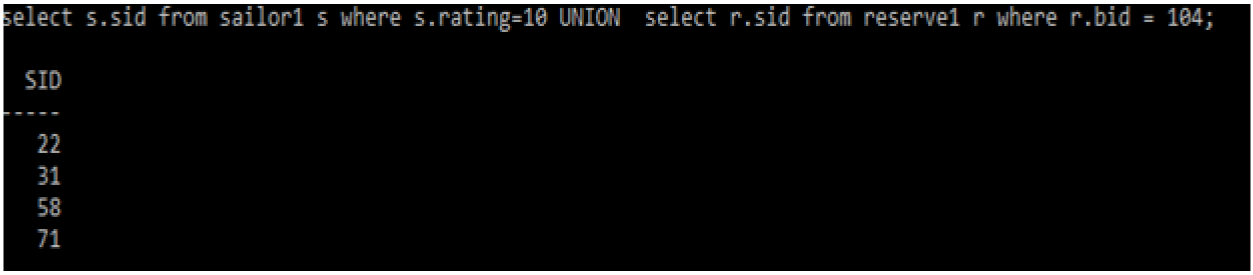
Output

2. **Find all sid's of sailors who have rating of 10 or reserved boat no.104.**

Query :

```
 select s.sid from sailor1 s where s.rating=10
UNION

 select r.sid from reserve1 r where r.bid = 104;
```

Output



```
select s.sid from sailor1 s where s.rating=10 UNION  select r.sid from reserve1 r where r.bid = 104;

  SID
-----
   22
   31
   58
   71
```

Us

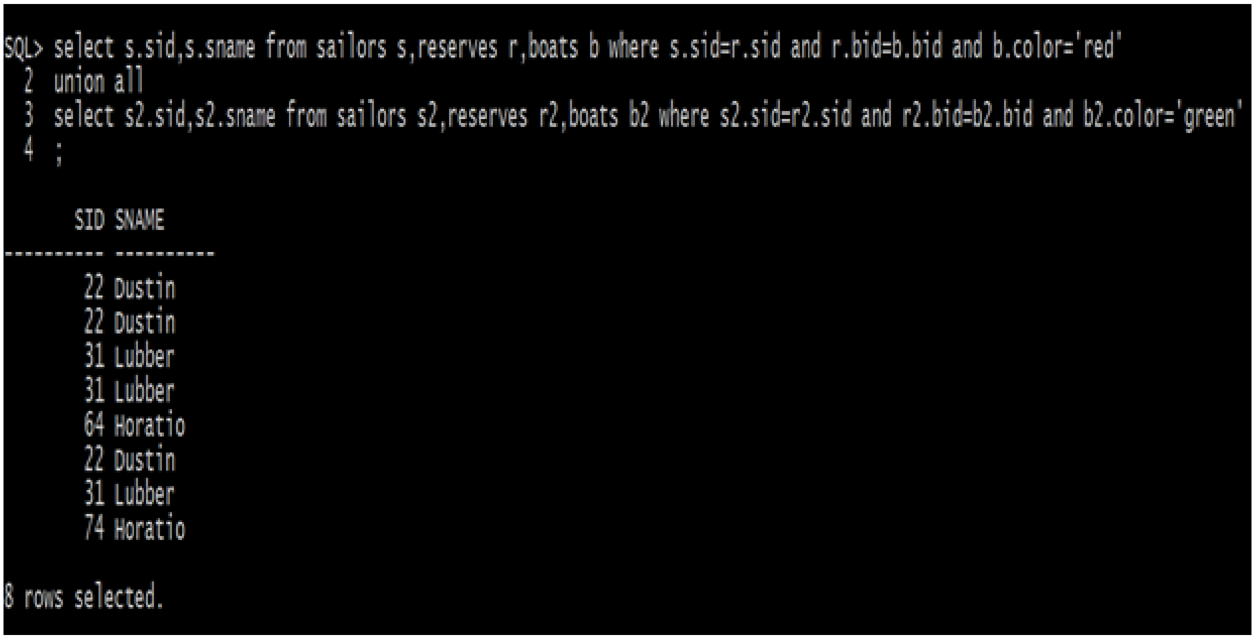
2

UNION ALL :

The UNION ALL command combines the result set of two or more SELECT statements (allows duplicate values).

Syntax :

Example :



```
SQL> select s.sid,s.sname from sailors s,reserves r,boats b where s.sid=r.sid and r.bid=b.bid and b.color='red'
  2  union all
  3  select s2.sid,s2.sname from sailors s2,reserves r2,boats b2 where s2.sid=r2.sid and r2.bid=b2.bid and b2.color='green'
  4  ;

       SID SNAME
---------- ----------
        22 Dustin
        22 Dustin
        31 Lubber
        31 Lubber
        64 Horatio
        22 Dustin
        31 Lubber
        74 Horatio

8 rows selected.
```

22

INTERSECT :

Let R and S are two union compatible relations then, intersect operation returns the tuples that are common in both the relations.

1. **Find the names of sailors who have reserved red and green boat.**

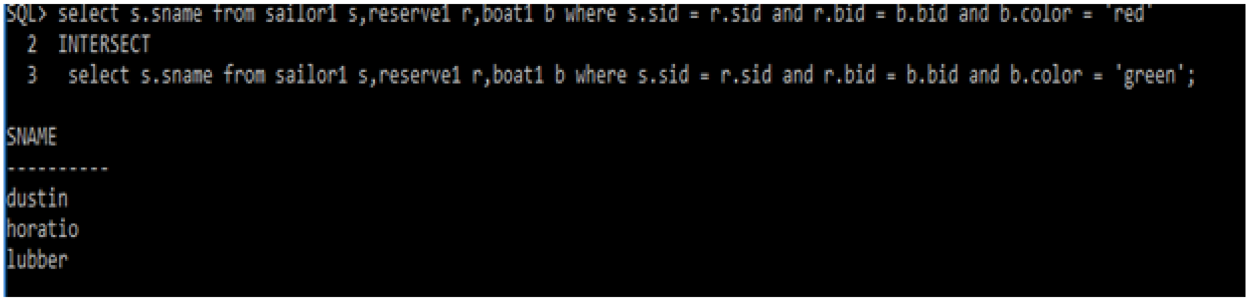
Query

select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'red'

INTERSECT

select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'green';

Output



```
SQL> select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'red'
  2  INTERSECT
  3   select s.sname from sailor1 s,reserve1 r,boat1 b where s.sid = r.sid and r.bid = b.bid and b.color = 'green';

SNAME
----------
dustin
horatio
lubber
```

MINUS :

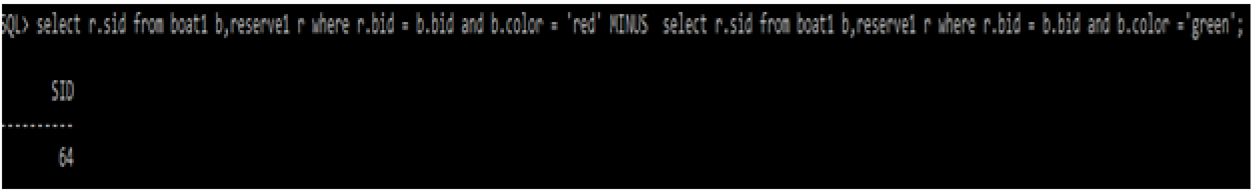
Let R and S are two union compatible relations then, intersect operation returns the tuples that are present in R but not in S.

1. **Find the sid's of sailors who have reserved red but not green boat.**

Query

select r.sid from boats b,reserves r where r.bid=b.bid and b.color='red' minus select r.sid from boats b,reserves r where r.bid=b.bid and b.color='green';

Output:



```
SQL> select r.sid from boat1 b,reserve1 r where r.bid = b.bid and b.color = 'red' MINUS  select r.sid from boat1 b,reserve1 r where r.bid = b.bid and b.color ='green';

       SID
----------
        64
```

VIVA QUESTION:

1. List various set Operations

2. Differentiate UNION and UNION ALL

EXERCISE : 2b

AIM : Execute Sub Queries and Co-Related Nested Queries on Relations.

• Implement

o Single-row subquery o Multiple-row subquery

• Using Group Functions in a Subquery

• Using HAVING Clause with Subqueries

• Using Null Values in a Subquery

• Data retrieval using Correlated Subqueries

o EXISTS Operator o NOT EXISTS Operator .

Description :

NESTED QUERIES

A query embedded inside another query is called a sub query. Inner query executes initially only once and that result will be used by all the tuples of outer query.

Co-Related nested queries: Correlated subquery is a query in which the inner query is executed for each row of the outer query.

Implement :

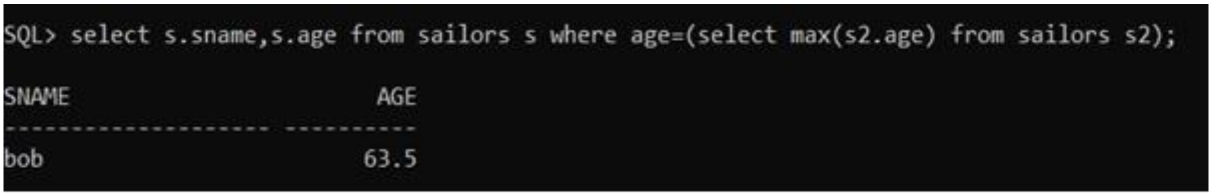
- **Single row subquery:**

A single row subquery returns zero or one row to the outer SQL statement. You can place a subquery in a WHERE clause, a HAVING clause, or a FROM clause of a SELECT statement.

EXAMPLE :

1. **Find the name and age of the oldest sailor.**

select s.sname,s.age from sailors s where age=(select max(s2.age) from sailors s2);



```
SQL> select s.sname,s.age from sailors s where age=(select max(s2.age) from sailors s2);

SNAME                       AGE
-------------------- ----------
bob                        63.5
```

- **Multiple row subquery :**

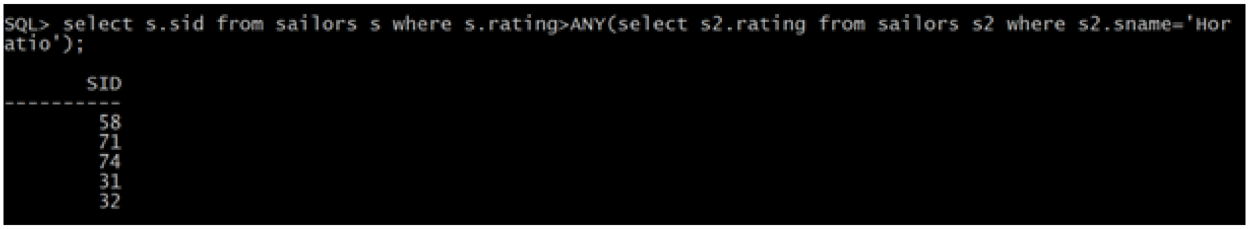
- **Multiple-row subqueries are nested queries that can return more than one row of results to the parent query. Multiple-row subqueries are used most commonly in WHERE and HAVING clauses.**
- **Since it returns multiple rows, it must be handled by set comparison operators (IN, ALL, ANY).**

- **While IN operator holds the same meaning as discussed in the earlier chapter, ANY operator compares a specified value to each value returned by the subquery while ALL compares a value to every value returned by a subquery.**
- **The below query will show the error because single-row subquery returns multiple rows**

EXAMPLE :

1. **Find sailors whose rating is better than some sailor called Horatio**

select s.sid from sailors s where s.rating select s.sid from sailors s where s.rating s2.sname='Ho



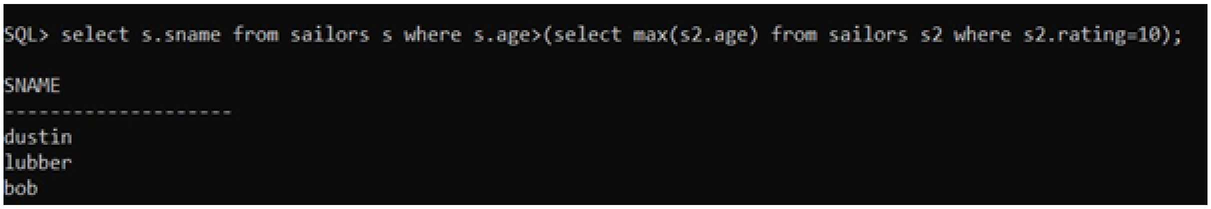
```
SQL> select s.sid from sailors s where s.rating>ANY(select s2.rating from sailors s2 where s2.sname='Hor
atio');

       SID
----------
        58
        71
        74
        31
        32
```

Using Group functions in a subqueries :

EXAMPLE :

1. **Find the names of sailors who are older than oldest sailor with a rating of 10.**

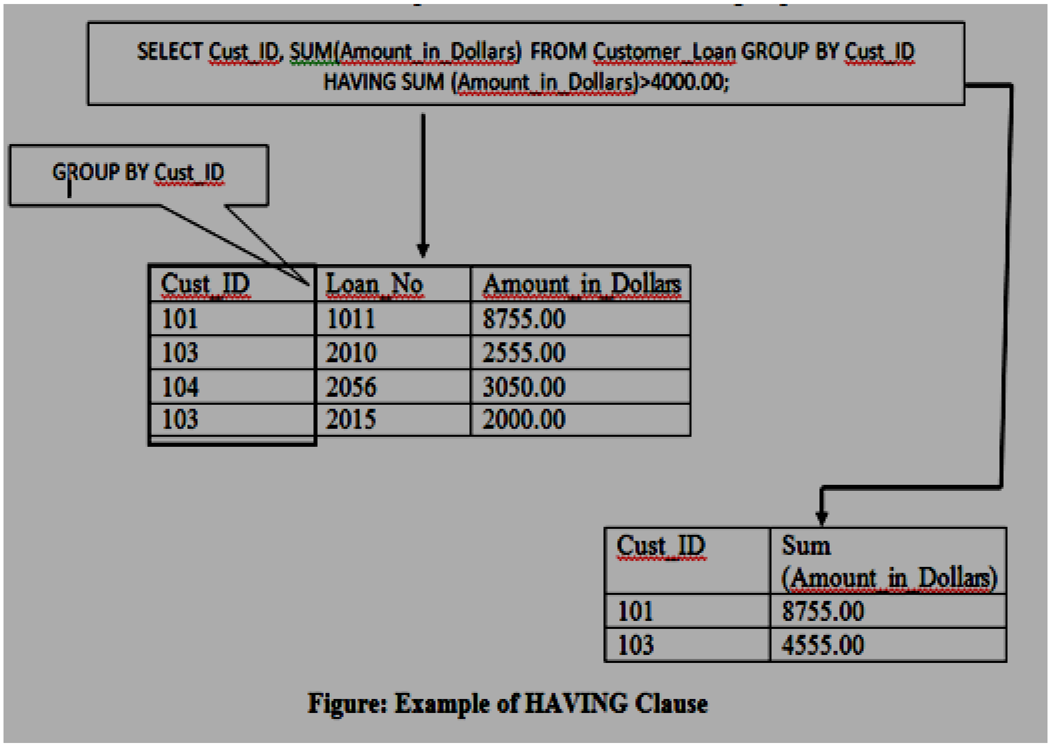
select s.sname from sailors s where s.age>(select max(s2.age) from sailors s2 where s2.rating=10);



```
SQL> select s.sname from sailors s where s.age>(select max(s2.age) from sailors s2 where s2.rating=10);

SNAME
--------------------
dustin
lubber
bob
```

Using HAVING clauses with subqueries :

- **The HAVING clause is used along with the GROUP BY clause. The HAVING clause can be used to select and reject row groups.**
- **The format of the HAVING clause is similar to the WHERE clause, consisting of the keyword HAVING followed by a search condition.**
- **The HAVING clause thus specifies a search condition for groups.**



```
SELECT Cust_ID, SUM(Amount_in_Dollars) FROM Customer_Loan GROUP BY Cust_ID
HAVING SUM (Amount_in_Dollars)>4000.00;
```

GROUP BY Cust_ID

Cust_ID	Loan_No	Amount_in_Dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

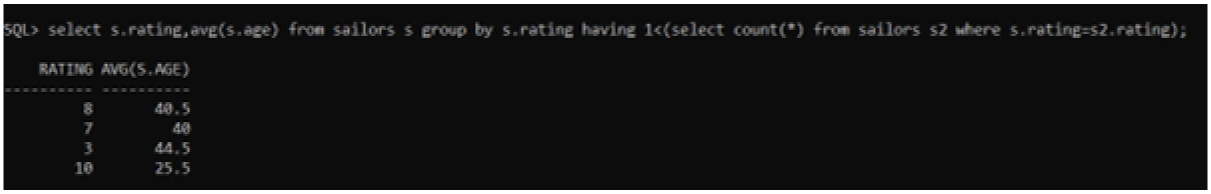
Cust_ID	Sum (Amount_in_Dollars)
101	8755.00
103	4555.00

Figure: Example of HAVING Clause

EXAMPLE :

1. **Find the average age of sailors for each rating level that has at least two sailors.**

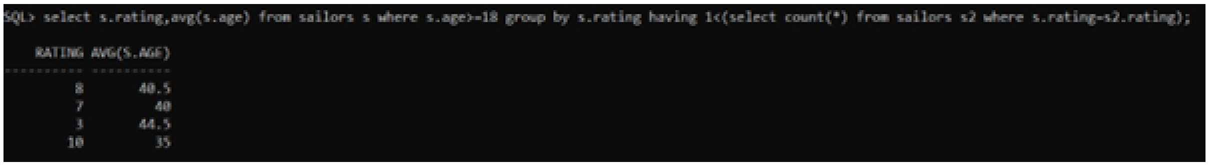
select s.rating,avg(s.age) from sailors s group by s.rating having 1 <(select count(*) from sailors s1 where s.rating=s1.rating);



```
SQL> select s.rating,avg(s.age) from sailors s group by s.rating having 1<(select count(*) from sailors s2 where s.rating=s2.rating);

    RATING AVG(S.AGE)
---------- ----------
         8       40.5
         7         40
         3       44.5
        10       25.5
```

2. **Find the average age of sailors who are of voting age (i.e., at least 18 years old) for each rating level that has at least two sailors.**

select s.rating,avg(s.age) from sailors s where s.age>=18 group by s.rating having 1<(select count(*) from sailors s2 where s.rating=s2.rating);



```
SQL> select s.rating,avg(s.age) from sailors s where s.age>=18 group by s.rating having 1<(select count(*) from sailors s2 where s.rating=s2.rating);

    RATING AVG(S.AGE)
---------- ----------
         8       40.5
         7         40
         3       44.5
        10         35
```

Using NULL values in a subquery :

A field with a NULL value is a field with no value.

Syntax :

a. IS NULL :

Syntax

SELECT column_names

FROM table_name

WHERE column_name IS NULL;

IS NOT NULL :

Syntax

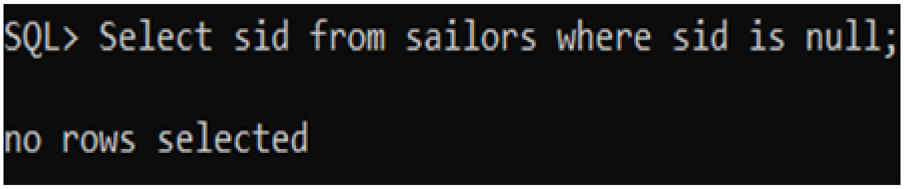
SELECT column_names

FROM table_name

WHERE column_name IS NOT NULL;

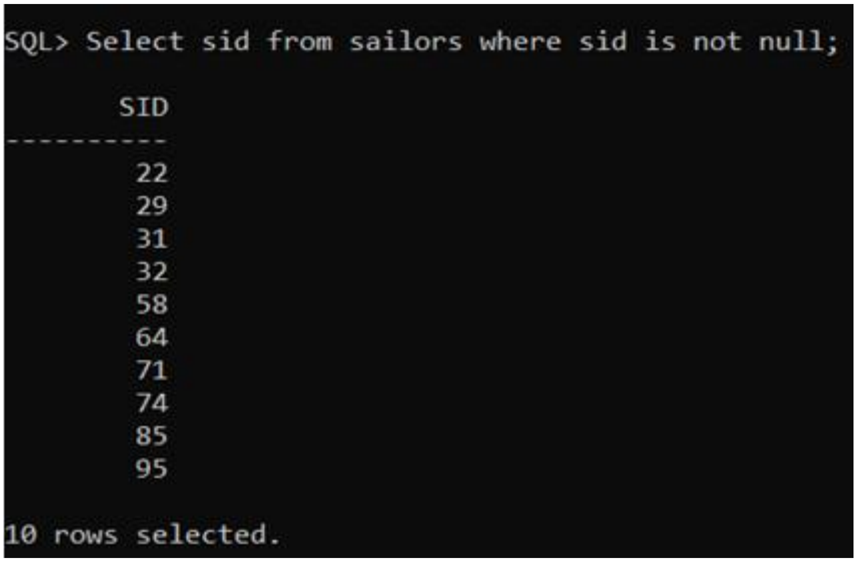
Example :

Select sid from sailors where sid is null;



```
SQL> Select sid from sailors where sid is null;

no rows selected
```

Select sid from sailors where sid is not null;



```
SQL> Select sid from sailors where sid is not null;

       SID
----------
        22
        29
        31
        32
        58
        64
        71
        74
        85
        95

10 rows selected.
```

Data retrieval using correlated sub queries:

- **EXISTS operator:**

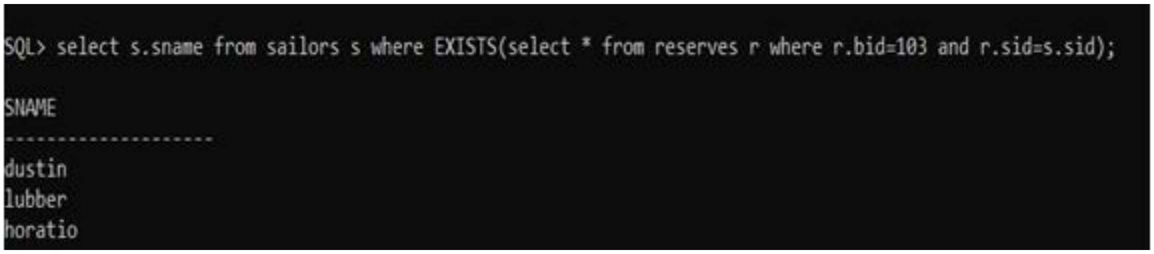
The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

Example:

1.Find the names of sailors who have reserved boat number 103.

select s.sname from sailors s where EXISTS(select * from reserves r where r.bid=103 and r.sid=s.sid);



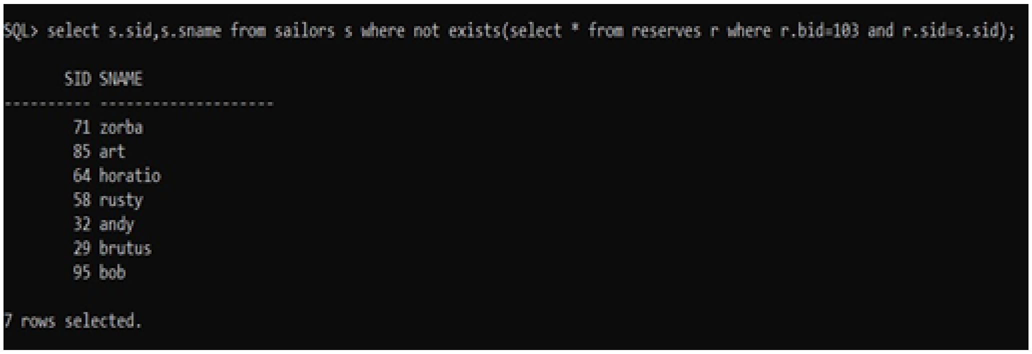
```
SQL> select s.sname from sailors s where EXISTS(select * from reserves r where r.bid=103 and r.sid=s.sid);

SNAME
--------------------
dustin
lubber
horatio
```

- **NOT EXISTS operator : Negated version of EXISTS**

Example:

1. Find the sid’s and names of sailors who have not reserved boat number 103.

select s.sid,s.sname from sailors s where not exists(select * from reserves r where r.bid=103 and r.sid=s.sid);



```
SQL> select s.sid,s.sname from sailors s where not exists(select * from reserves r where r.bid=103 and r.sid=s.sid);

       SID SNAME
---------- --------------------
        71 zorba
        85 art
        64 horatio
        58 rusty
        32 andy
        29 brutus
        95 bob

7 rows selected.
```

EXERCISE : 3a

QUERIES USING AGGREGATE FUNCTIC

(COUNT, SUM, AVG, MAX AND MIN) **GROUP BY and HAVING.**

AIM : Execute the following Multiple row functions (Aggregate Functions) on Relation

- **Group functions(AVG, COUNT, MAX, MIN, SUM)**
- **DISTINCT Keyword in Count Function**
- **Null Values in Group Functions**
- **NVL Function with Group Functions.**

Description :

AGGREGATE FUNCTIONS

In data base management system ,an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

The aggregate functions are:

1) MAX(): It returns the max value in the given column.

2) MIN(): It returns the max value in the given column.

3) SUM(): It returns the sum of all numeric values in the given column.

4) AVG(): It returns the average of all values in the given column.

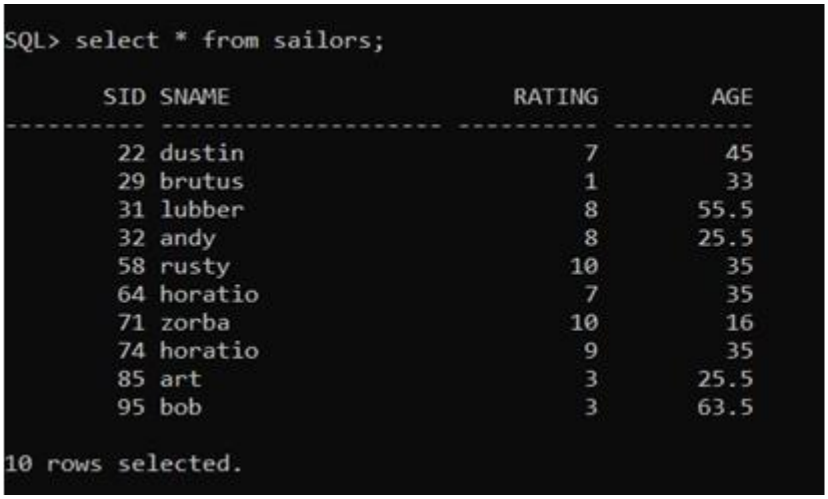
5) COUNT():It returns the total number of all values in the given column(excluding null values).

6) COUNT(*):It returns the number of all rows in the given table(including null values).

Group Functions(AVG , COUNT , MAX , MIN , SUM) :

Example :

```
SELECT * FROM sailors;
```

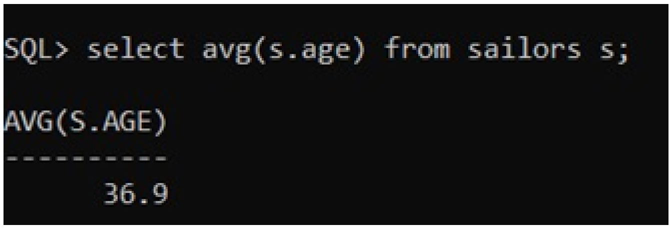


```
SQL> select * from sailors;

       SID SNAME                    RATING        AGE
---------- -------------------- ---------- ----------
        22 dustin                        7         45
        29 brutus                        1         33
        31 lubber                        8       55.5
        32 andy                          8       25.5
        58 rusty                        10         35
        64 horatio                       7         35
        71 zorba                        10         16
        74 horatio                       9         35
        85 art                           3       25.5
        95 bob                           3       63.5

10 rows selected.
```

AVERAGE (AVG):

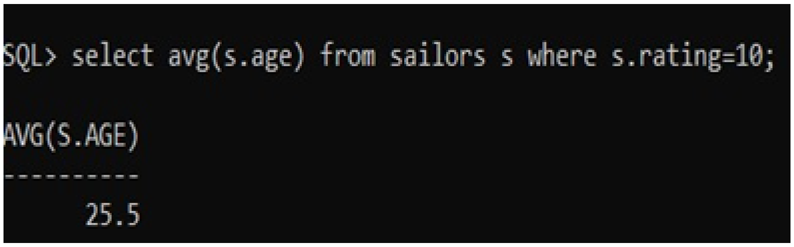
Example :

select avg(s.age) from sailors s;



```
SQL> select avg(s.age) from sailors s;

AVG(S.AGE)
----------
      36.9
```

select avg(s.age) from sailors s where s.rating=10;

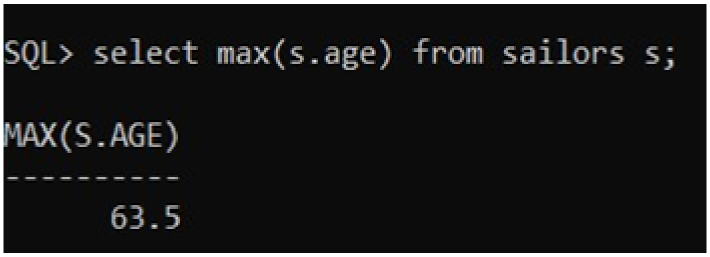


```
SQL> select avg(s.age) from sailors s where s.rating=10;

AVG(S.AGE)
----------
      25.5
```

MAXIMUN (MAX):

Example:

select max(s.age) from sailors s;

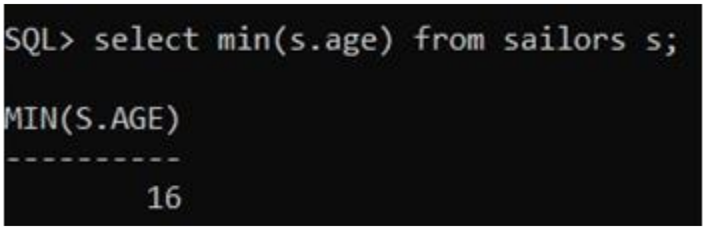


```
SQL> select max(s.age) from sailors s;

MAX(S.AGE)
----------
      63.5
```

MINIMUM (MIN):

Example:

select min(s.age) from sailors s;

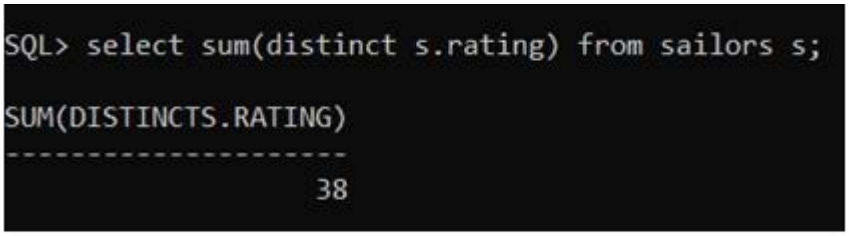


```
SQL> select min(s.age) from sailors s;

MIN(S.AGE)
----------
        16
```

SUM:

Example:

select sum(distinct s.rating) from sailors s;

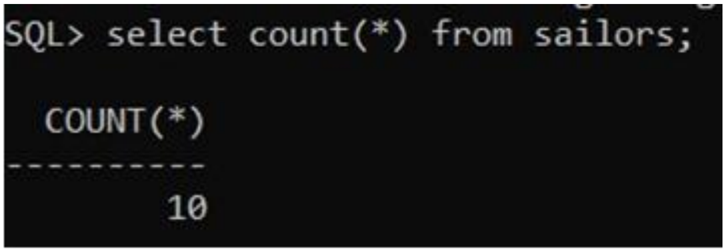


```
SQL> select sum(distinct s.rating) from sailors s;

SUM(DISTINCTS.RATING)
---------------------
                   38
```

COUNT:

Example:

select count(*) from sailors;

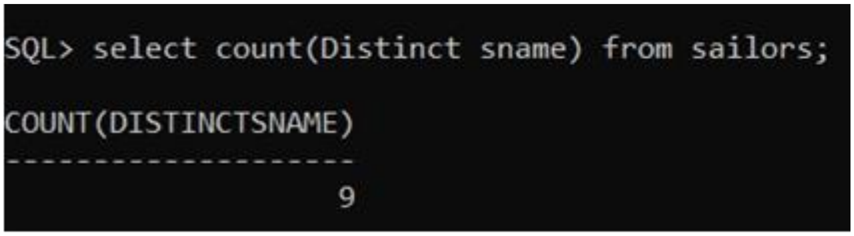


```
SQL> select count(*) from sailors;

  COUNT(*)
----------
        10
```

DISTINCT keyword in count function :

EXAMPLE :

select count(Distinct sname) from sailors;



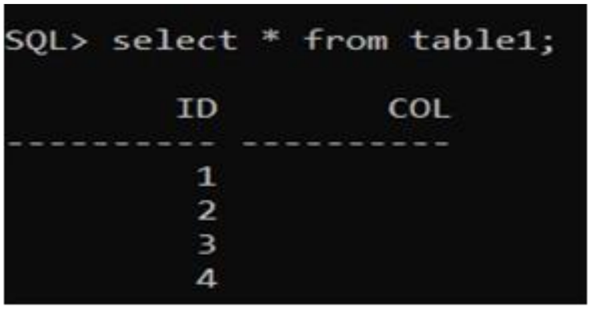
```
SQL> select count(Distinct sname) from sailors;

COUNT(DISTINCTSNAME)
--------------------
                   9
```

Null values in group functions :

EXAMPLE :

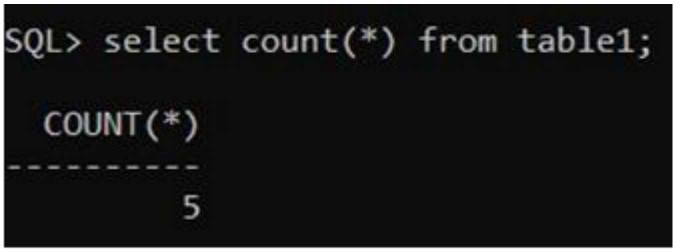
Create table table1 (id int, col int);

Select * from table1;



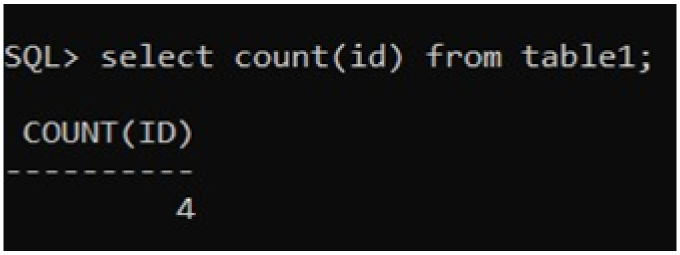
```
SQL> select * from table1;

        ID        COL
---------- ----------
         1
         2
         3
         4
```

select count(*) from table1;



```
SQL> select count(*) from table1;

  COUNT(*)
----------
         5
```

select count(id) from table1;



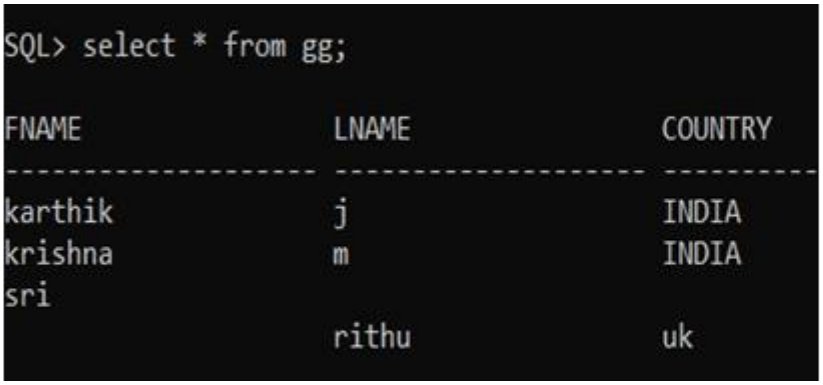
```
SQL> select count(id) from table1;

 COUNT(ID)
----------
         4
```

NVL function with group functions :

EXAMPLE :

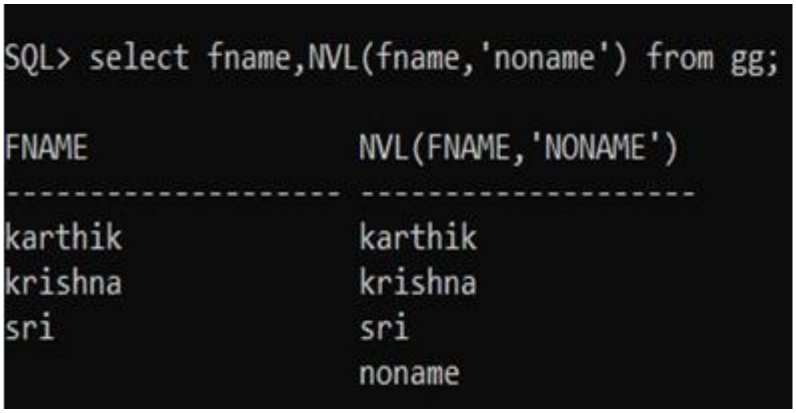
create table gg(fname varchar2(20),lname varchar2(20),country varchar2(10));

SELECT * FROM GG;



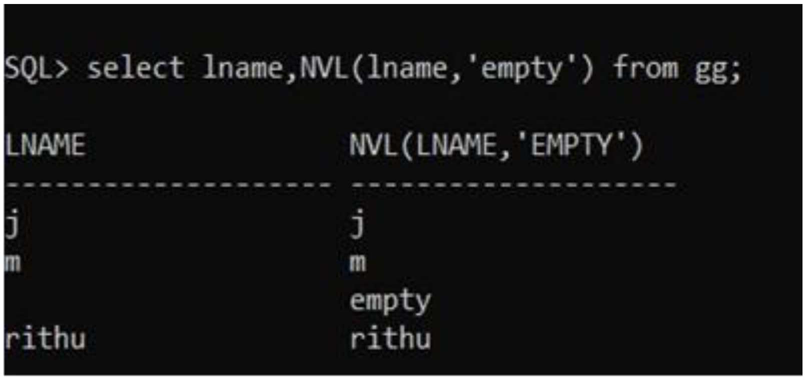
```
SQL> select * from gg;

FNAME                LNAME                COUNTRY
-------------------- -------------------- ----------
karthik              j                    INDIA
krishna              m                    INDIA
sri
                     rithu                uk
```

select fname,NVL(fname,'noname') from gg;



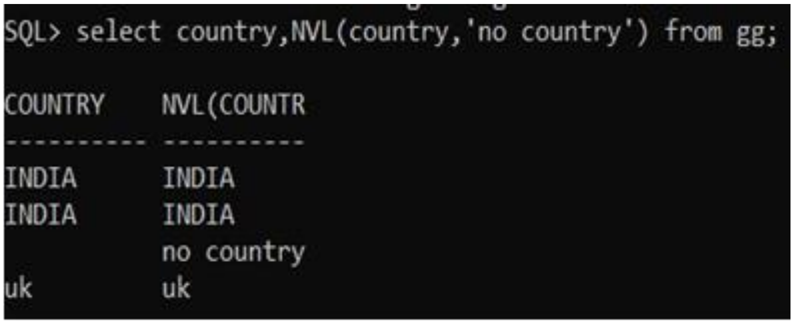
```
SQL> select fname,NVL(fname,'noname') from gg;

FNAME                NVL(FNAME,'NONAME')
-------------------- --------------------
karthik              karthik
krishna              krishna
sri                  sri
                     noname
```

select lname,NVL(lname,'empty') from gg;



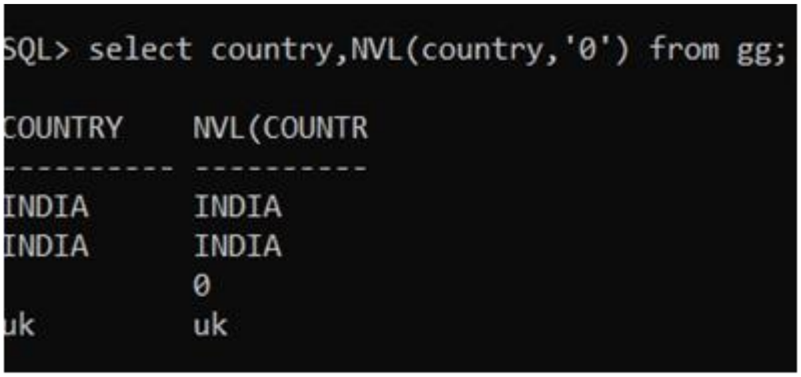
```
SQL> select lname,NVL(lname,'empty') from gg;

LNAME                NVL(LNAME,'EMPTY')
-------------------- --------------------
j                    j
m                    m
                     empty
rithu                rithu
```

select country,NVL(country,'no country') from gg;



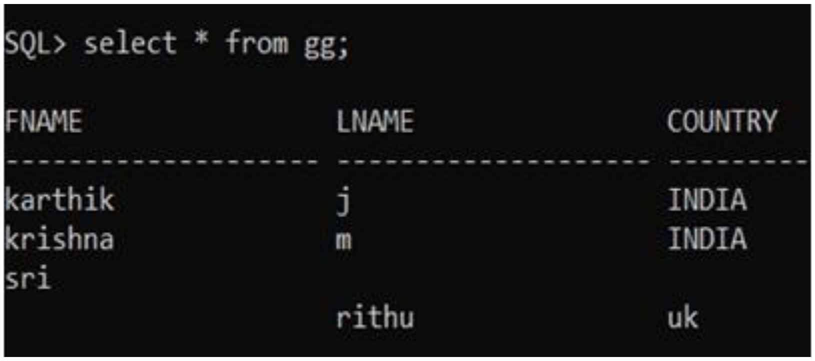
```
SQL> select country,NVL(country,'no country') from gg;

COUNTRY    NVL(COUNTR
---------- ----------
INDIA      INDIA
INDIA      INDIA
           no country
uk         uk
```

select country,NVL(country,'0') from gg;



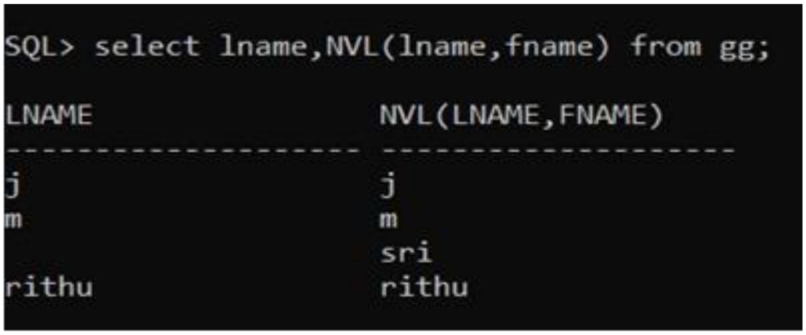
```
SQL> select country,NVL(country,'0') from gg;

COUNTRY    NVL(COUNTR
---------- ----------
INDIA      INDIA
INDIA      INDIA
           0
uk         uk
```

select * from gg;



```
SQL> select * from gg;

FNAME                LNAME                COUNTRY
-------------------- -------------------- ---------
karthik              j                    INDIA
krishna              m                    INDIA
sri
                     rithu                uk
```

select lname,NVL(lname,fname) from gg;



```
SQL> select lname,NVL(lname,fname) from gg;

LNAME                NVL(LNAME,FNAME)
-------------------- --------------------
j                    j
m                    m
                     sri
rithu                rithu
```

EXERCISE : 3b

AIM : Create Groups of Data using Group By clause

• Grouping by One Column

• Grouping by More Than One Column

• Illegal Queries Using Group Functions

• Restricting groups using HAVING Clause

• Nesting Group Functions.

Description :

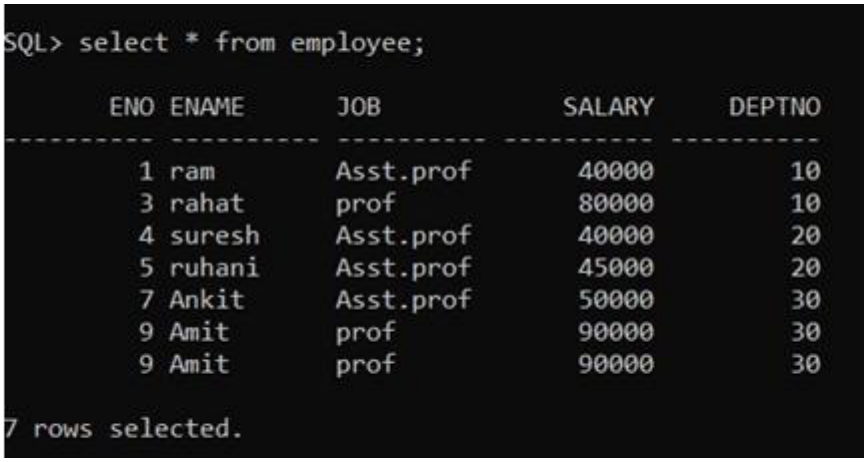
GROUP by clause :

- **The GROUP BY clause is used in a SELECT statement to collect data across multiple records and group the results by one or more columns.**
- **Sometimes it is required to get information not about each row, but about each group.**
- **Related rows can be grouped together by the GROUP BY clause by specifying a column as a grouping column.**
- **In the output table all the rows with an identical value in the grouping column will be grouped together. Hence, the number of rows in the output is equal to the number of distinct values of the grouping column.**

Grouping by More Than One Column

EXAMPLE :

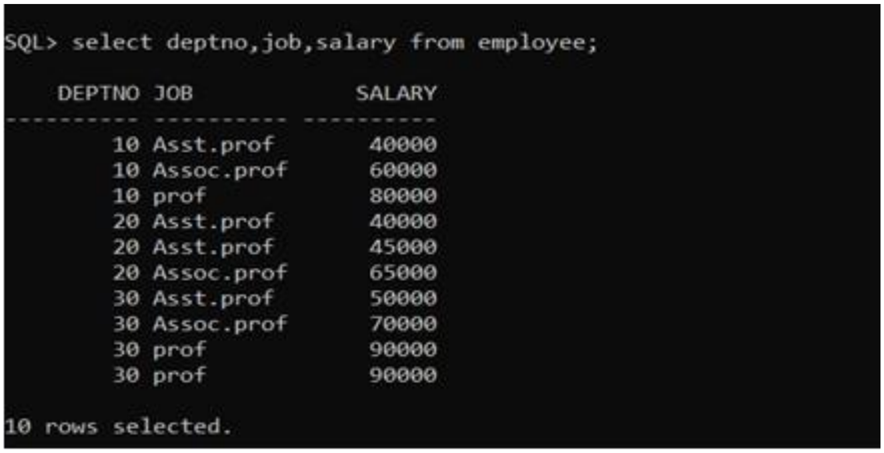
```
create table employee(eno int,ename varchar2(10),job varchar2(10),salary int,deptno int);
select * from employee;
```



```
SQL> select * from employee;

       ENO ENAME      JOB            SALARY     DEPTNO
---------- ---------- ---------- ---------- ----------
         1 ram        Asst.prof       40000         10
         3 rahat      prof            80000         10
         4 suresh     Asst.prof       40000         20
         5 ruhani     Asst.prof       45000         20
         7 Ankit      Asst.prof       50000         30
         9 Amit       prof            90000         30
         9 Amit       prof            90000         30

7 rows selected.
```

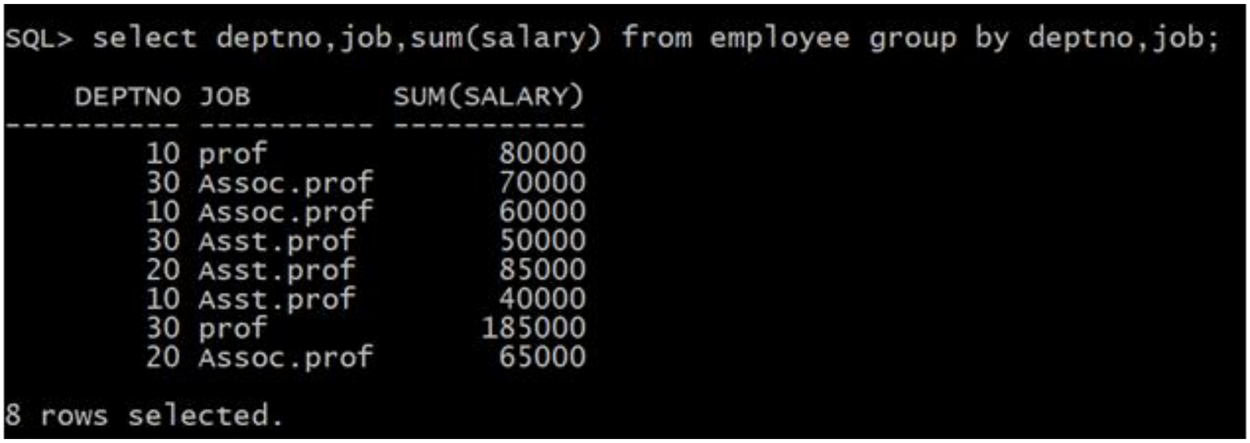
Query:Total salary paid to each job in each department

select Deptno,job,salary from employee;



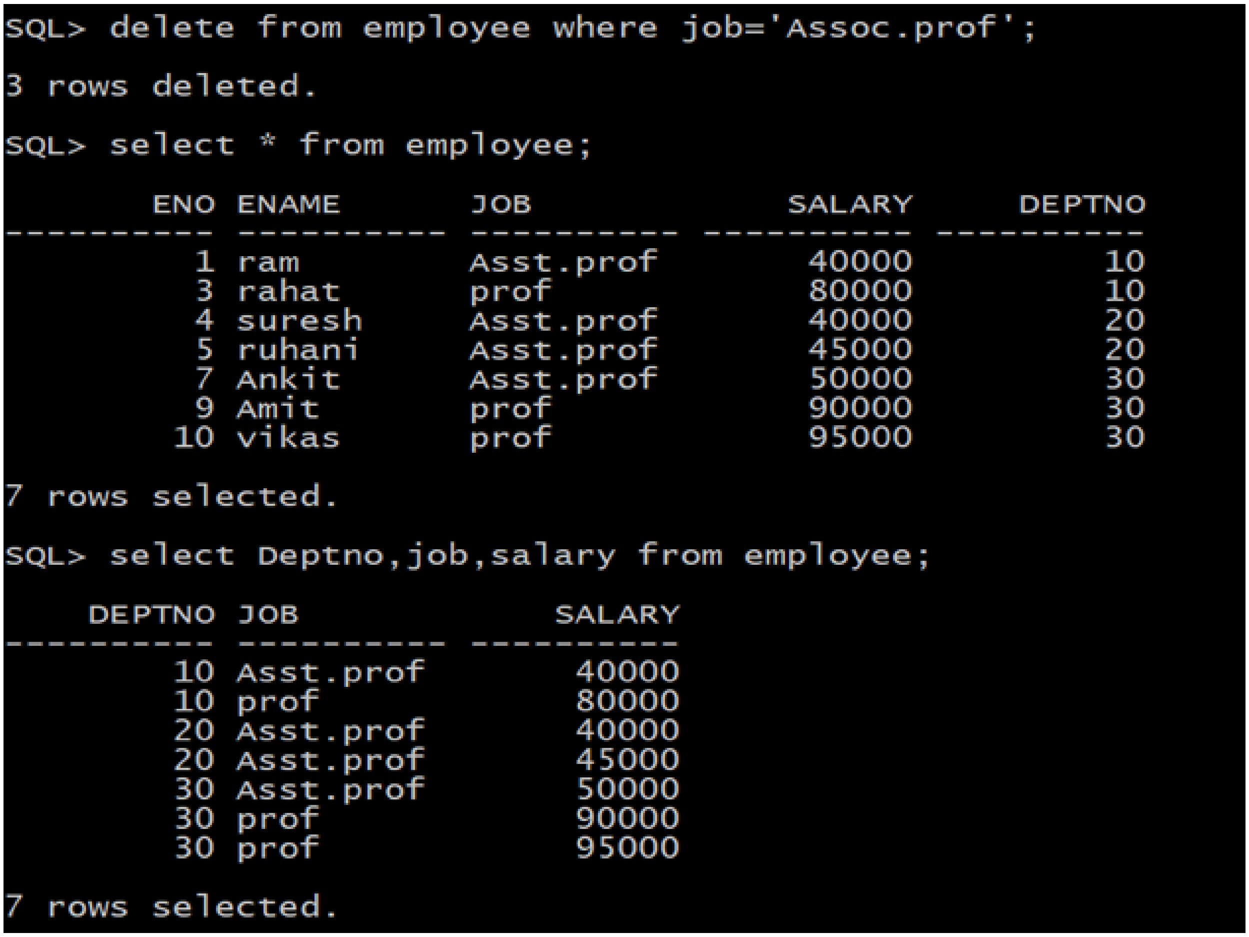
```
SQL> select deptno,job,salary from employee;

    DEPTNO JOB           SALARY
---------- --------- ----------
        10 Asst.prof      40000
        10 Assoc.prof     60000
        10 prof           80000
        20 Asst.prof      40000
        20 Asst.prof      45000
        20 Assoc.prof     65000
        30 Asst.prof      50000
        30 Assoc.prof     70000
        30 prof           90000
        30 prof           90000

10 rows selected.
```

select deptno,job,sum(salary) Total_Salary from employee group by deptno,job;



```
SQL> select deptno,job,sum(salary) from employee group by deptno,job;

    DEPTNO JOB        SUM(SALARY)
---------- ---------- -----------
        10 prof             80000
        30 Assoc.prof       70000
        10 Assoc.prof       60000
        30 Asst.prof        50000
        20 Asst.prof        85000
        10 Asst.prof        40000
        30 prof            185000
        20 Assoc.prof       65000

8 rows selected.
```

Query: Total Salary paid to each job in each department excluding Assoc Prof

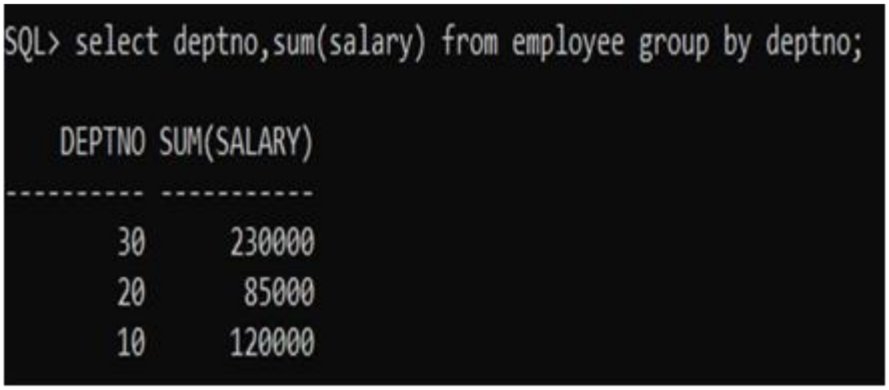


```
SQL> delete from employee where job='Assoc.prof';

3 rows deleted.

SQL> select * from employee;

       ENO ENAME      JOB            SALARY     DEPTNO
---------- ---------- ---------- ---------- ----------
         1 ram        Asst.prof       40000         10
         3 rahat      prof            80000         10
         4 suresh     Asst.prof       40000         20
         5 ruhani     Asst.prof       45000         20
         7 Ankit      Asst.prof       50000         30
         9 Amit       prof            90000         30
        10 vikas      prof            95000         30

7 rows selected.

SQL> select Deptno,job,salary from employee;

    DEPTNO JOB            SALARY
---------- ---------- ----------
        10 Asst.prof       40000
        10 prof            80000
        20 Asst.prof       40000
        20 Asst.prof       45000
        30 Asst.prof       50000
        30 prof            90000
        30 prof            95000

7 rows selected.
```

Grouping by one column :

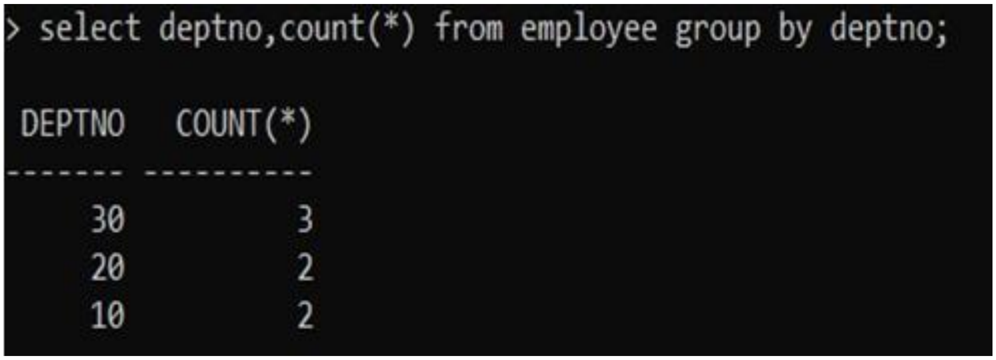
EXAMPLE :

select deptno,sum(salary) from employee group by deptno;



```
SQL> select deptno,sum(salary) from employee group by deptno;

    DEPTNO SUM(SALARY)
---------- -----------
        30      230000
        20       85000
        10      120000
```

select deptno,count(*) from employee group by deptno;

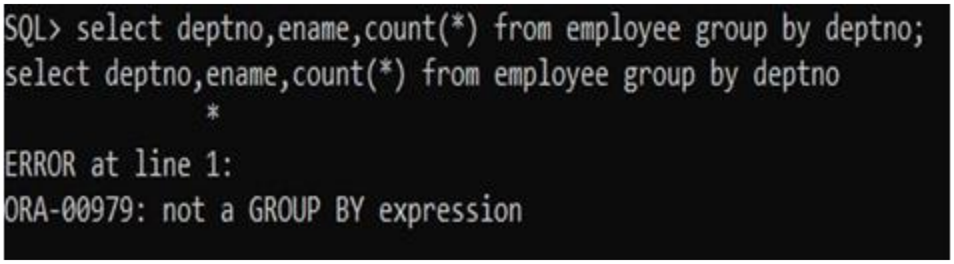


```
> select deptno,count(*) from employee group by deptno;

 DEPTNO   COUNT(*)
------- ----------
     30          3
     20          2
     10          2
```

Illegal Queries using Group functions :

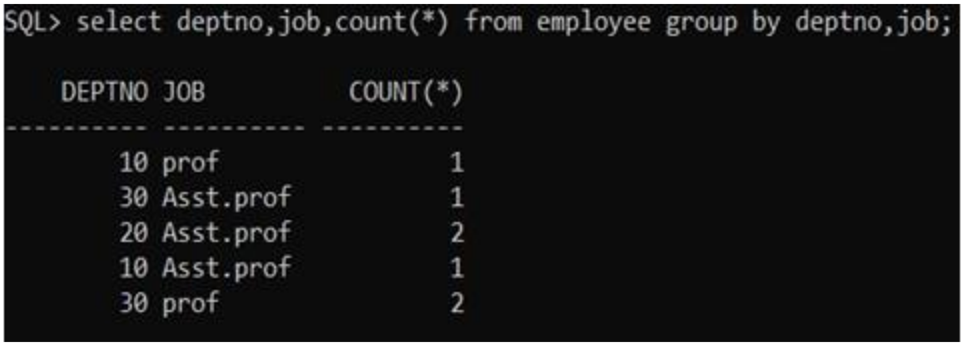
EXAMPLE :

select deptno,ename,count(*) from employee group by deptno;



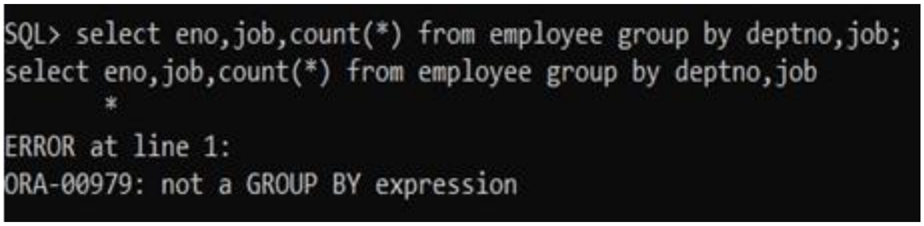
```
SQL> select deptno,ename,count(*) from employee group by deptno;
select deptno,ename,count(*) from employee group by deptno
              *
ERROR at line 1:
ORA-00979: not a GROUP BY expression
```

select deptno,job,count(*) from employee group by deptno,job;



```
SQL> select deptno,job,count(*) from employee group by deptno,job;

    DEPTNO JOB          COUNT(*)
---------- ---------- ----------
        10 prof                1
        30 Asst.prof           1
        20 Asst.prof           2
        10 Asst.prof           1
        30 prof                2
```

select eno,job,count(*) from employee group by deptno,job;

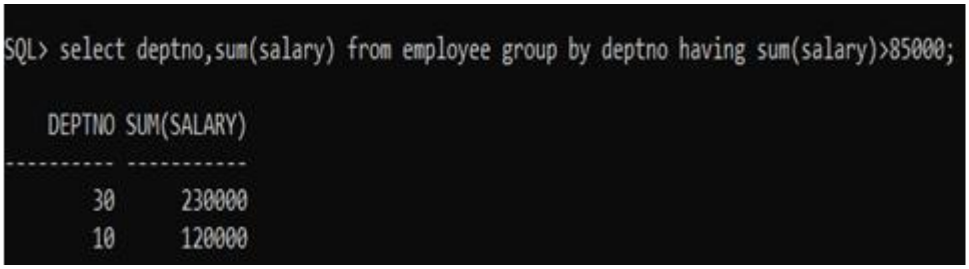


```
SQL> select eno,job,count(*) from employee group by deptno,job;
select eno,job,count(*) from employee group by deptno,job
       *
ERROR at line 1:
ORA-00979: not a GROUP BY expression
```

Restricting groups using HAVING clause :

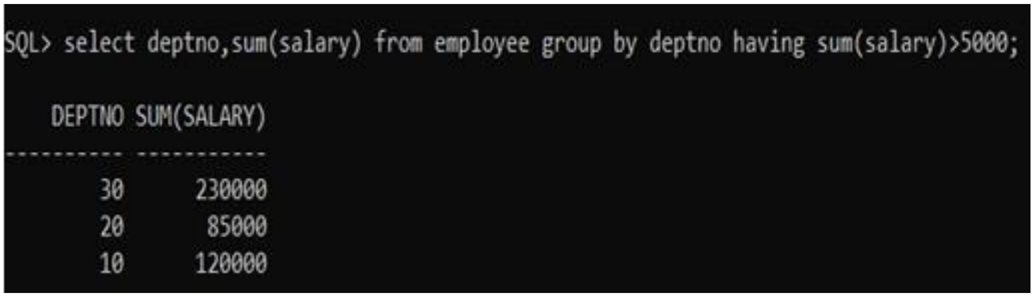
EXAMPLE :

select deptno,sum(salary) from employee group by deptno having sum(salary)>85000;



```
SQL> select deptno,sum(salary) from employee group by deptno having sum(salary)>85000;

    DEPTNO SUM(SALARY)
---------- ----------
        30     230000
        10     120000
```

select deptno,sum(salary) from employee group by deptno having sum(salary)>5000;

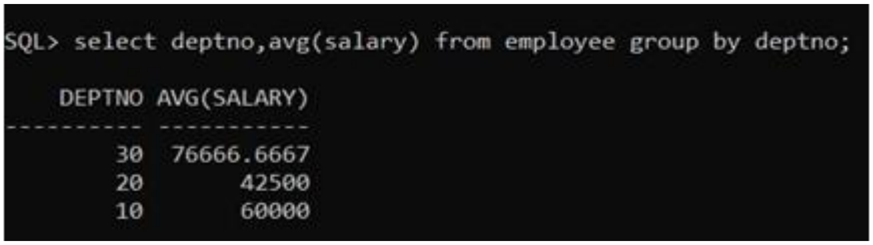


```
SQL> select deptno,sum(salary) from employee group by deptno having sum(salary)>5000;

    DEPTNO SUM(SALARY)
---------- ----------
        30     230000
        20      85000
        10     120000
```

Nesting group functions :

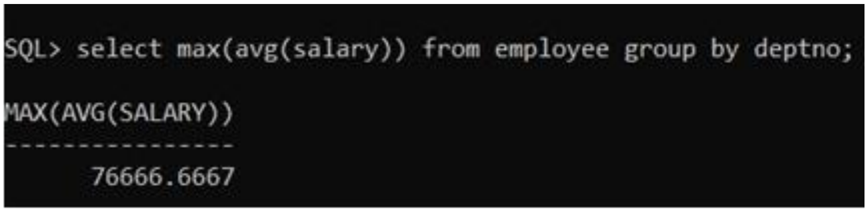
EXAMPLE :

select deptno,avg(slary) from employee group by deptno;



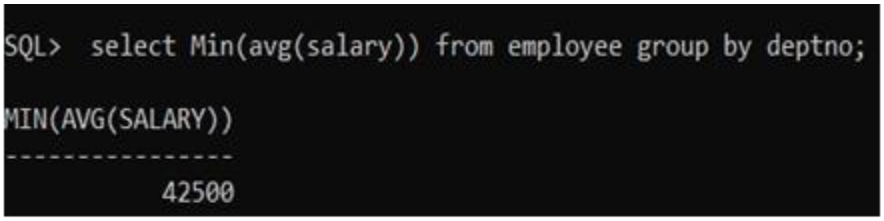
```
SQL> select deptno,avg(salary) from employee group by deptno;

    DEPTNO AVG(SALARY)
---------- -----------
        30  76666.6667
        20       42500
        10       60000
```

select max(avg(salary)) from employee group by deptno;



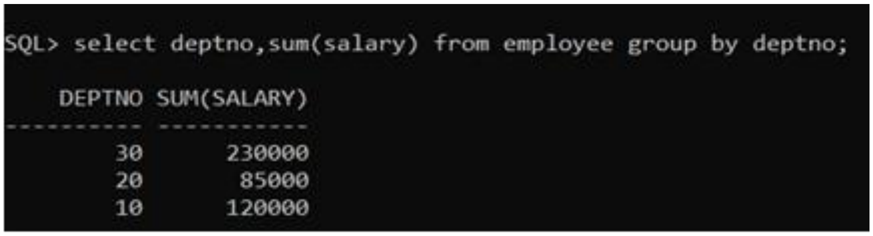
```
SQL> select max(avg(salary)) from employee group by deptno;

MAX(AVG(SALARY))
----------------
      76666.6667
```

select Min(avg(salary)) from employee group by deptno;



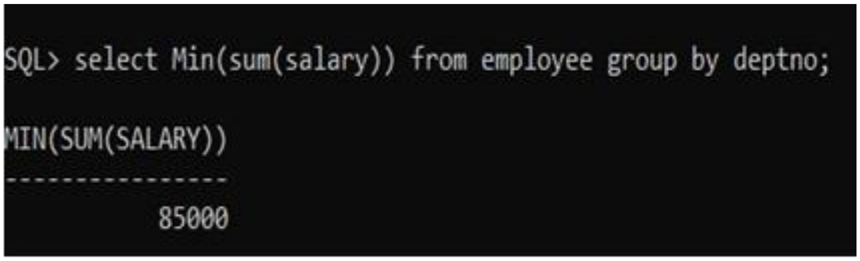
```
SQL>  select Min(avg(salary)) from employee group by deptno;

MIN(AVG(SALARY))
----------------
           42500
```

select deptno,sum(salary) from employee group by deptno;



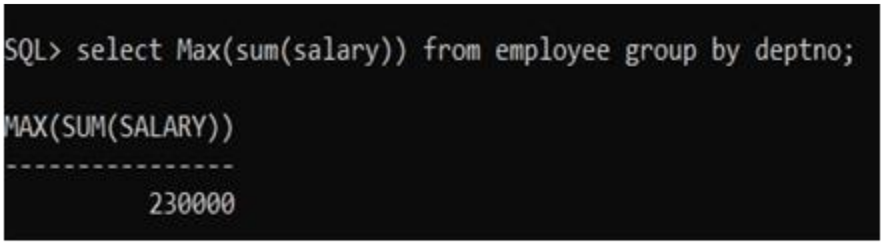
```
SQL> select deptno,sum(salary) from employee group by deptno;

    DEPTNO SUM(SALARY)
---------- -----------
        30      230000
        20       85000
        10      120000
```

select Min(sum(salary)) from employee group by deptno;



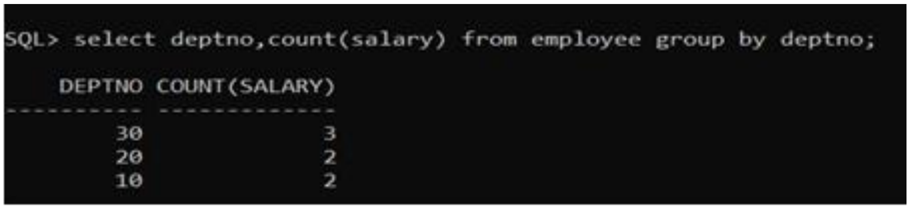
```
SQL> select Min(sum(salary)) from employee group by deptno;

MIN(SUM(SALARY))
----------------
           85000
```

select Max(sum(salary)) from employee group by deptno;



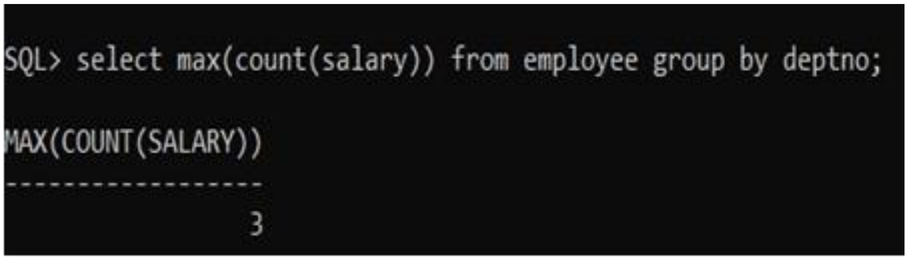
```
SQL> select Max(sum(salary)) from employee group by deptno;

MAX(SUM(SALARY))
----------------
          230000
```

select deptno,count(salary) from employee group by deptno;



```
SQL> select deptno,count(salary) from employee group by deptno;

    DEPTNO COUNT(SALARY)
---------- -------------
        30             3
        20             2
        10             2
```

select max(count(salary)) from employee group by deptno;



```
SQL> select max(count(salary)) from employee group by deptno;

MAX(COUNT(SALARY))
------------------
                 3
```

EXERCISE : 4

QUERIES USING CONVERSION FUNCTIONS (TO_CHAR, TO_NUMBER AND TO_DATE), STRING FUNCTIONS (CONCATENATION, LPAD, RPAD, LTRIM, RTRIM, LOWER, UPPER, INITCAP, LENGTH, SUBSTR AND INSTR), DATE FUNCTIONS (SYSDATE, NEXT_DAY, ADD_MONTHS, LAST_DAY, MONTHS_BETWEEN, LEAST, GREATEST, TRUNC, ROUND, TO_CHAR)

Description :

a. **Character functions :**

Case – manipulation functions (LOWER , UPPER , INITCAP) :

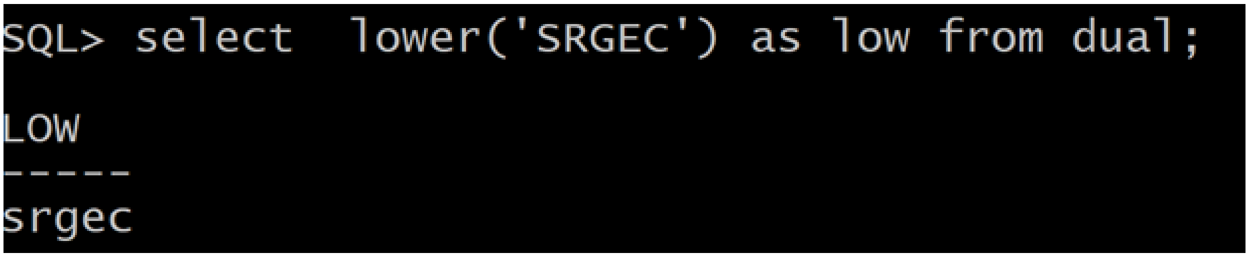
1. lower (): this function converts the uppercase letters to lower case letters what you are passed to the function.

Syntax:

lower(message)

Example

select lower('SRGEC') as low from dual;



```
SQL> select  lower('SRGEC') as low from dual;

LOW
-----
srgec
```

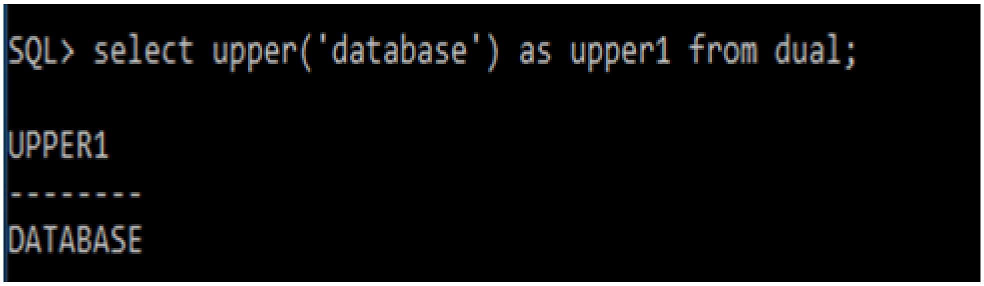
2.upper(): this function is used to convert the lower case letters into uppercase letters.

Syntax:

upper(message)

Example

select upper('database') as upper1 from dual;



```
SQL> select upper('database') as upper1 from dual;

UPPER1
--------
DATABASE
```

3. initcap():

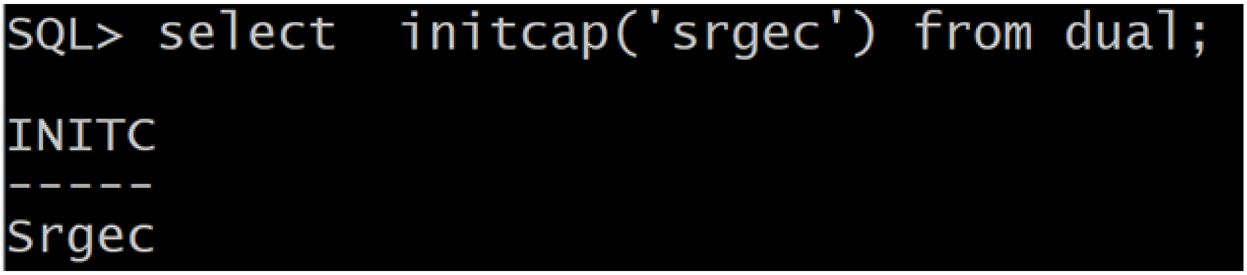
It make initial letter to capital letter what you have passed to the function.

Syntax:

initcap(message)

Example :

select initcap('srgec') from dual;



```
SQL> select  initcap('srgec') from dual;

INITC
-----
Srgec
```

Character manipulation functions (CONCAT , SUBSTR , LENGTH , INSTR , LPAD | RPAD , TRIM , REPLACE) :

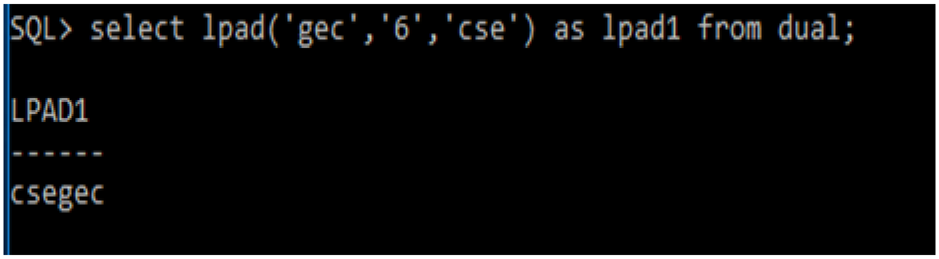
1. **lpad(): This function is used for attaching a new word to the original one at left side.**

Syntax:

lpad(word1,length,word2)

Example:

select lpad('gec',’6','cse') as lpad1 from dual;



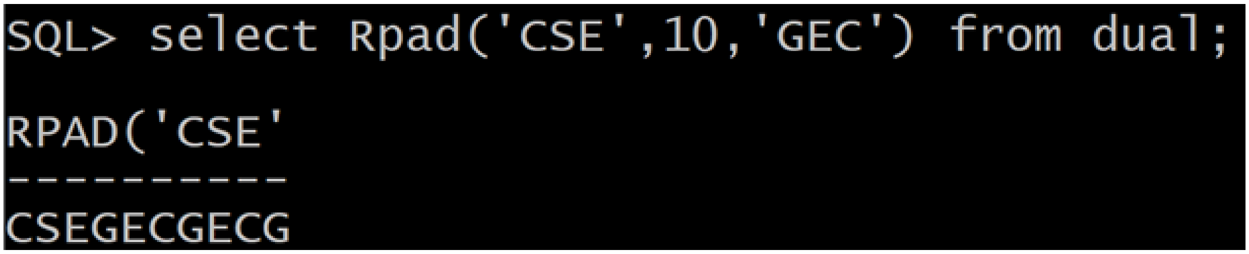
```
SQL> select lpad('gec','6','cse') as lpad1 from dual;

LPAD1
------
csegec
```

2. **rpad(): This function is used for attaching a new word to the original one at right side.**

Syntax:

rpad(word1,length,word2)

Example: select Rpad('CSE',10,'GEC') from dual;



```
SQL> select Rpad('CSE',10,'GEC') from dual;

RPAD('CSE'
----------
CSEGECGECG
```

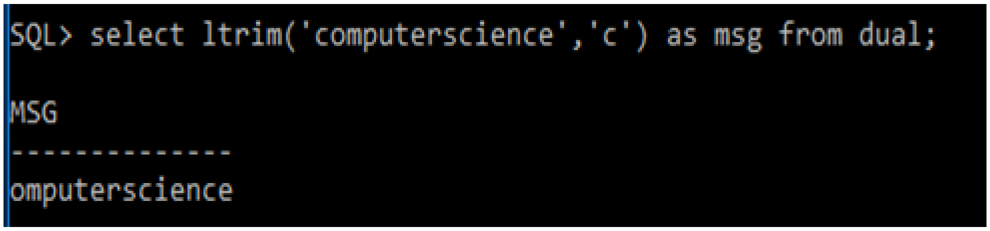
3. **ltrm():This function is used for left trimming i.e, it delete(cut) the left most letter.**

Syntax:

ltrim('message','character')

Example:

select ltrim('computerscience','c') as msg from dual;



```
SQL> select ltrim('computerscience','c') as msg from dual;

MSG
--------------
omputerscience
```

4. **rtrim()**

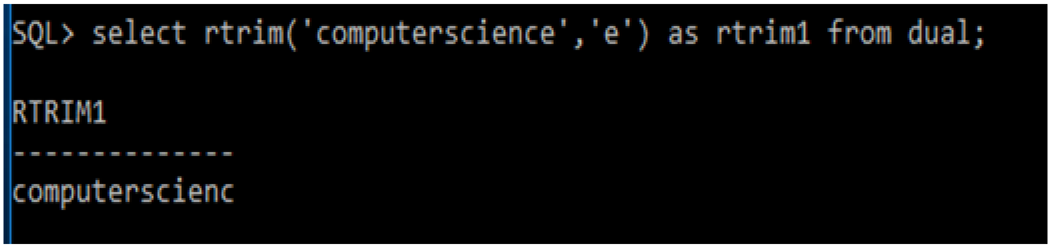
This function is used for right trimming.

Syntax:

rtrim('message','character')

Example:

select rtrim('computerscience','e') as rtrim1 from dual;



```
SQL> select rtrim('computerscience','e') as rtrim1 from dual;

RTRIM1
--------------
computerscienc
```

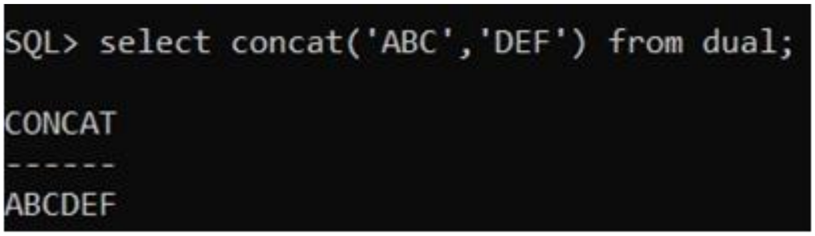
5. **concat():This function is used to add two strings.**

Syntax:

Concat('string1' , 'string 2')

Example :

select concat('ABC','DEF') from dual;



```
SQL> select concat('ABC','DEF') from dual;

CONCAT
------
ABCDEF
```

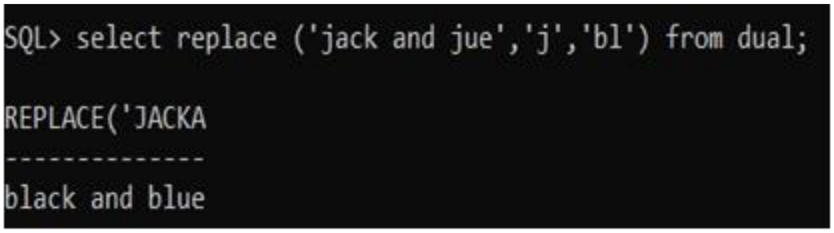
6. **Replace : This function is used to replace a particular character from a string.**

Syntax :

Replace(‘string’, ‘replaceable char’,char);

Example :

select replace ('jack and jue','j','bl') from dual;



```
SQL> select replace ('jack and jue','j','bl') from dual;

REPLACE('JACKA
--------------
black and blue
```

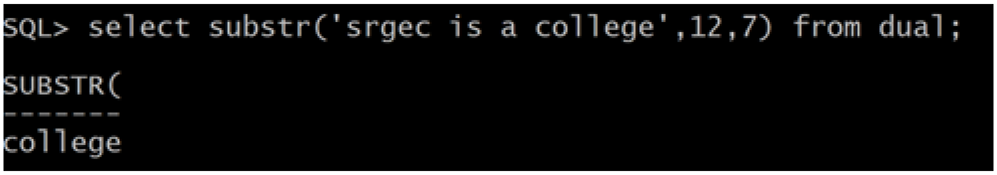
7. **substring :This function is used to extract the substring from a main string .**

syntax :

substr(string , indexing , size);

Example :

SQL> select substr('srgec is a college',12,7) from dual;



```
SQL> select substr('srgec is a college',12,7) from dual;

SUBSTR(
-------
college
```

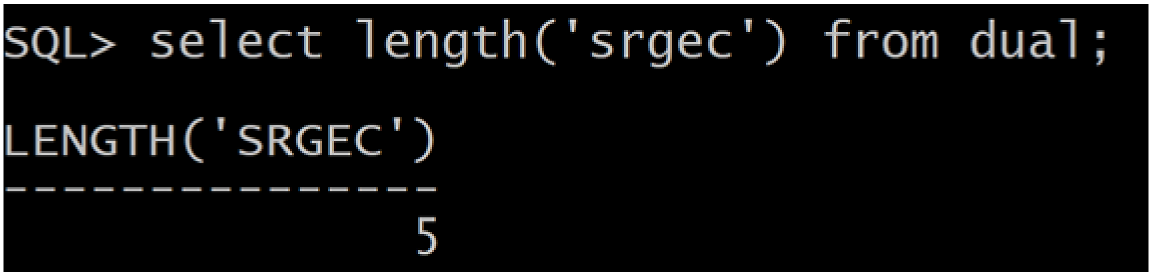
8. **length :This function is used to find the length of a given string.**

Syntax :

Length(string);

Example:

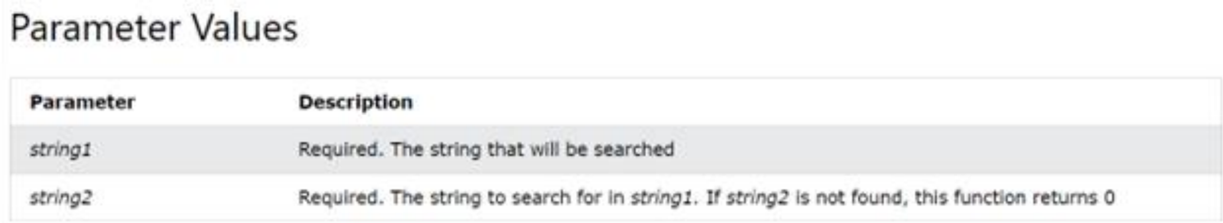
SQL> select length('srgec') from dual;



```
SQL> select length('srgec') from dual;

LENGTH('SRGEC')
---------------
              5
```

9. INSTR():The INSTR() function returns the position of the first occurrence of a string in another string. This function performs a case-insensitive search.

Syntax: INSTR(string1, string2)

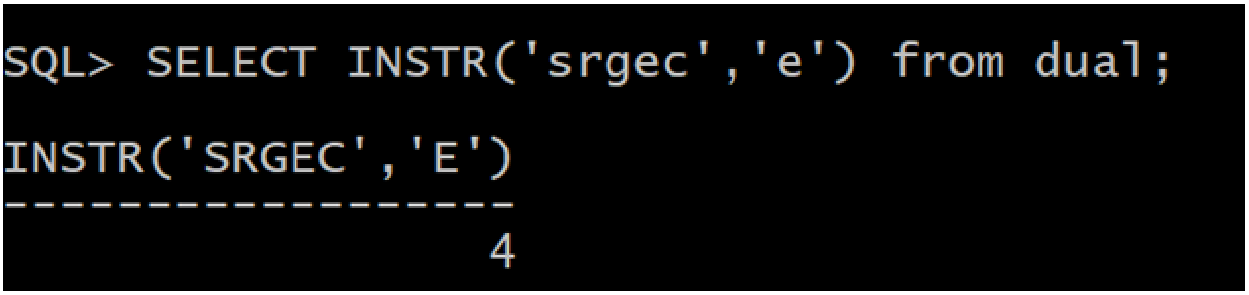


Parameter Values

Parameter	Description
string1	Required. The string that will be searched
string2	Required. The string to search for in *string1*. If *string2* is not found, this function returns 0

Example: SQL> SELECT INSTR('srgec','e') from dual;

Output:



```
SQL> SELECT INSTR('srgec','e') from dual;

INSTR('SRGEC','E')
------------------
                 4
```

b. **Number functions (ROUND , TRUNCATE , MOD) :**

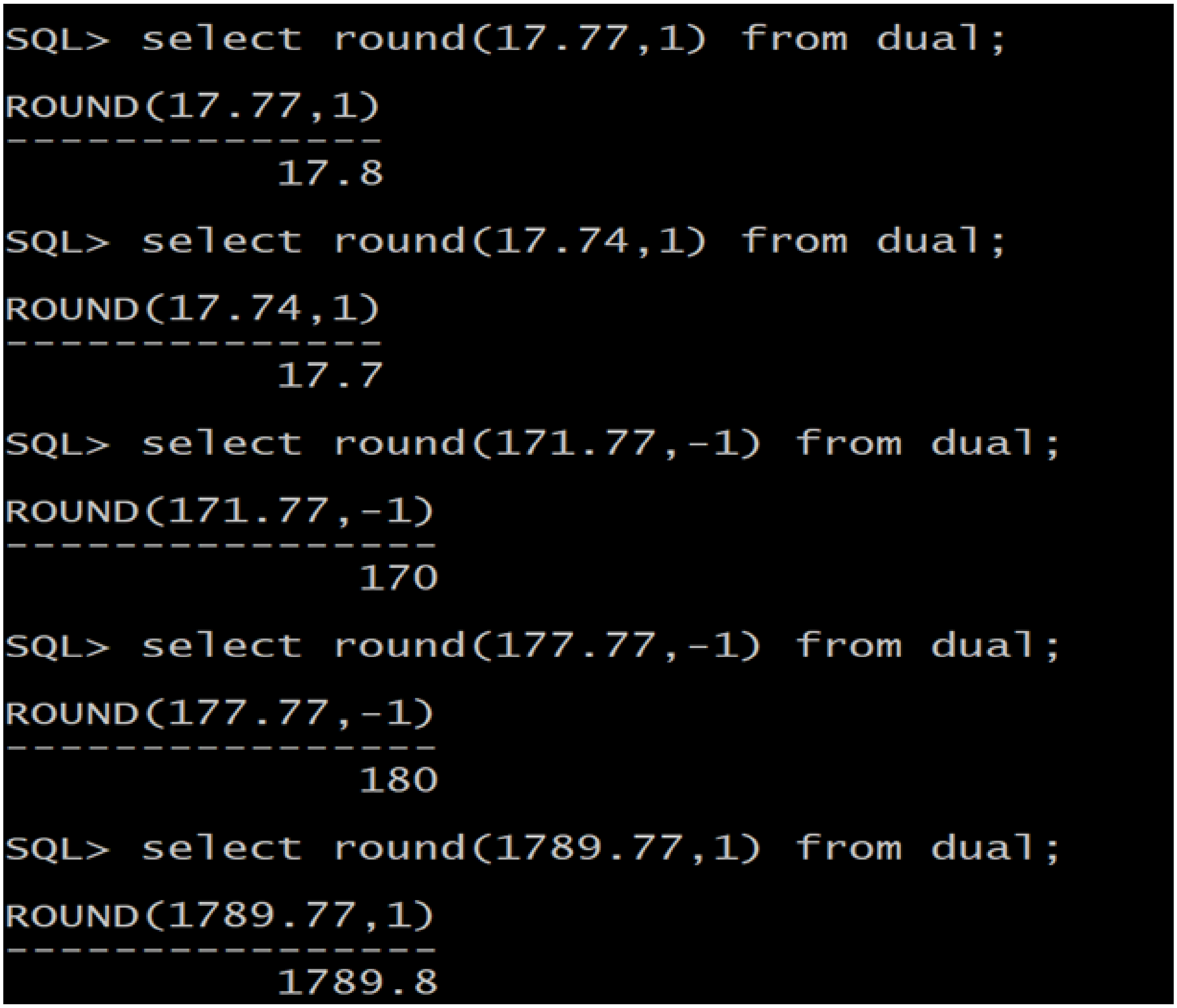
1. **ROUND :**

The ROUND() function rounds a number to a specified number of decimal places.

SYNTAX :

ROUND(*number*, *decimals*, *operation*)

Example :



```
SQL> select round(17.77,1) from dual;

ROUND(17.77,1)
--------------
          17.8

SQL> select round(17.74,1) from dual;

ROUND(17.74,1)
--------------
          17.7

SQL> select round(171.77,-1) from dual;

ROUND(171.77,-1)
----------------
             170

SQL> select round(177.77,-1) from dual;

ROUND(177.77,-1)
----------------
             180

SQL> select round(1789.77,1) from dual;

ROUND(1789.77,1)
----------------
          1789.8
```

2. **TRUNCATE :**

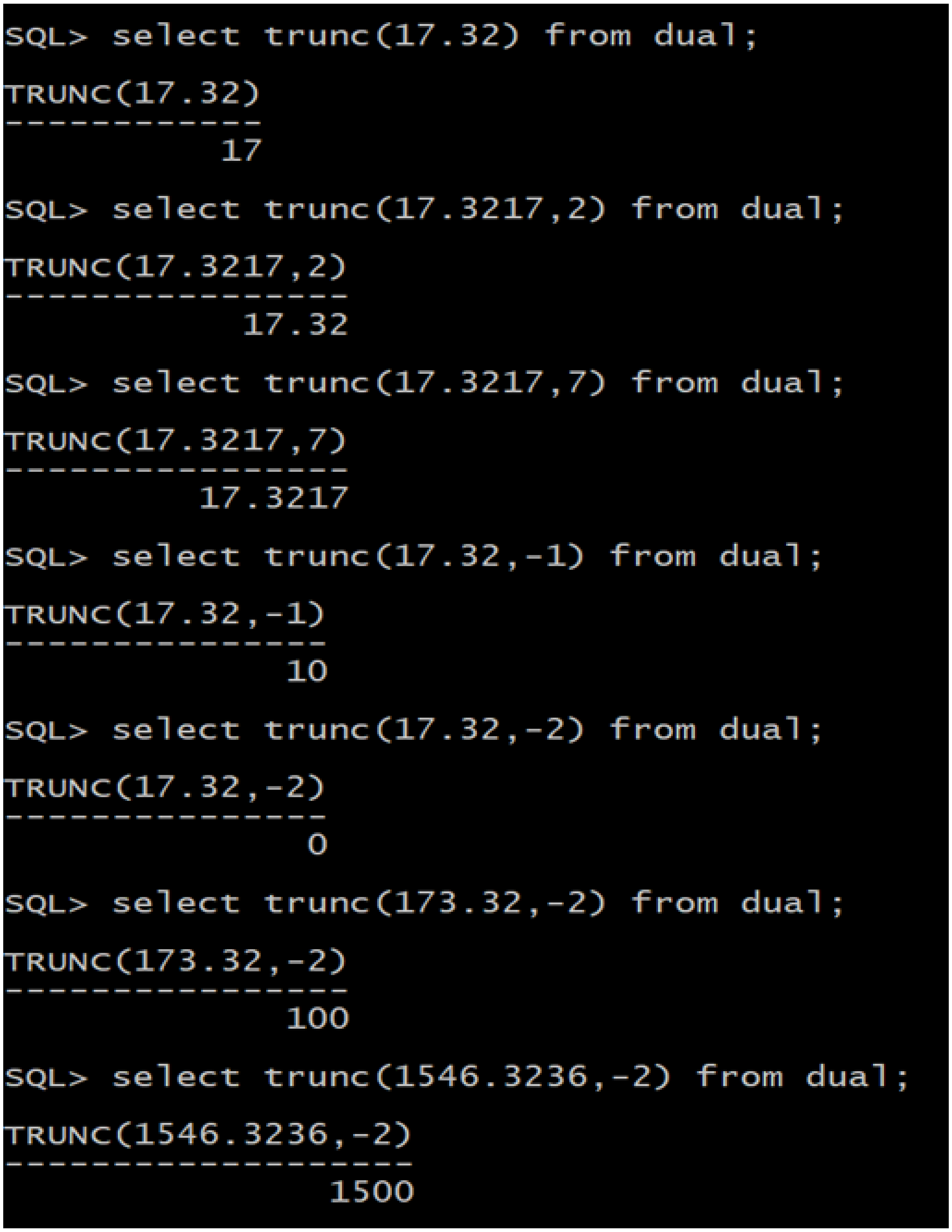
The TRUNCATE() function truncates a number to the specified number of decimal places.

Syntax :

TRUNCATE(*number*, *decimals*)

Example :

Output :



```
SQL> select trunc(17.32) from dual;

TRUNC(17.32)
------------
          17

SQL> select trunc(17.3217,2) from dual;

TRUNC(17.3217,2)
----------------
           17.32

SQL> select trunc(17.3217,7) from dual;

TRUNC(17.3217,7)
----------------
         17.3217

SQL> select trunc(17.32,-1) from dual;

TRUNC(17.32,-1)
---------------
             10

SQL> select trunc(17.32,-2) from dual;

TRUNC(17.32,-2)
---------------
              0

SQL> select trunc(173.32,-2) from dual;

TRUNC(173.32,-2)
----------------
             100

SQL> select trunc(1546.3236,-2) from dual;

TRUNC(1546.3236,-2)
-------------------
               1500
```

3. **MOD :**

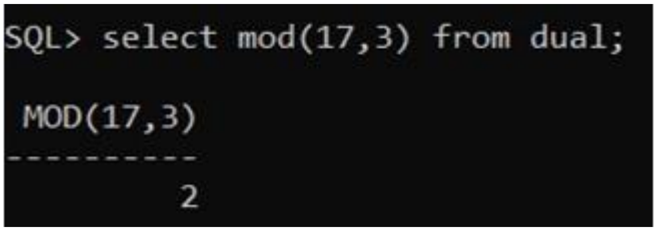
The MOD() function returns the remainder of a number divided by another number.

Syntax :

MOD(x , y)

Example :

select mod(17,3) from dual;



```
SQL> select mod(17,3) from dual;

 MOD(17,3)
----------
         2
```

c. **Date functions :**

Months _ Between :It gives the number of months between specified two dates.

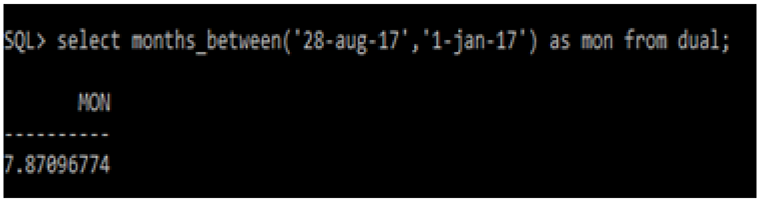
Result value	**Months_between(date-exp1,date-exp2)**
Negative result	**If date-exp1 is earlier than date-exp2**
Integer result	**If date-exp1 and date-exp2 have the same day,or both specify th the month.**
Decimal result	**If days are different and they are not both specify the last day o**
Fractional part	**Always calcilated as the difference between days divided by 31 number of days in the month.**

Syntax:

months_between(date1,date2)

Example:

select months_between('28-aug-17','1-jan-17') as mon from dual;

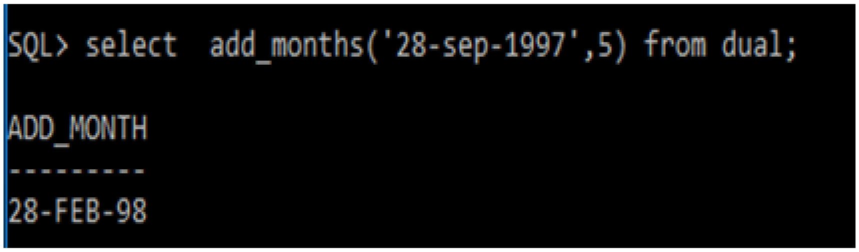


```
SQL> select months_between('28-aug-17','1-jan-17') as mon from dual;

       MON
----------
7.87096774
```

Add _Months : This function is used to add the 'n' number of months to a given date.

Example:

select add_months('28-sep-1997',5) from dual;



```
SQL> select  add_months('28-sep-1997',5) from dual;

ADD_MONTH
---------
28-FEB-98
```

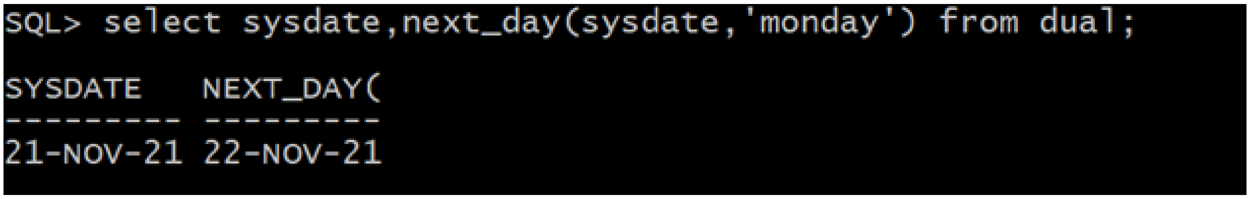
Next_Day :

Syntax :

next_day(date,dayname)

EXAMPLE :

SQL> select sysdate,next_day(sysdate,'monday') from dual;



```
SQL> select sysdate,next_day(sysdate,'monday') from dual;

SYSDATE   NEXT_DAY(
--------- ---------
21-NOV-21 22-NOV-21
```

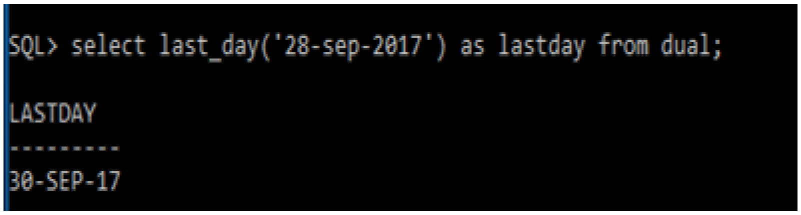
Last _ Day : It gives the last day of the specified month in a date.

Syntax:

last_date(date)

Example:

select last_day('28-sep-2017') as lastday from dual;



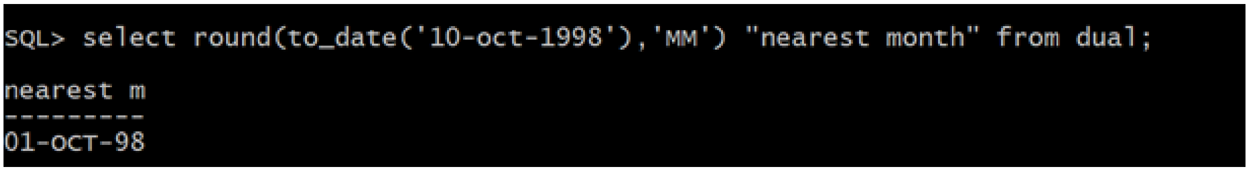
```
SQL> select last_day('28-sep-2017') as lastday from dual;

LASTDAY
---------
30-SEP-17
```

Round :

The Round() Returns the date rounded by the specified format unit

Example :

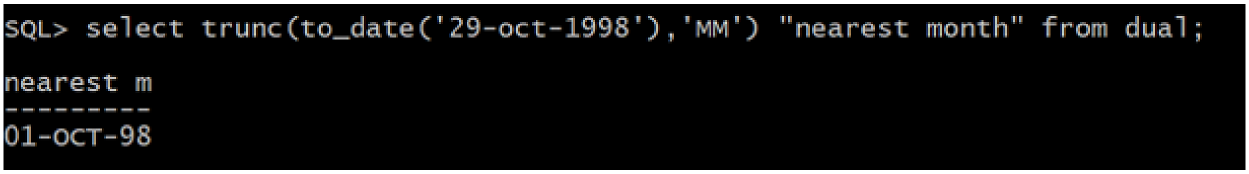
select round(to_date('10-oct-1998'),'MM') "nearest month" from dual;



```
SQL> select round(to_date('10-oct-1998'),'MM') "nearest month" from dual;

nearest m
---------
01-OCT-98
```

Trunc : Truncates the specified date of its time portion according to the format unit provided.

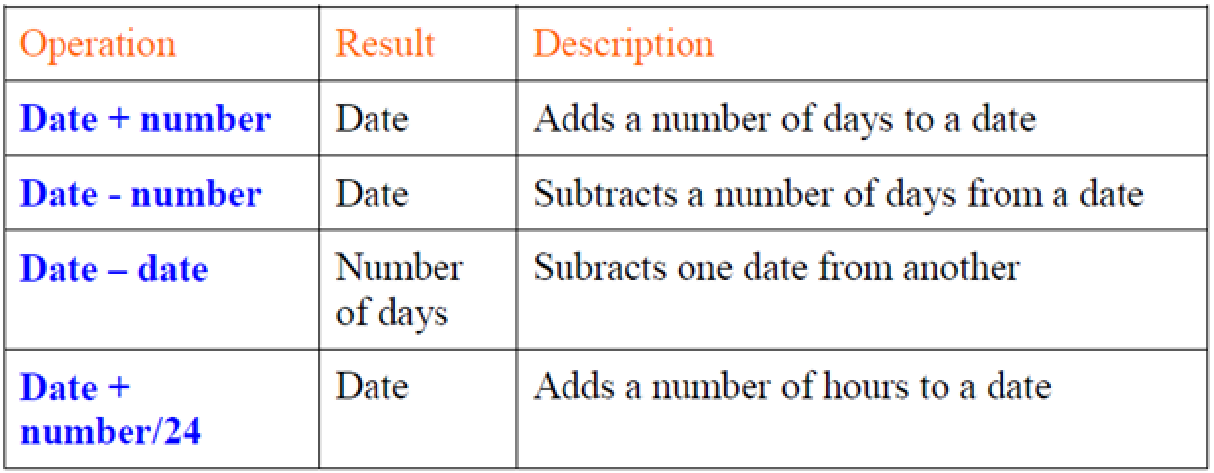
Example :



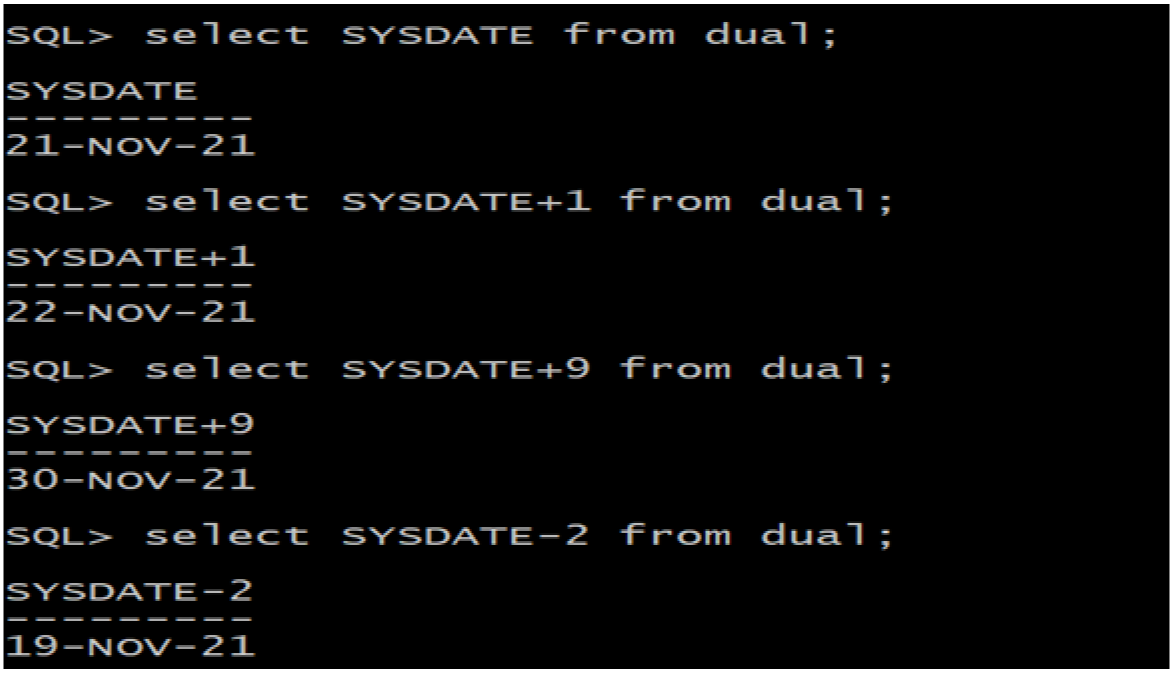
```
SQL> select trunc(to_date('29-oct-1998'),'MM') "nearest month" from dual;

nearest m
---------
01-OCT-98
```

Arithmetic with Dates:

- **Add or subtract a number to or from a date for a resultant date value**
- **Subtract two dates to find the number of days between those dates.**
- **Add hours to a date by dividing the number of hours by 24**
- **Since the database stores dates as numbers, you can perform calculations using arithmetic operators such as addition and subtraction. You can add and subtract number constants as well as dates. You can perform the following operations:**



Operation	Result	Description
Date + number	Date	Adds a number of days to a date
Date - number	Date	Subtracts a number of days from a date
Date – date	Number of days	Subracts one date from another
Date + number/24	Date	Adds a number of hours to a date



```
SQL> select SYSDATE from dual;

SYSDATE
---------
21-NOV-21

SQL> select SYSDATE+1 from dual;

SYSDATE+1
---------
22-NOV-21

SQL> select SYSDATE+9 from dual;

SYSDATE+9
---------
30-NOV-21

SQL> select SYSDATE-2 from dual;

SYSDATE-2
---------
19-NOV-21
```

Experiment 5

i. **Create a simple PL/SQL program which includes declaration section, executable section and exception –Handling section (Ex. Student marks can be selected from the table and printed for those who secured first class and an exception can be raised if no records were found).**

ii. **Insert data into student table and use COMMIT, ROLLBACK and SAVEPOINT in PL/SQL block..**

i). We have to create the student table and insert the records in to the table as follows:

```
SQL> create table student(sid number(10),sname varchar2(20),rank varchar(10));
Table created.



SQL> insert into student values(501,'Ravi','second');
1 row created.



SQL> insert into student values(502,'Raju','third');
1 row created.



SQL> insert into student values(503,'Ramu','');
1 row created.



SQL> select *from student;

SID SNAME  RANK

501 Ravi   second
502 Raju   third
503 Ramu
```

PL/SQL CODE:

```
SQL>ed 5a
```

Enter the following code into the text editor and save the file with .sql format

```
set serveroutput on; declare
temp1 number(10); temp2 varchar2(10);

begin
select sid,sname into temp1,temp2 from student where rank='first'; dbms_output.put_line('Student No:'||
temp1 ||' Name:'||temp2||' got first rank');
exception
when no_data_found then dbms_output.put_line('*******************************************');
dbms_output.put_line('# Error: there is no student got first rank');
 end;
/
```

```
SQL> @5a;
******************************************* # Error: there is no student got first rank
PL/SQL procedure successfully completed.

SQL> update student set rank='first' where sid=503; 1 row updated.

SQL> select *from student;
SID SNAME  RANK


----------- ---------------------- -----------
501 Ravi        second
```

```
502 Raju     third
503 Ramu     first
SQL> @5a
Student No:503 Name:Ramu got first rank PL/SQL procedure successfully completed.
```

ii)

```
SQL> select *from student;
```

```
SID SNAME  RANK
501 Ravi     second
502 Raju     third
503 Ramu     first
```

PL/SQL CODE:

```
SQL>ed 5b
```

Enter the following code into the text editor and save the file with .sql format set serverout

```
DECLARE
sno student.sid%type; name student.sname%type; srank student.rank%type;

BEGIN
sno := &sno; name := '&name'; srank := '&srank';
INSERT into student values(sno,name,srank); dbms_output.put_line('One record inserted
COMMIT;
-- adding savepoint SAVEPOINT s1;
-- second time asking user for input sno := &sno;
name := '&name'; srank := '&srank';
INSERT into student values(sno,name,srank); dbms_output.put_line('One record inserted
ROLLBACK TO SAVEPOINT s1;
END;
/
```

```
SQL> @5b;
```

EXPERIMENT: 6

Develop a program that includes the features NESTED IF, CASE and CASE expression.

The programcan be extended using the NULLIF and COALESCE functions.

A. NESTED IF:

A nested if-then is an if statement that is the target of another if statement. Nested if-then statements mean an if statement inside another if statement

Syntax:-

```
if (condition1) then
   -- Executes when
   condition1 is true if
   (condition2) then
     -- Executes when
   condition2 is true end if;
end if;
```

PL/SQL CODE: PL/SQL Program to find biggest of three number using nested if.

```
 SQL>ed 6a
```

Enter the following code into the text editor and save the file with .sql format

```
declare
        a number:=10;
        b number:=12;
        c number:=5;
begin
        dbms_output.put_line('a='||a||' b='||b||' c='||c);
        if a>b AND a>c then
                dbms_output.put_line('a is greatest');
        else
                if b>a AND b>c then
                        dbms_output.put_line('b is greatest');
                else
                        dbms_output.put_line('c is greatest');
                end if;
        end if;
end;
/
```

```
SQL> @ "D:\dbms\lab experiments\6a.sql";
```

a=10 b=12 c=5

b is greatest

PL/SQL procedure success

B. CASE and CASE Expression : **CASE statement** selects one sequence of statements to execute. However, to select the sequence, the **CASE** statement uses a selector rather than multiple Boolean expressions. A selector is an expression, the value of which is used to select one of several alternatives. **Syntax**

```
CASE selector
  WHEN       'value1'
  THEN  S1;  WHEN
  'value2'  THEN  S2;
  WHEN       'value3'
  THEN S3;
  ...
  ELSE Sn;  -- default
```

case END CASE;

SQL> create table emp(eno number(5), ename varchar2(10), loc varchar(10), salary number(10,2));

Table created.

SQL > insert into emp values(101,'ali','vja',15000);

1 row created.

SQL> insert into emp values(102,'ravi','hyd',25000);

1 row created.

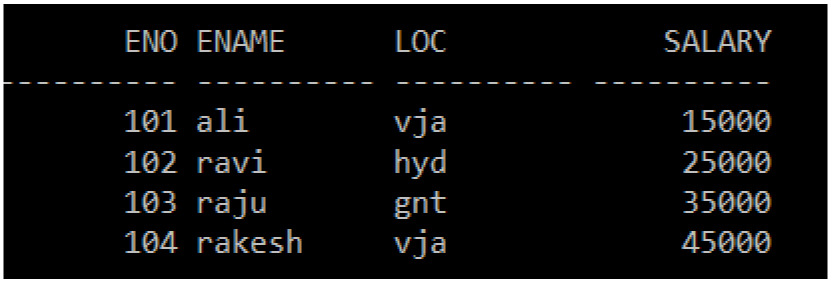
SQL> insert into emp values(103,'raju','gnt',35000);

1 row created.

SQL> insert into emp values(104,'rakesh','vja',45000);

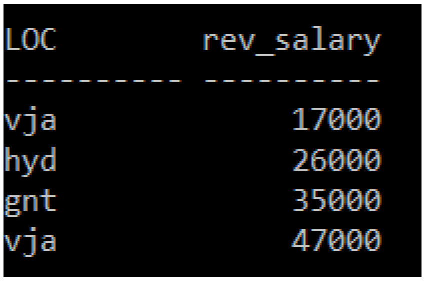
1 row created.

SQL> select *from emp;



```
       ENO ENAME      LOC            SALARY
---------- ---------- ---------- ----------
       101 ali        vja             15000
       102 ravi       hyd             25000
       103 raju       gnt             35000
       104 rakesh     vja             45000
```

SQL> select loc, case(loc) when 'vja' then salary+2000 when 'hyd' then salary+1000 else salary end "rev_salary" from emp;



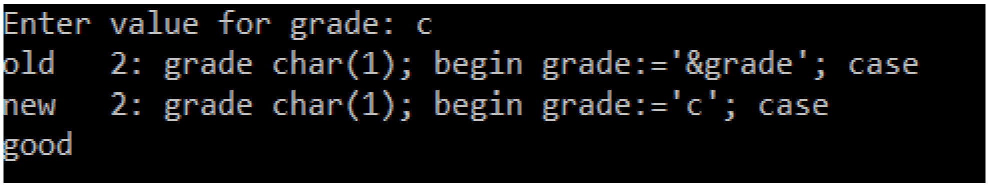
```
LOC        rev_salary
---------- ----------
vja             17000
hyd             26000
gnt             35000
vja             47000
```

PL/SQL CODE: PL/SQL CODE to demonstrate CASE

SQL> ed 6b

```
declare
grade char(1); begin grade:='&grade'; case
when grade='a' then
dbms_output.put_line('Excellent'); when grade='b' then
dbms_output.put_line('very good'); when grade='c' then
dbms_output.put_line('good'); when grade='d' then
dbms_output.put_line('fair'); when grade='f' then
dbms_output.put_line('poor');
else
 dbms_output.put_line('No such grade');
end case;
end;
/
```

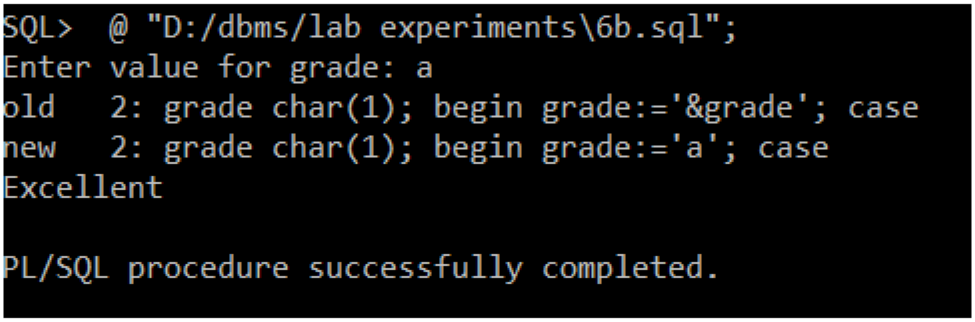
SQL> @ "D:/dbms/lab experiments\6b.sql";



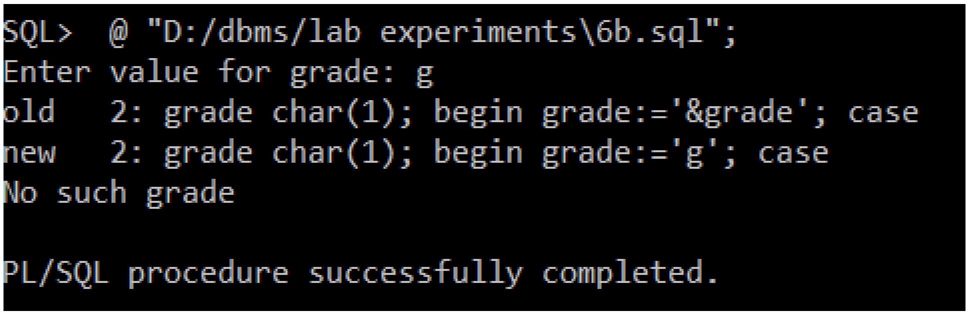
```
Enter value for grade: c
old   2: grade char(1); begin grade:='&grade'; case
new   2: grade char(1); begin grade:='c'; case
good
```

PL/SQL procedure successfully completed.

SQL> @ "D:/dbms/lab experiments\6b.sql";



```
SQL>  @ "D:/dbms/lab experiments\6b.sql";
Enter value for grade: a
old   2: grade char(1); begin grade:='&grade'; case
new   2: grade char(1); begin grade:='a'; case
Excellent

PL/SQL procedure successfully completed.
```



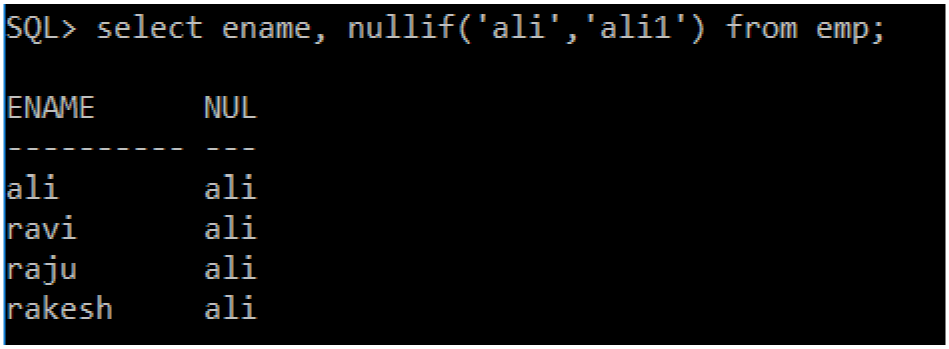
```
SQL>  @ "D:/dbms/lab experiments\6b.sql";
Enter value for grade: g
old   2: grade char(1); begin grade:='&grade'; case
new   2: grade char(1); begin grade:='g'; case
No such grade

PL/SQL procedure successfully completed.
```

C. NULLIF: Takes two arguments. If the two arguments are equal, then NULL is returned. otherwise the first argument is returned.

Syntax: **select column_name, NULLIF(argument1,arguement2) from table_name;**

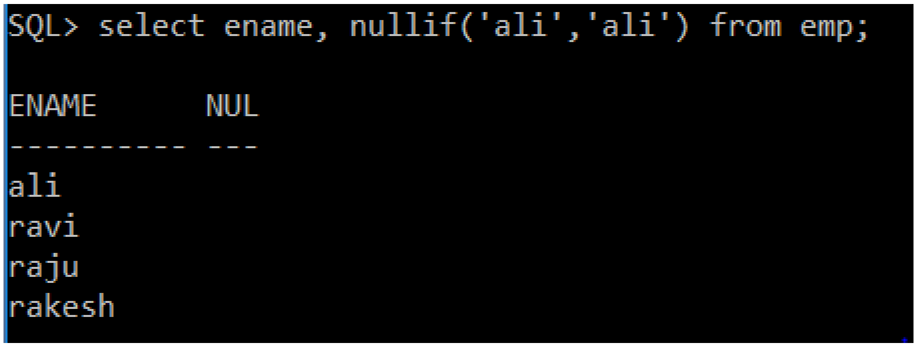
Example:

SQL> select ename, nullif('ali','ali1') from emp;



```
SQL> select ename, nullif('ali','ali1') from emp;

ENAME      NUL
---------- ---
ali        ali
ravi       ali
raju       ali
rakesh     ali
```

SQL>select ename, nullif('ali','ali') from emp;



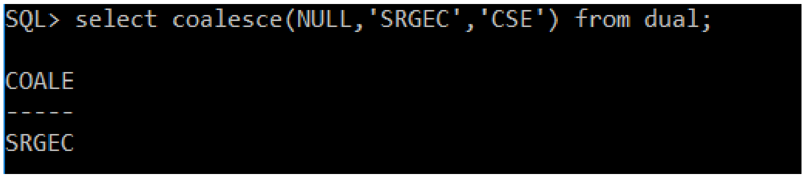
```
SQL> select ename, nullif('ali','ali') from emp;

ENAME      NUL
---------- ---
ali
ravi
raju
rakesh
```

D. COALESCE: COALESCE () function accepts a list of arguments and returns the first one that evaluates to a non-null value.

Syntax: coalesce("expression1","expression2",...);

Example:

SQL> select coalesce(NULL,'SRGEC','CSE') from dual;



```
SQL> select coalesce(NULL,'SRGEC','CSE') from dual;

COALE
-----
SRGEC
```

EXPERIMENT:7

Program development using WHILE LOOPS, numeric FOR LOOPS, nested loops using ERROR Handling, Built–In Exceptions, User defined Exceptions, Raise Application Error

A. **WHILE LOOP: AWHILE LOOP statementin PL/SQL programming language repeatedly executes a target statement as long as a given condition is true.**

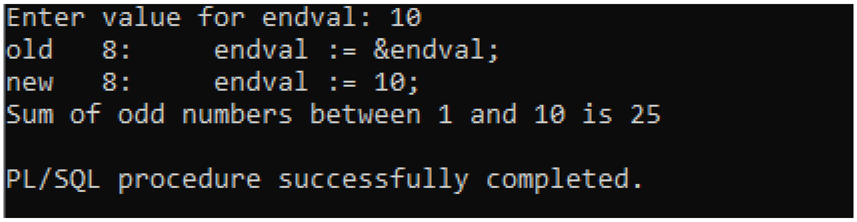
Syntax:

```
WHILE condition LOOP

sequence_of_statements

END LOOP;
```

PL/SQL Code:APL/SQL Program to find sum of ODD number upto given umber using While loop

```
Set server output on;

declare

inval number;

endval number;

s number default 0;

begin

inval := 1;

  -- Prompt the user for the value of endval
```

```
endval := &endval;
while inval<endval
loop
s := s + inval;
inval := inval + 2;  -- Move to the next odd number
end loop;
        -- Output the result
dbms_output.put_line('Sum of odd numbers between 1 and ' || endval || ' is ' || s);
end;
/
```



```
Enter value for endval: 10
old   8:     endval := &endval;
new   8:     endval := 10;
Sum of odd numbers between 1 and 10 is 25

PL/SQL procedure successfully completed.
```

B. For Loop

In PL/SQL, a FOR loop allows you to execute a block of code a specific number of times. It works by iterating through a range of values, which you define

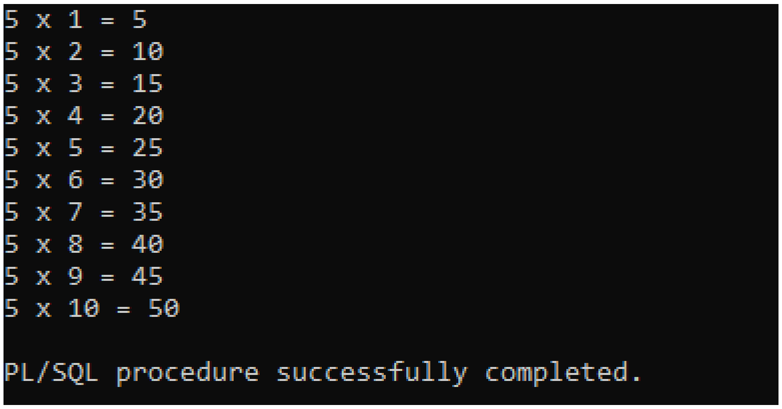
Syntax

```
FOR counter IN initial_value..final_value LOOP sequence_of_statements;
ENDLOOP;
```

PL/SQLCODE:APL/SQL code to print multiplication table using for loop

```
SET SERVEROUTPUT ON;
DECLARE
i NUMBER;
```

```
num NUMBER := 5;  -- You can change the number to get the multiplication table for a different number

BEGIN

   FOR i IN 1..10 LOOP

      DBMS_OUTPUT.PUT_LINE(num || ' x ' || i || ' = ' || (num * i));

   END LOOP;

END;

/
```



```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

PL/SQL procedure successfully completed.
```

C.NESTED LOOP: PL/SQL allows using one loop inside another loop.It may be either basic,while or for loop.

Syntax:

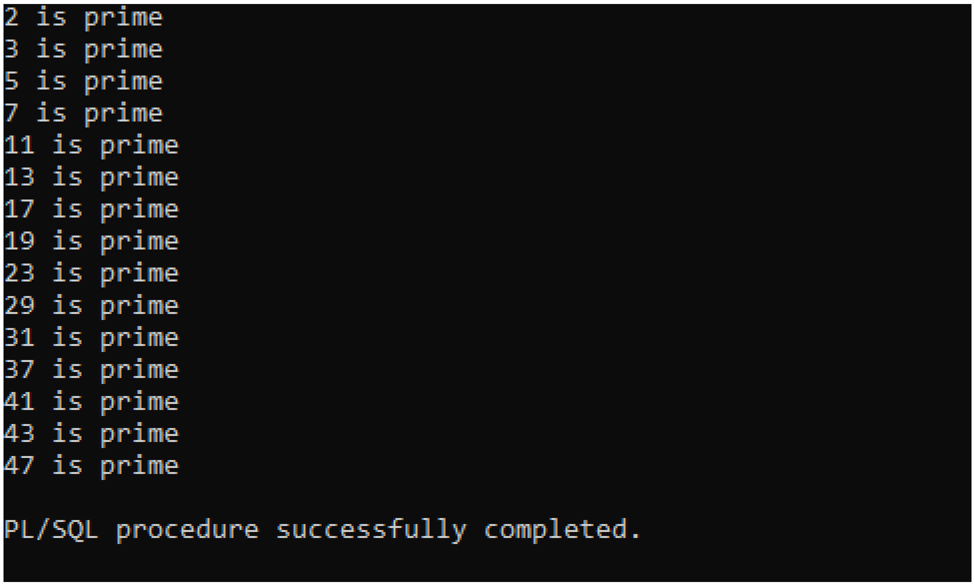
```
WHILE condition1 LOOP sequence_of_statements1

WHILE condition2 LOOP sequence_of_statements2

ENDLOOP;

END LOOP;
```

PL/SQLCODE:A PL/SQL program to print n prime number using nested loop using Error Handling.

```
SET SERVEROUTPUT ON;

DECLARE
```

```
i NUMBER := 2;  -- Starting from 2

j NUMBER;  -- Divisor variable

BEGIN

   -- Loop to find prime numbers

   LOOP

j := 2;  -- Start checking divisibility from 2

            -- Nested loop to check if 'i' is prime

      LOOP

         BEGIN

 -- Exit the loop if 'i' is divisible by 'j' or if 'j' reaches 'i'

            EXIT WHEN MOD(i, j) = 0 OR j = i;

j := j + 1;

         EXCEPTION

            WHEN OTHERS THEN

               DBMS_OUTPUT.PUT_LINE('Error while checking number ' || i || ': ' || SQLERRM);

               EXIT;  -- Exit the inner loop in case of an error

         END;

      END LOOP;

-- If 'i' is prime (j == i), print it

      IF j = i THEN

         DBMS_OUTPUT.PUT_LINE(i || ' is prime');

      END IF;

 -- Increment 'i' to check the next number

i := i + 1;
```

```
      -- Exit the outer loop when 'i' reaches 50
      EXIT WHEN i = 50;
   END LOOP;
END;
/
```

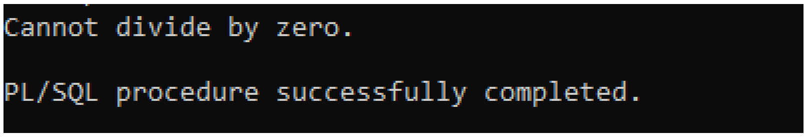


```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime

PL/SQL procedure successfully completed.
```

D.Built–In Exceptions::In PL/SQL, built-in exceptions are predefined exceptions that represent common error conditions.

```
DECLARE
result NUMBER;
BEGIN
result := 10 / 0;  -- Division by zero
EXCEPTION
   WHEN ZERO_DIVIDE THEN
      DBMS_OUTPUT.PUT_LINE('Cannot divide by zero.');
```

```
END;

/
```

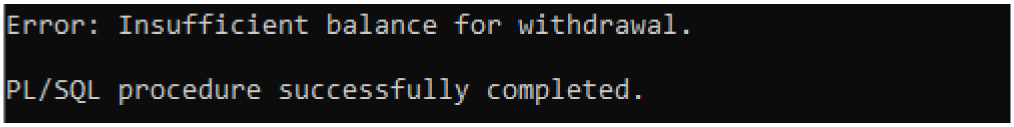


```
Cannot divide by zero.

PL/SQL procedure successfully completed.
```

E:USER defined Exceptions:In PL/SQL, user-defined exceptions are exceptions that you can define explicitly in your code.

```
DECLARE

   -- Declaring the user-defined exception

insufficient_balance EXCEPTION;

balance NUMBER := 50;

withdrawal_amount NUMBER := 100;

BEGIN

   -- Check if balance is sufficient for withdrawal

   IF balance <withdrawal_amount THEN

      RAISE insufficient_balance;  -- Raise exception if balance is insufficient

   END IF;

      -- If no exception, proceed with the withdrawal

balance := balance - withdrawal_amount;

   DBMS_OUTPUT.PUT_LINE('Withdrawal successful! New balance: ' || balance);


EXCEPTION

   WHEN insufficient_balance THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Error: Insufficient balance for withdrawal.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
END;
/
```

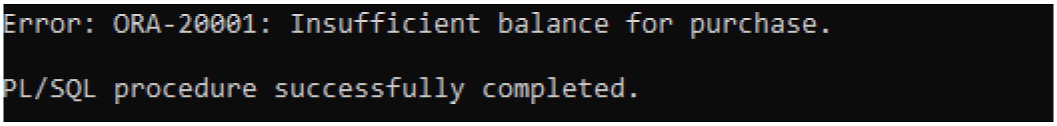


```
Error: Insufficient balance for withdrawal.

PL/SQL procedure successfully completed.
```

F::Raise Application Error::The RAISE_APPLICATION_ERROR procedure in PL/SQL is used to raise user-defined exceptions with a custom error number and message.

```
DECLARE
v_balance NUMBER := 50;
v_purchase_amount NUMBER := 100;
BEGIN
    -- Check if the balance is sufficient for the purchase
    IF v_balance<v_purchase_amount THEN
        -- Raise a custom error if the balance is insufficient
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance for purchase.');
    END IF;

    -- Continue with the purchase if balance is sufficient
```

```
v_balance := v_balance - v_purchase_amount;
    DBMS_OUTPUT.PUT_LINE('Purchase successful! Remaining balance: ' || v_balance);

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```



```
Error: ORA-20001: Insufficient balance for purchase.

PL/SQL procedure successfully completed.
```

8. Programs development using creation of procedures, passing parameters IN and OUT of Procedures.

A Stored procedure is a set of SQL statements stored in the database and executed as a single unit. It helps in improving performance, reducing redundancy, and ensuring data security.

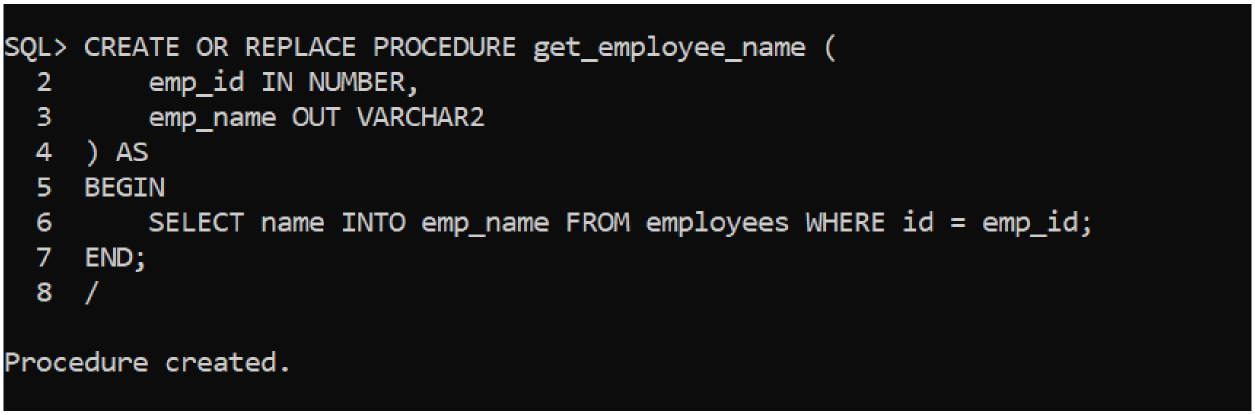
Step 1: Create the Table

```
CREATE TABLE employees (
   id NUMBER PRIMARY KEY,
   name VARCHAR2(100),
   salary NUMBER
);

INSERT INTO employees VALUES (101, 'Alice', 50000);
INSERT INTO employees VALUES (102, 'Bob', 60000);
INSERT INTO employees VALUES (103, 'Charlie', 55000);
COMMIT;
```

Step 2: Create a Procedure with IN and OUT Parameters

This procedure takes an employee ID (IN parameter) and returns the employee's name (OUT parameter).

```
CREATE OR REPLACE PROCEDURE get_employee_name (
   emp_id IN NUMBER,
   emp_name OUT VARCHAR2
) AS
BEGIN
   SELECT name INTO emp_name FROM employees WHERE id = emp_id;
END;
/
```



```
SQL> CREATE OR REPLACE PROCEDURE get_employee_name (
         emp_id IN NUMBER,
         emp_name OUT VARCHAR2
     ) AS
     BEGIN
         SELECT name INTO emp_name FROM employees WHERE id = emp_id;
     END;
     /

Procedure created.
```

Step 3: Execute the Procedure

DECLARE v_name VARCHAR2(100);

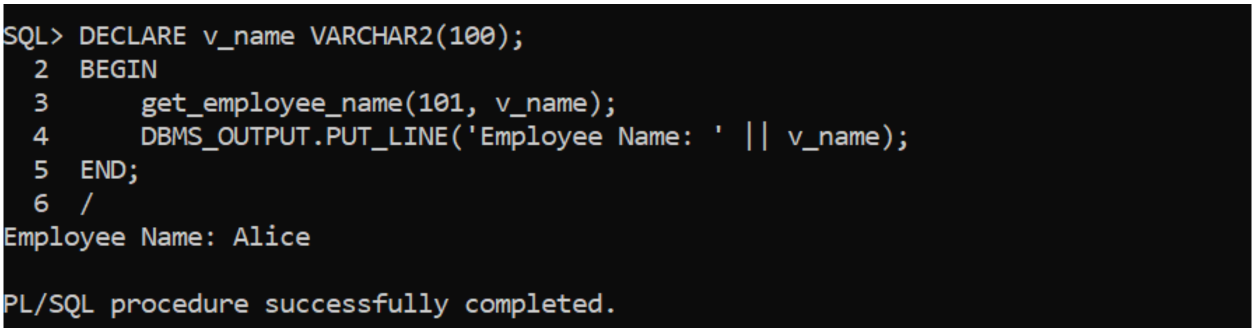
BEGIN

get_employee_name(101, v_name);

DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_name);

END;

/



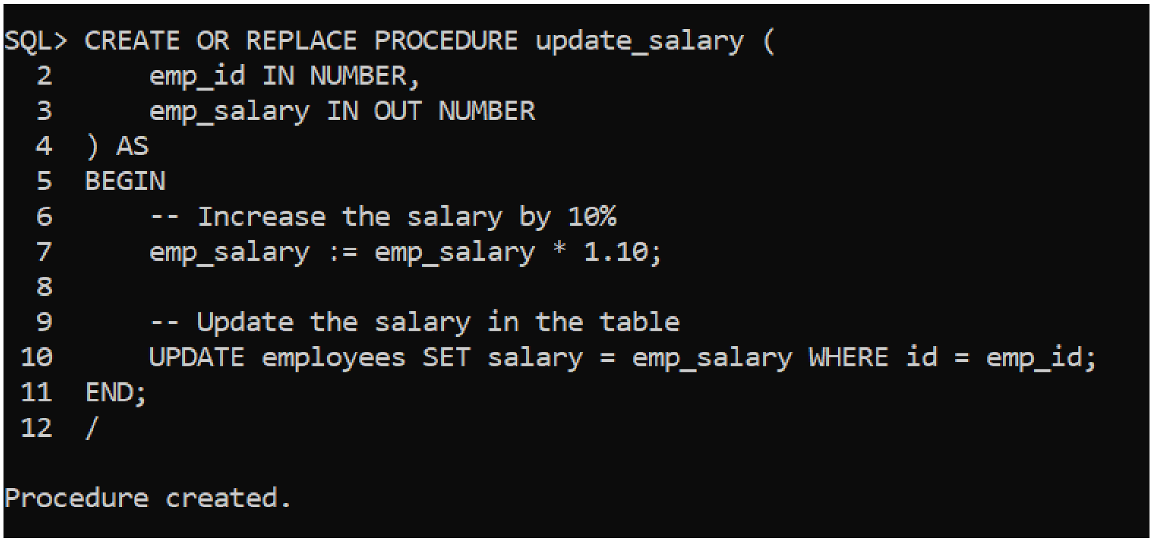
```
SQL> DECLARE v_name VARCHAR2(100);
     BEGIN
         get_employee_name(101, v_name);
         DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_name);
     END;
     /
Employee Name: Alice

PL/SQL procedure successfully completed.
```

Step 4: Create a Procedure with an INOUT Parameter

This procedure:

- **Takes an employee ID (IN parameter).**
- **Takes an existing salary (INOUT parameter).**
- **Increases the salary by 10% and updates the table.**

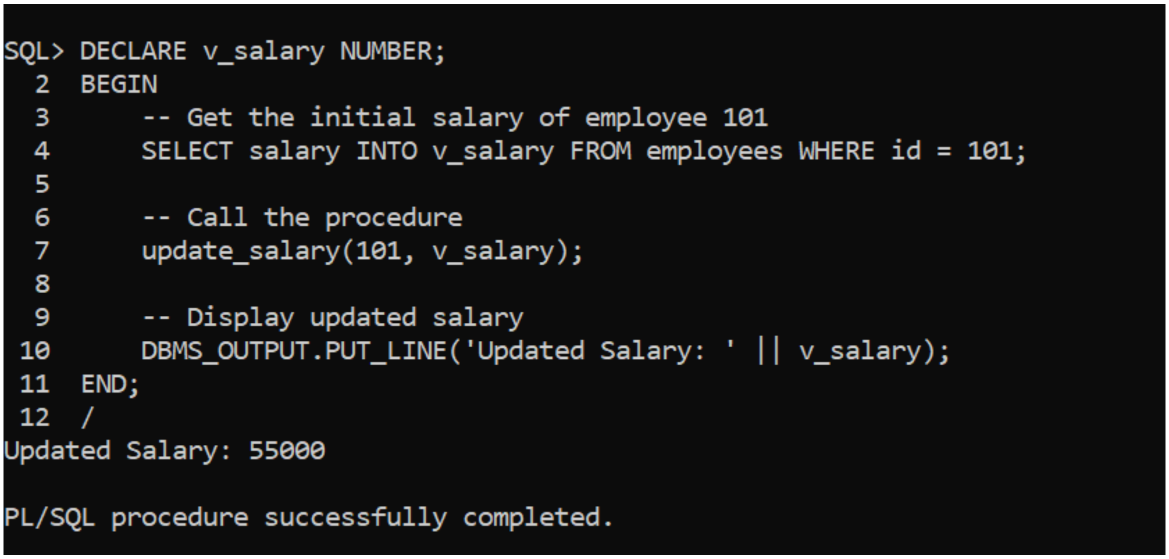
```
CREATE OR REPLACE PROCEDURE update_salary (

   emp_id IN NUMBER,

   emp_salary IN OUT NUMBER

) AS

BEGIN

   -- Increase the salary by 10%

   emp_salary := emp_salary * 1.10;

   -- Update the salary in the table

   UPDATE employees SET salary = emp_salary WHERE id = emp_id;

END;

/
```



```
SQL> CREATE OR REPLACE PROCEDURE update_salary (
         emp_id IN NUMBER,
         emp_salary IN OUT NUMBER
     ) AS
     BEGIN
         -- Increase the salary by 10%
         emp_salary := emp_salary * 1.10;

         -- Update the salary in the table
         UPDATE employees SET salary = emp_salary WHERE id = emp_id;
     END;
     /

Procedure created.
```

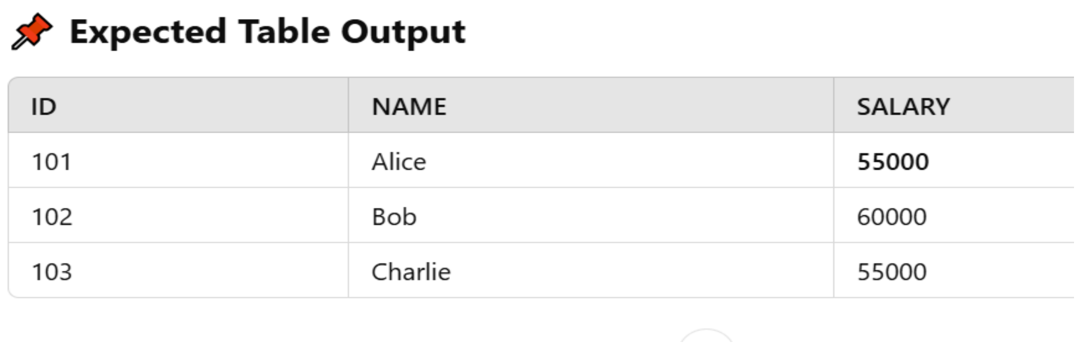
Step 5: Execute the Procedure

```
DECLARE v_salary NUMBER;

BEGIN

   -- Get the initial salary of employee 101

   SELECT salary INTO v_salary FROM employees WHERE id = 101;



   -- Call the procedure

   update_salary(101, v_salary);



   -- Display updated salary

   DBMS_OUTPUT.PUT_LINE('Updated Salary: ' || v_salary);

END;

/
```



```
SQL> DECLARE v_salary NUMBER;
     BEGIN
         -- Get the initial salary of employee 101
         SELECT salary INTO v_salary FROM employees WHERE id = 101;

         -- Call the procedure
         update_salary(101, v_salary);

         -- Display updated salary
         DBMS_OUTPUT.PUT_LINE('Updated Salary: ' || v_salary);
     END;
     /
Updated Salary: 55000

PL/SQL procedure successfully completed.
```

Step 6: Verify Updated Table Data



📌 Expected Table Output

ID	NAME	SALARY
101	Alice	**55000**
102	Bob	60000
103	Charlie	55000

Benefits of Stored Procedures

Improves Performance – Reduces SQL execution time.
Encapsulation – Business logic is stored in one place.
Security – Users can execute procedures without direct table access.

Reusability – The same procedure can be executed multiple times.

EXPERIMENT – 11

AIM : Develop programs using before and after triggers, row and statement triggers and instead of triggers.

Trigger : A trigger in SQL is a special type of stored procedure that is automatically executed or "triggered" when certain events occur on a specific table or view in a database. Triggers are typically used to enforce business rules, perform validation, or automate certain actions like logging or updating related data.

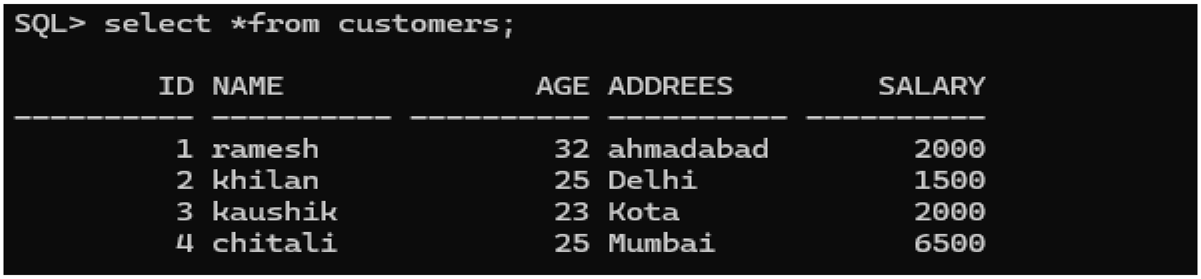
- **BEFORE Trigger: Executes before the operation (e.g., before inserting or updating data).**
- **AFTER Trigger: Executes after the operation (e.g., after inserting, updating, or deleting data).**
- **ROW Trigger: Executes once for each row affected.**
- **STATEMENT Trigger: Executes once for the entire SQL statement.**
- **INSTEAD OF Trigger: Replaces the operation (e.g., in views to insert, update, or delete underlying data).**

These triggers allow you to automate various actions such as validation, logging, and even preventing certain operations.

create table customers and insert values into it.

```
CREATE  table  CUSTOMERS(

                ID number(3),

                Name varchar2(10),

                Age number(3),

                Address varchar2(10),

                 Salary number(10,2)

);

SQL> insert into customers values(1,'ramesh',32,'ahmedabad',2000);

1 row created.

SQL> insert into customers values(2,'khilan',25,'Delhi',1500);
```

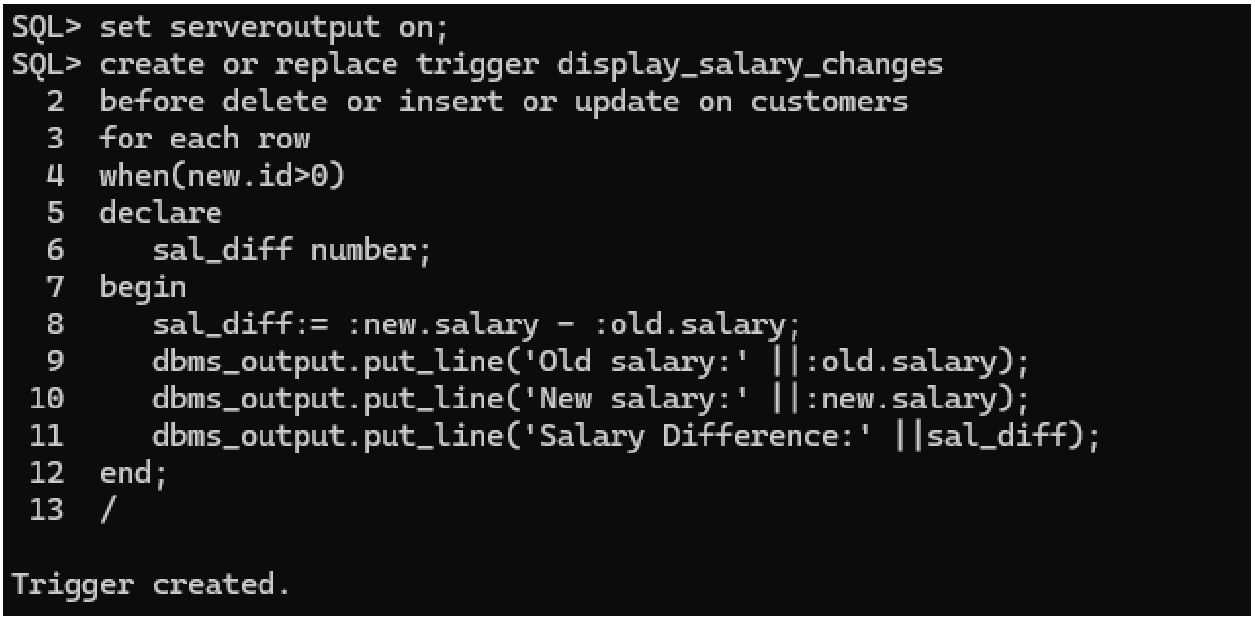
```
1 row created.

SQL> insert into customers values(3,'kaushik',23,'Kota',2000);

1 row created.

SQL> insert into customers values(4,'chitali',25,'Mumbai',6500);

1 row created.

SQL> select *from customers;
```



```
SQL> select *from customers;

        ID NAME              AGE ADDREES        SALARY
---------- ---------- ---------- ---------- ----------
         1 ramesh             32 ahmadabad        2000
         2 khilan             25 Delhi            1500
         3 kaushik            23 Kota             2000
         4 chitali            25 Mumbai           6500
```

<u>PL/SQL Code</u> : for creation of trigger while insert / update records into a table.

```
SQL> ed 11a

CREATE OR REPLACE TRIGGER display_salary_changes BEFORE DELETE OR INSERT OR UPDATE
ON customers FOR EACH ROW

WHEN (NEW.ID > 0) DECLARE

sal_diff number; BEGIN

sal_diff := :NEW.salary - :OLD.salary; dbms_output.put_line('Old salary: ' || :OLD.salary);
dbms_output.put_line('New salary: ' || :NEW.salary); dbms_output.put_line('Salary
difference: ' || sal_diff);

END;

/
```



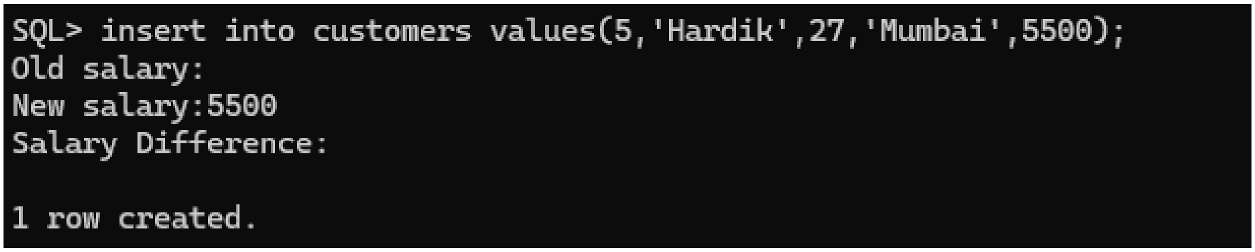
```
SQL> set serveroutput on;
SQL> create or replace trigger display_salary_changes
     before delete or insert or update on customers
     for each row
     when(new.id>0)
     declare
        sal_diff number;
     begin
        sal_diff:= :new.salary - :old.salary;
        dbms_output.put_line('Old salary:' ||:old.salary);
        dbms_output.put_line('New salary:' ||:new.salary);
        dbms_output.put_line('Salary Difference:' ||sal_diff);
     end;
     /

Trigger created.
```

Sql> @11a

Trigger created

1) **Query to insert a new customer detail into the customer table.**

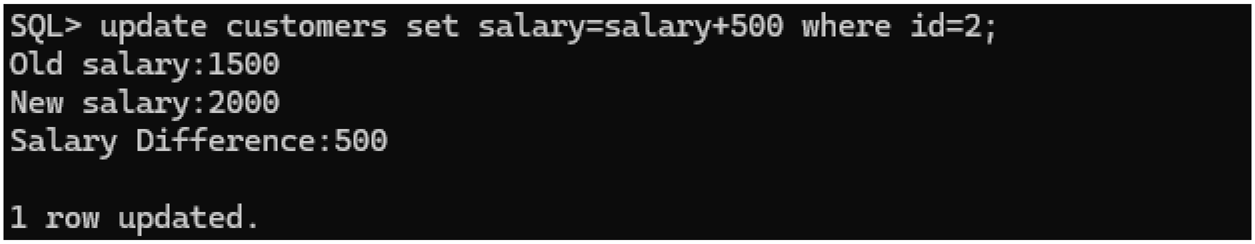
    insert into customers values(5,'Hardik',27,'Mumbai',5500);



```
SQL> insert into customers values(5,'Hardik',27,'Mumbai',5500);
Old salary:
New salary:5500
Salary Difference:

1 row created.
```

2) **Query to update customers salary as salary+500 for customer id=2**

SQL> update customers set salary=salary+500 where id=2;



```
SQL> update customers set salary=salary+500 where id=2;
Old salary:1500
New salary:2000
Salary Difference:500

1 row updated.
```

EXPERIMENT-12

Create a table and perform the search operation on table using indexing and non-indexing techniques

An index is a database object that improves the speed of data retrieval operations on a table.

The DBMS can use indexes to find data more quickly.

- **Without an index: The query might take more time, especially with a large dataset.**

- **With an index: The database can quickly locate the relevant rows based on the index, significantly reducing the search time.**

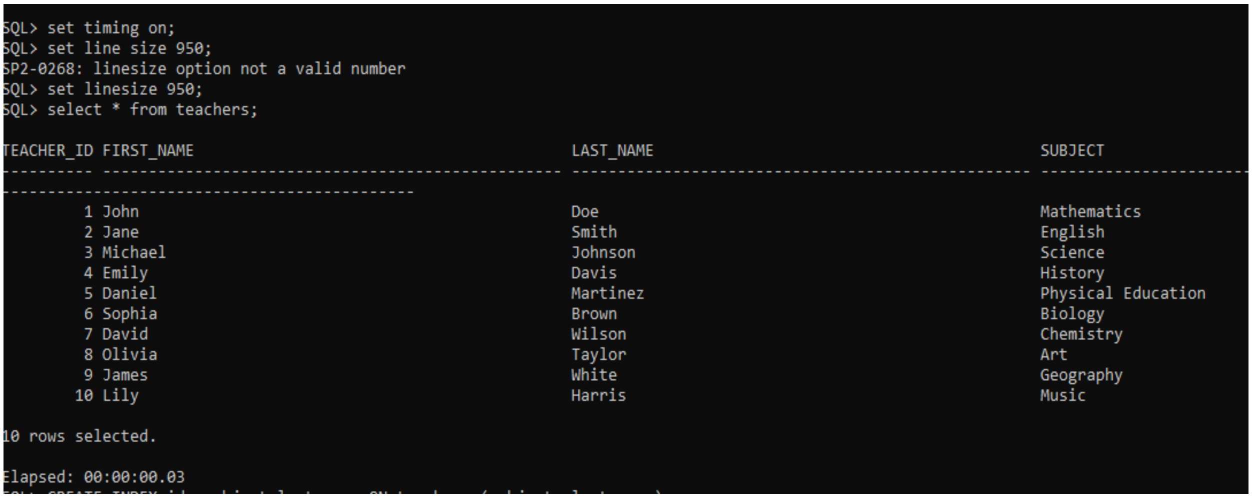
```
CREATE TABLE teachers (

   teacher_id INT PRIMARY KEY,

   first_name VARCHAR(50) NOT NULL,

   last_name VARCHAR(50) NOT NULL,

   subject VARCHAR(100)

);


INSERT INTO teachers (teacher_id, first_name, last_name, subject)

VALUES (1, 'John', 'Doe', 'Mathematics');
```

```
INSERT INTO teachers (teacher_id, first_name, last_name, subject)
VALUES (2, 'Jane', 'Smith', 'English');

INSERT INTO teachers (teacher_id, first_name, last_name, subject)
VALUES (3, 'Michael', 'Johnson', 'Science');

INSERT INTO teachers (teacher_id, first_name, last_name, subject)
VALUES (4, 'Emily', 'Davis', 'History');
INSERT INTO teachers (teacher_id, first_name, last_name, subject)
VALUES (5, 'Daniel', 'Martinez', 'Physical Education');

INSERT INTO teachers (teacher_id, first_name, last_name, subject)
VALUES (6, 'Sophia', 'Brown', 'Biology');

INSERT INTO teachers (teacher_id, first_name, last_name, subject)
VALUES (7, 'David', 'Wilson', 'Chemistry');

INSERT INTO teachers (teacher_id, first_name, last_name, subject)
VALUES (8, 'Olivia', 'Taylor', 'Art');

INSERT INTO teachers (teacher_id, first_name, last_name, subject)
VALUES (9, 'James', 'White', 'Geography');

INSERT INTO teachers (teacher_id, first_name, last_name, subject)
```

VALUES (10, 'Lily', 'Harris', 'Music');

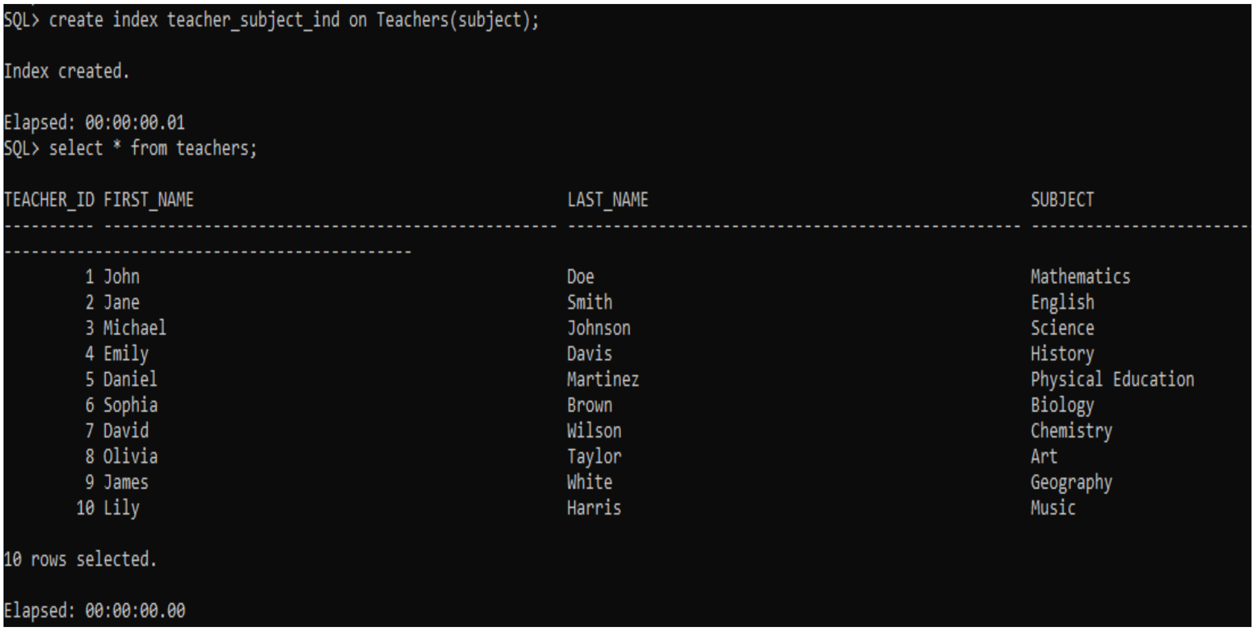
Set timing on;

Select * from teachers;



```
SQL> set timing on;
SQL> set line size 950;
SP2-0268: linesize option not a valid number
SQL> set linesize 950;
SQL> select * from teachers;

TEACHER_ID FIRST_NAME                                         LAST_NAME                                          SUBJECT
---------- -------------------------------------------------- -------------------------------------------------- -----------------------
---------------------------------------------
         1 John                                               Doe                                                Mathematics
         2 Jane                                               Smith                                              English
         3 Michael                                            Johnson                                            Science
         4 Emily                                              Davis                                              History
         5 Daniel                                             Martinez                                           Physical Education
         6 Sophia                                             Brown                                              Biology
         7 David                                              Wilson                                             Chemistry
         8 Olivia                                             Taylor                                             Art
         9 James                                              White                                              Geography
        10 Lily                                               Harris                                             Music

10 rows selected.

Elapsed: 00:00:00.03
```

create index teacher_subject_ind on Teachers(subject);



```
SQL> create index teacher_subject_ind on Teachers(subject);

Index created.

Elapsed: 00:00:00.01
SQL> select * from teachers;

TEACHER_ID FIRST_NAME                                         LAST_NAME                                          SUBJECT
---------- -------------------------------------------------- -------------------------------------------------- -----------------------
---------------------------------------------
         1 John                                               Doe                                                Mathematics
         2 Jane                                               Smith                                              English
         3 Michael                                            Johnson                                            Science
         4 Emily                                              Davis                                              History
         5 Daniel                                             Martinez                                           Physical Education
         6 Sophia                                             Brown                                              Biology
         7 David                                              Wilson                                             Chemistry
         8 Olivia                                             Taylor                                             Art
         9 James                                              White                                              Geography
        10 Lily                                               Harris                                             Music

10 rows selected.

Elapsed: 00:00:00.00
```