FYIT SEM II

SUBJECT: PL/SQL –I PRACTICAL MANUAL

BY:PROF.AJAY PASHANKAR

NOTE: IN THIS MANUAL YOU WILL FIND MANY EXTRA PRACTICAL'S APART FROM YOUR SYLLABUS, IT MIGHT HAPPEN THAT SOME TOPICS DON'T MATCH EXACTLY TO SYLLABUS IT JUST A GUIDELINE TO IMPLEMENT PRACTICAL

FOLLOW THIS INDEX AFTER PAGE NO :46

Writing PL/SQL block with basic programming constructs

1) Print "Hello World" using PL/SQL block.

```
Set Serveroutput on;
BEGIN
dbms_output.put_line('Hello World');
end;
/
```

OUTPUT

Hello World

2) Print "Loop exited" after while loop completed

```
Set Serveroutput on;
 DECLARE
i number :=1;
BEGIN
loop
i:=i+1;
EXIT WHEN I>4;
end loop;
dbms_output.put_line('loop exited');
end;
/
```

OUTPUT:

Loop exited

3) Print "hello" five times

```
Set Serveroutput on;
DECLARE
i number :=1;
BEGIN
loop
i:=i+1;
dbms_output.put_line('hello');
EXIT WHEN i>5;
end loop;
end;
/
```

OUTPUT:

```
hello
hello
hello
hello
hello
```

4) Print hello 5 times with numbering

```
Set Serveroutput on;
```

```
DECLARE
i number :=0;
BEGIN
loop
i:=i+1;
dbms_output.put_line(TO_CHAR(i) || ' hello');
if i>5 then
Exit;
end if;
end loop;
end;
/
```

OUTPUT:

1 hello
2 hello
3 hello
4 hello
5 hello

5) Write a PL/SQL code block that will accept an account number from the user, check if the users balance is less than the minimum balance, only then deduct Rs. 100/- from the balance. The process is fired on acc_mstr table.

```
SQL> create table acc_mstr2(acc_no number
  2  (10),curr_bal number(10,2));

SQL> select * from acc_mstr2;


   ACC_NO   CURR_BAL
---------- ----------
       1      1134
       2     12567
       5     65434
       3    657687
       6     34567


declare
  mcur number(10,2);
  mac number(10);
  begin
 mac:=&mac;
  select curr_bal into mcur from acc_mstr where acc_no=mac;
 if mcur<2000 then
 update acc_mstr set curr_bal=curr_bal-100 where acc_no=mac;
dbms_output.put_line('Updated');
end if;
end;
/
```

```
Enter value for mac: 1
old   5:  mac:=&mac;
new   5:  mac:=1;

PL/SQL procedure successfully completed.

SQL> select * from acc_mstr;

   ACC_NO   CURR_BAL
---------- ----------
       1      1134
       2     12567
       5     65434
       3    657687
       6     34567
```

1) **Create a table employee(empid,empname,doj,salary). Write a PL/SQL block increased the salary of the employee if his date of joining is before specified day by 15% otherwise increased by 5% (In the if condition).**

```
CREATE TABLE Employee
 (  Empid varchar2(6) constraint emp_pk primary key,
 Empname varchar2(15),
 DOJ Date,
 Salary number(10,2)  );
Table created.


INSERT INTO Employee
VALUES('E001','Deepti','26-Feb-2010',10000);

INSERT INTO Employee
VALUES ('E002','Harshada','15-May-2010',10000);

INSERT INTO Employee
VALUES ('E003','Roshan','7-Aug-2009',10000);
```

```
SQL> select * from employee;
EMPID  EMPNAME        DOJ         SALARY
------ --------------- --------- ---------- -----------
E001   Deepti          26-FEB-10      10000
E002   Harshada        15-MAY-10      10000
E003   Roshan          07-AUG-09      10000
```

```
DECLARE
eid varchar2(6);
doj date;
dt Date;
BEGIN
dt:='26-Feb-2010';
eid:=&eid;
SELECT DOJ INTO doj FROM Employee WHERE eid=Empid;
IF (doj<=dt) THEN
UPDATE Employee set Salary=Salary+0.15*Salary WHERE eid=Empid;
ELSE
UPDATE Employee set Salary=Salary+0.05*Salary WHERE eid=Empid;
END IF;
END;
/
```

OUTPUT:

a. Salary increased by 15%

```
Enter value for eid: 'E001'
old   7: eid:=&eid;
new   7: eid:='E001';
PL/SQL procedure successfully completed.

SQL> select * from Employee;
EMPID  EMPNAME        DOJ         SALARY
```

```
------ --------------- --------- ---------- -----------
E001   Deepti           26-FEB-10      11500
E002   Harshada         15-MAY-10      10000
E003   Roshan          07-AUG-09      10000
```

b. Salary increased by 5%

```
Enter value for eid: 'E002'
old   7: eid:=&eid;
new   7: eid:='E002';
PL/SQL procedure successfully completed.
```

```
SQL> select * from Employee;

EMPID  EMPNAME        DOJ         SALARY
------ --------------- --------- ----------
E001   Deepti         26-FEB-10        11500
E002   Harshada        15-MAY-10      10500
E003   Roshan         07-AUG-09      10000
```

2) Create a table place(floor,room_number,number_of_seats). Write a PL/SQL block to comment on type place as fairly small, little bigger, biggest depending on number_of_seats for a given room_no.

```
CREATE TABLE place
(Floor Number(4),
Room_no Number(4) CONSTRAINT p_1 PRIMARY KEY,
No_of_seats Number(6) );
Table created.

INSERT INTO place VALUES(1,101,65);

INSERT INTO place VALUES (2,201,135);

INSERT INTO place VALUES (3,301,40);
```

```
DECLARE
rmid number(4);
seats number(6);
BEGIN
rmid:=&rmid;
SELECT No_of_seats INTO seats FROM place WHERE rmid=Room_no;
DBMS_OUTPUT.PUT_LINE('No. of seats '||seats);
IF (seats<=60) THEN
DBMS_OUTPUT.PUT_LINE ('Fairly small');
ELSIF (seats>60 and seats<=100) THEN
DBMS_OUTPUT.PUT_LINE ('Little Bigger');
ELSE
DBMS_OUTPUT.PUT_LINE ('Biggest');
END IF;
END;
/
```

OUTPUT:

```
Enter value for rmid: 201
old   5: rmid:=&rmid;
new   5: rmid:=201;
No. of seats 135
Biggest

PL/SQL procedure successfully completed.
```

3) Create a table lecturer (id,name,major_subject,doj). Write a PL/SQL block with case when statement which print course name depending on major subject for specified lecture id.

```
CREATE TABLE lecturer
( Lecturer_id Varchar2(4) CONSTRAINT z PRIMARY KEY,
 Name Varchar2(15),
 Major_sub Varchar2(10),
 DOJ Date );
Table created.

INSERT INTO lecturer VALUES('L001','Mayekar','Accounts','25-Jun-1999');

INSERT INTO lecturer VALUES('L002','Kulkarni','Sanskrit','26-Feb-2000');

INSERT INTO lecturer VALUES('L003','Adarkar mam','Physics','12-Jan-1990');




DECLARE
lect_id Varchar2(4);
sub Varchar2(10);
BEGIN
lect_id:=’&lect_id’;
SELECT Major_sub INTO sub FROM lecturer WHERE lect_id=Lecturer_id;
CASE upper(sub)
WHEN 'ACCOUNTS' THEN
DBMS_output.put_line('Commerce');
WHEN 'SANSKRIT' THEN
DBMS_output.put_line('Arts');
WHEN 'PHYSICS' THEN
DBMS_output.put_line(‘Science');
ELSE
DBMS_output.put_line('Invalid subject');
END CASE;
END;
/
```

OUTPUT:

```
Enter value for lect_id: 'L001'
old   5: lect_id:=&lect_id;
```

```
new   5: lect_id:='L001';
Commerce

PL/SQL procedure successfully completed.
```

4) Write a PL/SQL code block to calculate the area of circle for a value of radius 2 to 7 and corresponding value of calculated area in an empty table area(radius, area).

```
CREATE TABLE area
(radius number(2),
area number(10,2)
);
Table created.
```

Using Unconstrain loop

```
DECLARE
rad area.radius% Type:=2;
area area.area% Type;
BEGIN
LOOP
area:=3.142*rad*rad;
insert into area VALUES(rad,area);
rad:=rad+1;
EXIT WHEN rad>7;
END LOOP;
END;
/
```

Using Unconstraint loop with if

```
DECLARE
rad area.radius% Type:=2;
area area.area% Type;
BEGIN
LOOP
rad:=rad+1;
```

```
IF (rad>7) THEN
EXIT;
END IF;
area:=3.142*rad*rad;
INSERT INTO area VALUES(rad,area);
END LOOP;
END;
/
```

Using While loop

```
DECLARE
rad area.radius% Type:=3;
area area.area% Type;
BEGIN
WHILE(rad<8)



LOOP
area:=3.142*rad*rad;
INSERT INTO area VALUES(rad,area);
rad:=rad+1;
END LOOP;
END;
```

Using for loop

```
DECLARE
rad area.radius% Type:=3;
area area.area% Type;
BEGIN
FOR rad IN 3..7
LOOP
area:=3.142*rad*rad;
INSERT INTO area VALUES(rad,area);
END LOOP;
```

```
END;
/
```

Using unconstraint loop with goto statement

```
DECLARE
rad area.radius% Type:=2;
area area.area% Type;
BEGIN
LOOP
rad:=rad+1;
area:=3.142*rad*rad;
IF(rad>7) THEN
GOTO display;
END IF;
INSERT INTO area VALUES(rad,area);
END LOOP;
<<display>>
DBMS_OUTPUT.PUT_LINE('U can check table details');
END;
/
```

5)Using Null in conditional statement
Create table Mytable(total_col,Num_col,Char_col,Date_col).Insert into the table using Pl/SQL block.

```
CREATE TABLE Mytable
(total_col number(2),
num_col number(2),
char_col number(2),
date_col number );
```
Table created.

```
DECLARE
tc number(2);
nc number(2);
cc number(2);
dc number(2);
BEGIN
tc:=&tc;
nc:=&nc;
cc:=&cc;
dc:=&dc;
IF(nc=NULL) THEN
nc:=0;
END IF;
IF(dc=NULL) THEN
dc:=0;
END IF;
IF(nc=NULL) THEN
nc:=0;
END IF;
INSERT INTO Mytable VALUES(tc,nc,cc,dc);
END;
/
```

OUTPUT:

```
Enter value for tc: 2
old   7: tc:=&tc;
new   7: tc:=2;
Enter value for nc: NULL
old   8: nc:=&nc;
new   8: nc:=NULL;
Enter value for cc: 5
old   9: cc:=&cc;
new   9: cc:=5;
Enter value for dc: NULL
old  10: dc:=&dc;
new  10: dc:=NULL;

PL/SQL procedure successfully completed.
```

Procedures and Functions in PL/SQL Block

1) Write a Procedure that displays "Hello World" on prompt.

```
CREATE OR REPLACE PROCEDURE display
AS
BEGIN
DBMS_OUTPUT.PUT_LINE('Hello World');
END;
/
Procedure created.
```

OUTPUT:

```
SQL> call display();
Hello World
Call completed.
```

2) Write a Procedure that accepts two nos. and swap them . Display the swap nos.

Defining the function outside the block

```
CREATE OR REPLACE PROCEDURE swap(a IN OUT number,b IN OUT number)
AS
 t number;
BEGIN
t:=a;
a:=b;
b:=t;
END;
/
```

```
DECLARE
a number(2);
b number(2);
BEGIN
a:=&a;
b:=&b;
DBMS_OUTPUT.PUT_LINE('Before swapping Value of a'||a||'  Value of b'||b);
swap(a,b);
DBMS_OUTPUT.PUT_LINE('Value of a'||a||'  Value of b'||b);
END;
/
```

Defining the function inside the block

```
DECLARE
a number(2);
b number(2);
PROCEDURE swap(a IN OUT number,b IN OUT number)
AS t number;
BEGIN
t:=a;
a:=b;
b:=t;
END;
BEGIN
a:=&a;
b:=&b;
DBMS_OUTPUT.PUT_LINE('Before swapping  a= '||a||' b= '||b);
swap(a,b);
DBMS_OUTPUT.PUT_LINE('After swapping  a= '||a||'   b= '||b);
END;
/
```

OUTPUT:

```
SQL> call swap(10,20);
Before swapping  a=10  b=20
```

```
After swapping  a=20   b=10
Call completed.
```

3) Create a table lecturer and write a procedure that accepts a lecturer id and displays the major subject.

```
CREATE OR REPLACE PROCEDURE lec(id in out varchar2)
as sub varchar2(10);
BEGIN
SELECT Major_sub INTO sub FROM lecturer WHERE id=Lecturer_id;
id:=sub;
END;
/
DECLARE
id varchar2(10);
BEGIN
id:=&id;
lec(id);
DBMS_OUTPUT.PUT_LINE('Major subjects :'||id);
END;
/
```

OUTPUT:

```
Enter value for id: 'L001'
old   4: id:=&id;
new   4: id:='L001';
Major subjects :Accounts
PL/SQL procedure successfully completed
```

---

4) Write a PL/SQL block to define procedure to insert data in the employee table

```
DECLARE
eid varchar2(6):='e34';
ename varchar2(15):='fsaf';
```

```
dt date:='31-Aug-2009';
sal number(10,2):=10000;
PROCEDURE insert01(e varchar2,nm varchar2,dt date,sal number)
AS
BEGIN
INSERT INTO employee VALUES(e,nm,dt,sal);
END;
BEGIN
Insert01(eid,ename,dt,sal);
END;
/

PL/SQL procedure successfully completed.
```

OUTPUT:

```
SQL> SELECT * FROM employee;
EMPID  EMPNAME        DOJ         SALARY
------ -------------- --------- ----------
E001   Deepti         26-FEB-10     11500
E002   Harshada       15-MAY-10     10500
E003   Roshan         07-AUG-09     11500
e65    deepti         15-JAN-10      4000
E1     harshu         26-FEB-90     45555
e34    fsaf           31-AUG-09     10000
6 rows selected.
```

---

1) Create a function that compares two given numbers and displays which one is greater than the other.

```
SQL>  create or replace function
  compare(a in number,b in number)
  return boolean
  as
  begin
```

```
 if a>b then
 return true;
 else
 return false;
 end if;
 end compare;
Function created.


SQL> declare
 a boolean;
 begin
 a:=compare(55,50);
 if a=true then
 dbms_output.put_line('a is greater than b');
 else
 dbms_output.put_line('a is less than b');
 end if;
 end;
/
```

OUTPUT:

```
a is greater than b
PL/SQL procedure successfully completed.
```

---

2)Create a function that calculates the factorial of the given positive number.

```
SQL> create or replace function
 factorial(num number)
  return number
 as
 fact number:=1;
 res number:=1;
 begin
 if num=0 then
 return 1;
 else
 while(fact <= num)
 loop
 res:= res * fact;
```

```
 fact:=fact+1;
 end loop;
 return res;
 end if;
end factorial;
```

Function created.

OUTPUT:

```
SQL> exec dbms_output.put_line(factorial(5));
120
```

3) Create a function that counts the total number of records from a table .

```
SQL> create or replace function
  count
   return number
   as
   vcount number;
   begin
   select count(*) into vcount from employee;
   return vcount;
  end count;
```

Function created.

OUTPUT:

```
SQL> select count from dual;
COUNT
----------
2
```

4) **Create a function that generates the Fibonacci sequence till the given range.**

```
SQL> create or replace  function  fibonacci (b number)
 return number
 As
 ft number:=0;
 st number:=1;
 nt number;
 begin
 dbms_output.put_line(ft);
 dbms_output.put_line(st);
 for i in 1..b-2
 loop
 nt :=st+ft;
 ft:=st;
 st:=nt;
 dbms_output.put_line(nt);
 end loop;
 return null;
 end fibonacci;

Function created.
```

OUTPUT:

```
SQL> declare
begin
fibonacci(5);
end
0
1
1
2
3
end
```

Study of transaction and locks

Commit and entire rollback:

1)Create a table Ticket. Insert 5 meaningful records in it. Perform the action of booking ticket. Either Commit transaction or roll the whole transaction.

```
SQL> create table ticket(train_id number(4), t_name varchar2(20), avail_tickets number(5));
Table created.
SQL> insert into ticket values(1001,'Udyaan',200);
1 row created.

SQL>  insert into ticket values(1002,'Mandvi',100);
1 row created.

SQL>  insert into ticket values(1003,'Maharashtra',300);
1 row created.
SQL> select * from ticket;

 TRAIN_ID T_NAME              AVAIL_TICKETS
---------- -------------------- -------------
     1001 Udyaan                       200
     1002 Mandvi                       100
     1003 Maharashtra                  300
```

```
Declare
 num number(5);
id number(5);
num1 number(5);
begin
id :=&id;
num :=&num;
update ticket set avail_tickets=avail_tickets - num where train_id = id;
select avail_tickets INTO num1 from ticket where train_id = id;
if num1<=0 then
```

```
    ROLLBACK;
   else
    COMMIT;
  end if;
   end;
  /
```

OUTPUT:

```
Enter value for id: 1001
old   6:  id :=&id;
new   6:  id :=1001;
Enter value for num: 10
old   7:  num :=&num;
new   7:  num :=10;

PL/SQL procedure successfully completed.

SQL> select * from ticket;

 TRAIN_ID T_NAME            AVAIL_TICKETS
---------- ------------------- -------------
    1001 Udyaan                 190
    1002 Mandvi                 100
    1003 Maharashtra              300
```

Partial Rollback:

2)Create a table Employee and customer. Insert 5 meaningful records in both tables. Perform following operations on tables by single transaction query. Committee Employee transaction and rollback Customer transaction.(Partial Rollback)

a) Update Employee by increasing salary of all employees by 5000. Increase Bonus By 1000 for specified employee.

b)Update Customer by increasing purchase by 5000 for a specified customer id.

```
 declare
 cid number(5);
```

```
eid varchar2(10);
begin
 cid:=&cid;
 eid:='&eid';
update employee set salary=salary+5000;
update employee set salary=salary+1000 where empid=eid;
savepoint s1;
update customer set purchase=purchase+5000 where cust_id=cid;
rollback to savepoint s1;
commit;
end ;
/
```

a)

```
1     declare
2     empavg number(10);
3     begin
4     update employee11 set esalary=esalary+5000;
5     savepoint s1;
6     update employee11 set ebonus=ebonus+1000;
7     select avg(esalary) into empavg from employee11;
8     if empavg>5000 then
9     rollback to savepoint s1;
10     end if;
11     commit;
12* end;
13  /
```

OUTPUT:

PL/SQL procedure successfully completed.

SQL> select * from employee11;

EMP_ID	ENAME	ESALARY	EBONUS
1	rahul	40000	1500
2	prashad	270000	2000
3	abhishek	270000	2001
4	pandey	520000	26000
5	robinwood	27000	1500

Experiencing Deadlock:

3)Create Employee and customer table. Simultaneously access Employee and customer in updated mode on two separate windows and ask access of other table to experience a deadlock.

Select – FOR UPDATE without NOWAIT Clause.

CLIENT 1

```
SQL> select * from employee14 FOR UPDATE;

E_NAME               E_SALARY
-------------------- ----------
abcd                  10000
xyz                  31000
lmn                   20000

SQL>
```

CLIENT 2

```
SQL> select * from customer14 FOR UPDATE;

    C_ID C_NAME
---------- ----------
       1 abc
       2 xyz
       3 lmn
       4 pqr
       5 wns
       6 srt

6 rows selected.
```

CLIENT 1

```
SQL> select * from customer14 FOR UPDATE;
(Wait state)
```

CLIENT 2

SQL> select * from employee14;
(Wait state)

Select – FOR UPDATE with NOWAIT Clause

CLIENT 1

SQL> select * from employee14 FOR UPDATE;

```
E_NAME               E_SALARY
-------------------- ----------
abcd                  10000
xyz                   31000
lmn                   20000

SQL>
```

CLIENT 2

SQL> select * from employee14 for update Nowait;

Implementing Cursors & Sequences

Implicit Cursors:

1)Create employee table,insert 5 meaningful record to it,Using Implicit cursor.Update the sql of specified employee id,if that id is available in the table.Using SQL%FOUND return the appropriate message.

```
create table employee(id number(5) primary key,first_name varchar2(10),last_name
varchar2(10),salary number(10));
insert into employee values(&id,'&first_name','&last_name',&salary);
Enter value for id: 3
Enter value for first_name: Deepali
Enter value for last_name: Patil
Enter value for salary: 343432
old   1: insert into employee values(&id ,'&first_name','&last_name',&salary)
new   1: insert into employee values(3 ,'Deepali','Patil',343432)
1 row created.
SQL> /
Enter value for id: 4
Enter value for first_name: Sukaya
Enter value for last_name: Shinde
Enter value for salary: 342323
old   1: insert into employee values(&id ,'&first_name','&last_name',&salary)
new   1: insert into employee values(4 ,'Sukaya','Shinde',342323)
1 row created.
SQL> /
Enter value for id: 5
Enter value for first_name: Snehal
Enter value for last_name: Sawant
Enter value for salary: 23456
old   1: insert into employee values(&id ,'&first_name','&last_name',&salary)
new   1: insert into employee values(5 ,'Snehal','Sawant',23456)

1 row created.
```

```
declare
eid number(10);
sal number(10);
begin
update employee set salary=&sal where id=&eid;
if SQL%found then
dbms_output.put_line('Salary Updated');
else
dbms_output.put_line('id not found');
end if;
end;
/
```

OUTPUT:

```
SQL> /
Enter value for sal: 343243
Enter value for eid: 1
old   5: update employee set salary=&sal where id=&eid;
new   5: update employee set salary=343243 where id=1;
Salary Updated

PL/SQL procedure successfully completed.
```

Explicit Cursor:

2)create a table employee ,insert 5 meaningful records to it. Write a cursor to accept id of the employee and print its first name and last name.

```
declare
fst_name varchar2(10);
eid varchar2(10);
cursor emp_curs is select empname from employee where empid='&eid';
```

```
 begin
open emp_curs;
 loop
fetch emp_curs into fst_name;
exit when emp_curs%NOTFOUND;
dbms_output.put_line(fst_name||' ');
 end loop;
close emp_curs;
end;
SQL> /
```

OUTPUT:

```
Enter value for eid: 3
old   5:  cursor emp_curs is select first_name,last_name from employee where id=&eid;
new   5:  cursor emp_curs is select first_name,last_name from employee where id=3;
Deepali  Patil

PL/SQL procedure successfully completed.
```

3)create a table employee,insert 5 meaningful records to it.Create a cursor to display first name and last name of employee till the last record is found by for loop.

```
declare
fst_name varchar2(10);
sal number(14,2);
counter varchar2(5);
cursor emp_curs is select empname,salary from employee;
begin
for e_curs in emp_curs
loop
dbms_output.put_line(e_curs.empname||' '||e_curs.salary);
counter:=to_char(emp_curs%rowcount);
end loop;
dbms_output.put_line('no of row processed='||counter);
end;
/
```

OUTPUT:

```
Gauri  Kudtarkar
Manorama  Chendge
Deepali  Patil
Sukaya  Shinde
Snehal  Sawant
no of rows processed=5
```

4) Create a table employees. Insert 5 meaning full records in it. With the help of cursor loop display the employee id from the given table.

```
 1      declare
 2      eid number(10);
 3       cursor emp_curs is select id from employee;
 4       begin
 5      open emp_curs;
 6      loop
 7      fetch emp_curs into eid;
 8      exit when emp_curs %notfound;
 9      dbms_output.put_line(eid);
10       end loop;
11      dbms_output.put_line('No of records processed='||to_char(emp_curs%rowcount));
12      close emp_curs;
13     end;
SQL> /
```

OUTPUT:

```
1
2
3
4
5
No of records processed =5

PL/SQL procedure successfully completed.
```

---

5)Create a table employee. Insert 5 meaning full records in it. Use PL/SQL, cursor and for loop to count no of records in the given table.

Using Cursor loop:

```
 1  declare
```

```
 2     eid number(10);
 3      cursor emp_curs is select id from employee;
 4      begin
 5     open emp_curs;
 6     loop
 7     fetch emp_curs into eid;
 8     exit when emp_curs %notfound;
 9     end loop;
10     dbms_output.put_line('No of records processed ='||to_char(emp_curs%rowcount));
11     close emp_curs;
12     end;
13  /
```

Using Cursor for loop:

```
SQL>   declare
         fst_name varchar2(10);
         lst_name varchar2(10);
         counter varchar2(5);
         cursor emp_curs is select first_name,last_name from employee;
         begin
        for e_curs in emp_curs
         loop
            counter:=to_char(emp_curs%rowcount);
       end loop;
       dbms_output.put_line('no of row processed='||counter);
  end;
/
```

OUTPUT:

```
no of row processed=5
PL/SQL procedure successfully completed.
```

Use of Subquery in the From Clause of Select Statement:

3) The bank manager has decided to mark all those accounts as Inactive (I) on which there are no transactions performed in the last 365 days. Whenever any such update takes place, a record for the same is maintained in the inactive_acct_mstr table comprising of the account number, the opening date and the type of account. Write a PL/SQL block to do the same.

```
SQL> select * from trans_mstr ;
TRNAS ACC_NO    TRANS_TYPE  TRANS_AMT TRANSDT
----- ---------- ---------- ---------- ---------
t1    1         credit        15000 25-AUG-10
t2    2         debit         24000 20-FEB-09
t3    3         debit         20000 23-FEB-08
t4    4         credit        17000 20-NOV-05


SQL> select * from acct_mstr1;

ACC_NO    STATUS    OPNDT
---------- ---------- ---------
1         a         23-JUL-10
2         a         10-JAN-09
3         a         05-DEC-06
4         a         05-NOV-04



declare
 cursor curs_notrans is
 select acc_no,status,opndt from acct_mstr1
 where acc_no in (select acc_no from trans_mstr
 group by acc_no having max(sysdate-transdt)>365);
 str_acc_no acct_mstr1.acc_no%type;
 str_status acct_mstr1.status%type;
 dt_opndt acct_mstr1.opndt%type;

 begin
 open curs_notrans;
 if curs_notrans%isopen then
```

```
loop
fetch curs_notrans into str_acc_no,str_status,dt_opndt;
exit when curs_notrans%notfound;
if curs_notrans%found then
update acct_mstr1 set status='s' where acc_no=str_acc_no;
insert into inactive_acct_mstr values(str_acc_no,dt_opndt);
end if;
end loop;

commit;
else
dbms_output.put_line('unable to open the cursor');
end if;
close curs_notrans;
end;
```

```
SQL>  select * from inactive_acct_mstr ;


ACC_NO    OPNDT
---------- ---------
2         10-JAN-09
3         05-DEC-06
4         05-NOV-04

SQL> select * from acct_mstr1;

ACC_NO    STATUS    OPNDT
---------- ---------- ---------
1         a         23-JUL-10
2         s         10-JAN-09
3         s         05-DEC-06
4         s         05-NOV-04
```

Parameterized Cursors:

7) Create a table employee. Insert 5 meaning full records in it. Create a parameterized cursor to find given employee in the table.

```
Declare
chkid varchar2(10);
Cursor curs_emp is select * from employee;
Cursor curs_chk (eid varchar2) is select empname from employee where empid=eid;
emp_name varchar2(24);
BEGIN
chkid:='&chkid';
open curs_emp;
open curs_chk(chkid);
fetch curs_chk into emp_name;
if curs_chk %found then
dbms_output.put_line('emp name:'|| emp_name);
else
dbms_output.put_line('no records found');
end if;
close curs_chk;
close curs_emp;
end;
/
```

OUTPUT:

```
Enter value for chkid: 1
old   7:  chkid:=&chkid;
new   7:  chkid:=1;
emp name:abc

PL/SQL procedure successfully completed.
```

Strongly Types Cursor Variables:

8)Create table Employee. Insert 3 meaningful records to it. Using strongly-typed cursor variable print the names of the employees whose salary is greater than 30000.

SQL> Select * from Employee14;

E_NAME	E_SALARY
abcd	10000
xyz	31000
lmn	20000

```
declare
type emp_type IS REF CURSOR RETURN EMPLOYEE%ROWTYPE;
curs3 emp_type;
emp_rec EMPLOYEE%ROWTYPE;
begin
open curs3 for select * from employee where salary<30000;
 dbms_output.put_line('name of employee having salary more than 30000');
loop
```

```
    fetch curs3 INTO emp_rec;
  EXIT WHEN curs3%NOTFOUND;
      dbms_output.put_line(emp_rec.empname);
   end loop;
  close curs3;
end;
/
```

Output:

```
name of employee having salary more than 30000
xyz

PL/SQL procedure successfully completed.
```

Weakly Typed Cursor Variable:

9) **Create table Employee. Insert 3 meaningful records to it. Using weakly-typed cursor variable print the names of the employees whose salary is greater than 30000. Create Customer table. Insert 5 meaningful records to it. Print the names of customers whose id is greater than 2 using same cursor variable.**

```
SQL> select * from customer14;

    C_ID C_NAME
---------- ----------
       1 abc
```

```
2 xyz
3 lmn
4 pqr
5 wns
6 srt

6 rows selected.

SQL> select * from employee14;

E_NAME              E_SALARY
-------------------- ----------
abcd                 10000
xyz                  31000
lmn                  20000
```

```
 1  declare
 2  type cust_type IS REF CURSOR ;
 3  curs3  cust_type;
 4  cust_rec  CUSTOMER14%ROWTYPE;
 5 emp_rec EMPLOYEE14%ROWTYPE;
 6  begin
 7  open curs3  for select * from customer14 where c_id>2;
 8  dbms_output.put_line('name of customer having ID  greater than 2');
 9  loop
10   fetch curs3 INTO cust_rec;
11   EXIT WHEN curs3%NOTFOUND;
12      dbms_output.put_line(cust_rec.c_name);
13   end loop;
14   close curs3;
15    open curs3 for select * from employee14 where e_salary>30000;
16    dbms_output.put_line('name of employee having salary more than 30000');
17    loop




18     fetch curs3 INTO emp_rec;
```

```
 19   EXIT WHEN curs3%NOTFOUND;
 20        dbms_output.put_line(emp_rec.e_name);
 21    end loop;
 22   close curs3;
 23    end;
SQL> /
```

Output:

```
name of customer having ID  greater than 2
lmn
pqr
wns
srt
name of employee having salary more than 30000
xyz

PL/SQL procedure successfully completed.
```

Sequences:

1) **Create a sequence by name seq_start1, which will generate numbers from 1to 1000 in ascending order with an interval of 1.**

```
SQL>  create sequence seq_start1
  start with 1
  increment by 1
  maxvalue 1000
SQL> /
```

OUTPUT:

Sequence created

```
SQL> select seq_start1.nextval from dual;
```

OUTPUT:

```
 NEXTVAL
---------
    1
```

```
SQL> select seq_start1.nextval from dual;
```

OUTPUT:

```
 NEXTVAL
----------
   2
```

2)Alter the sequence seq_start1to change the interval between two numbers as 2.

```
SQL> alter sequence seq_start1
  increment by 2;
```

OUTPUT:

Sequence altered.

```
SQL> select seq_start1.nextval from dual;
```

OUTPUT:

```
NEXTVAL
----------
    4
```

```
SQL> select seq_start1.nextval from dual;
```

OUTPUT:

```
 NEXTVAL
----------
     6
```

3)Destroy the sequence seq_start1.

SQL> drop sequence seq_start1;

OUTPUT:

Sequence dropped.

4)Create table supp1 .Create a sequence seq_supp1 which will generate numbers from 100 in ascending order with an interval of 1.Use seq_supp1 to generate id for inserting data into supp1 table using procedure.

```
SQL> create table supp1 (id number(5),name varchar2(10));
Table created.
SQL> create sequence seq_supp1
start with 100
increment by 1;
Sequence created.
SQL> create or replace procedure val_ins1
(P_name IN varchar2) is
begin
insert into supp1 values (seq_supp1.nextval, P_name);
commit;
end val_ins1;
/
```

OUTPUT:

```
Procedure created.
SQL> exec val_ins1('ABC');
PL/SQL procedure successfully completed.
SQL> exec val_ins1('XYZ');
PL/SQL procedure successfully completed.
SQL> select * from supp1;
ID NAME
---------- ----------
100 ABC
    101        Z
```

5)Create a sequence by name seq_cycle, which will generate numbers from 1000 to 100 in descending order with an interval of 100 (decrementing sequence).

```
SQL> create sequence seq_cycle
maxvalue 10000
start with 300
minvalue 100
increment by -100
nocycle;
```

OUTPUT:

```
Sequence created.
SQL> select seq_cycle.nextval from dual;
NEXTVAL
----------
300
SQL> select seq_cycle.nextval from dual;
NEXTVAL
----------
200
Sequence created.
SQL> select seq_cycle.nextval from dual;
NEXTVAL
----------
100

SQL> select seq_cycle.nextval from dual;
```

PRACTICAL -1

AIM: Basics of triggers & its syntax.

- **A trigger is a PL/SQL block structure or Stored procedures that defines an action when same database event occurs.**
- **A trigger is fired when a DML statements like Insert, Delete, and Update is executed on a database table.**
- **A trigger is triggered automatically when an associated DML statement is executed.**

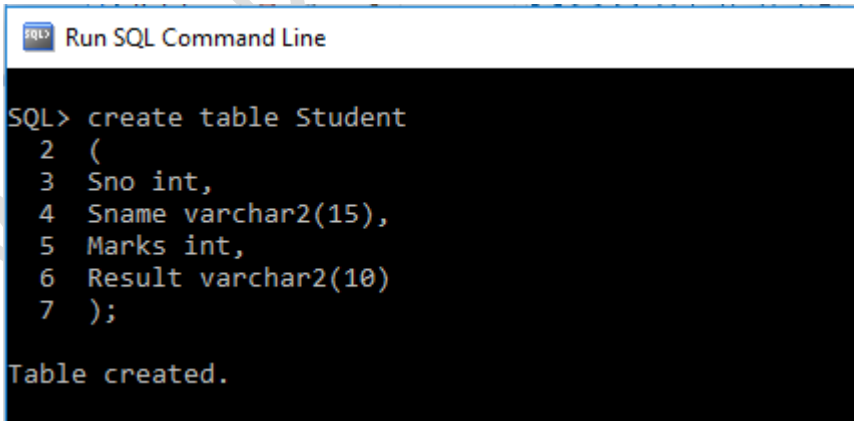
Syntax for Creating Triggers-

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] | UPDATE [OR] | DELETE} [OF col_name] ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
BEGIN
 --- sql statements
END;
```

Creation of table: -

```
create table Student
(
Sno int,
Sname varchar2(15),
Marks int,
Result varchar2(10)
);
```

OUTPUT: -

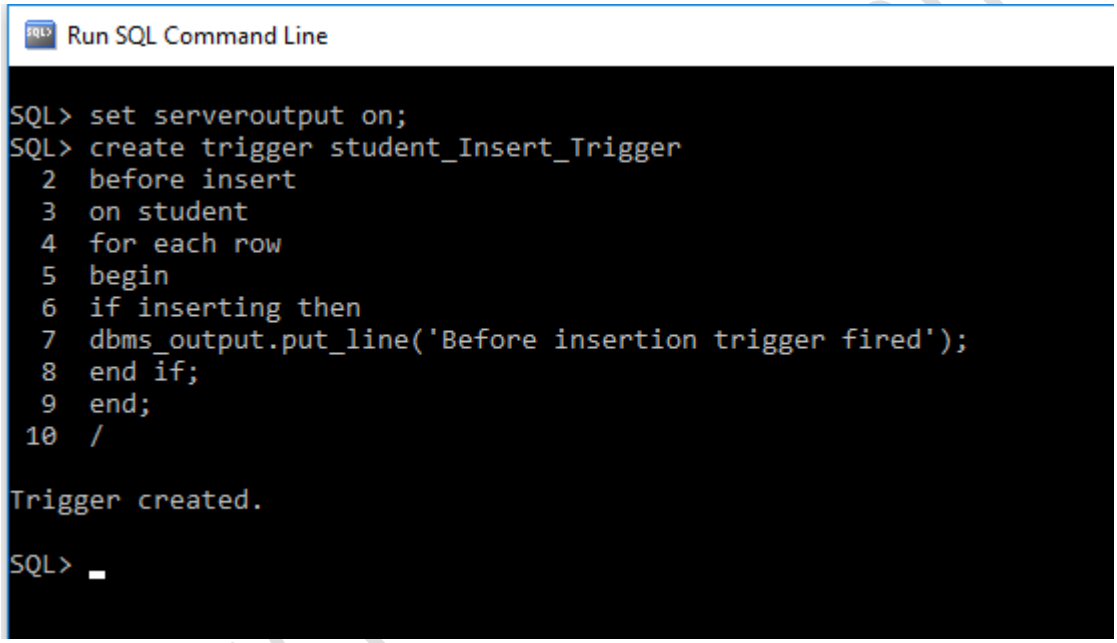


Creation of trigger: -

```
set serveroutput on;
```

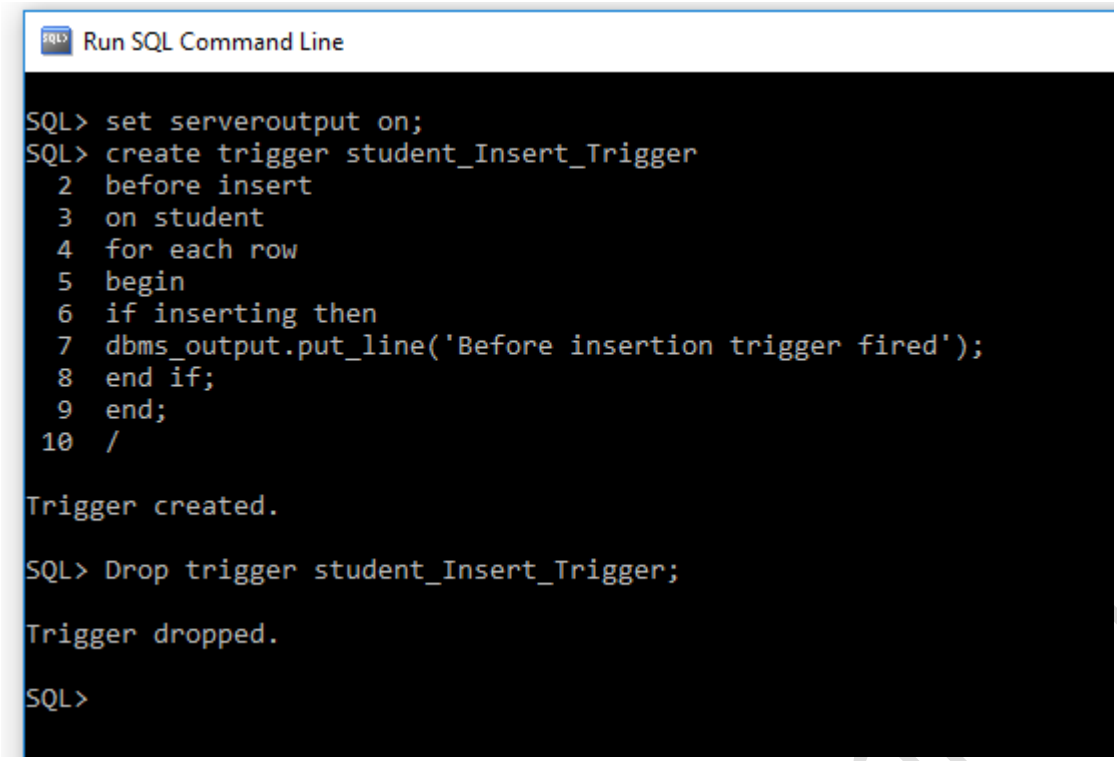
```
create trigger student_Insert_Trigger
Before insert
on student
for each row
begin
if inserting then
dbms_output.put_line('After insertion trigger fired');
end if;
end;
/
```

OUTPUT: -



Dropping of triggers: -

Drop trigger Student_Insert_Trigger



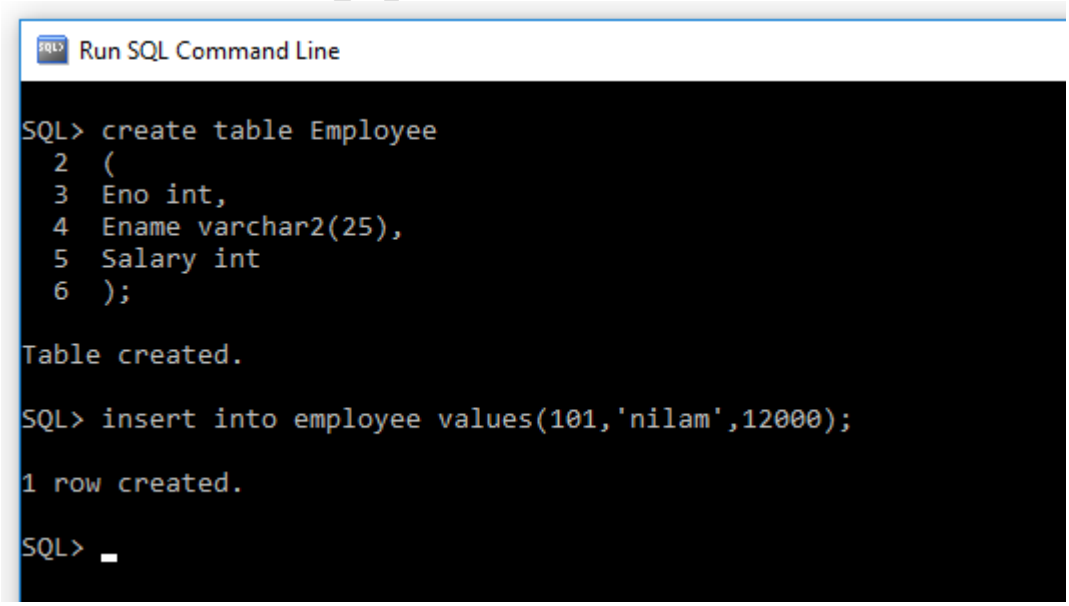
PRACTICAL -2

AIM: Create a trigger before update statement level.

Creation of table: -

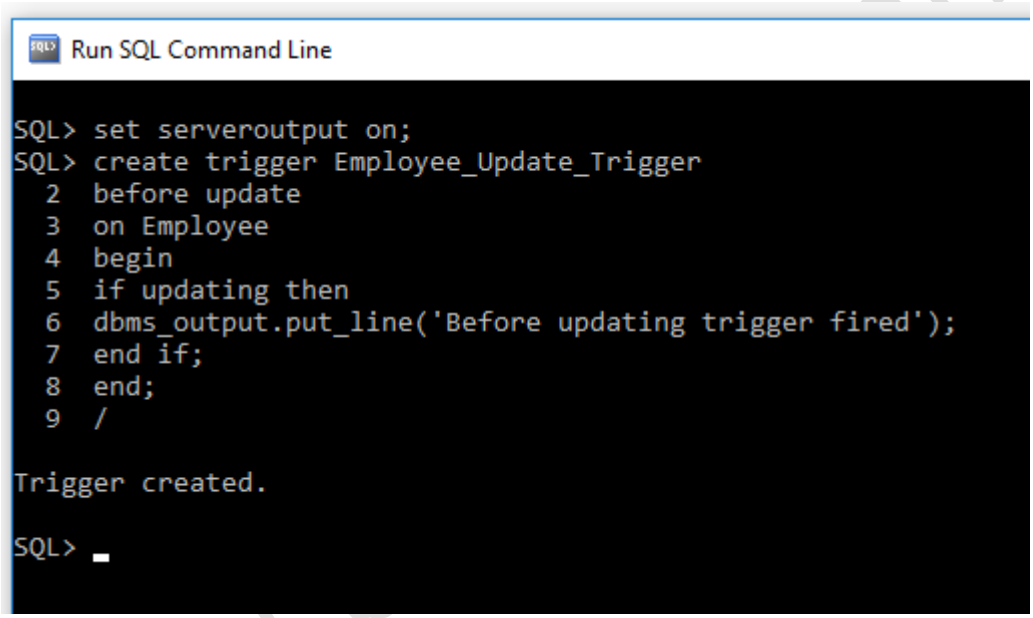
```
create table Employee
(
Eno int,
Ename varchar2(25),
Salary int
);
```

OUTPUT: -



Creation of trigger: -

```
set serveroutput on;
create trigger Employee_Update_Trigger
before update
on Employee
begin
if updating then
dbms_output.put_line('Before updating trigger fired');
end if;
end;
/
```



OUTPUT: -

Run SQL Command Line

```
SQL> set serveroutput on;
SQL> create trigger Employee_Update_Trigger
  2  before update
  3  on Employee
  4  begin
  5  if updating then
  6  dbms_output.put_line('Before updating trigger fired');
  7  end if;
  8  end;
  9  /

Trigger created.

SQL> select* from employee;

       ENO ENAME                          SALARY
---------- ------------------------- ----------
       101 nilam                          12000

SQL> update Employee set Salary=15000 where Eno=101;
Before updating trigger fired

1 row updated.

SQL> select*from employee;

       ENO ENAME                          SALARY
---------- ------------------------- ----------
       101 nilam                          15000
```

PRACTICAL -3

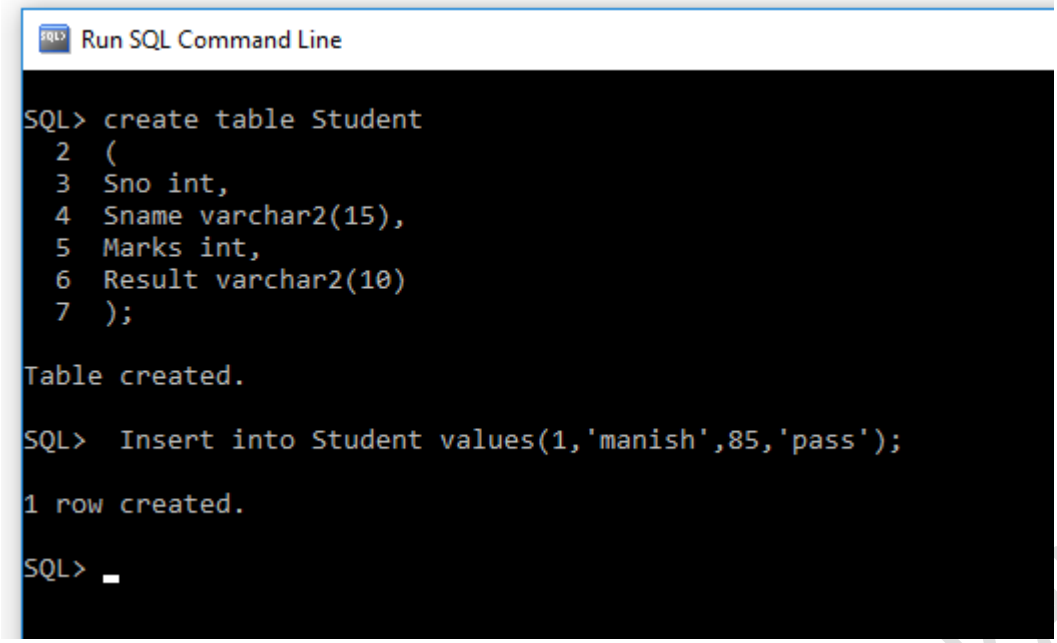
AIM: Create a trigger: after insert, update, delete statement level.

Creation of table: -

```
create table Student
(
Sno int,
Sname varchar2(15),
Marks int,
Result varchar2(10)
);
```

Insertion of data in table: -

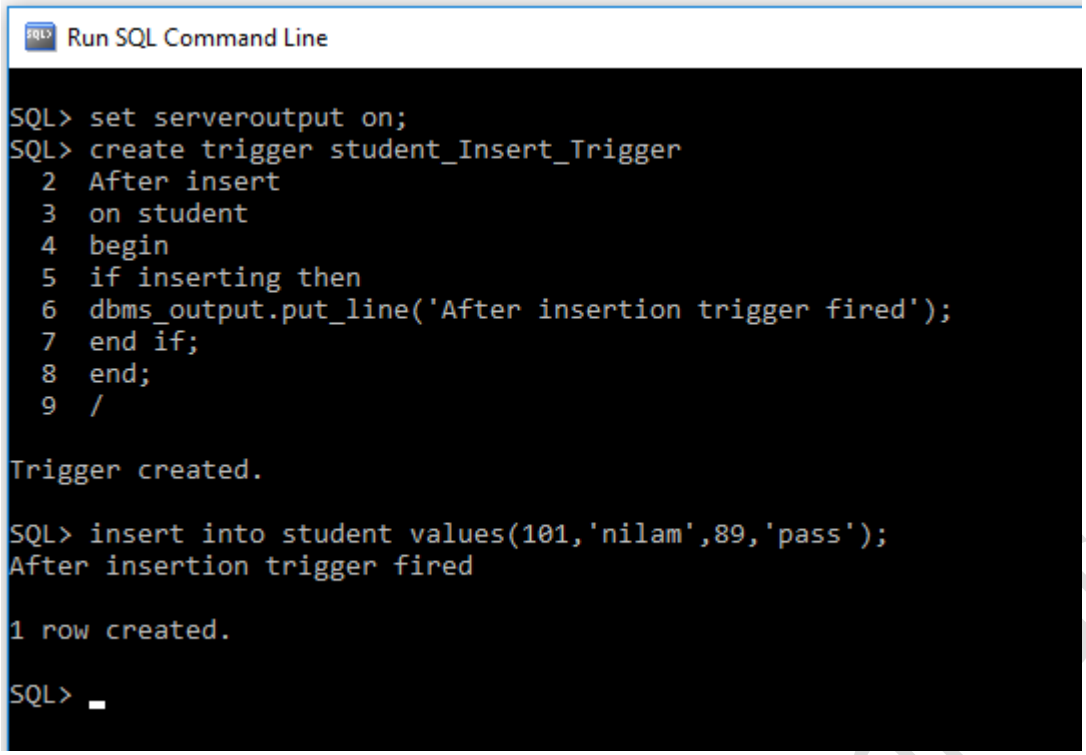
Insert into student values(1,'manish',85,'pass');



Creation of trigger for insert command: -

```
set serveroutput on;
create trigger student_Insert_Trigger
After insert
on student
begin
if inserting then
dbms_output.put_line('After insertion trigger fired');
end if;
end;
/
```

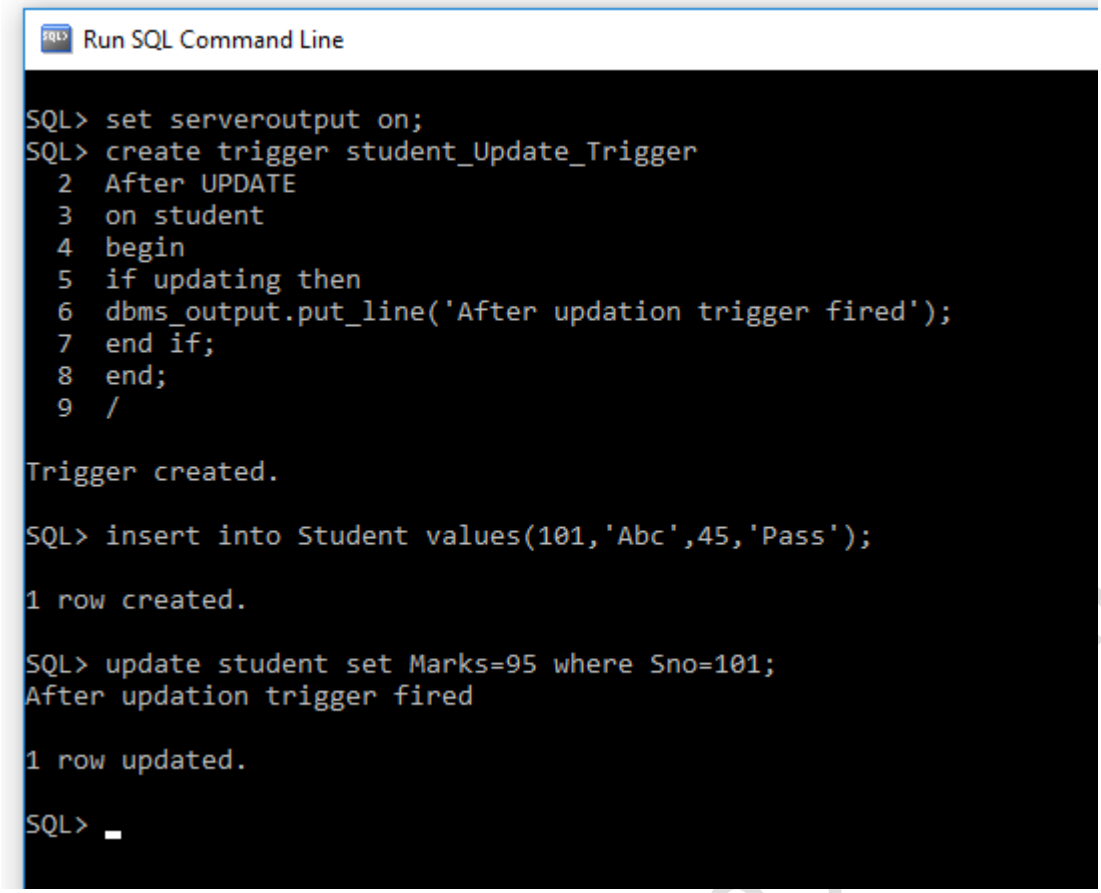
OUTPUT: -



Creation of trigger for update command: -

```
set serveroutput on;
create trigger student_Update_Trigger
After UPDATE
on student
begin
if updating then
dbms_output.put_line('After updation trigger fired');
end if;
end;
/
```

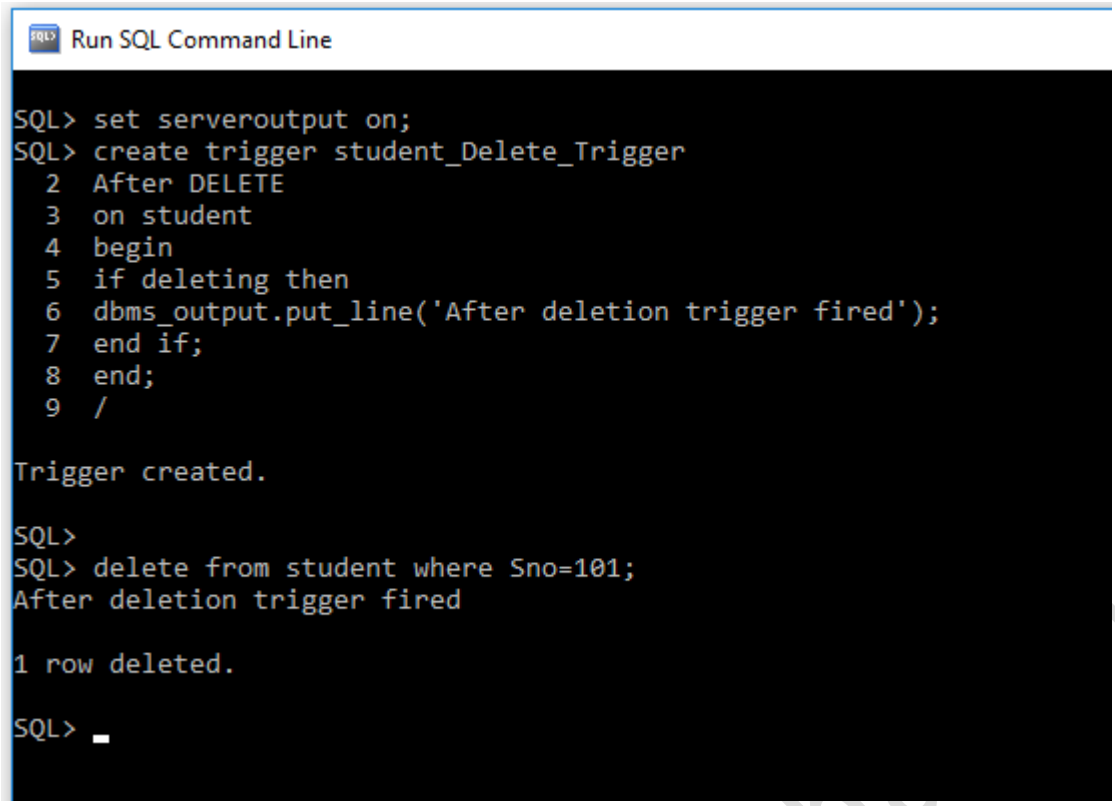
OUTPUT: -



Creation of trigger for delete command: -

```
set serveroutput on;
create trigger student_Delete_Trigger
After DELETE
on student
begin
if deleting then
dbms_output.put_line('After deletion trigger fired');
end if;
end;
/
```

OUTPUT: -



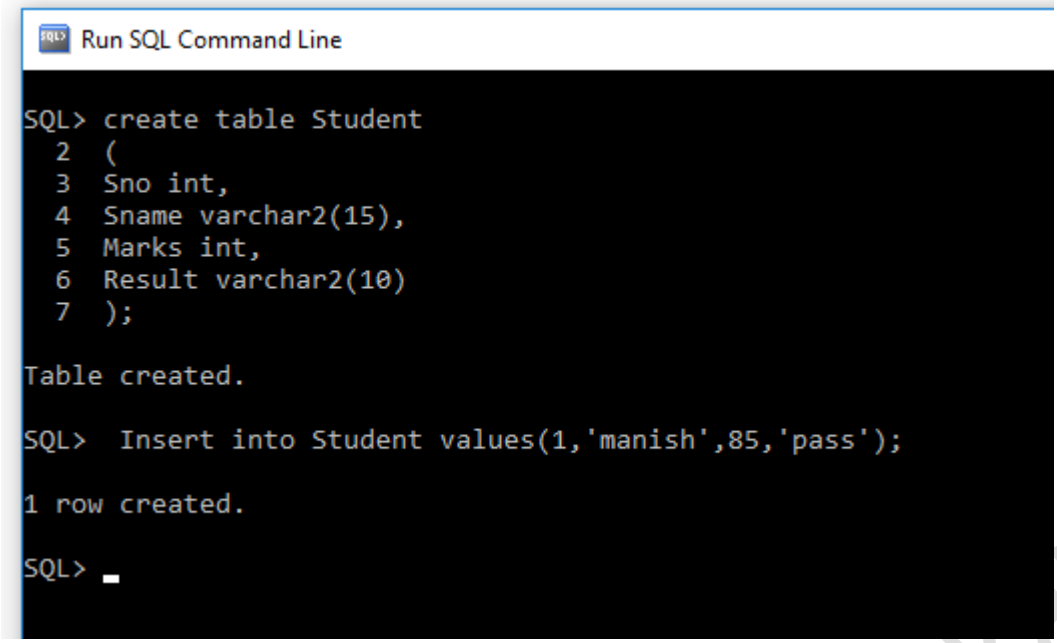
PRACTICAL -4

AIM: Create a trigger after insertion row-level.

Creation of table: -
CODE: -

```
create table Student
(
Sno int,
Sname varchar2(15),
Marks int,
Result varchar2(10)
);
```

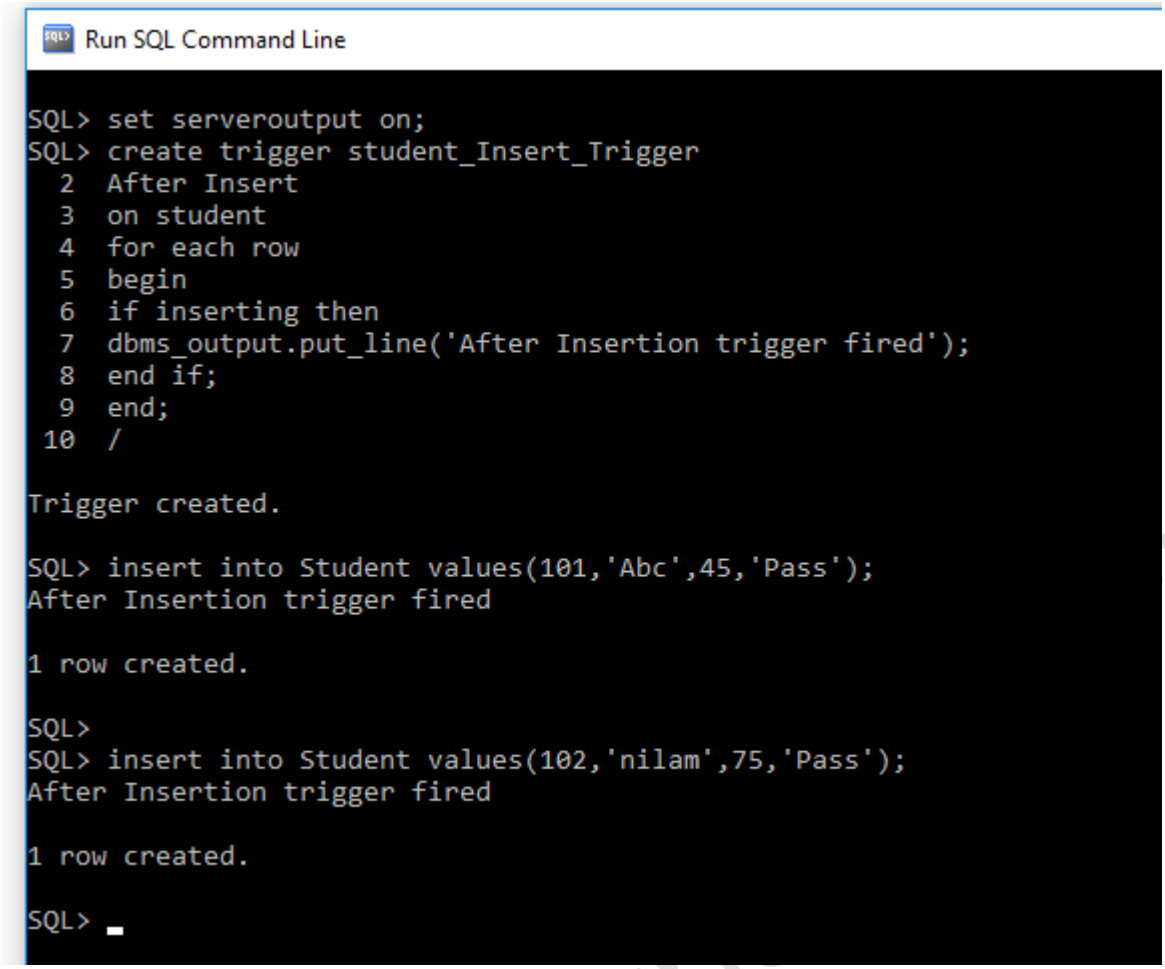
Insertion of data in table: -
Insert into student values(1,'manish',85,'pass');



Creation of trigger for Insert command for row-level: -

```
set serveroutput on;
create trigger student_Insert_Trigger
After Insert
on student
for each row
begin
if inserting then
dbms_output.put_line('After Insertion trigger fired');
end if;
end;
/
```

OUTPUT: -



PRACTICAL -5

AIM: Create a trigger for row-level which demonstrate the use of co-relation identifiers (: new :old).

Creation of table: -

create table Employee
(
Eno int,
Ename varchar2(25),
Salary int
);

create table Salary_Change
(
Saldif int
);

Run SQL Command Line

```
SQL> create table Employee
  2  (
  3  Eno int,
  4  Ename varchar2(25),
  5  Salary int
  6  );

Table created.

SQL> create table Salary_Change
  2  (
  3  Saldif int
  4  );

Table created.
```

Creation of trigger: -

```
create trigger Emp_Salary_Trigger
after update of Salary
on Employee
for each row
declare
a int;
begin
a:=:new.salary-:old.salary;
insert into Salary_Change values(a);
end;
/
```

OUTPUT: -

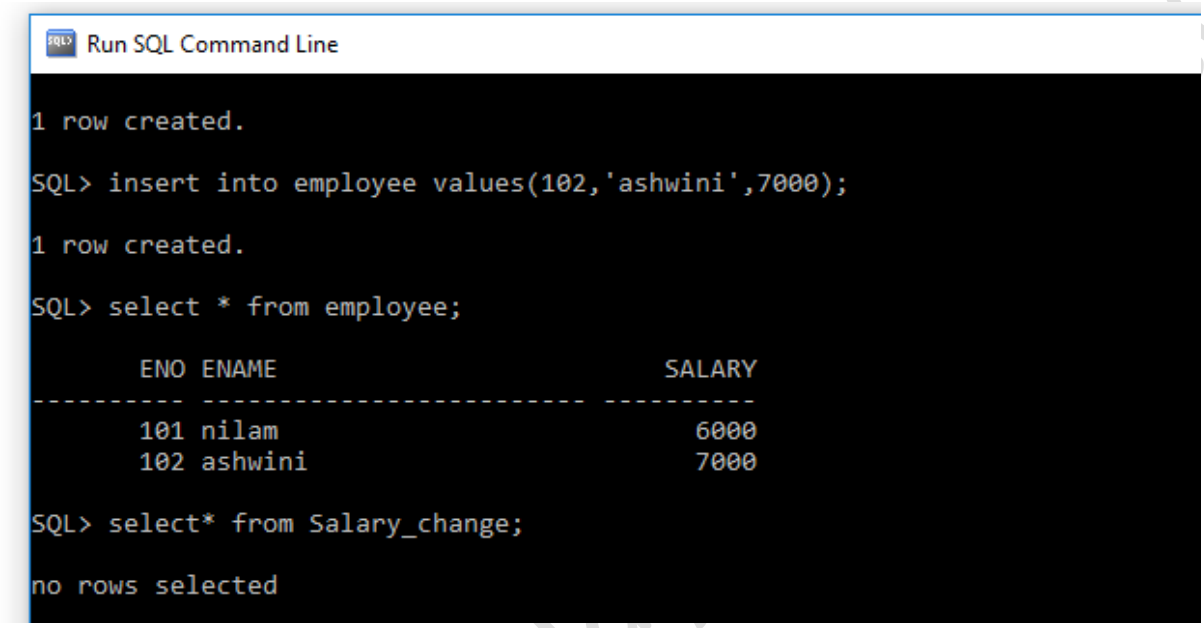
Run SQL Command Line

```
SQL> create trigger Emp_Salary_Trigger
  2  after update of Salary
  3  on Employee
  4  for each row
  5  declare
  6  a int;
  7  begin
  8  a:=:new.salary-:old.salary;
  9  insert into Salary_Change values(a);
 10  end;
 11  /

Trigger created.
```

Insertion of data & selection: -

```
insert into Employee values(101,'nilam',6000);
insert into Employee values(102,'ashwini',7000);

select * from employee;
select * from Salary_Change;
```

OUTPUT: -

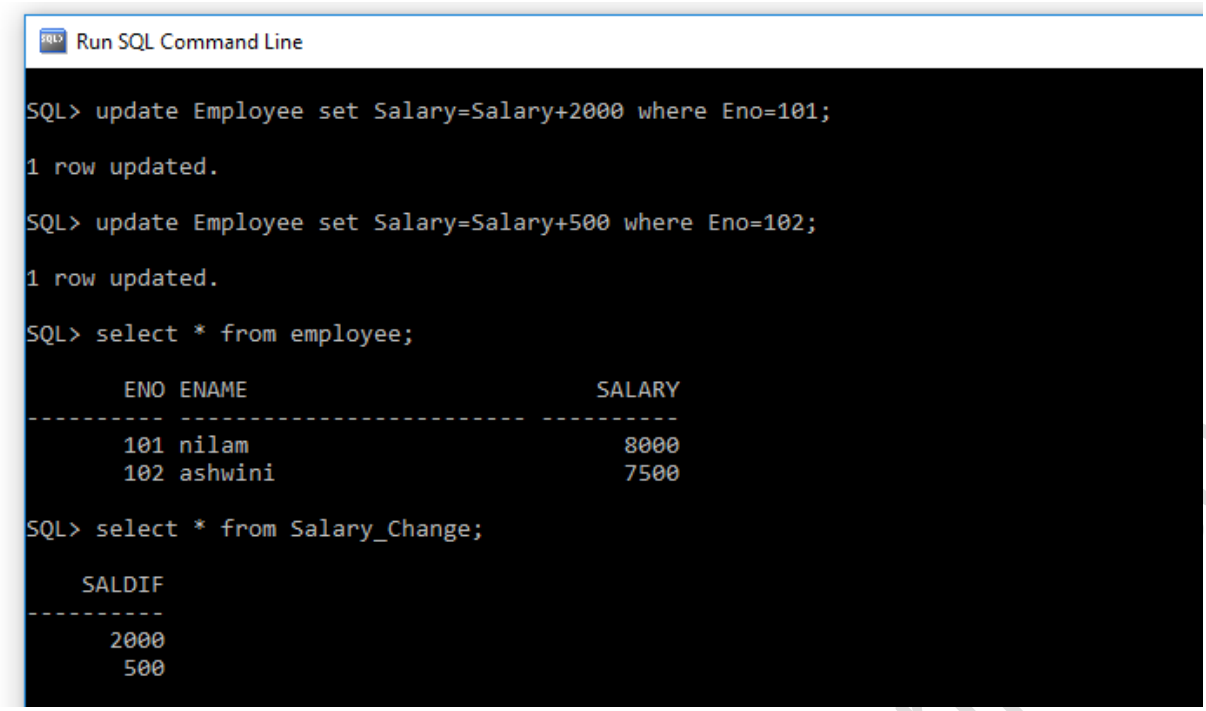


Updating of data & selection: -

```
update Employee set Salary=Salary+2000 where Eno=101;
update Employee set Salary=Salary+500 where Eno=102;

select * from employee;
select * from Salary_Change;
```

Salary difference: -

- 2000
- 500



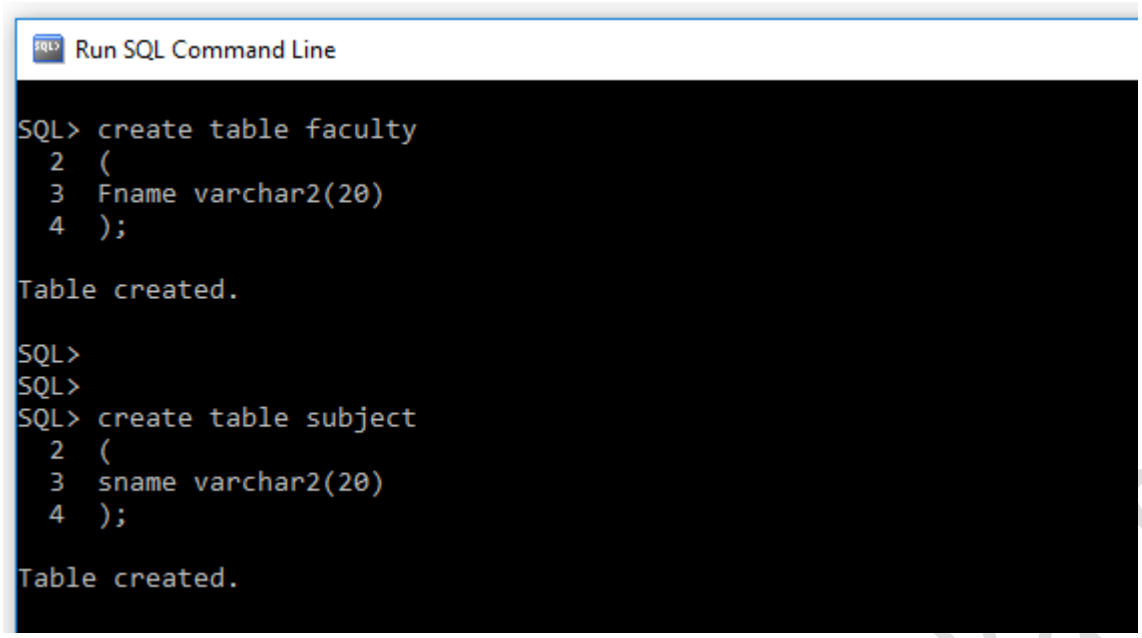
+

PRACTICAL -6

AIM: Create a trigger which demonstrate the use of instead of (View)

Creation of table: -

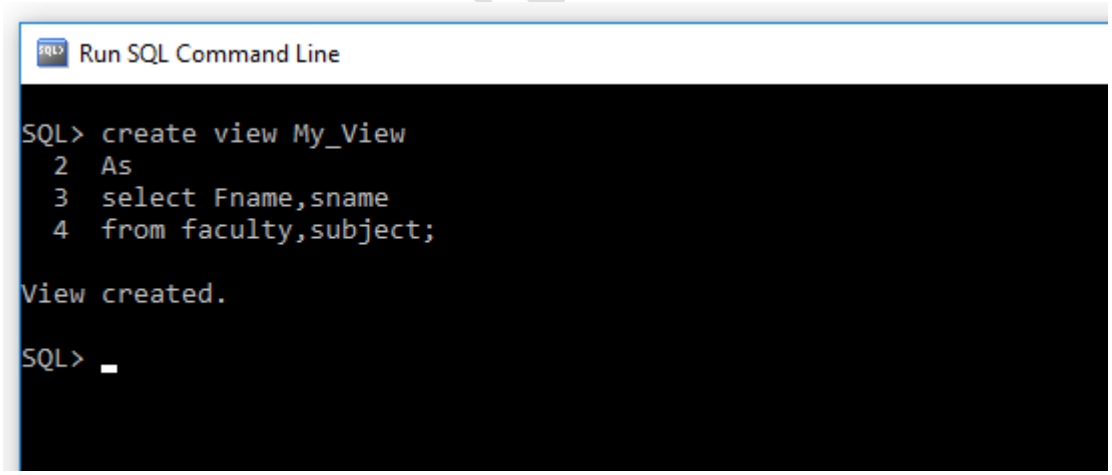
```
create table faculty
(
Fname varchar2(20)
);

create table subject
(
sname varchar2(20)
);
```



Definition of view: -
View is a virtual table created from the physical table.

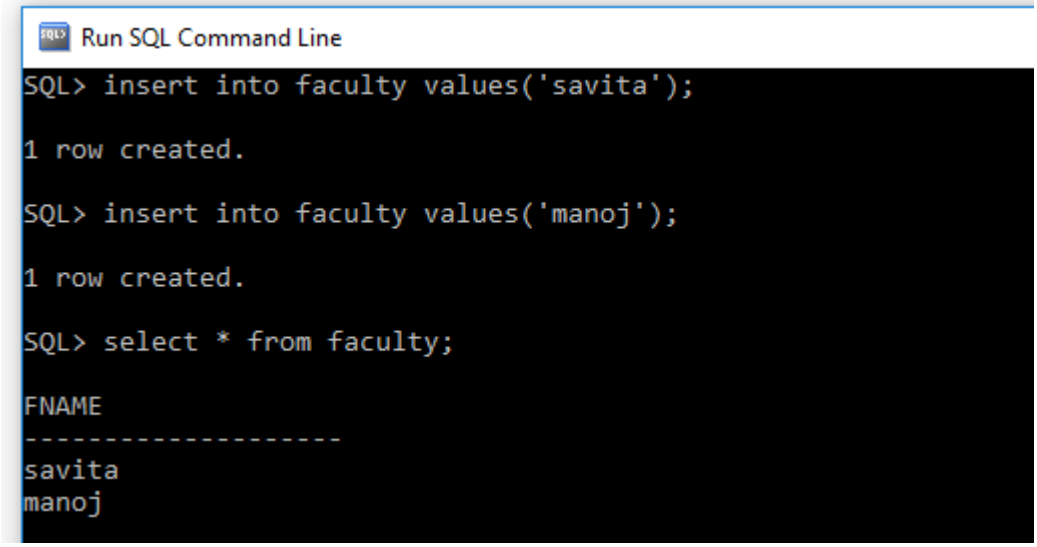
Creation of view: -

```
create view My_View
As
select Fname,sname
from faculty,subject;
```



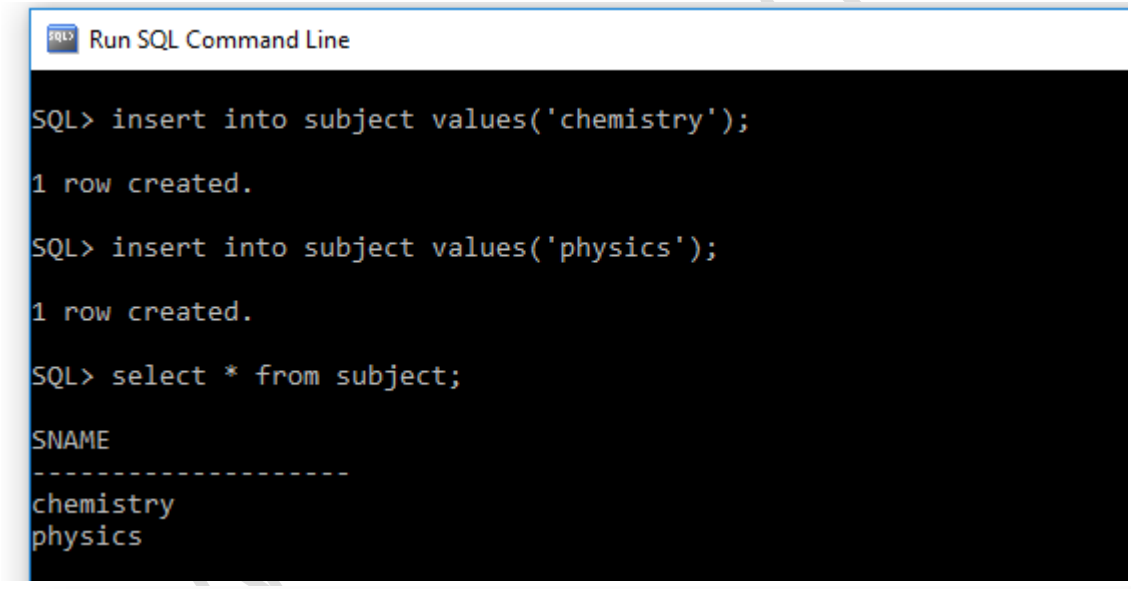
Insertion of data in a table faculty: -

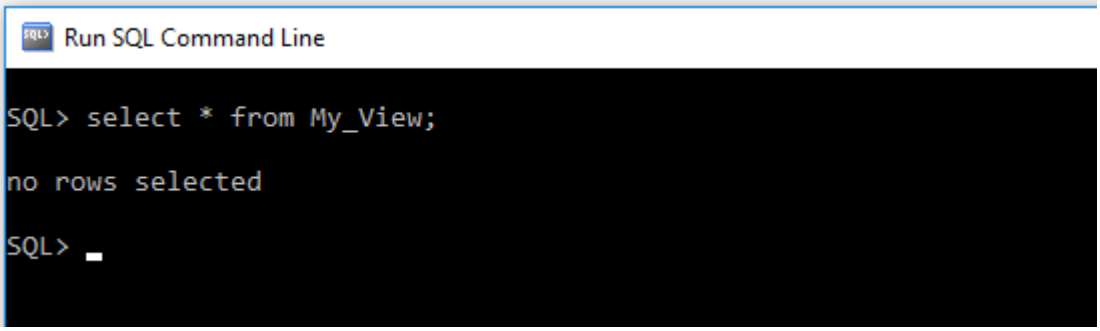
```
insert into faculty values('savita');
insert into faculty values('manoj');
```



Insertion of data in a table Subject: -

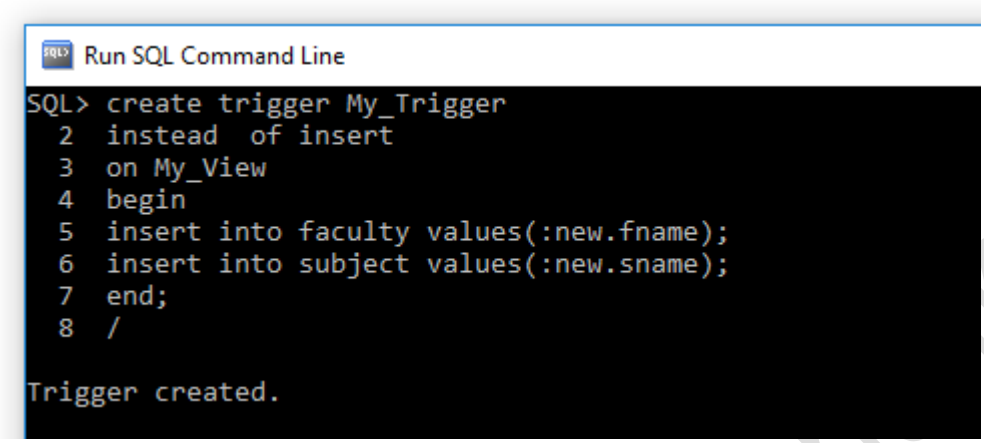
insert into subject values('chemistry');
insert into subject values('physics');

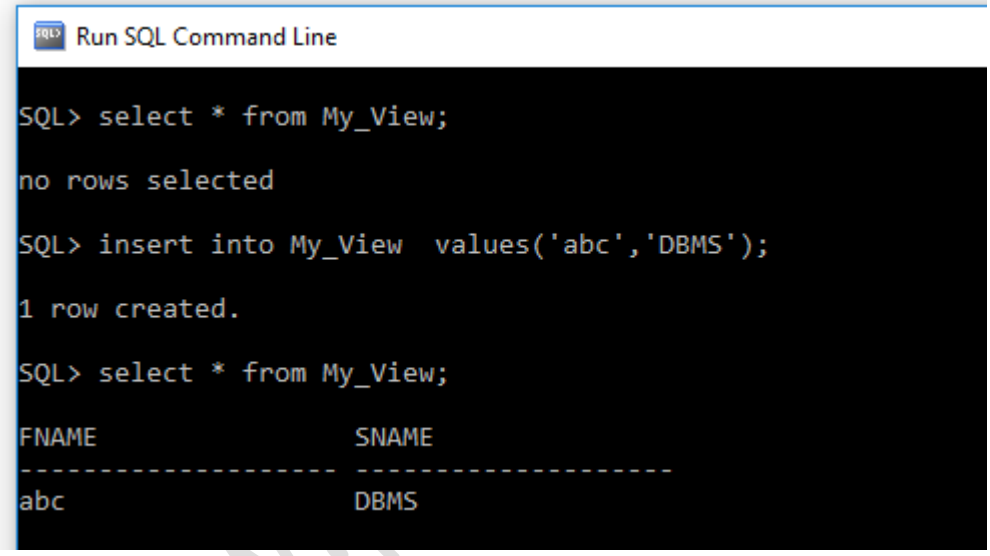




Creation of Trigger: -

```
create trigger My_Trigger
instead of insert
on My_View
begin
insert into faculty values(:new.fname);
insert into subject values(:new.sname);
end;
/
```





PRACTICAL - 7

AIM: With respect to trigger perform following operations;

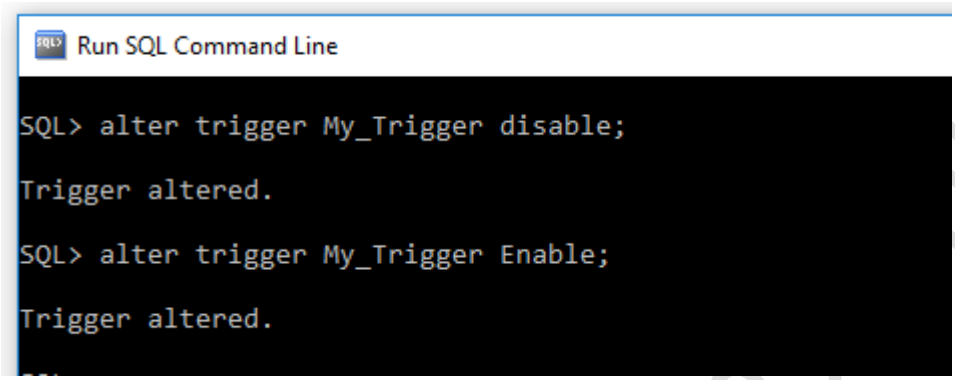
a) **Disable the trigger**
b) **Enable the trigger**
c) **Display the list of user defined triggers which consist of name of triggers and type of triggers.**

Enabling or Disabling Trigger: -

alter trigger trigger_name enable;
alter trigger trigger_name disable;

Ex:

alter trigger My_Trigger disable;



List of user-defined trigger: -

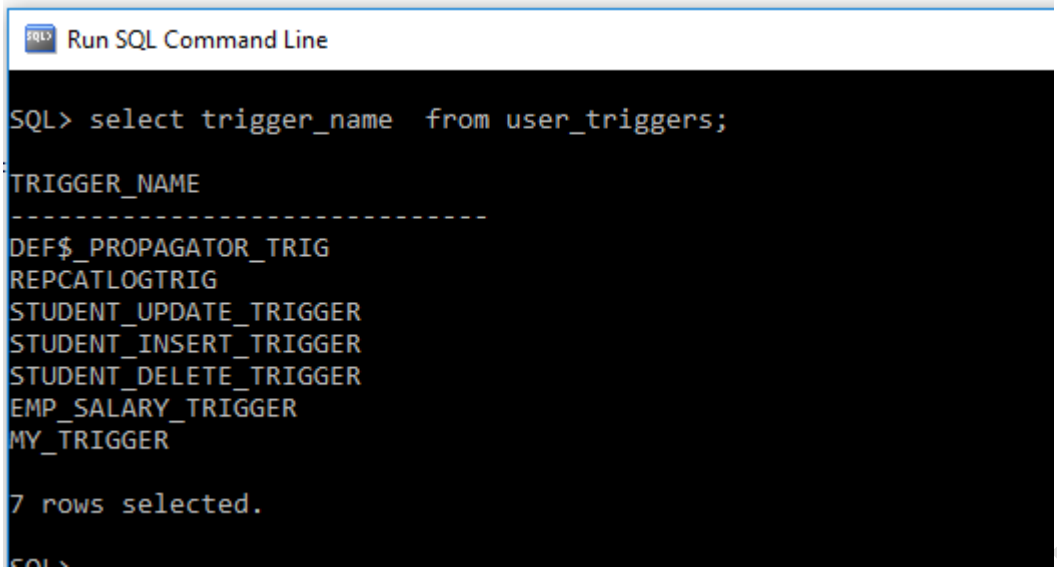
select * from user_triggers;
select trigger_name from user_triggers;
desc user_triggers;
select trigger_name,trigger_type from user_triggers;

select * from user_triggers;

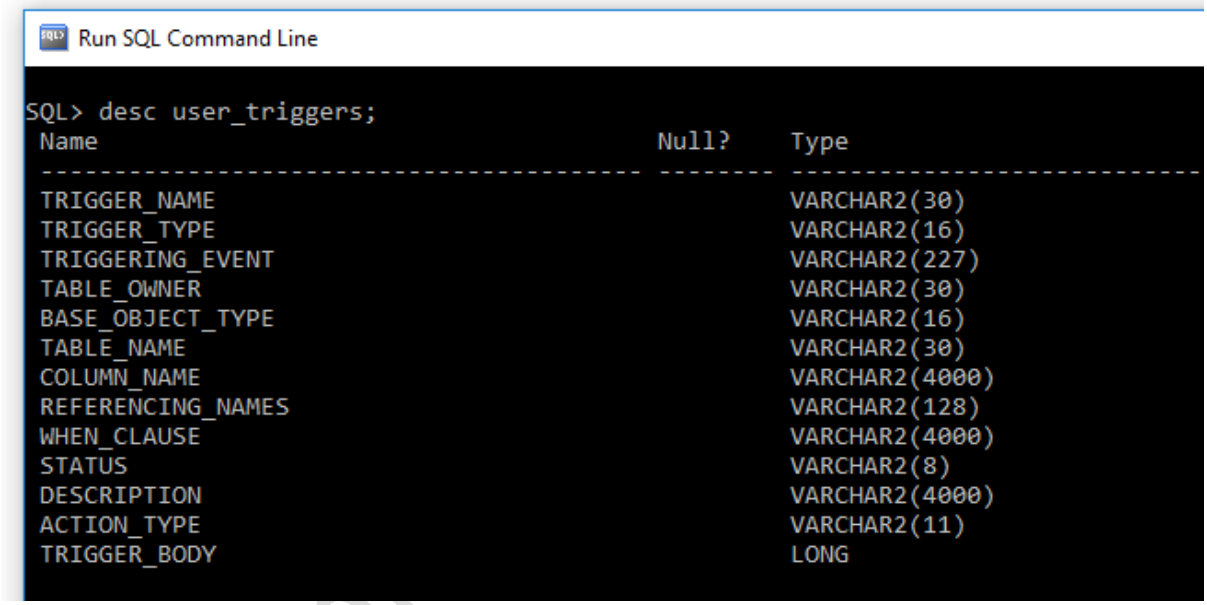
Run SQL Command Line

```
SQL> select * from user_triggers;

TRIGGER_NAME                   TRIGGER_TYPE
------------------------------ ----------------
TRIGGERING_EVENT
--------------------------------------------------------------------------------
TABLE_OWNER                    BASE_OBJECT_TYPE TABLE_NAME
------------------------------ ---------------- ------------------------------
COLUMN_NAME
--------------------------------------------------------------------------------
REFERENCING_NAMES
--------------------------------------------------------------------------------
WHEN_CLAUSE
--------------------------------------------------------------------------------
STATUS
--------
DESCRIPTION
--------------------------------------------------------------------------------
ACTION_TYPE
-----------
TRIGGER_BODY
--------------------------------------------------------------------------------
DEF$_PROPAGATOR_TRIG           BEFORE STATEMENT

TRIGGER_NAME                   TRIGGER_TYPE
------------------------------ ----------------
TRIGGERING_EVENT
--------------------------------------------------------------------------------
TABLE_OWNER                    BASE_OBJECT_TYPE TABLE_NAME
------------------------------ ---------------- ------------------------------
COLUMN_NAME
--------------------------------------------------------------------------------
```

select trigger_name from user_triggers;



desc user_triggers;



select trigger_name,trigger_type from user_triggers;

Run SQL Command Line

```
SQL> select trigger_name,trigger_type  from user_triggers;

TRIGGER_NAME                   TRIGGER_TYPE
------------------------------ ----------------
DEF$_PROPAGATOR_TRIG           BEFORE STATEMENT
REPCATLOGTRIG                  AFTER STATEMENT
STUDENT_UPDATE_TRIGGER         AFTER STATEMENT
STUDENT_INSERT_TRIGGER         AFTER EACH ROW
STUDENT_DELETE_TRIGGER         AFTER STATEMENT
EMP_SALARY_TRIGGER             AFTER EACH ROW
MY_TRIGGER                     INSTEAD OF

7 rows selected.
```

PRACTICAL - 8

AIM: Create a simple index for employee table on salary column.

An index which is created on single column of a table is refer to as simple index.

SYNTAX: -

CREATE INDEX <INDEX NAME> ON <TABLE NAME> (<COLUMN NAME>);

CREATION OF TABLE: -

create table Employee
(
Eno int,
Ename varchar2(25),
Salary int
);

Run SQL Command Line

```
SQL> create table Employee
  2  (
  3  Eno int,
  4  Ename varchar2(25),
  5  Salary int
  6  );

Table created.
```

Insertion of data in a table Employee: -

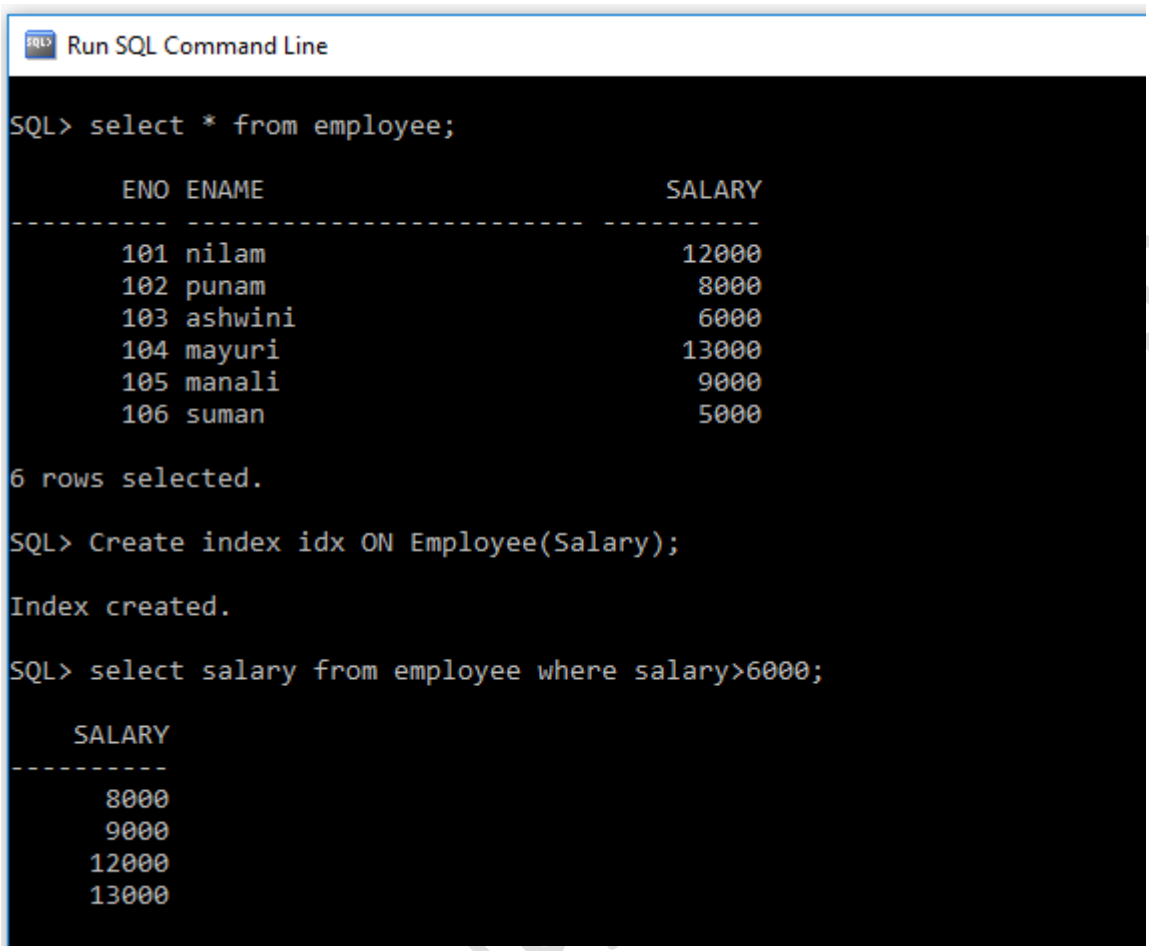
insert into employee values(101,'nilam',12000);
insert into employee values(102,'punam',8000);
insert into employee values(103,'ashwini',6000);
insert into employee values(104,'mayuri',13000);
insert into employee values(105,'manali',9000);

insert into employee values(106,'suman',5000);

EXAMPLE: -

Create index idx ON Employee(Salary);

OUTPUT: -



```
Run SQL Command Line

SQL> select * from employee;

       ENO ENAME                          SALARY
---------- ------------------------- ----------
       101 nilam                          12000
       102 punam                           8000
       103 ashwini                         6000
       104 mayuri                         13000
       105 manali                          9000
       106 suman                           5000

6 rows selected.

SQL> Create index idx ON Employee(Salary);

Index created.

SQL> select salary from employee where salary>6000;

    SALARY
----------
      8000
      9000
     12000
     13000
```


PRACTICAL - 9

AIM: Create a composite index for employee table on salary & commission column.

An index which is created on more than one column of a table is refer to as composite index.

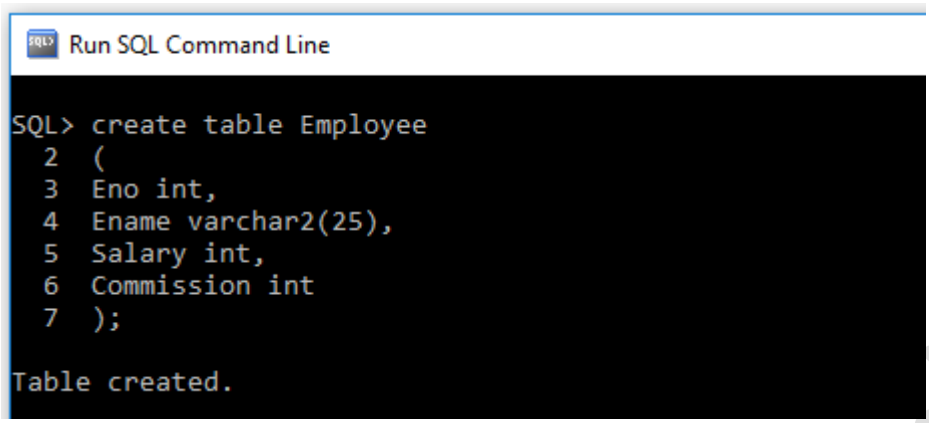
SYNTAX: -

CREATE INDEX <INDEX NAME> ON <TABLE NAME> (<COLUMN NAME1, COLUMN NAME2>);

CREATION OF TABLE: -

create table Employee

```
(
Eno int,
Ename varchar2(25),
Salary int,
Commission int
);
```



Insertion of data in a table Employee: -

insert into employee values(101,'nilam',12000,1200);
insert into employee values(102,'punam',8000,800);
insert into employee values(103,'ashwini',6000,600);
insert into employee values(104,'mayuri',13000,1300);
insert into employee values(105,'manali',9000,900);
insert into employee values(106,'suman',5000,500);

EXAMPLE: -
Create index idx_sal_comm ON Employee(Salary,Commission);

OUTPUT: -

Run SQL Command Line

```
SQL> select * from employee;

       ENO ENAME                          SALARY COMMISSION
---------- ------------------------- ---------- ----------
       101 nilam                          12000       1200
       102 punam                           8000        800
       103 ashwini                         6000        600
       104 mayuri                         13000       1300
       105 manali                          9000        900
       106 suman                           5000        500

6 rows selected.

SQL> Create index idx_sal_comm ON Employee(Salary,Commission);

Index created.

SQL> select salary,commission from employee where salary>6000;

    SALARY COMMISSION
---------- ----------
      8000        800
      9000        900
     12000       1200
     13000       1300
```


PRACTICAL - 10

AIM: Create a unique index for employee table on mobile number column.

A unique index an also be created on one or more column.

If an index is created on single column then it is referred to as simple unique index.

If an index is created on more than one column then it is referred to as composite unique index.

SYNTAX: -

FOR SINGLE COLUMN;

CREATE UNIQUE INDEX <INDEX NAME> ON <TABLE NAME> (<COLUMN NAME>);

FOR MORE THAN ONE COLUMN;

CREATE INDEX <INDEX NAME> ON <TABLE NAME> (<COLUMN NAME1, COLUMN NAME2>);

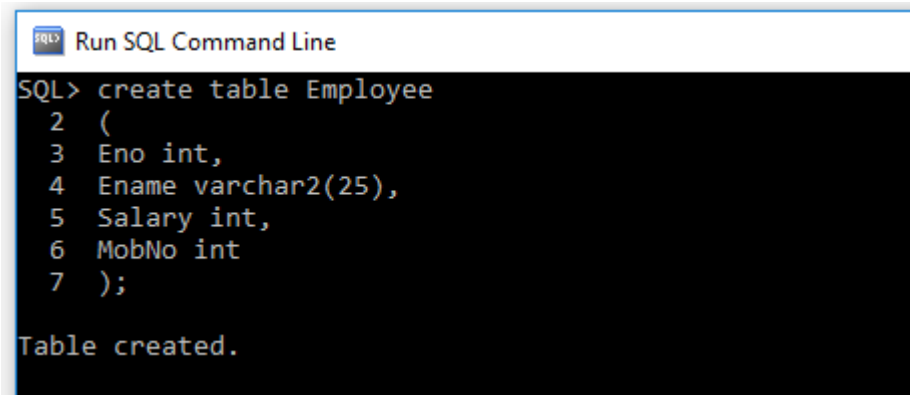
CREATION OF TABLE: -

create table Employee

(

Eno int,

Ename varchar2(25),
Salary int,
Mob_no. int
);



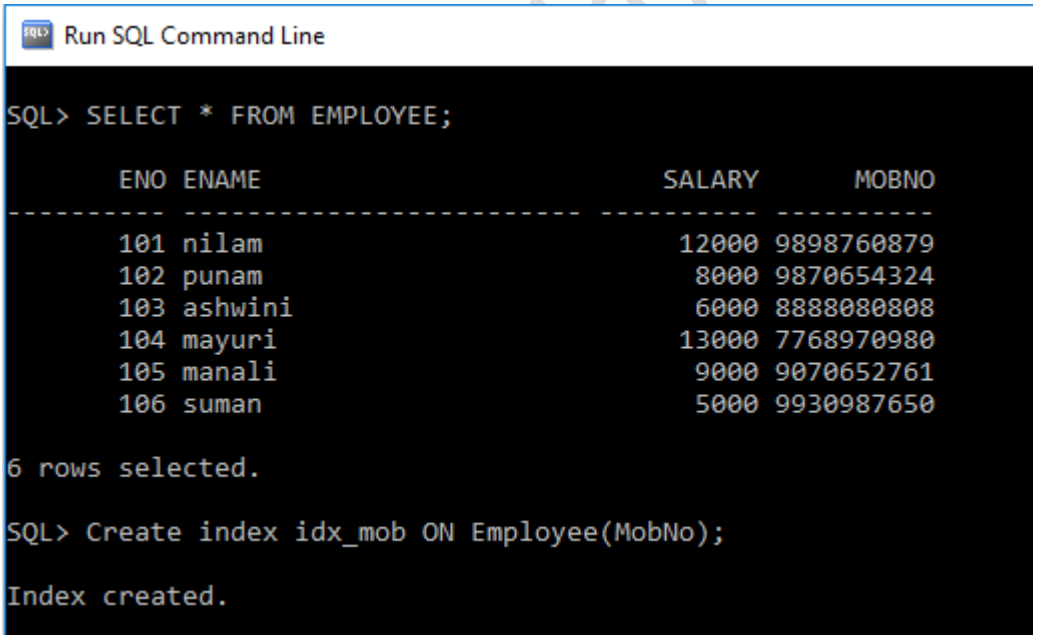
Insertion of data in a table Employee: -

insert into employee values(101,'nilam',12000,9898760879);
insert into employee values(102,'punam',8000,9870654324);
insert into employee values(103,'ashwini',6000,8888080808);
insert into employee values(104,'mayuri',13000,7768970980);
insert into employee values(105,'manali',9000,9070652761);
insert into employee values(106,'suman',5000,9930987650);

EXAMPLE: -

Create index idx_mob ON Employee(MobNo);

OUTPUT: -



Run SQL Command Line

```
SQL> select MobNo from employee where Eno=102;

     MOBNO
----------
9870654324
```

PRACTICAL - 11

AIM: Create a reverse index for employee on salary column.

It is mainly used for finding the highest value like salary, commission etc.

SYNTAX: -

CREATE INDEX <INDEX NAME> ON <TABLE NAME> (<COLUMN NAME>) REVERSE;

CREATION OF TABLE: -

create table Employee

(

Eno int,

Ename varchar2(25),

Salary int,

Mob_no. int

);

Run SQL Command Line

```
SQL> create table Employee
  2  (
  3  Eno int,
  4  Ename varchar2(25),
  5  Salary int,
  6  MobNo int
  7  );

Table created.
```

Insertion of data in a table Employee: -

insert into employee values(101,'nilam',12000,9898760879);
insert into employee values(102,'punam',8000,9870654324);
insert into employee values(103,'ashwini',6000,8888080808);
insert into employee values(104,'mayuri',13000,7768970980);
insert into employee values(105,'manali',9000,9070652761);
insert into employee values(106,'suman',5000,9930987650);

EXAMPLE: -

Create index idx_reverse ON Employee(salary)reverse;

OUTPUT: -

Run SQL Command Line

```
SQL> SELECT * FROM EMPLOYEE;

       ENO ENAME                          SALARY      MOBNO
---------- ------------------------- ---------- ----------
       101 nilam                          12000 9898760879
       102 punam                           8000 9870654324
       103 ashwini                         6000 8888080808
       104 mayuri                         13000 7768970980
       105 manali                          9000 9070652761
       106 suman                           5000 9930987650

6 rows selected.
```

Run SQL Command Line

```
SQL> Create index idx_reverse ON Employee(salary)reverse;

Index created.

SQL> select salary from employee where salary>8000;

    SALARY
----------
     12000
     13000
      9000
```

PRACTICAL - 12

AIM: Create a function-based index for employee table on salary column.

A column index will not be used when same column is express in arithmetic expression or function in where clause.

SYNTAX: -

CREATE INDEX <INDEX NAME> ON <TABLE NAME> (FUNCTION NAME<COLUMN NAME>);

CREATION OF TABLE: -

```
create table Employee
(
Eno int,
Ename varchar2(25),
Salary int
);
```

Run SQL Command Line

```
SQL> create table Employee
  2  (
  3  Eno int,
  4  Ename varchar2(25),
  5  Salary int
  6  );

Table created.
```

Insertion of data in a table Employee: -

insert into employee values(101,'nilam',12000);
insert into employee values(102,'punam',8000);
insert into employee values(103,'ashwini',6000);
insert into employee values(104,'mayuri',13000);
insert into employee values(105,'manali',9000);
insert into employee values(106,'suman',5000);

EXAMPLE: -
Create index idx_fun_salary ON Employee(Salary,(0.10*Salary),InitCap(Ename));

QUERY: -
Select salary,(0.10*salary),InitCap(Ename) from employee where salary>10000;

OUTPUT: -

Run SQL Command Line

```
SQL> Create index idx_fun_salary ON Employee(Salary,(0.10*Salary),InitCap(Ename));

Index created.

SQL> select * from employee;

       ENO ENAME                          SALARY
---------- ------------------------- ----------
       101 nilam                          12000
       102 punam                           8000
       103 ashwini                         6000
       104 mayuri                         13000
       105 manali                          9000
       106 suman                           5000

6 rows selected.
```

Run SQL Command Line

```
SQL> Select salary,(0.10*salary),InitCap(Ename) from employee where salary>10000;

    SALARY (0.10*SALARY) INITCAP(ENAME)
---------- ------------- ------------------------
     12000          1200 Nilam
     13000          1300 Mayuri
```

PRACTICAL - 13

AIM: Create a bitmap index for employee table on gender column.

An index has only two values (ex-gender) are called as bitmap index.

SYNTAX: -

CREATE BITMAP INDEX <INDEX NAME> ON <TABLE NAME> (COLUMN NAME);

CREATION OF TABLE: -

```
create table Employee
(
Eno int,
Ename varchar2(25),
Salary int,
Gender varchar2(10)
);
```

Run SQL Command Line

```
SQL> create table Employee
  2  (
  3  Eno int,
  4  Ename varchar2(25),
  5  Salary int,
  6  Gender varchar2(10)
  7  );

Table created.
```

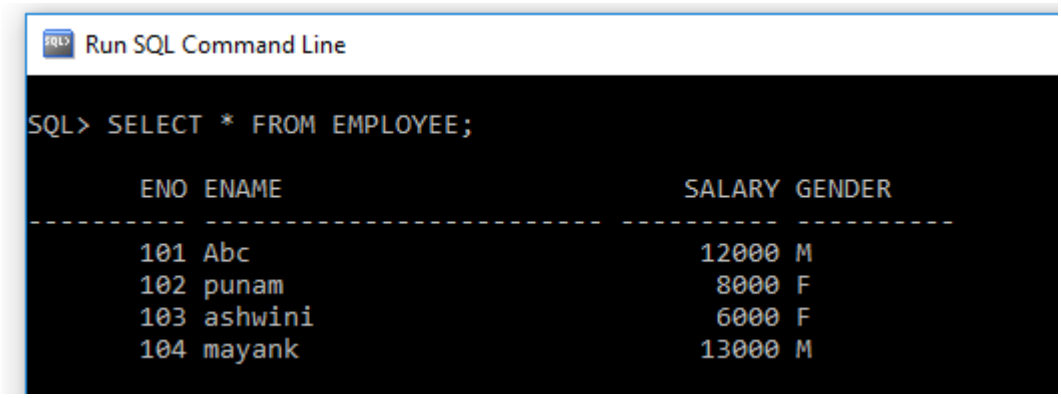
Insertion of data in a table Employee: -

```
insert into employee values(101,'Abc',12000,'M');
insert into employee values(102,'punam',8000,'F');
insert into employee values(103,'ashwini',6000,'F');
insert into employee values(104,'mayank',13000,'M');
```

EXAMPLE: -

Create bitmap index idx_gender ON Employee(Gender);

OUTPUT: -



PRACTICAL - 14

AIM: Drop all indexes.

PRACTICAL.15

AIM : Create a user call TYCS and password 1234 with the help of system user

The CREATE USER statement creates a database account that allows you to log into the oracle database.

Syntax:

create user user_name identified by password;

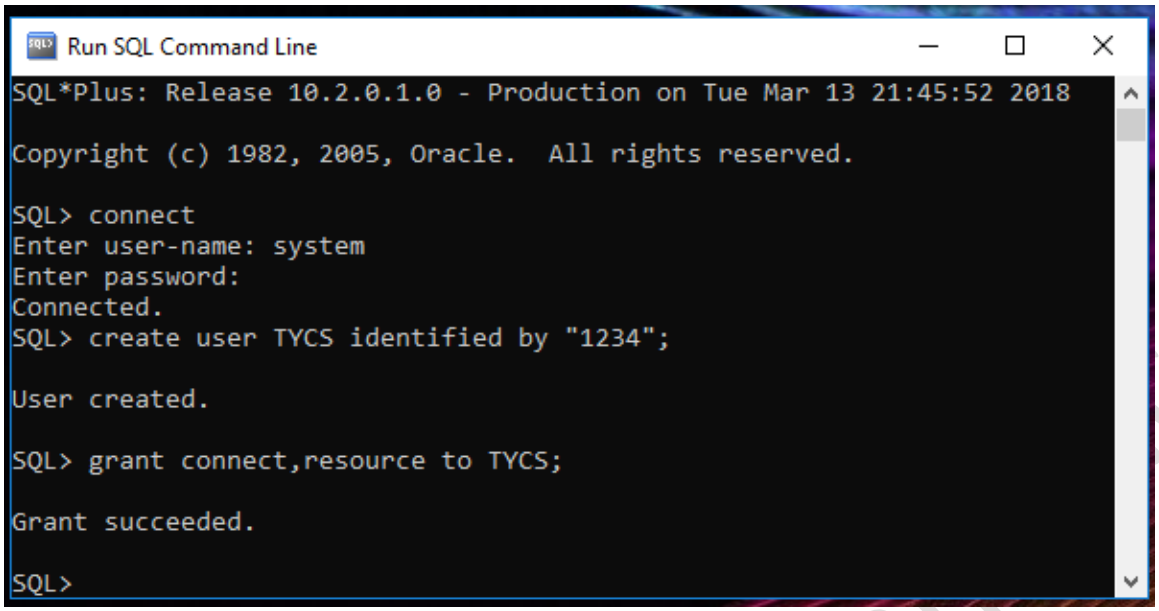
grant connect,resource to user_name;

CREATION OF USER:-

create user TYCS identified by "1234";

grant connect,resource to TYCS;

OUTPUT:



PRACTICAL.16

AIM: Demonstrate the use of following data dictionary.

1)user_users
2)user_tables
3)user_tab_columns
4)user_views
5)user_indexes
6)user_triggers
7)dba_users

8)dba_tables

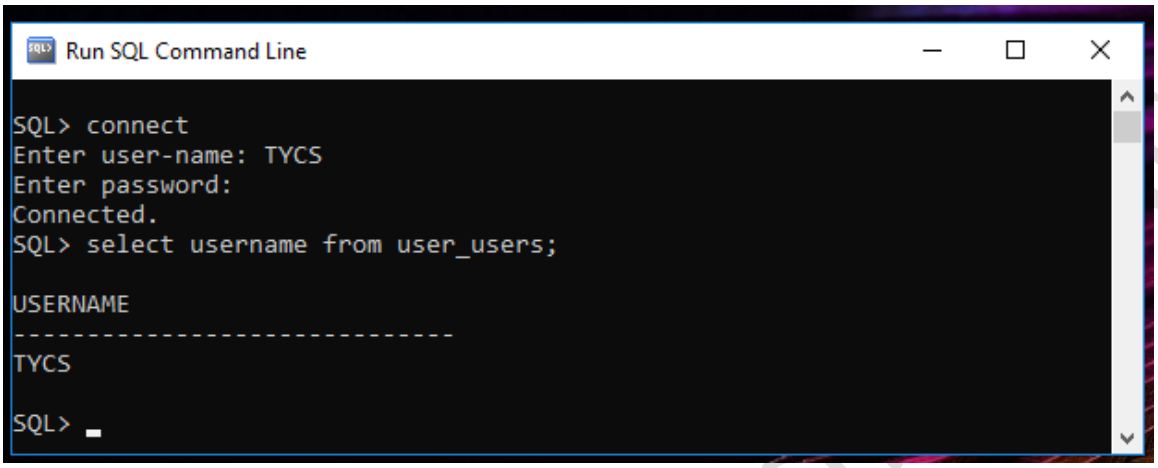
1) USER_USERS:

USER_USERS describes the current user.

Syntax:

Select user_name from user_users;

OUTPUT:



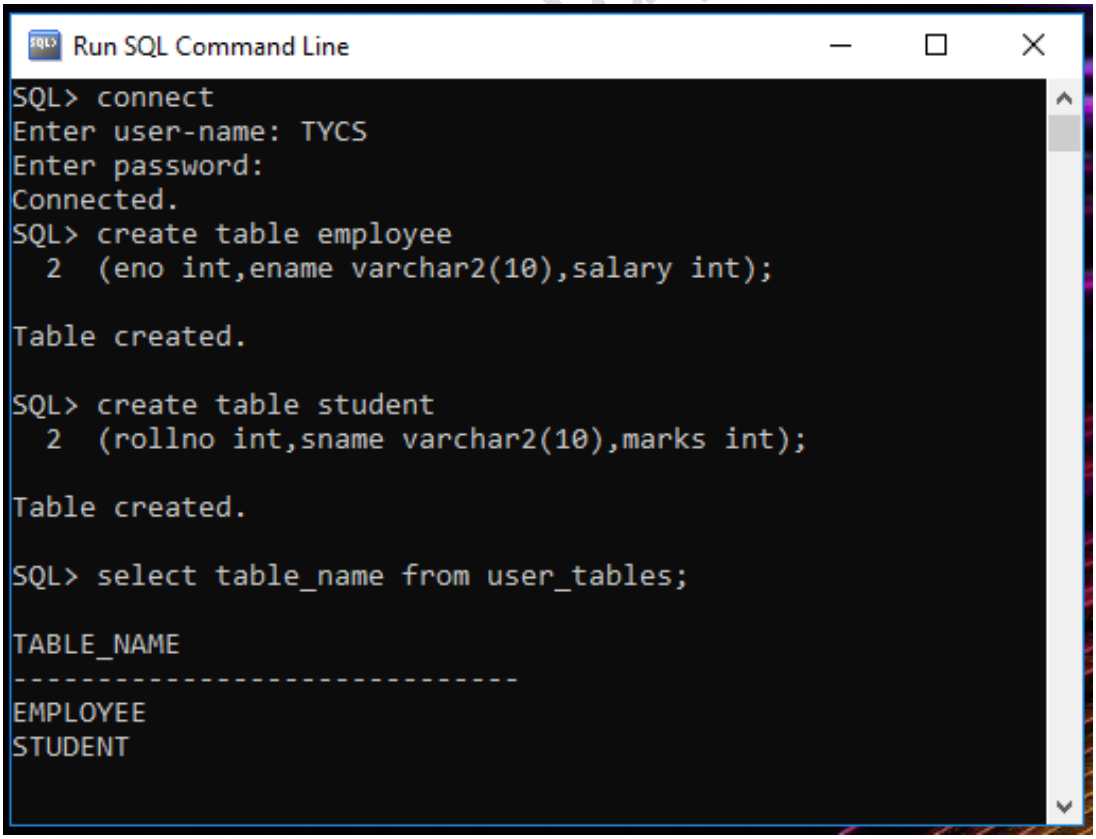
2) USER_TABLES:

USER_TABLES describes the relational tables owned by the current user.

Syntax:

Select table_name from user_tables;

OUTPUT:



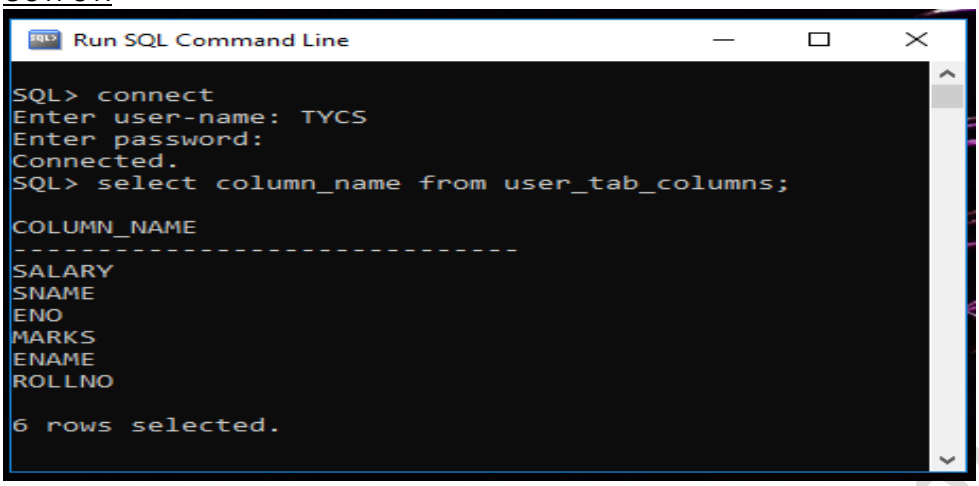
3) USER_TAB_COLUMNS:

USER_TAB_COLUMNS describes the columns of the tables, views, and clusters owned by the current user.

Syntax:

Select column_name from user_tab_columns;

OUTPUT:



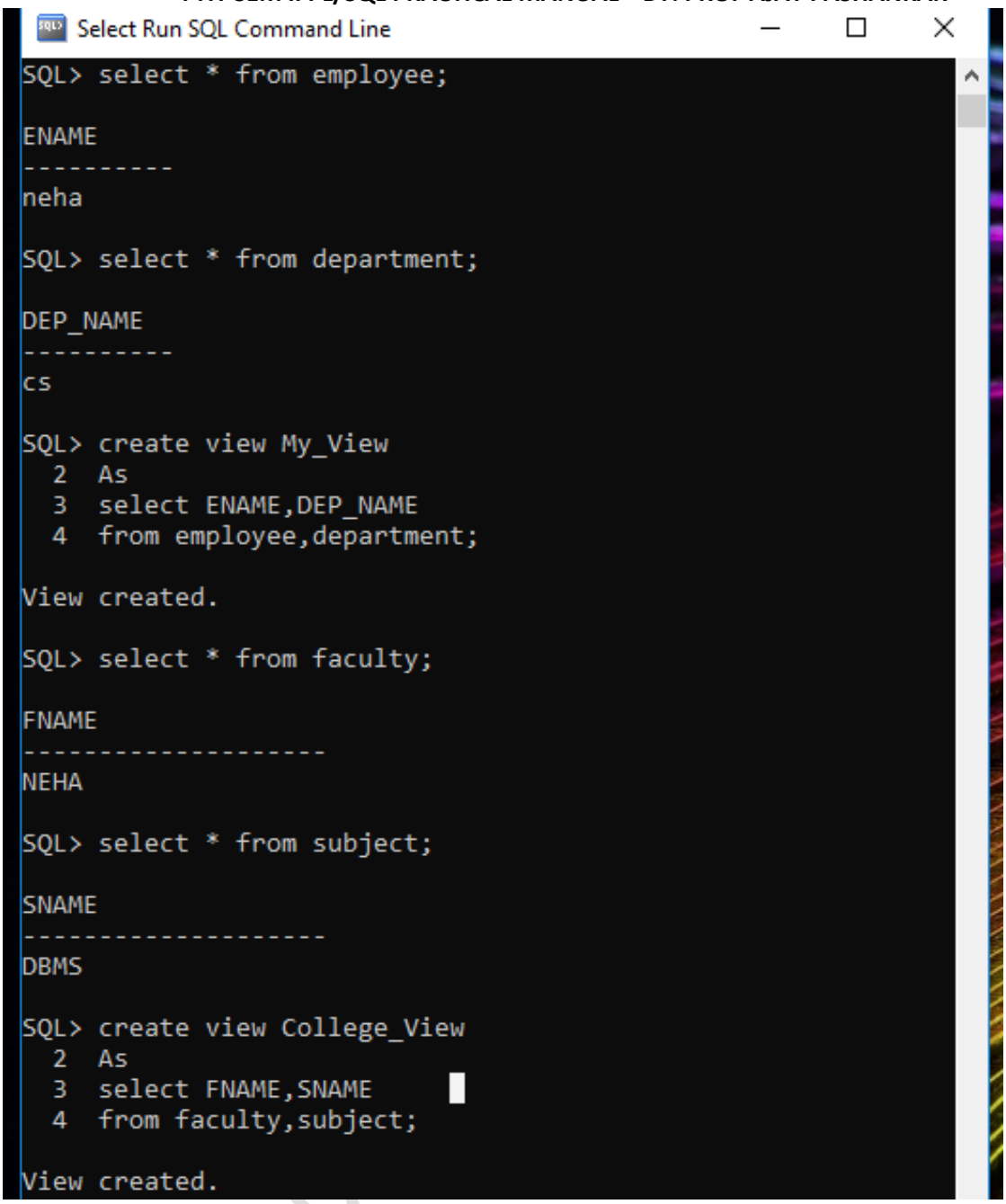
4)USER_VIEWS

USER_VIEWS describes the views owned by the current user.

Syntax:

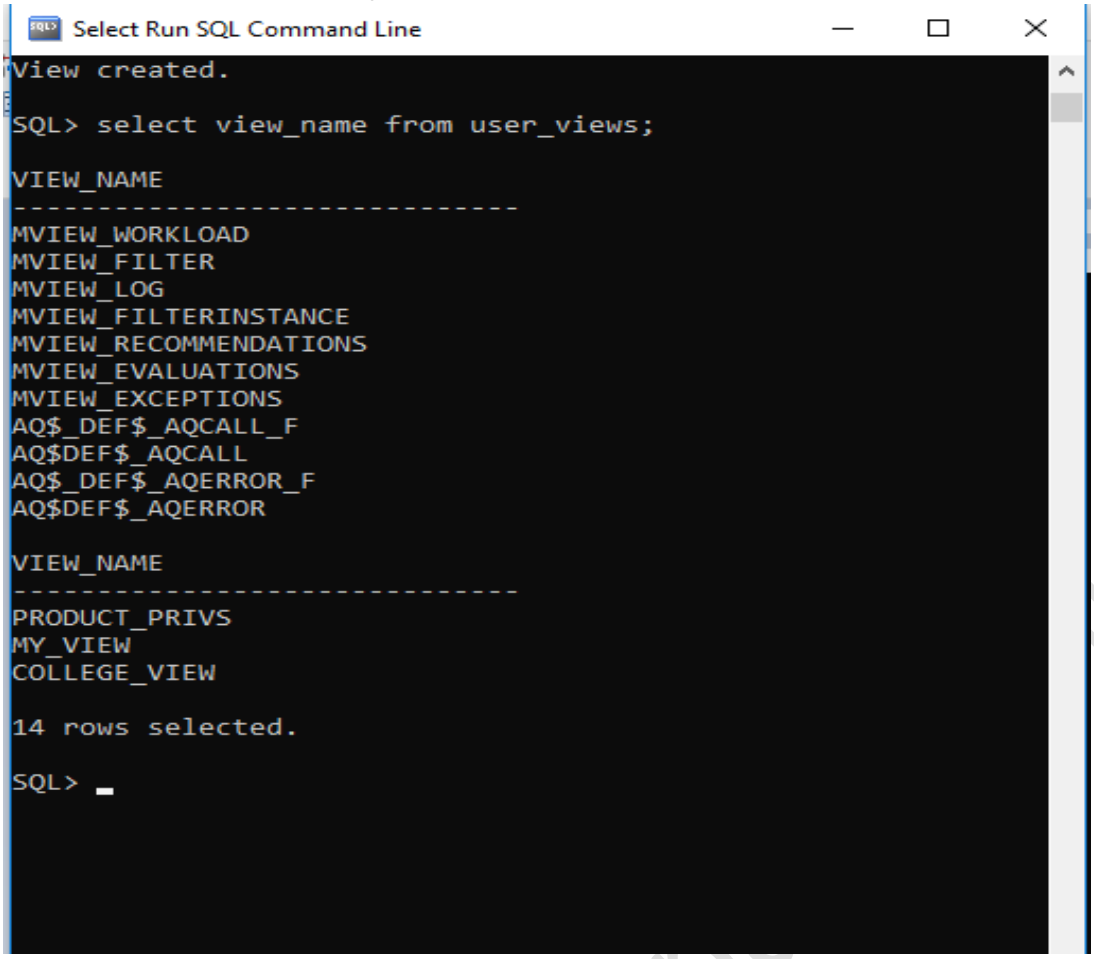
Select view_name from user_views;

OUTPUT:


Select Run SQL Command Line
SQL> select * from employee;
ENAME
----
neha
SQL> select * from department;
DEP_NAME
----
cs
SQL> create view My_View
2 As
3 select ENAME,DEP_NAME
4 from employee,department;
View created.
SQL> select * from faculty;
FNAME

NEHA
SQL> select * from subject;
SNAME

DBMS
SQL> create view College_View
2 As
3 select FNAME,SNAME
4 from faculty,subject;
View created.



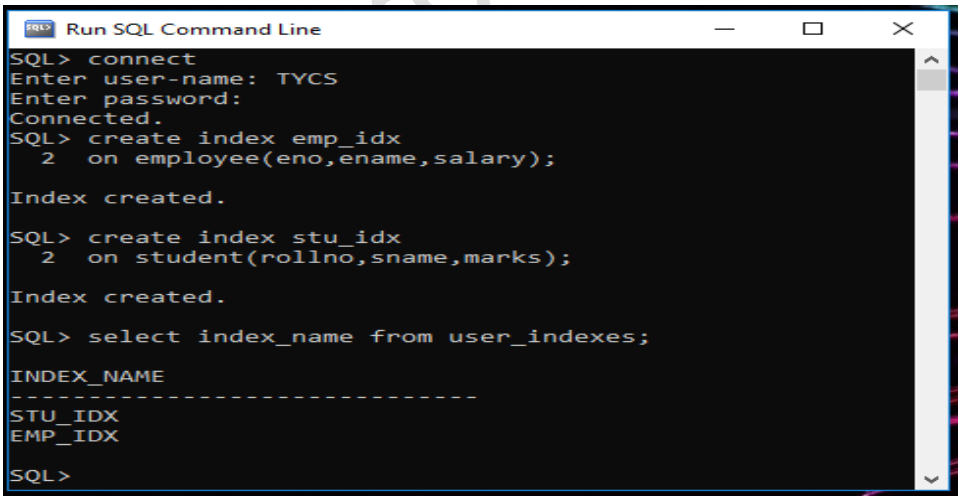
5) USER_INDEXES

USER_INDEXES describes indexes owned by the current user.

Syntax:

Select index_namefrom user_indexes;

OUTPUT:



6) USER_TRIGGERS:

USER_TRIGGERS describes the triggers owned by the current user.

Syntax:

Select trigger_name from user_triggers;

OUTPUT:

```
SQL> set serveroutput on;
SQL> create trigger student_Insert_Trigger_Neha
  2  Before insert
  3  on student
  4  for each row
  5  begin
  6  if inserting then
  7  dbms_output.put_line('After insertion trigger fired');
  8  end if;
  9  end;
 10  /

Trigger created.

SQL> set serveroutput on;
SQL> create trigger employee_Insert_Trigger_Neha
  2  Before insert
  3  on employee
  4  for each row
  5  begin
  6  if inserting then
  7  dbms_output.put_line('After insertion trigger fired');
  8  end if;
  9  end;
 10  /

Trigger created.

SQL> select trigger_name from user_triggers;

TRIGGER_NAME
------------------------------
STUDENT_INSERT_TRIGGER_NEHA
EMPLOYEE_INSERT_TRIGGER_NEHA

SQL>
```

7) DBA_USERS:

DBA_USERS describes all users of the database.

Syntax:

Select username,created from dba_users;

OUTPUT:

```
SQL> connect
Enter user-name: system
Enter password:
Connected.
SQL> select username,created from dba_users;

USERNAME                       CREATED
------------------------------ ---------
TYIT                           14-MAR-18
SYCS                           14-MAR-18
TYCS                           13-MAR-18
SYS                            07-FEB-06
SYSTEM                         07-FEB-06
ANONYMOUS                      07-FEB-06
MDSYS                          07-FEB-06
OUTLN                          07-FEB-06
DIP                            07-FEB-06
TSMSYS                         07-FEB-06
FLOWS_FILES                    07-FEB-06

USERNAME                       CREATED
------------------------------ ---------
CTXSYS                         07-FEB-06
DBSNMP                         07-FEB-06
FLOWS_020100                   07-FEB-06
XDB                            07-FEB-06
HR                             07-FEB-06

16 rows selected.
```

8) DBA_TABLES :

DBA_TABLES describes all relational tables in the database.

Syntax:

Select table_name from dba_tables;

OUTPUT: