# EXPERIMENT-1

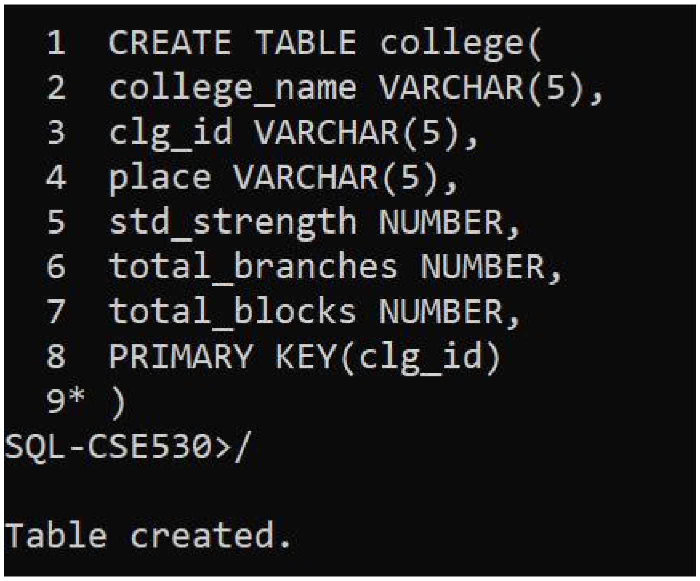
**AIM : Write SQL queries to CREATE TABLES for various databases using DDL commands (i.e.CREATE, ALTER, DROP, TRUNCATE).**

## CREATE TABLE:

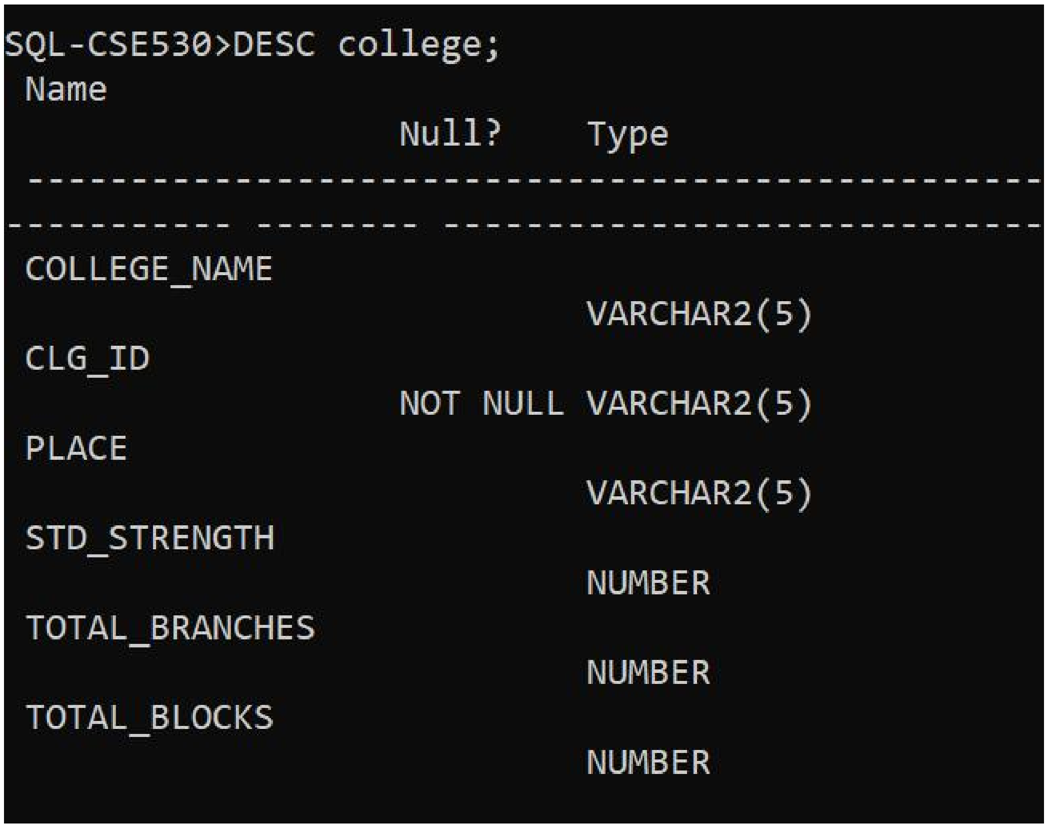
**Creates a table with specified constraints**

## SYNTAX:

**CREATE TABLE tablename (**
**column1 data_ type [constraint] [,**
**column2 data_ type [constraint] ] [,**
**PRIMARY KEY (column1 [, column2]) ] [,**
**FOREIGN KEY (column1 [, column2]) REFERENCES tablename] [,CONSTRAINT constraint]);**



```
  1  CREATE TABLE college(
  2  college_name VARCHAR(5),
  3  clg_id VARCHAR(5),
  4  place VARCHAR(5),
  5  std_strength NUMBER,
  6  total_branches NUMBER,
  7  total_blocks NUMBER,
  8  PRIMARY KEY(clg_id)
  9* )
SQL-CSE530>/

Table created.
```



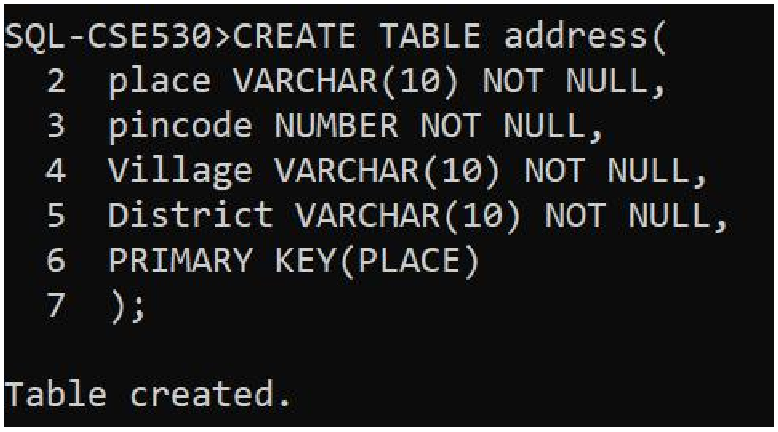
```
SQL-CSE530>DESC college;
 Name
                                  Null?    Type
 ----------------------------------------------------
----------- -------- ---------------------------
 COLLEGE_NAME
                                           VARCHAR2(5)
 CLG_ID
                                  NOT NULL VARCHAR2(5)
 PLACE
                                           VARCHAR2(5)
 STD_STRENGTH
                                           NUMBER
 TOTAL_BRANCHES
                                           NUMBER
 TOTAL_BLOCKS
                                           NUMBER
```

## ALTER TABLE :

**Used to add or modify table details like column names and data types, column constraints.**

## EXPERIMENT-2

**AIM : TO Write SQL queries to MANIPULATE TABLES for various databases using DML commands(i.e. INSERT, SELECT, UPDATE, DELETE,).**
**Creating table :**



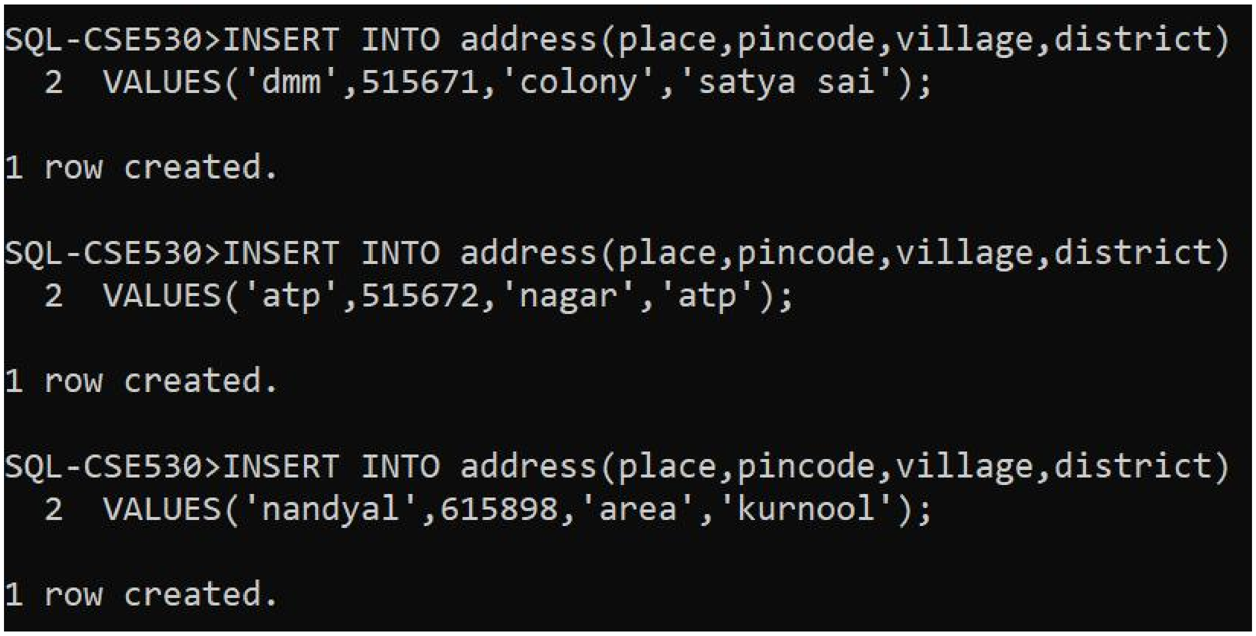
```
SQL-CSE530>CREATE TABLE address(
  2  place VARCHAR(10) NOT NULL,
  3  pincode NUMBER NOT NULL,
  4  Village VARCHAR(10) NOT NULL,
  5  District VARCHAR(10) NOT NULL,
  6  PRIMARY KEY(PLACE)
  7  );

Table created.
```

## INSERT COMMAND:

**It is used to add values to a table.**

## SYNTAX:

**INSERT INTO tablename**
**VALUES (value1,value2,...,valuen);**
**INSERT INTO tablename (column1, column2,...,column)**
**VALUES (value1, value2,...,valuen);**



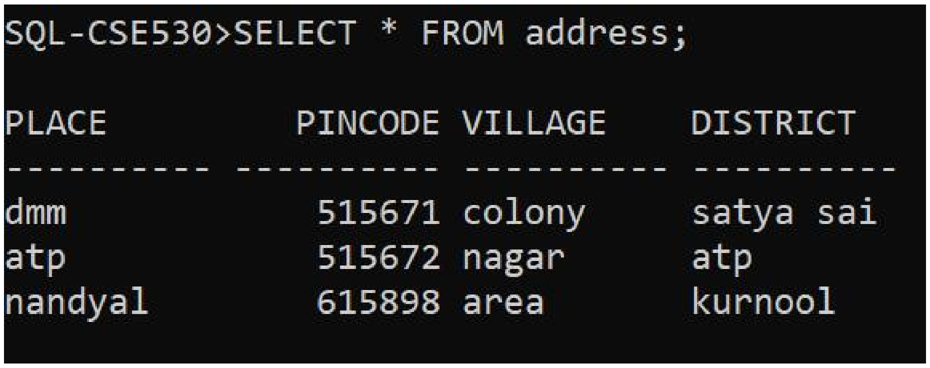
```
SQL-CSE530>INSERT INTO address(place,pincode,village,district)
  2  VALUES('dmm',515671,'colony','satya sai');

1 row created.

SQL-CSE530>INSERT INTO address(place,pincode,village,district)
  2  VALUES('atp',515672,'nagar','atp');

1 row created.

SQL-CSE530>INSERT INTO address(place,pincode,village,district)
  2  VALUES('nandyal',615898,'area','kurnool');

1 row created.
```

## SELECT COMMAND:

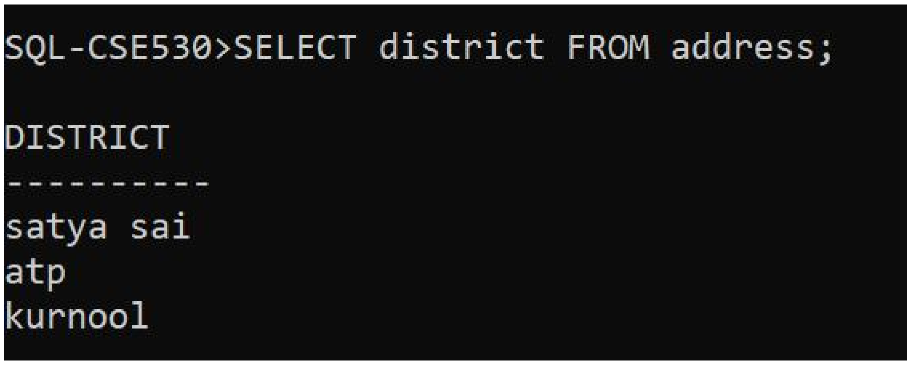
**The SELECT command used to list the contents of a table.**

## SYNTAX:

**Select * from table_name;**
**Select col_name from table_name;**



```
SQL-CSE530>SELECT * FROM address;

PLACE         PINCODE VILLAGE    DISTRICT
---------- ---------- ---------- ----------
dmm            515671 colony     satya sai
atp            515672 nagar      atp
nandyal        615898 area       kurnool
```



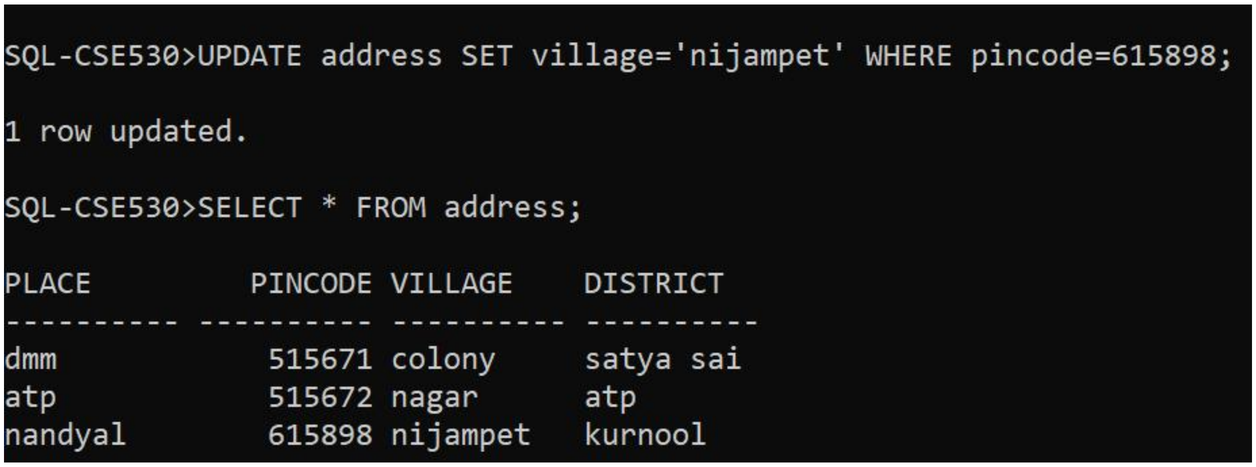
```
SQL-CSE530>SELECT district FROM address;

DISTRICT
----------
satya sai
atp
kurnool
```

## UPDATE COMMAND:

**The update command used to modify the contents of specified table.**

## SYNTAX:

**UPDATE tablename**
**SET column_name = value[,**
**Column_name = value ]**
**[ WHERE condition_list ];**



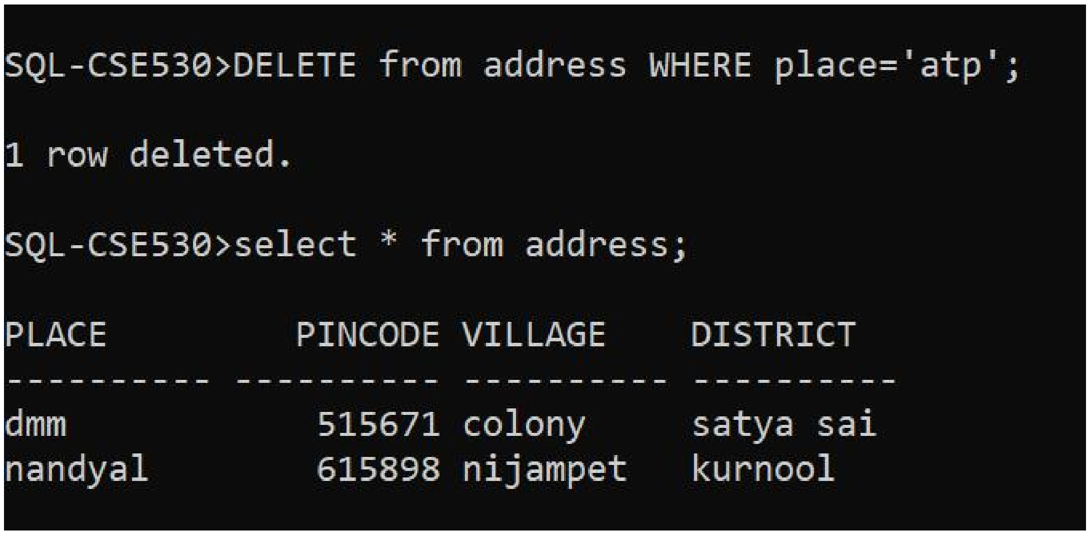
```
SQL-CSE530>UPDATE address SET village='nijampet' WHERE pincode=615898;

1 row updated.

SQL-CSE530>SELECT * FROM address;

PLACE         PINCODE VILLAGE    DISTRICT
---------- ---------- ---------- ----------
dmm            515671 colony     satya sai
atp            515672 nagar      atp
nandyal        615898 nijampet   kurnool
```

## DELETE COMMAND:

**To delete all rows or specified rows in a table.**

## SYNTAX:

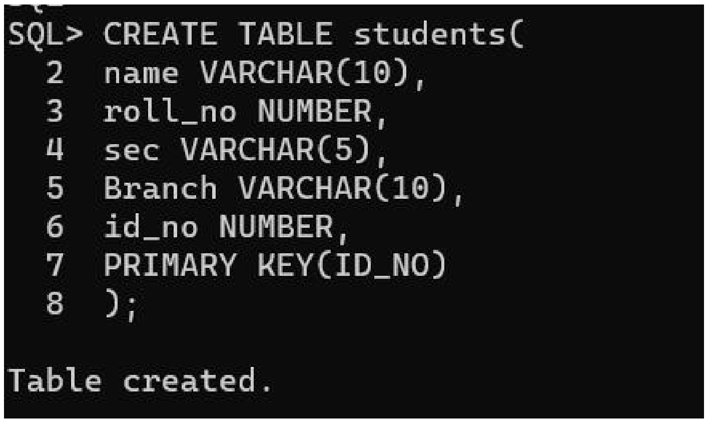
**DELETE FROM tablename [ WHERE condition_list];**



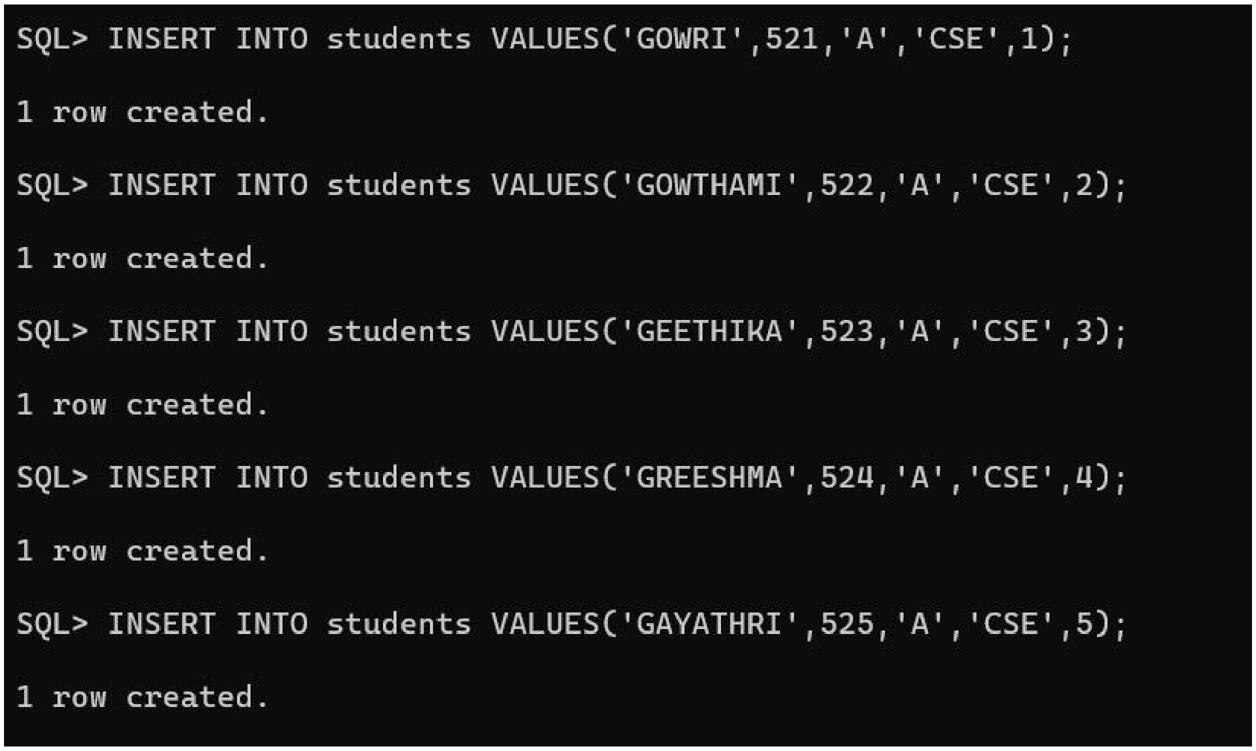
```
SQL-CSE530>DELETE from address WHERE place='atp';

1 row deleted.

SQL-CSE530>select * from address;

PLACE         PINCODE VILLAGE    DISTRICT
---------- ---------- ---------- ----------
dmm            515671 colony     satya sai
nandyal        615898 nijampet   kurnool
```

## EXPERIMENT – 3

**Aim:** To implement a view level design using CREATE VIEW,ALTER VIEW and DELETE VIEW ddl commands.

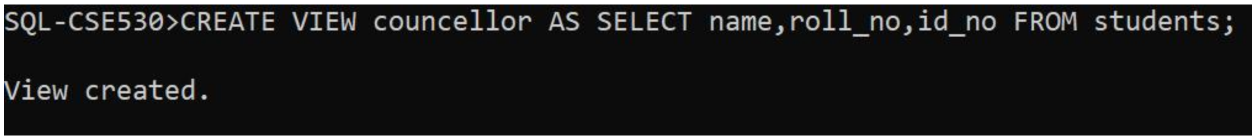


```
SQL> CREATE TABLE students(
  2  name VARCHAR(10),
  3  roll_no NUMBER,
  4  sec VARCHAR(5),
  5  Branch VARCHAR(10),
  6  id_no NUMBER,
  7  PRIMARY KEY(ID_NO)
  8  );

Table created.
```



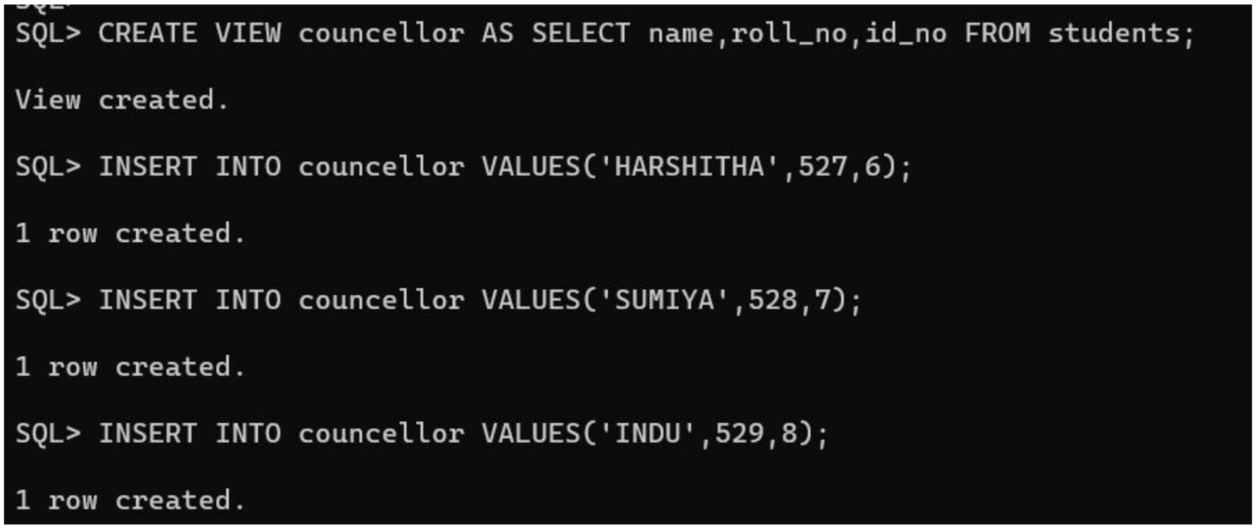
```
SQL> INSERT INTO students VALUES('GOWRI',521,'A','CSE',1);

1 row created.

SQL> INSERT INTO students VALUES('GOWTHAMI',522,'A','CSE',2);

1 row created.

SQL> INSERT INTO students VALUES('GEETHIKA',523,'A','CSE',3);

1 row created.

SQL> INSERT INTO students VALUES('GREESHMA',524,'A','CSE',4);

1 row created.

SQL> INSERT INTO students VALUES('GAYATHRI',525,'A','CSE',5);

1 row created.
```

## Creating view councellor:

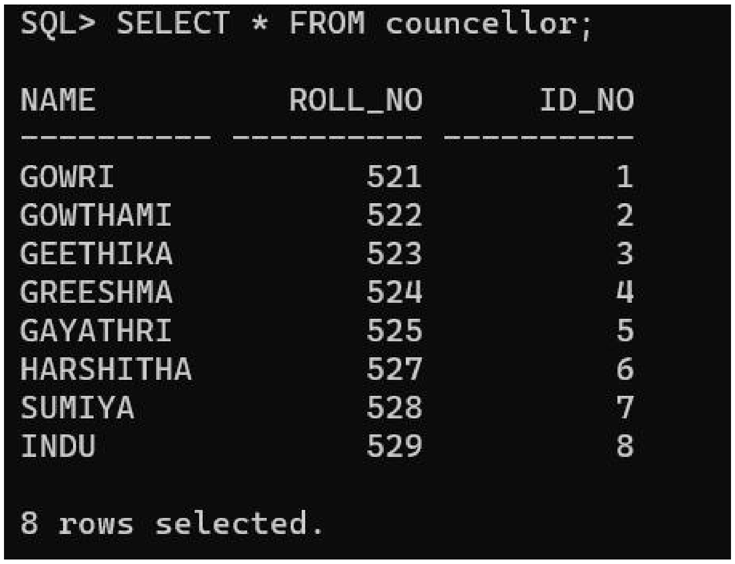


```
SQL-CSE530>CREATE VIEW councellor AS SELECT name,roll_no,id_no FROM students;

View created.
```

Inserting values into councellor:



```
SQL> CREATE VIEW councellor AS SELECT name,roll_no,id_no FROM students;

View created.

SQL> INSERT INTO councellor VALUES('HARSHITHA',527,6);

1 row created.

SQL> INSERT INTO councellor VALUES('SUMIYA',528,7);

1 row created.

SQL> INSERT INTO councellor VALUES('INDU',529,8);

1 row created.
```



```
SQL> SELECT * FROM councellor;

NAME          ROLL_NO      ID_NO
---------- ---------- ----------
GOWRI             521          1
GOWTHAMI          522          2
GEETHIKA          523          3
GREESHMA          524          4
GAYATHRI          525          5
HARSHITHA         527          6
SUMIYA            528          7
INDU              529          8

8 rows selected.
```

Selecting specific row :

Program Description:

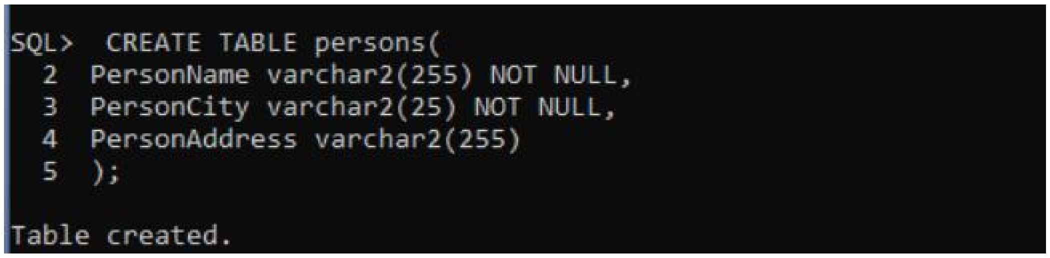
In this lab, we study the high-level design implementation of databases by using various view commands like create view, alter view and delete view.

A relation that is not part of a logical model, but it is made visible to a user as a virtual relation, is called a view. Therefore, view is referred as virtual relation.

## CREATE TABLE syntax:

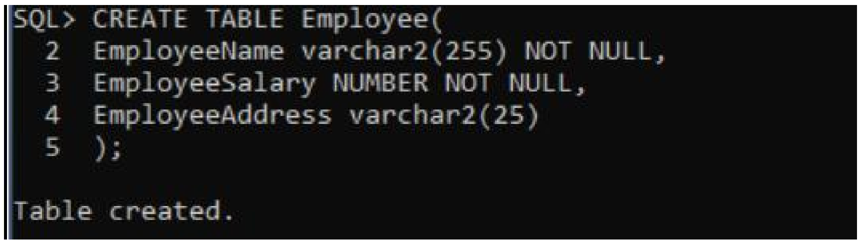
```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

EXAMPLE-1:



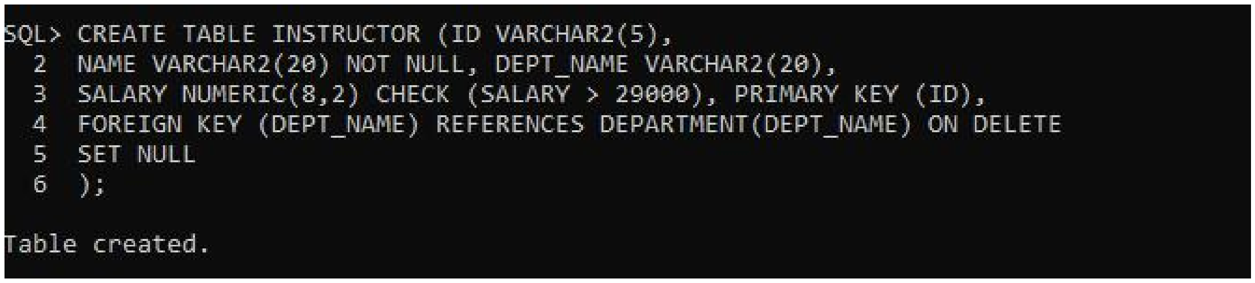
```
SQL>  CREATE TABLE persons(
  2  PersonName varchar2(255) NOT NULL,
  3  PersonCity varchar2(25) NOT NULL,
  4  PersonAddress varchar2(255)
  5  );

Table created.
```

EXAMPLE -2:



```
SQL> CREATE TABLE Employee(
  2  EmployeeName varchar2(255) NOT NULL,
  3  EmployeeSalary NUMBER NOT NULL,
  4  EmployeeAddress varchar2(25)
  5  );

Table created.
```

EXAMPLE-3



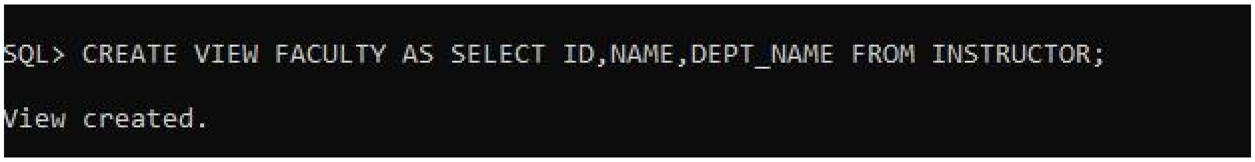
```
SQL> CREATE TABLE INSTRUCTOR (ID VARCHAR2(5),
  2  NAME VARCHAR2(20) NOT NULL, DEPT_NAME VARCHAR2(20),
  3  SALARY NUMERIC(8,2) CHECK (SALARY > 29000), PRIMARY KEY (ID),
  4  FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME) ON DELETE
  5  SET NULL
  6  );

Table created.
```

CREATE VIEW Syntax:

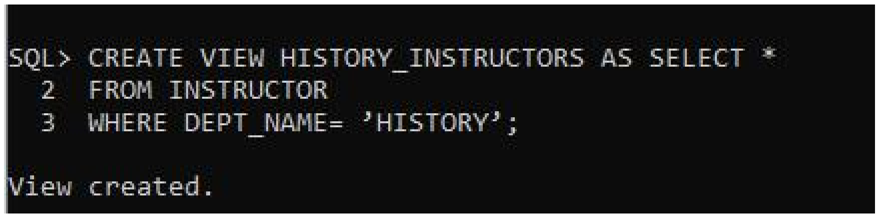
## View syntax:

CREATE VIEW VIEW_NAME AS <QUERY EXPRESSION>

## EXAMPLE:



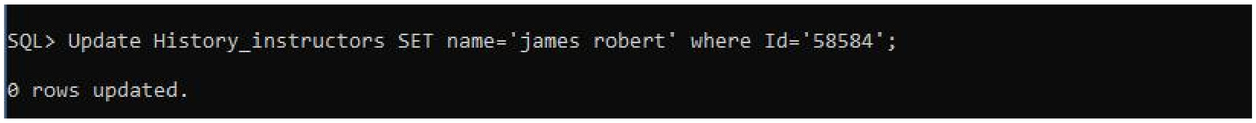
```
SQL> CREATE VIEW FACULTY AS SELECT ID,NAME,DEPT_NAME FROM INSTRUCTOR;

View created.
```



```
SQL> CREATE VIEW HISTORY_INSTRUCTORS AS SELECT *
  2  FROM INSTRUCTOR
  3  WHERE DEPT_NAME= 'HISTORY';

View created.
```

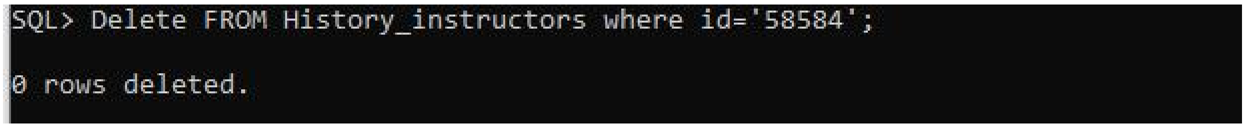
## UPDATE :

## EXAMPLE:



```
SQL> Update History_instructors SET name='james robert' where Id='58584';

0 rows updated.
```

## DELETE:



```
SQL> Delete FROM History_instructors where id='58584';

0 rows deleted.
```

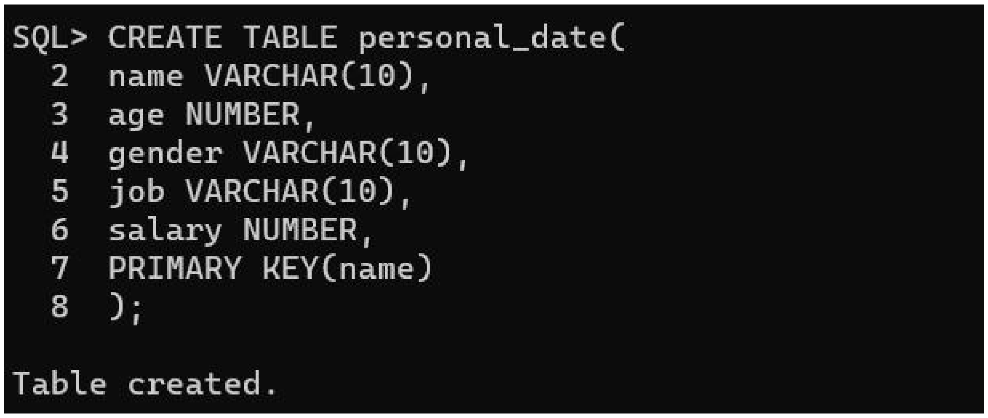
CONCLUSION:

In this Lab, We practice how to create view ,alter view , delete views.

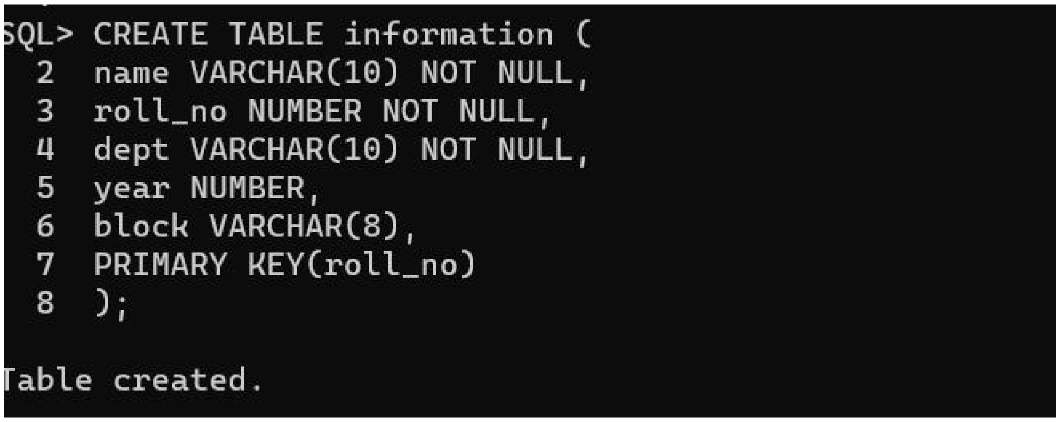
## EXPERIMENT-4

AIM : To create/perform relational set operations(i.e UNION UNIONALL,INTERSECT,MINUS,CROSS JOIN,NATURAL , JOIN.)

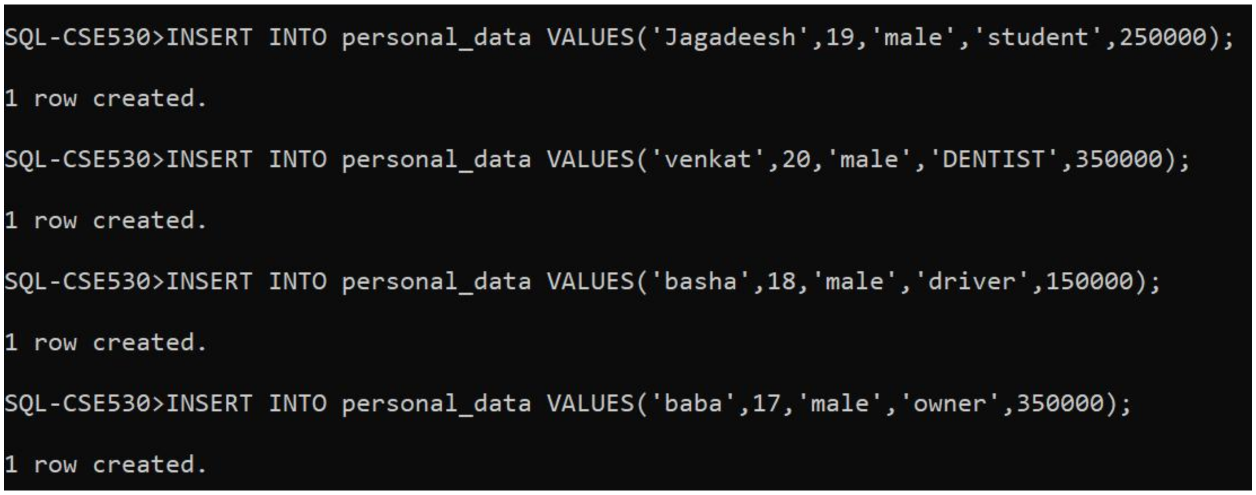
Creating tables :



```
SQL> CREATE TABLE personal_date(
name VARCHAR(10),
age NUMBER,
gender VARCHAR(10),
job VARCHAR(10),
salary NUMBER,
PRIMARY KEY(name)
);

Table created.
```

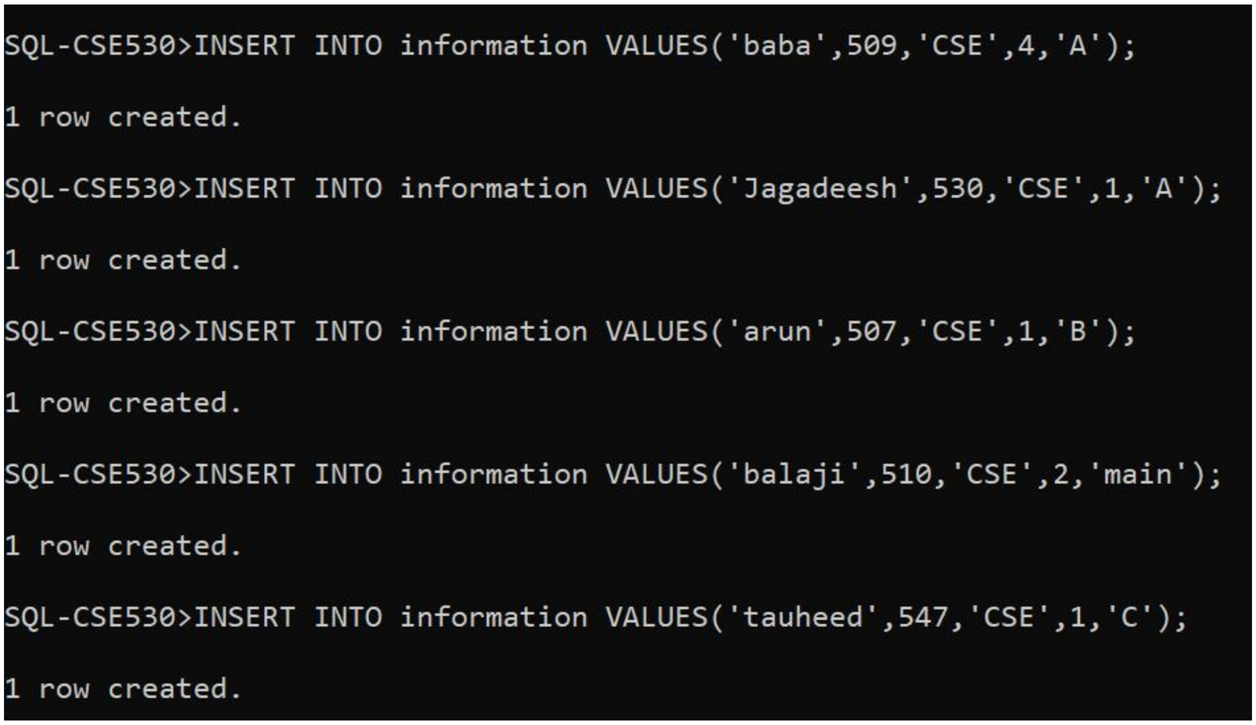


```
SQL> CREATE TABLE information (
name VARCHAR(10) NOT NULL,
roll_no NUMBER NOT NULL,
dept VARCHAR(10) NOT NULL,
year NUMBER,
block VARCHAR(8),
PRIMARY KEY(roll_no)
);

Table created.
```



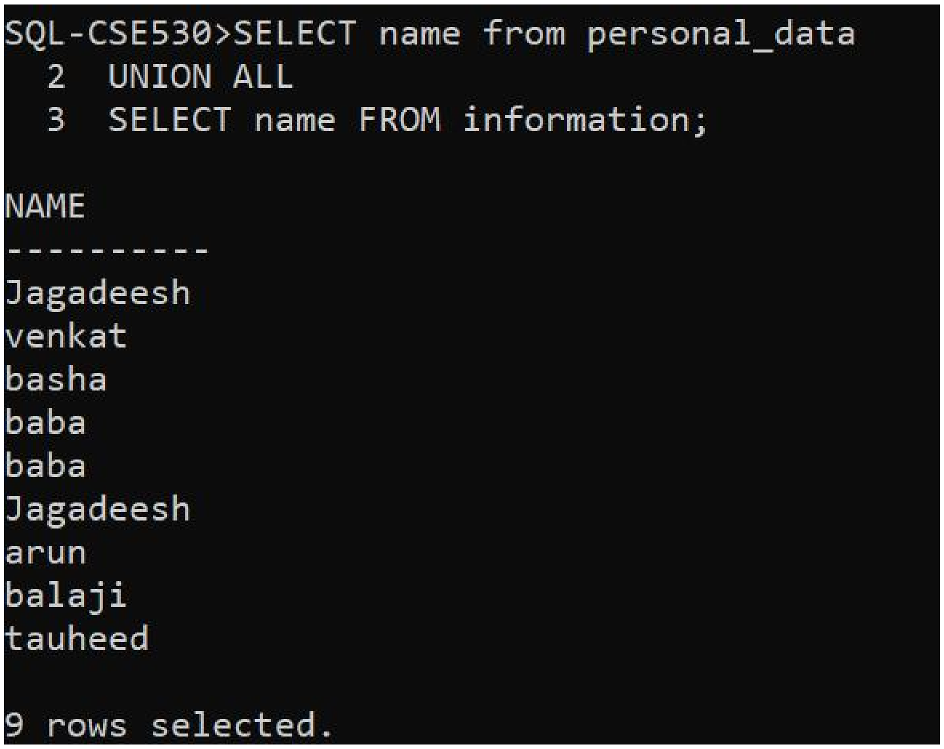
```
SQL-CSE530>INSERT INTO personal_data VALUES('Jagadeesh',19,'male','student',250000);

1 row created.

SQL-CSE530>INSERT INTO personal_data VALUES('venkat',20,'male','DENTIST',350000);

1 row created.

SQL-CSE530>INSERT INTO personal_data VALUES('basha',18,'male','driver',150000);

1 row created.

SQL-CSE530>INSERT INTO personal_data VALUES('baba',17,'male','owner',350000);

1 row created.
```

## Inserting values into **information** table :



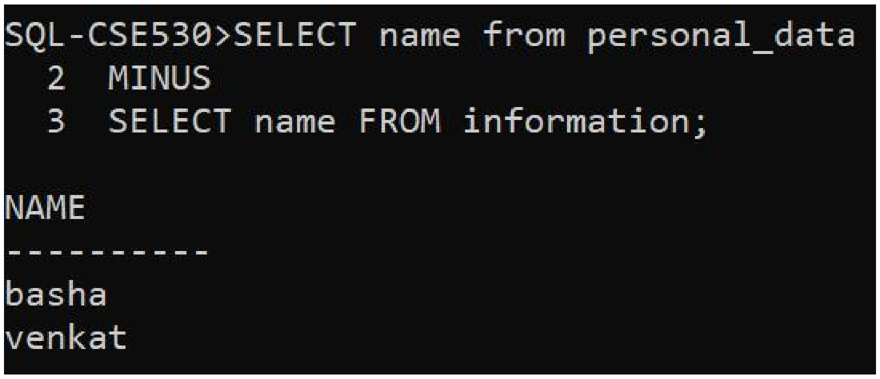
```
SQL-CSE530>INSERT INTO information VALUES('baba',509,'CSE',4,'A');

1 row created.

SQL-CSE530>INSERT INTO information VALUES('Jagadeesh',530,'CSE',1,'A');

1 row created.

SQL-CSE530>INSERT INTO information VALUES('arun',507,'CSE',1,'B');

1 row created.

SQL-CSE530>INSERT INTO information VALUES('balaji',510,'CSE',2,'main');

1 row created.

SQL-CSE530>INSERT INTO information VALUES('tauheed',547,'CSE',1,'C');

1 row created.
```

## Union all operation :



```
SQL-CSE530>SELECT name from personal_data
  2  UNION ALL
  3  SELECT name FROM information;

NAME
----------
Jagadeesh
venkat
basha
baba
baba
Jagadeesh
arun
balaji
tauheed

9 rows selected.
```

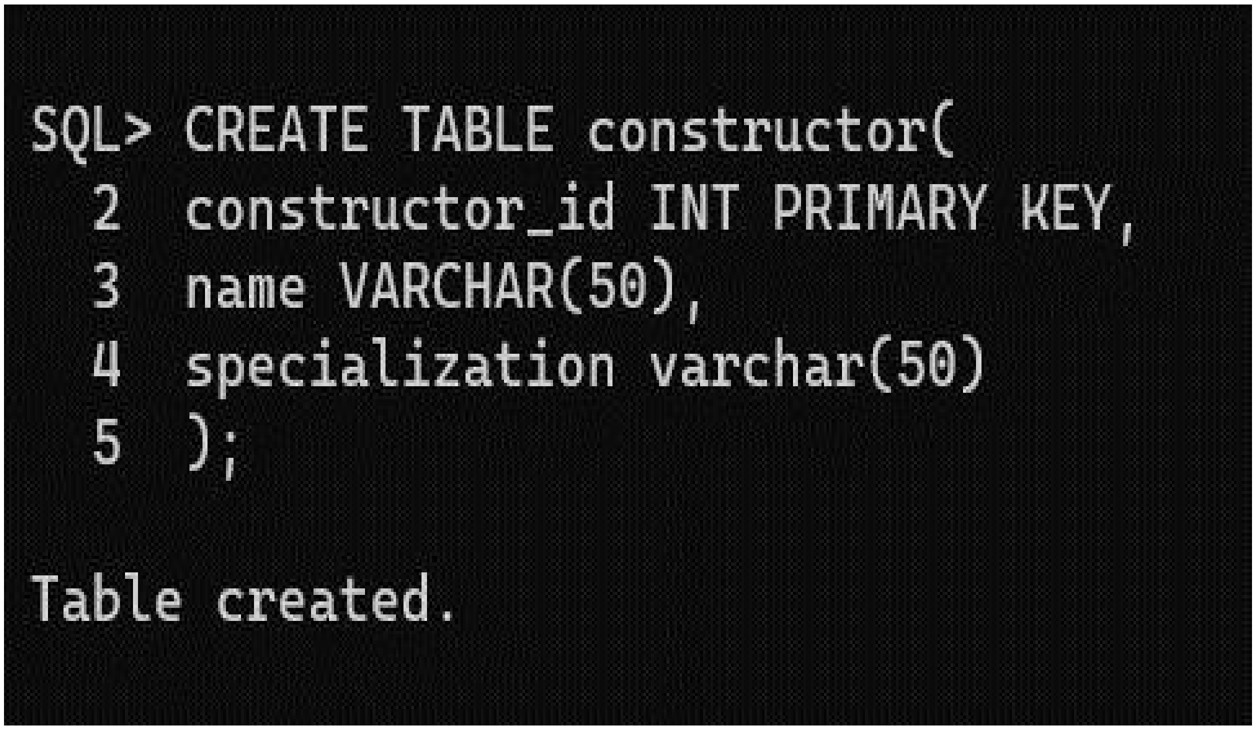
## Intersect operation:



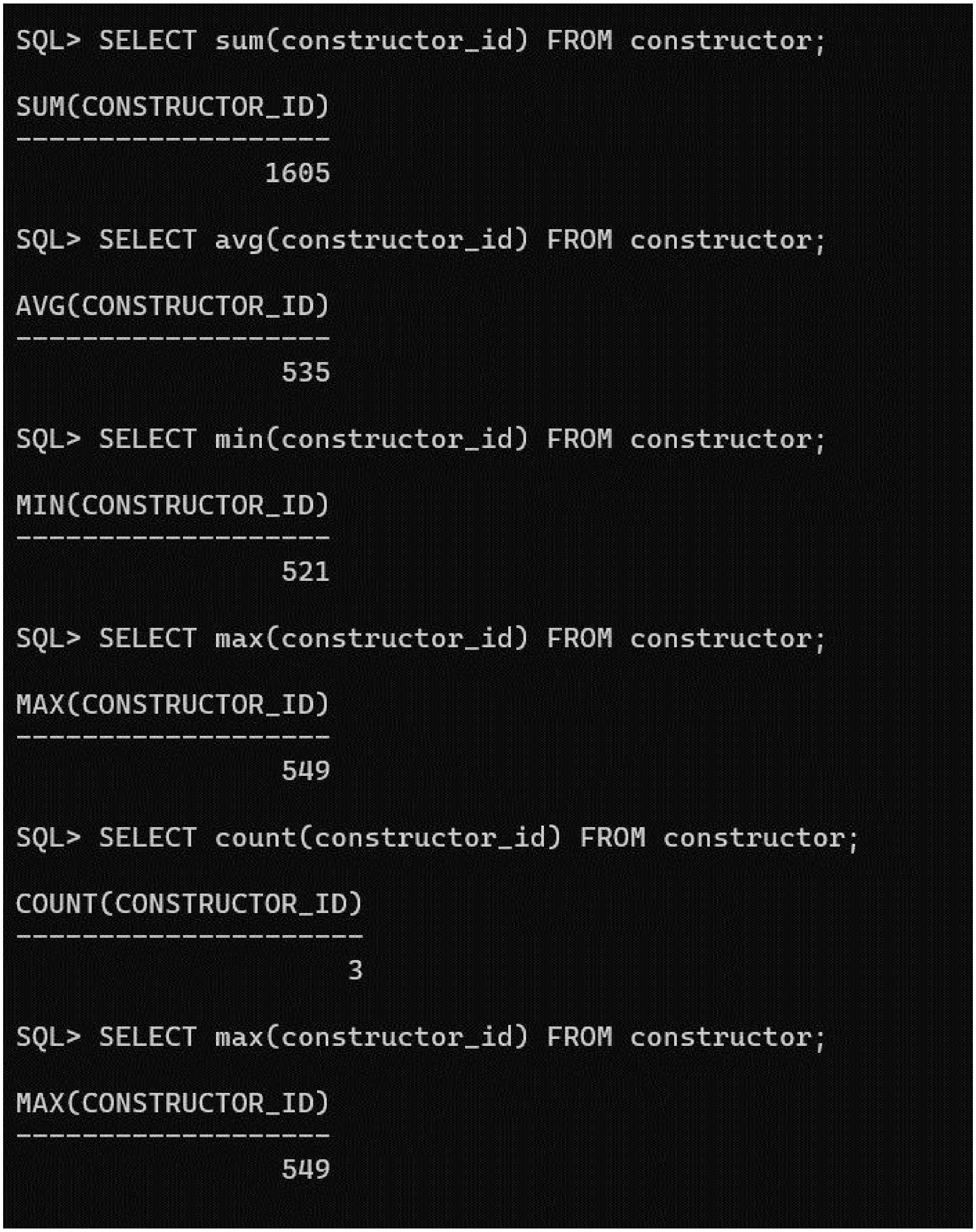
```
SQL-CSE530>SELECT name from personal_data
  2  MINUS
  3  SELECT name FROM information;

NAME
----------
basha
venkat
```

# EXPERIMENT-5

QUETSION : Write the SQL queries for Aggregate Functions(i.e. COUNT,AVG,MAX,SUM).



```
SQL> CREATE TABLE constructor(
  2  constructor_id INT PRIMARY KEY,
  3  name VARCHAR(50),
  4  specialization varchar(50)
  5  );

Table created.
```



```
SQL> SELECT sum(constructor_id) FROM constructor;

SUM(CONSTRUCTOR_ID)
-------------------
               1605

SQL> SELECT avg(constructor_id) FROM constructor;

AVG(CONSTRUCTOR_ID)
-------------------
                535

SQL> SELECT min(constructor_id) FROM constructor;

MIN(CONSTRUCTOR_ID)
-------------------
                521

SQL> SELECT max(constructor_id) FROM constructor;

MAX(CONSTRUCTOR_ID)
-------------------
                549

SQL> SELECT count(constructor_id) FROM constructor;

COUNT(CONSTRUCTOR_ID)
---------------------
                    3

SQL> SELECT max(constructor_id) FROM constructor;

MAX(CONSTRUCTOR_ID)
-------------------
                549
```

CONCLUSION : in this lab we studied about the Aggreagate functions.

# EXPERIMENT - 6

**Write SQL queries to perform JOIN OPERATIONS (i.e. CONDITIONAL JOIN, EQUI JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN)**
**Natural JOIN**
**For all instructors in the university who have taught some course, find their names and the course ID of all courses they taught"**

```
select name, course id
from instructor natural join teaches;
```

**An Equivalent Query**

```
select name, course id
from instructor, teaches
where instructor.ID = teaches.ID;
```

**List the names of instructors along with the the titles of courses that they teach.**

```
select name, title
from (instructor natural join teaches) join course using (course id);
```

**An Equivalent Query**

```
select name, title
from instructor natural join teaches, course
where teaches.course_id= course.course_id;
```

**Did the Following Query works fine or not Justify your Answer.**

```
select name, title
from instructor natural join teaches natural join course;
```

**JOIN with ON keyworkd refers it as Conditional JOIN**

```
select *
from student join takes on student.ID = takes.ID ;
select student.ID as ID , name, dept_name, tot_cred,
course_id, sec_id, semester, year, grade
from student join takes on student.ID = takes.ID ;
```

**OUTER JOINS**
**LEFT OUTER JOIN preserves tuples only in the relation named before (to the left of) the left outer join operation.**

```
select *
from student natural left outer join takes;
```

**To Find all students who have not taken a course.**

```
select ID
from student natural left outer join takes
where course id is null;
```

**RIGHT OUTER JOIN preserves tuples only in the relation named after (to the right of) the right outer join operation.**

```
select *
from takes natural right outer join student;
```

**Full Outer JOIN preserves tuples in both relations.**
**Display a list of all students in the Comp. Sci. department, along with the course sections, if any, that they have taken in Spring 2009; all course sections from Spring 2009 must be displayed, even if no student from the Comp. Sci. department has taken the course section."**

```
select *
from (select *
from student
where dept name= ’Comp. Sci’)
natural full outer join
(select *
from takes
where semester = ’Spring’ and year = 2009);
```

# EXPERIMENT – 8

QUESTION: Write SQL queries to perform ORACLE BUILT-IN FUNCTIONS (i.e. DATE, TIME)

AIM: in this experiment we have to know about the how Write SQL queries to perform ORACLE BUILT-IN FUNCTIONS (i.e. DATE, TIME)

Built-in Functions

Character Functions

case-conversion functions

character manipulation functions

Number Functions DATE functions

## case-conversion functions:

```
SELECT LOWER('SQL Course')
 FROM DUAL;
 SELECT UPPER('SQL Course')
 FROM DUAL;
SELECT INITCAP('SQL course')
FROM DUAL;
```

## character manipulation functions:

```
SELECT CONCAT('HELLO', 'WORLD')
 FROM DUAL;
 SELECT SUBSTR('HELLO WORLD',1,5)
 FROM DUAL;
 SELECT LENGTH('HELLO WORLD');
FROM DUAL;
SELECT INSTR('HELLO WORLD', 'WORLD')
 FROM DUAL;
SELECT LPAD(SALARY, 10, '*')
 FROM INSTRUCTOR;
 SELECT RPAD(SALARY, 10, '*')
 FROM INSTRUCTOR;
 SELECT REPLACE('JACK and JUE','J','BL')
FROM DUAL;
 SELECT TRIM('H' FROM 'HelloWorld'
 FROM DUAL;
```

## Number Functions:

```
SELECT ROUND(45.626,2)
 FROM DUAL;
 SELECT ROUND(45.626,0)
 FROM DUAL;
 SELECT ROUND(45.626,-1)
FROM DUAL;
SELECT ROUND(45.626,-2)
 FROM DUAL;
 SELECT TRUNC(45.626, 2)
 FROM DUAL;
 SELECT TRUNC(45.626, 0)
 FROM DUAL;
SELECT TRUNC(45.626, -1)
 FROM DUAL;
SELECT TRUNC(45.626, -2)
 FROM DUAL;
 SELECT MOD(1600,300)
FROM DUAL;
```

## Date Functions:

Oracle database stores date in the internal numeric format of Century, year, month, day, hours, minutes and seconds;

The Default display format is DD-MON-RR

```
        SELECT SYSDATE
 FROM DUAL;
 SELECT MONTHS_BETWEEN(SYSDATE,'15-FEB-20')
 FROM DUAL;
SELECT ADD_MONTHS(SYSDATE, 2)
FROM DUAL;
 SELECT NEXT_DAY(SYSDATE,'THRUSDAY')
 FROM DUAL;
 SELECT LAST_DAY(SYSDATE)
FROM DUAL;
 SELECT ROUND('25-JUL-03','MONTH')
 FROM DUAL;
SELECT ROUND('25-JUL-03','YEAR')
FROM DUAL;
SELECT TRUNC('25-JUL-03','YEAR')
 FROM DUAL;
SELECT TRUNC('25-JUL-03','MONTH')
```

FROM DUAL;

## CONCLUTION:

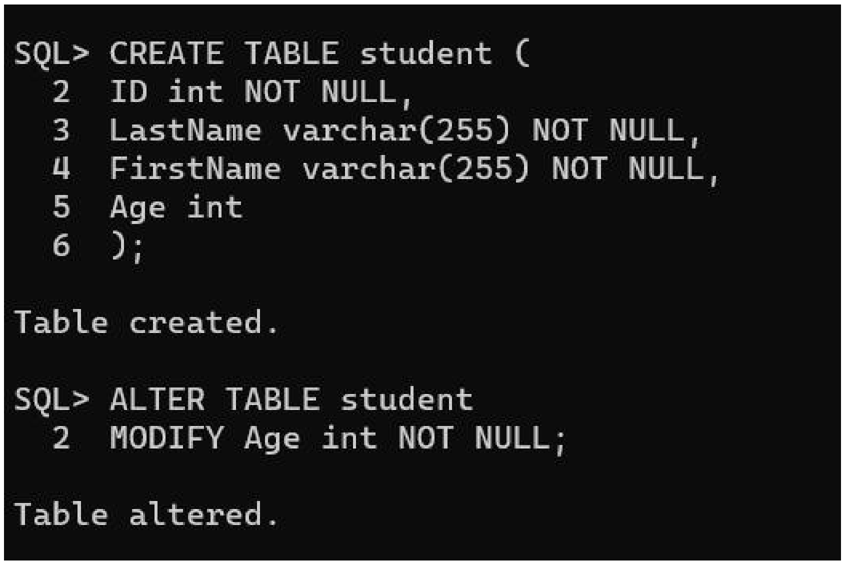
in this experiment we have to know about the how Write SQL queries to perform ORACLE BUILT-IN FUNCTIONS (i.e. DATE, TIME)

## EXPERIMENT - 9

AIM: Write SQL queries to perform KEY CONSTRAINTS (i.e. PRIMARY KEY, FOREIGN KEY, UNIQUE NOT NULL, CHECK, DEFAULT). Constraints SQL constraints are used to specify rules for the data in a table.

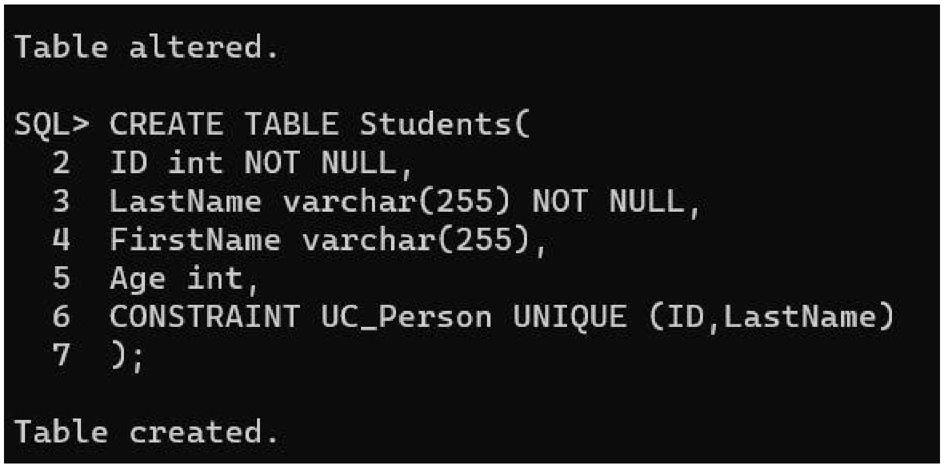
### Types of SQL Constraints:

NOT NULL Constraint Example:

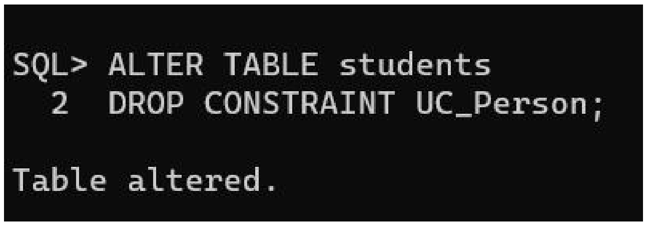


```
SQL> CREATE TABLE student (
  2  ID int NOT NULL,
  3  LastName varchar(255) NOT NULL,
  4  FirstName varchar(255) NOT NULL,
  5  Age int
  6  );

Table created.

SQL> ALTER TABLE student
  2  MODIFY Age int NOT NULL;

Table altered.
```

UNIQUE CONSTRAINT Example:

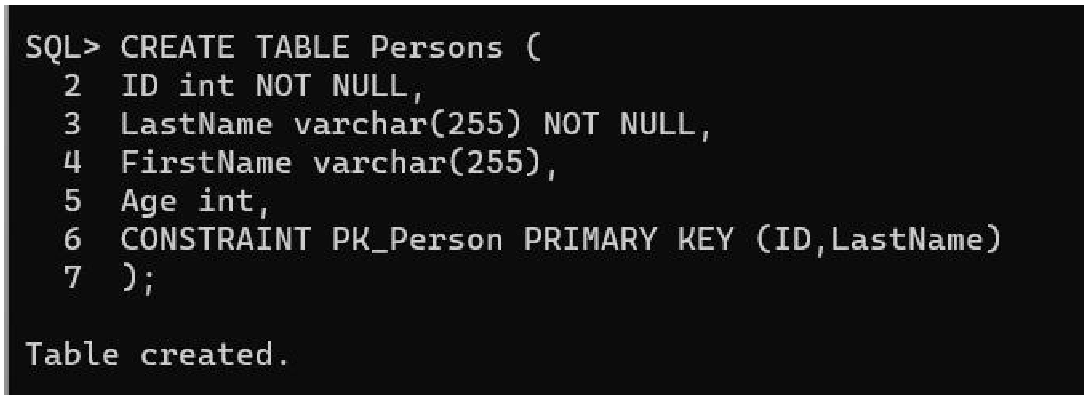


```
Table altered.

SQL> CREATE TABLE Students(
  2  ID int NOT NULL,
  3  LastName varchar(255) NOT NULL,
  4  FirstName varchar(255),
  5  Age int,
  6  CONSTRAINT UC_Person UNIQUE (ID,LastName)
  7  );

Table created.
```

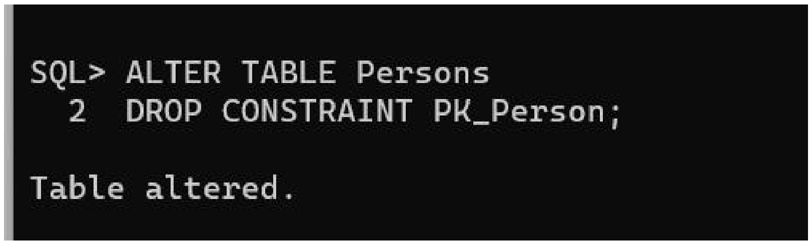


```
SQL> ALTER TABLE students
  2  DROP CONSTRAINT UC_Person;

Table altered.
```

PRIMARY KEY CONSTRAINT Example:

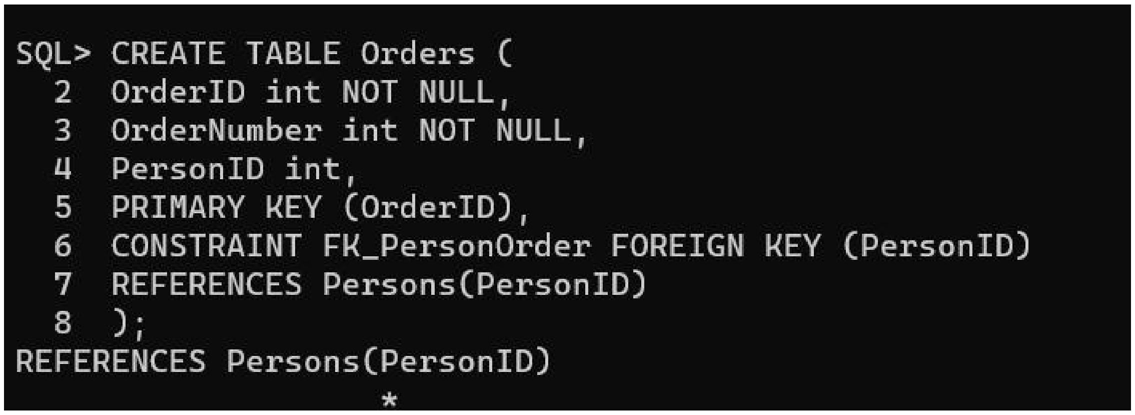


```
SQL> CREATE TABLE Persons (
  2  ID int NOT NULL,
  3  LastName varchar(255) NOT NULL,
  4  FirstName varchar(255),
  5  Age int,
  6  CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
  7  );

Table created.
```

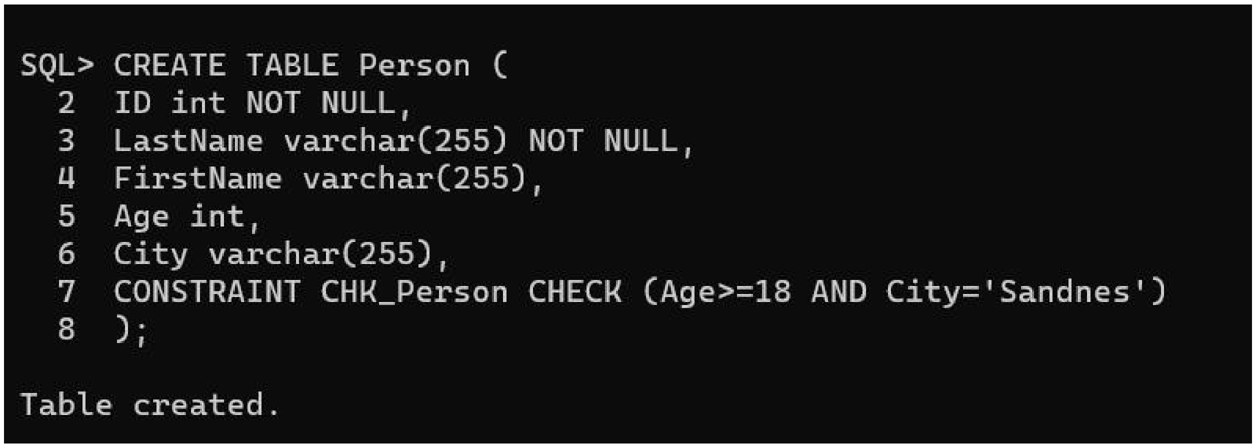


```
SQL> ALTER TABLE Persons
  2  DROP CONSTRAINT PK_Person;

Table altered.
```

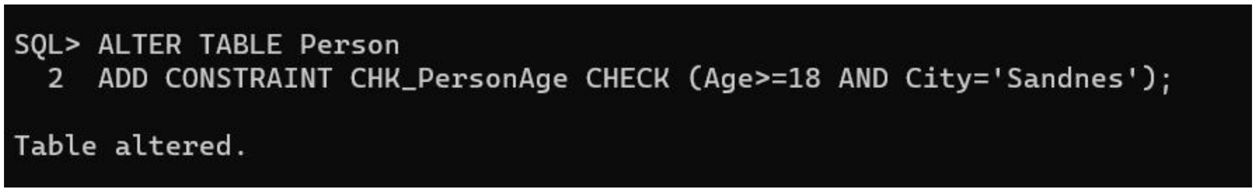
FORIEGN KEY CONSTRAINTS Example:

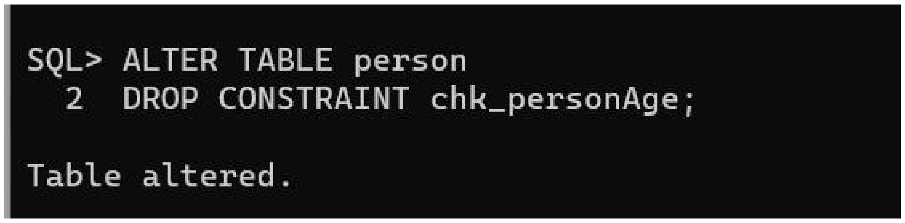

```
SQL> CREATE TABLE Orders (
  2  OrderID int NOT NULL,
  3  OrderNumber int NOT NULL,
  4  PersonID int,
  5  PRIMARY KEY (OrderID),
  6  CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
  7  REFERENCES Persons(PersonID)
  8  );
REFERENCES Persons(PersonID)
                   *
```

CHECK CONSTRAINTS Example:

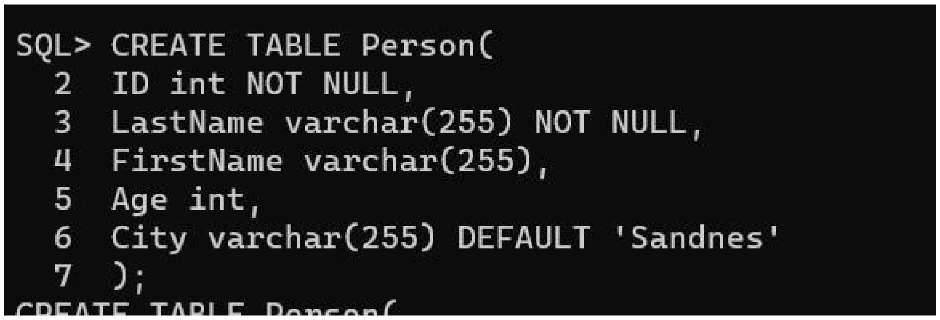

```
SQL> CREATE TABLE Person (
  2  ID int NOT NULL,
  3  LastName varchar(255) NOT NULL,
  4  FirstName varchar(255),
  5  Age int,
  6  City varchar(255),
  7  CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
  8  );

Table created.
```

ALTER TABLE Persons ADD CONSTRAINT_PersonAge CHECK (Age>=18 AND City='Sandnes');


```
SQL> ALTER TABLE Person
  2  ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');

Table altered.
```


```
SQL> ALTER TABLE person
  2  DROP CONSTRAINT chk_personAge;

Table altered.
```

DEFAULT CONSTRAINTS Example:



```
SQL> CREATE TABLE Person(
  2  ID int NOT NULL,
  3  LastName varchar(255) NOT NULL,
  4  FirstName varchar(255),
  5  Age int,
  6  City varchar(255) DEFAULT 'Sandnes'
  7  );
CREATE TABLE Person(
```

Drop cannot be dropped, So make it null value to remove default:

ALTER TABLE Persons MODIFY city DEFAULT NULL;

## CONCLUSION:

Write SQL queries to perform KEY CONSTRAINTS (i.e. PRIMARY KEY, FOREIGN KEY, UNIQUE NOT NULL, CHECK, DEFAULT). Constraints SQL constraints are used to specify rules for the data in a table.

# EXPERIMENT – 11

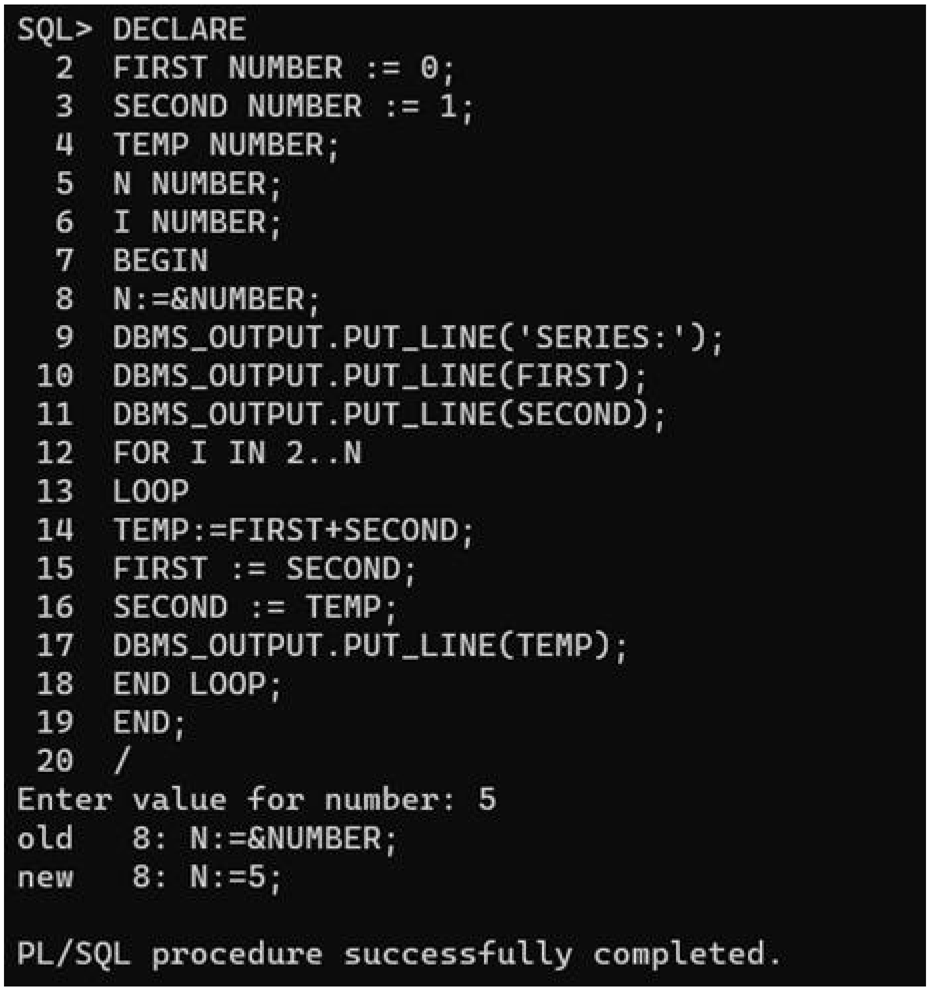
## QUESTION:

Write a PL/SQL program for finding the given number is prime number or not.

## AIM:

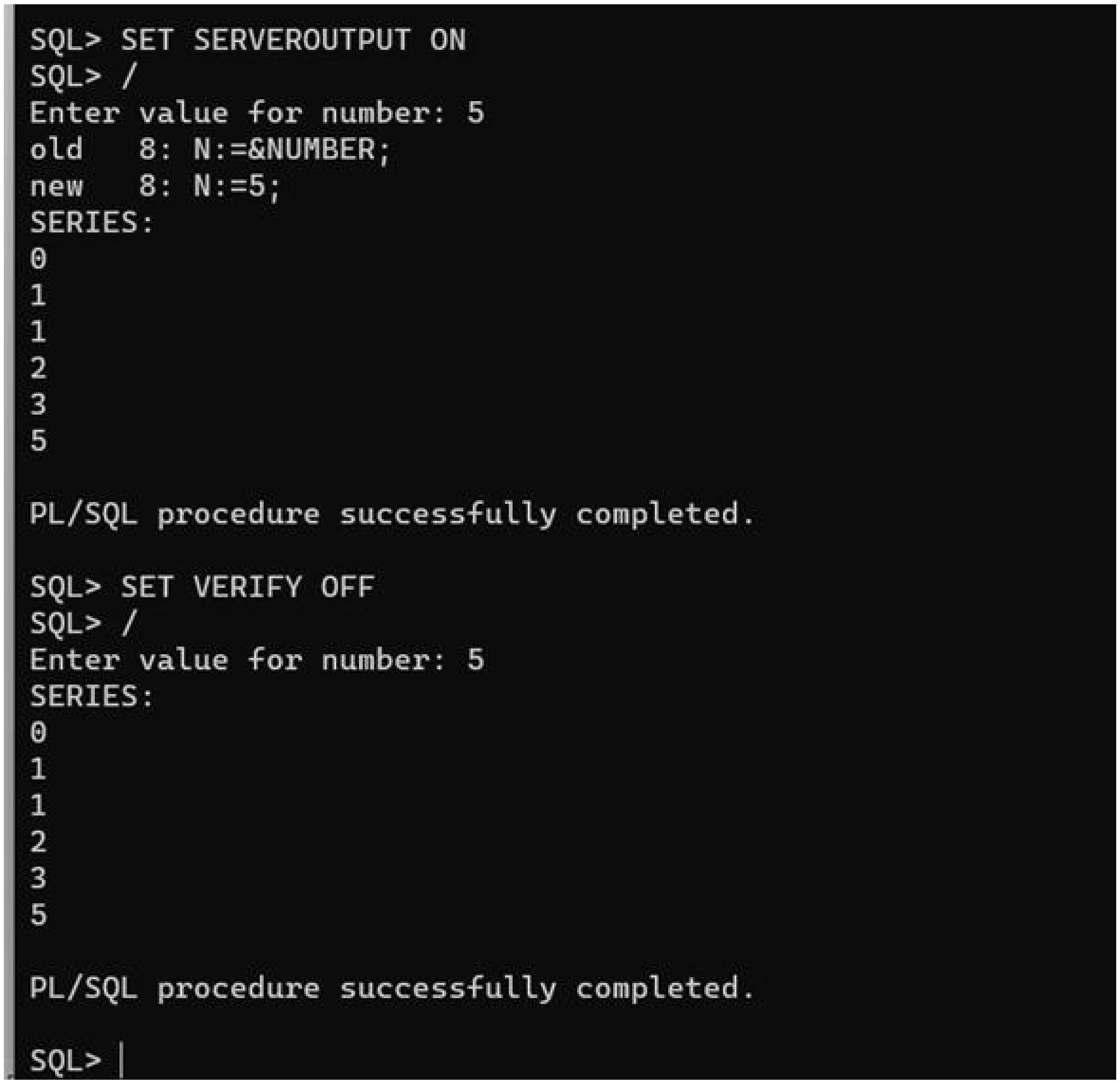
in this experiment we have to know about the how to Write a PL/SQL program for finding the given number is prime number or not.

Program Description:



```
SQL> DECLARE
  2  FIRST NUMBER := 0;
  3  SECOND NUMBER := 1;
  4  TEMP NUMBER;
  5  N NUMBER;
  6  I NUMBER;
  7  BEGIN
  8  N:=&NUMBER;
  9  DBMS_OUTPUT.PUT_LINE('SERIES:');
 10  DBMS_OUTPUT.PUT_LINE(FIRST);
 11  DBMS_OUTPUT.PUT_LINE(SECOND);
 12  FOR I IN 2..N
 13  LOOP
 14  TEMP:=FIRST+SECOND;
 15  FIRST := SECOND;
 16  SECOND := TEMP;
 17  DBMS_OUTPUT.PUT_LINE(TEMP);
 18  END LOOP;
 19  END;
 20  /
Enter value for number: 5
old   8: N:=&NUMBER;
new   8: N:=5;

PL/SQL procedure successfully completed.
```

OUTPUT:



```
SQL> SET SERVEROUTPUT ON
SQL> /
Enter value for number: 5
old   8: N:=&NUMBER;
new   8: N:=5;
SERIES:
0
1
1
2
3
5

PL/SQL procedure successfully completed.

SQL> SET VERIFY OFF
SQL> /
Enter value for number: 5
SERIES:
0
1
1
2
3
5

PL/SQL procedure successfully completed.

SQL> |
```

## CONCLUSION:

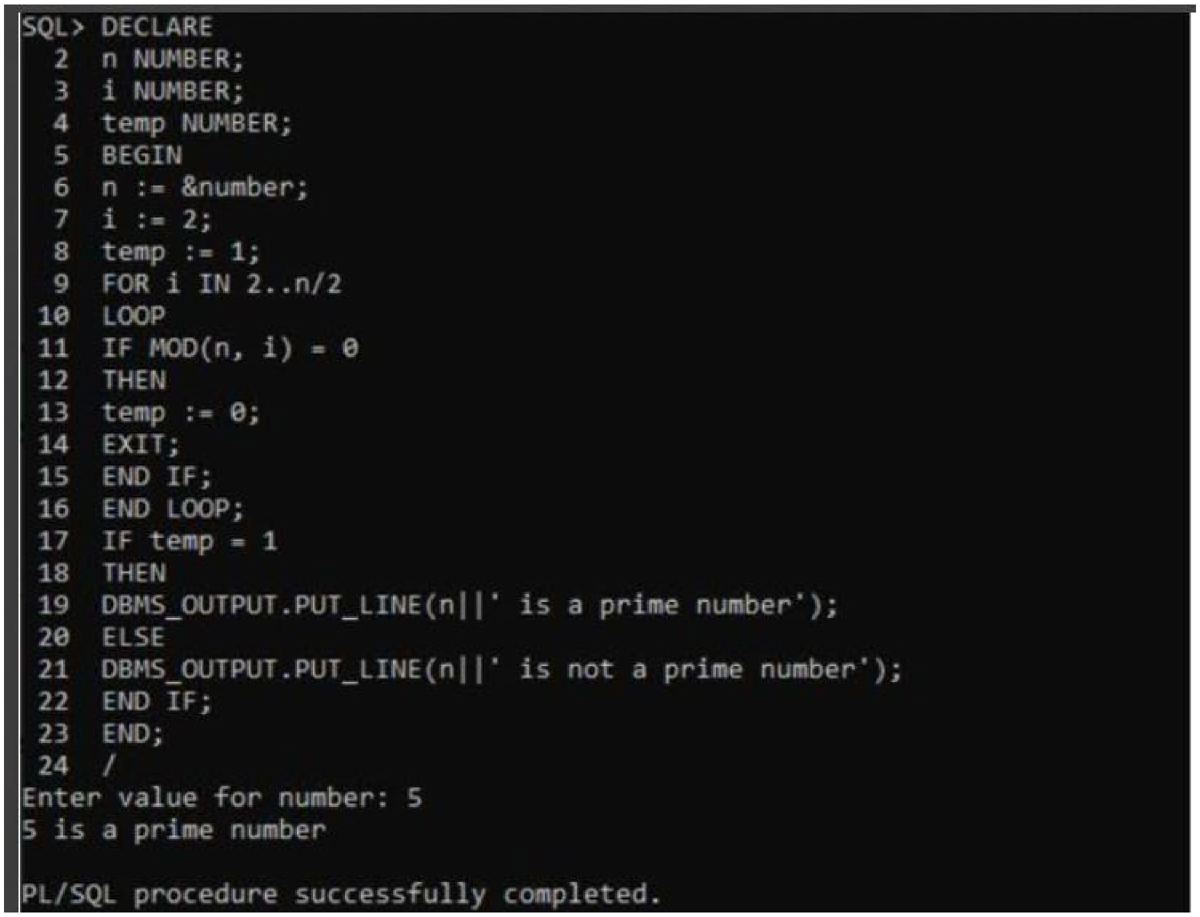
in this experiment we have to know about the how to Write a PL/SQL program for finding the given number is prime number or not.

## EXPERIMENT – 12

QUESTION: Write a PL/SQL program for displaying the Fibonacci series up to an integer.

AIM : Write a PL/SQL program for displaying the Fibonacci series up to an integer.

Program Description:



```
SQL> DECLARE
n NUMBER;
i NUMBER;
temp NUMBER;
BEGIN
n := &number;
i := 2;
temp := 1;
FOR i IN 2..n/2
LOOP
IF MOD(n, i) = 0
THEN
temp := 0;
EXIT;
END IF;
END LOOP;
IF temp = 1
THEN
DBMS_OUTPUT.PUT_LINE(n||' is a prime number');
ELSE
DBMS_OUTPUT.PUT_LINE(n||' is not a prime number');
END IF;
END;
/
Enter value for number: 5
5 is a prime number

PL/SQL procedure successfully completed.
```

OUTPUT:

CONCLUTION:

In this experiment we learnt about how to Write a PL/SQL program for displaying the Fibonacci series up to an integer.

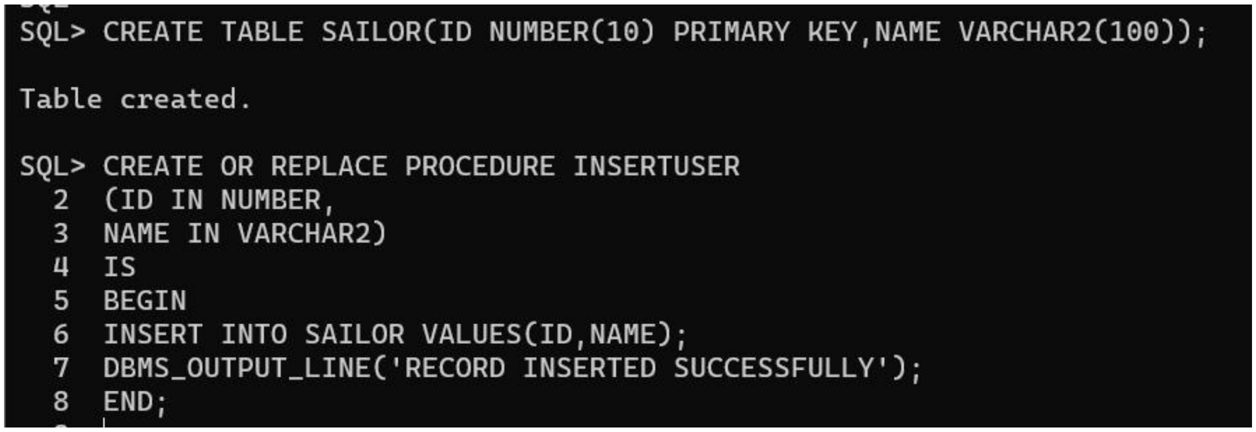
## EXPERIMENT-13

## Write PL/SQL program to implement Stored Procedure on table.

**AIM:** Write PL/SQL program to implement Stored Procedure on table.

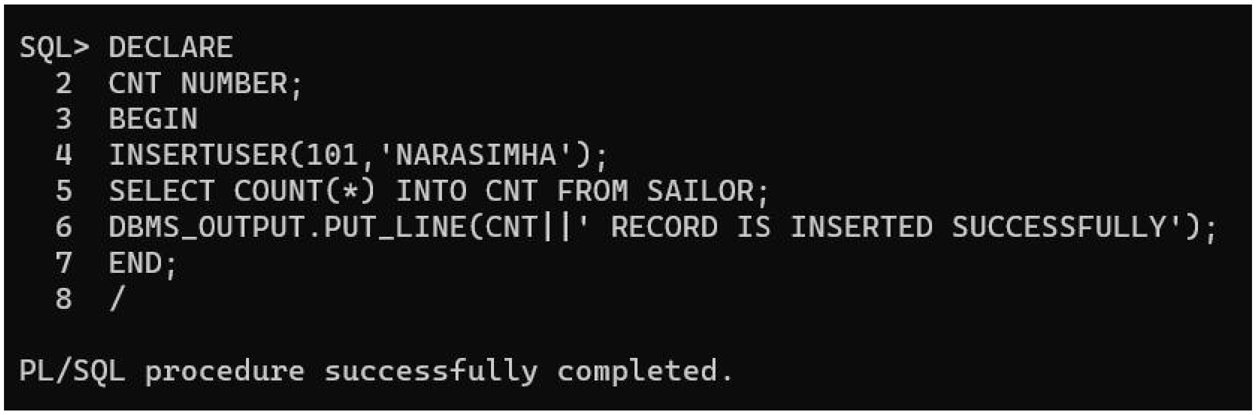
**PL/SQL Procedure**

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.
The procedure contains a header and a
body



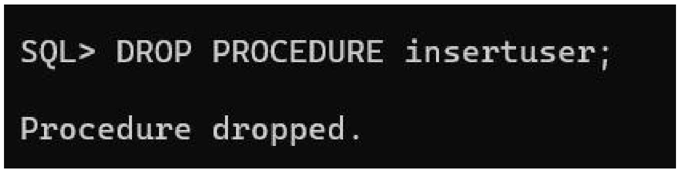
```
SQL> CREATE TABLE SAILOR(ID NUMBER(10) PRIMARY KEY,NAME VARCHAR2(100));

Table created.

SQL> CREATE OR REPLACE PROCEDURE INSERTUSER
(ID IN NUMBER,
NAME IN VARCHAR2)
IS
BEGIN
INSERT INTO SAILOR VALUES(ID,NAME);
DBMS_OUTPUT_LINE('RECORD INSERTED SUCCESSFULLY');
END;
```

**Execution Procedure:**



```
SQL> DECLARE
CNT NUMBER;
BEGIN
INSERTUSER(101,'NARASIMHA');
SELECT COUNT(*) INTO CNT FROM SAILOR;
DBMS_OUTPUT.PUT_LINE(CNT||' RECORD IS INSERTED SUCCESSFULLY');
END;
/

PL/SQL procedure successfully completed.
```

**DROP PROCEDURE:**



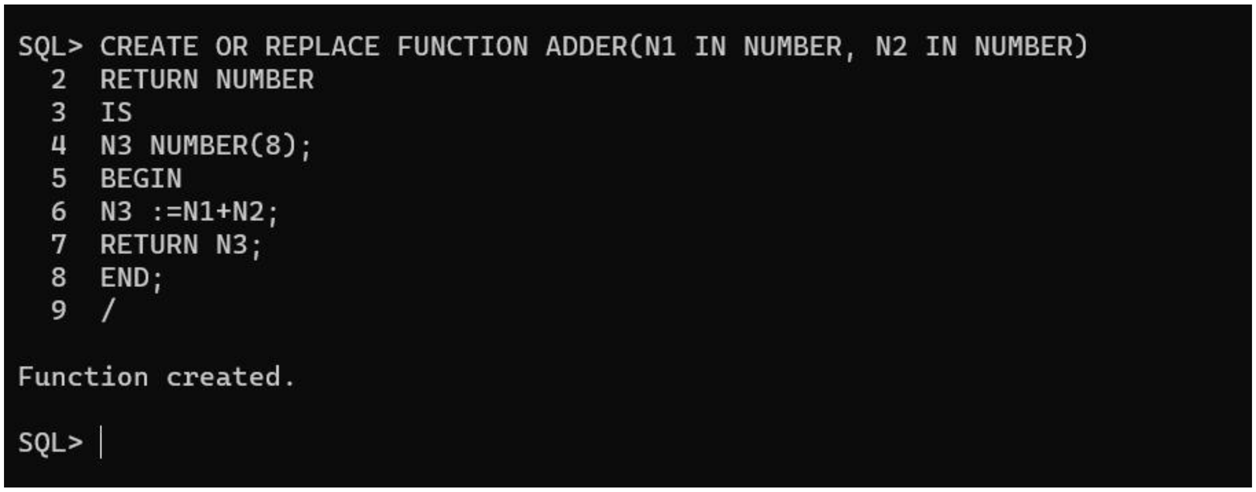
```
SQL> DROP PROCEDURE insertuser;

Procedure dropped.
```

## EXPERIMENT-14

AIM :TO Write PL/SQL program to implement Stored Function on table.

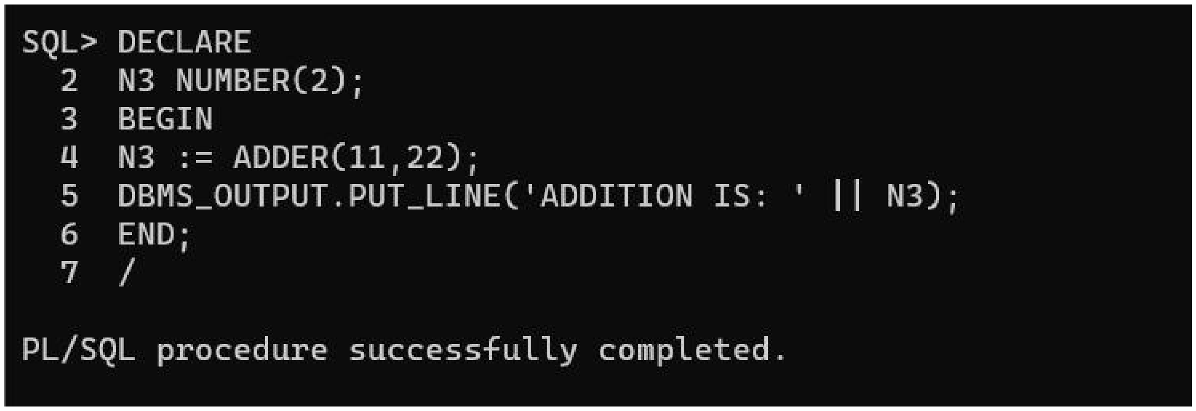
## PL/SQL Function:

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between
procedure and a function is, a function must always return a value, and on the other hand a
procedure may or may not return a value. Except this, all the other things of PL/SQL procedure
are true for PL/SQL function too.

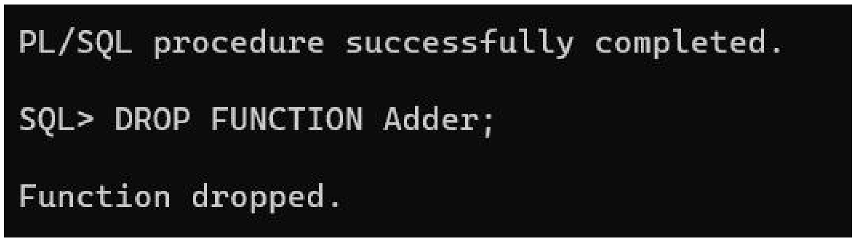


```
SQL> CREATE OR REPLACE FUNCTION ADDER(N1 IN NUMBER, N2 IN NUMBER)
RETURN NUMBER
IS
N3 NUMBER(8);
BEGIN
N3 :=N1+N2;
RETURN N3;
END;
/

Function created.

SQL>
```

### Execution Procedure:

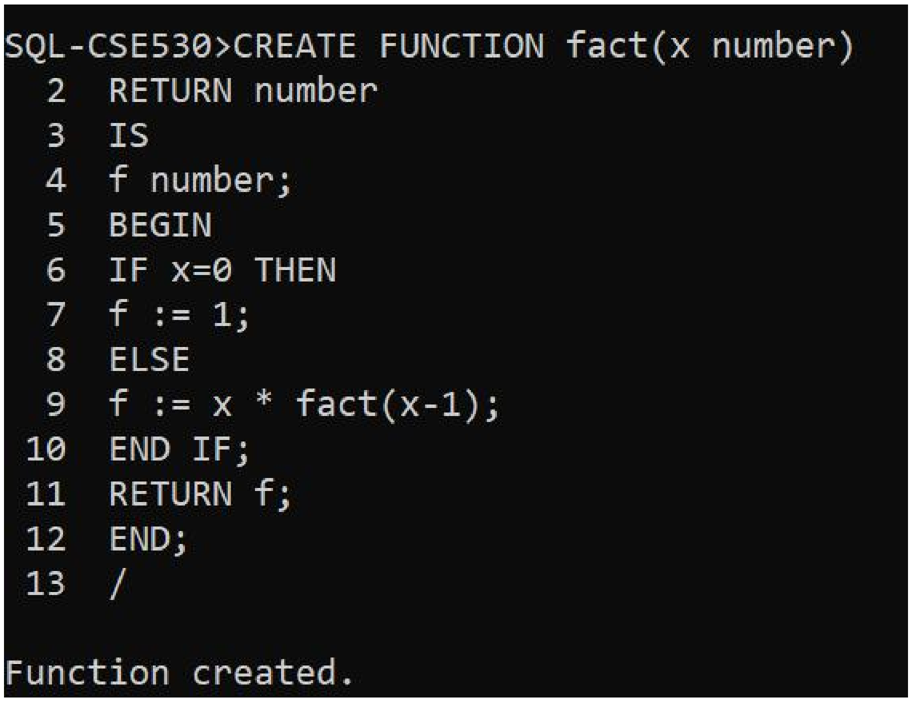


```
SQL> DECLARE
N3 NUMBER(2);
BEGIN
N3 := ADDER(11,22);
DBMS_OUTPUT.PUT_LINE('ADDITION IS: ' || N3);
END;
/

PL/SQL procedure successfully completed.
```



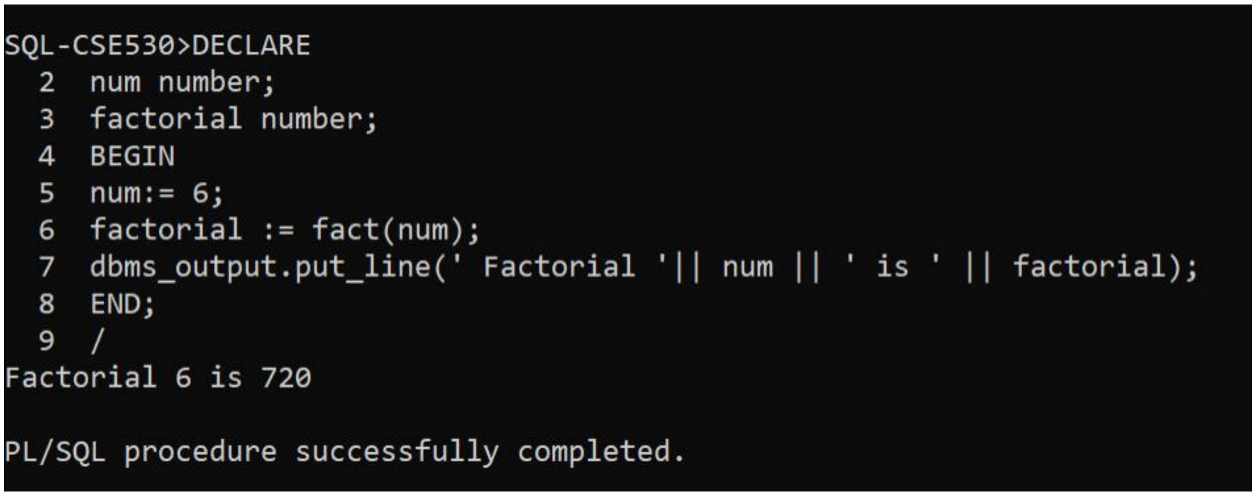
```
PL/SQL procedure successfully completed.

SQL> DROP FUNCTION Adder;

Function dropped.
```

## EXAMPLE-2

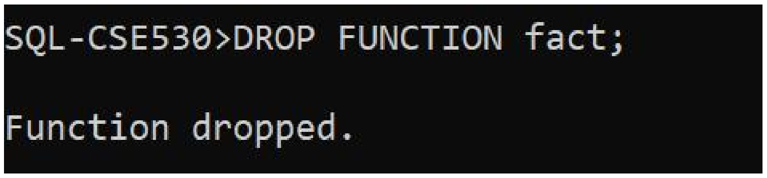


```
SQL-CSE530>CREATE FUNCTION fact(x number)
RETURN number
IS
f number;
BEGIN
IF x=0 THEN
f := 1;
ELSE
f := x * fact(x-1);
END IF;
RETURN f;
END;
/

Function created.
```

Execution Procedure:



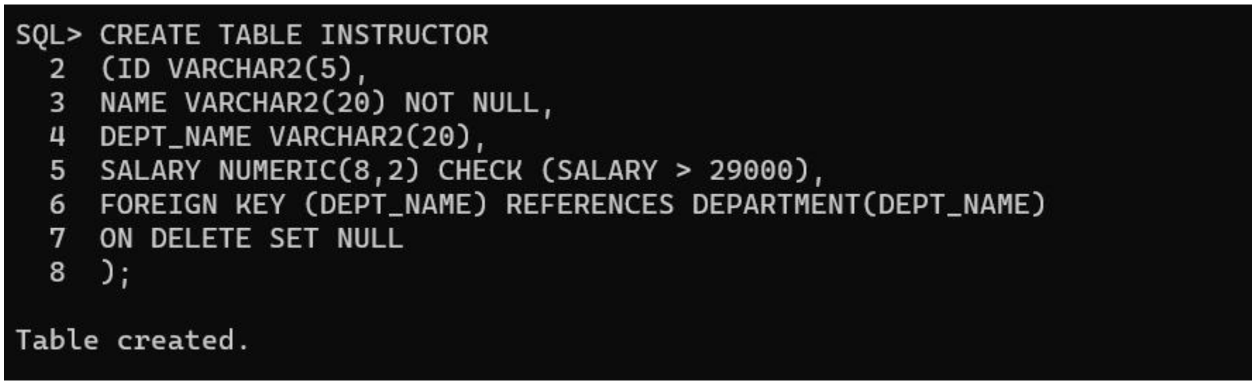
```
SQL-CSE530>DECLARE
num number;
factorial number;
BEGIN
num:= 6;
factorial := fact(num);
dbms_output.put_line(' Factorial '|| num || ' is ' || factorial);
END;
/
Factorial 6 is 720

PL/SQL procedure successfully completed.
```



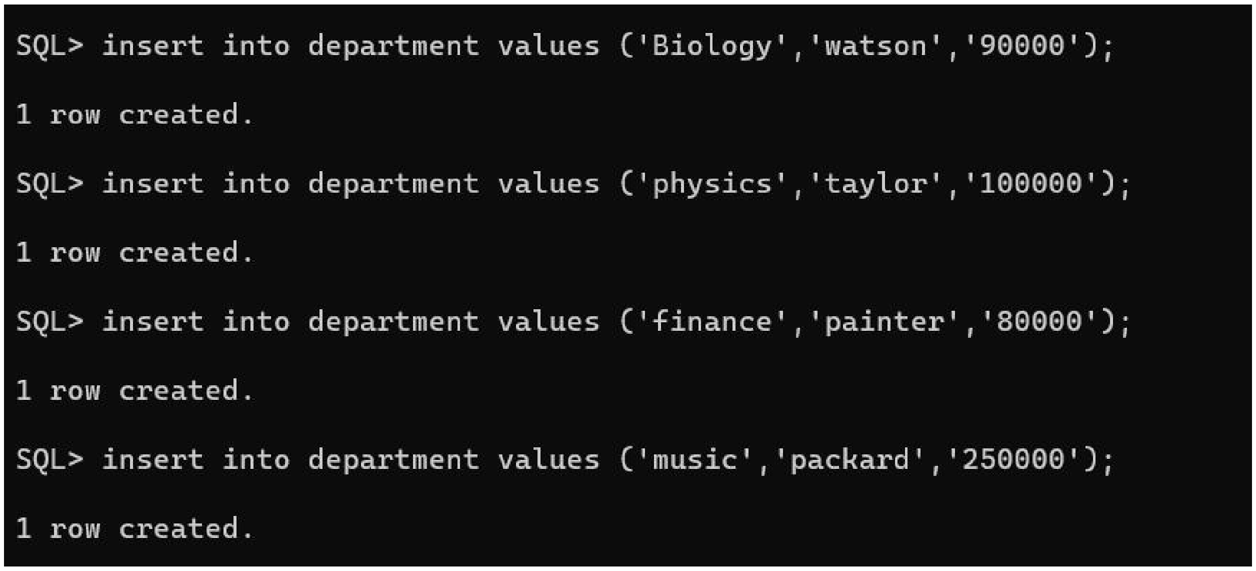
```
SQL-CSE530>DROP FUNCTION fact;

Function dropped.
```

## EXPERIMENT-15

AIM : TO Write PL/SQL program to implement Trigger on table.
Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored into database and invoked repeatedly, when specific condition match.
Triggers are stored programs, which are automatically executed or fired when some event
Occurs.
Triggers are written to be executed in response to any of the following events.
A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
A database definition (DDL) statement (CREATE, ALTER, or DROP).
A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN)

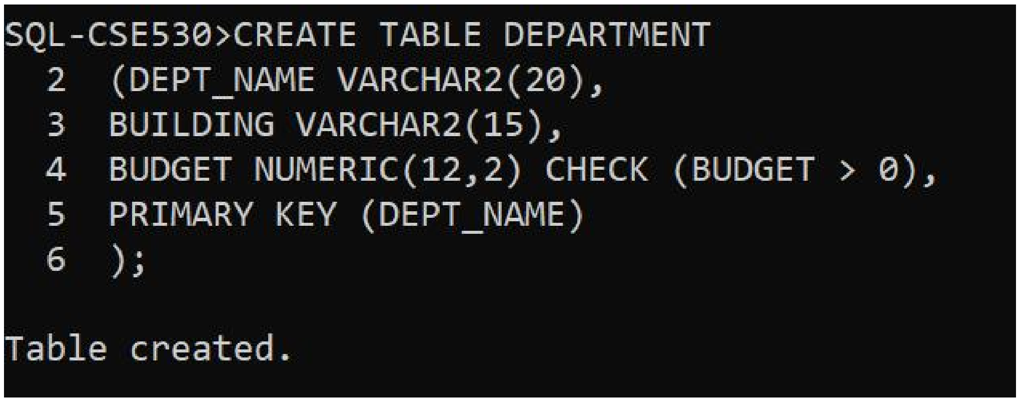


```
SQL> CREATE TABLE INSTRUCTOR
(ID VARCHAR2(5),
NAME VARCHAR2(20) NOT NULL,
DEPT_NAME VARCHAR2(20),
SALARY NUMERIC(8,2) CHECK (SALARY > 29000),
FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME)
ON DELETE SET NULL
);

Table created.
```



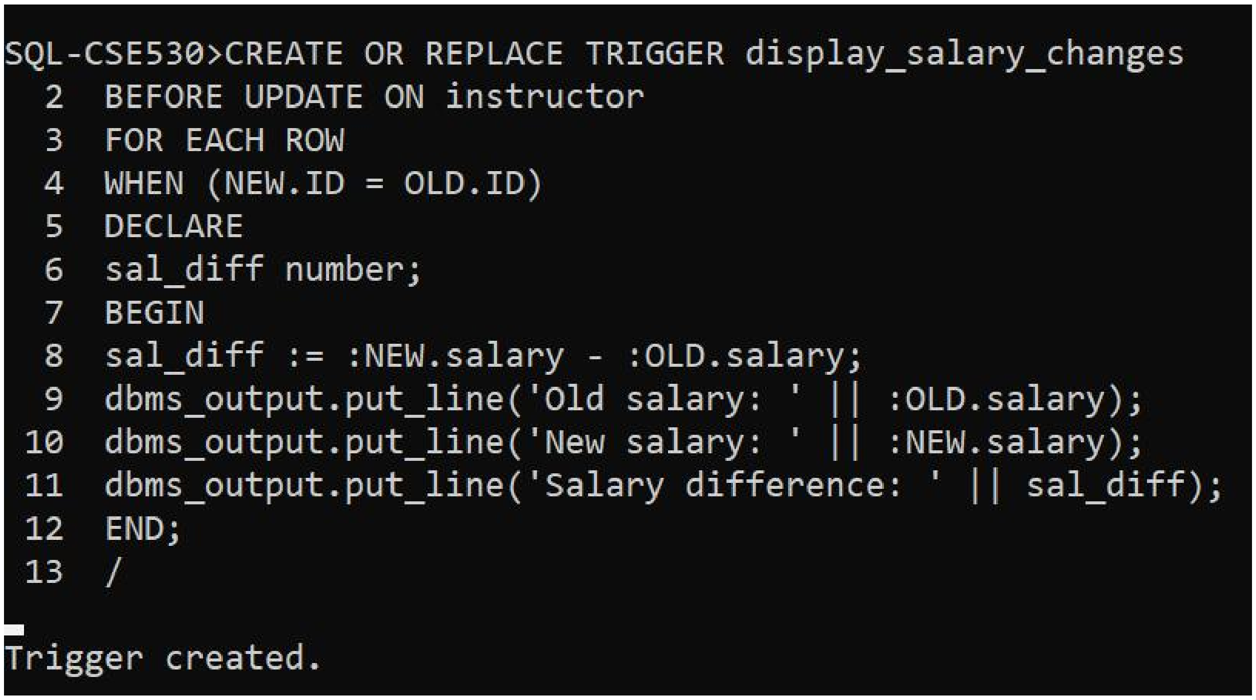
```
SQL> insert into department values ('Biology','watson','90000');

1 row created.

SQL> insert into department values ('physics','taylor','100000');

1 row created.

SQL> insert into department values ('finance','painter','80000');

1 row created.

SQL> insert into department values ('music','packard','250000');

1 row created.
```

CREATING DEPARTMENT TABLE :



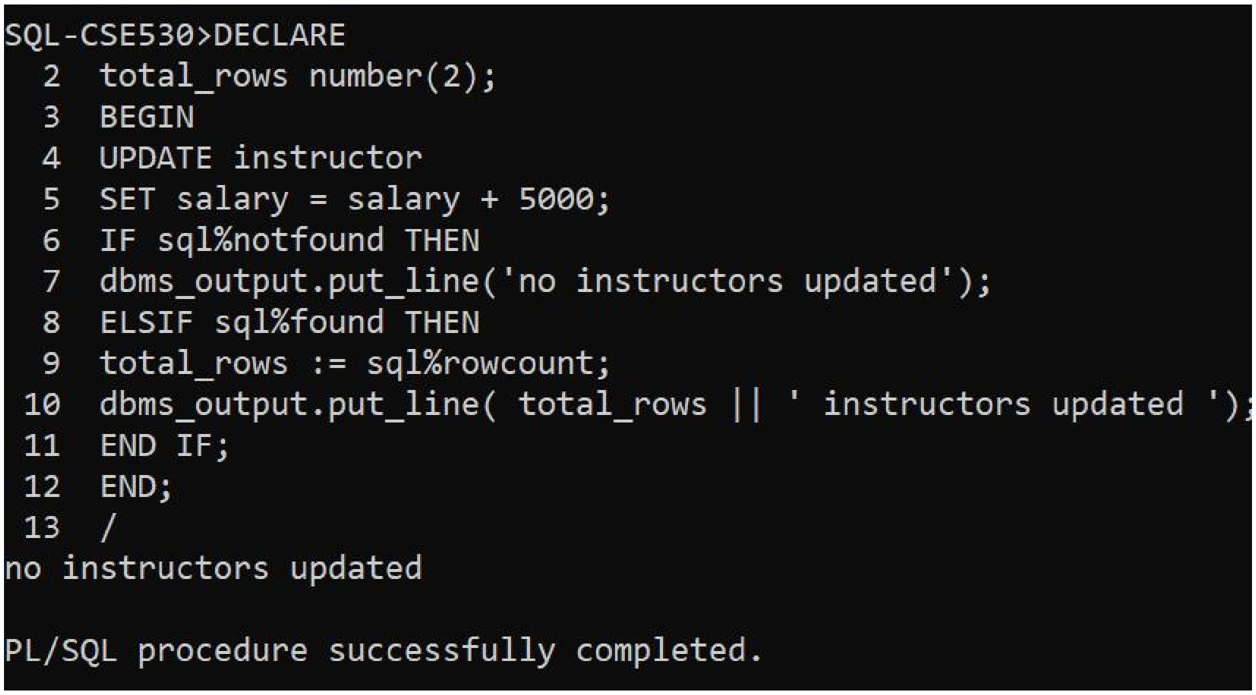
```
SQL-CSE530>CREATE TABLE DEPARTMENT
(DEPT_NAME VARCHAR2(20),
BUILDING VARCHAR2(15),
BUDGET NUMERIC(12,2) CHECK (BUDGET > 0),
PRIMARY KEY (DEPT_NAME)
);

Table created.
```

**An example to create Trigger :**



```
SQL-CSE530>CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE UPDATE ON instructor
FOR EACH ROW
WHEN (NEW.ID = OLD.ID)
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.
```

**A PL/SQL Procedure to execute a trigger:**

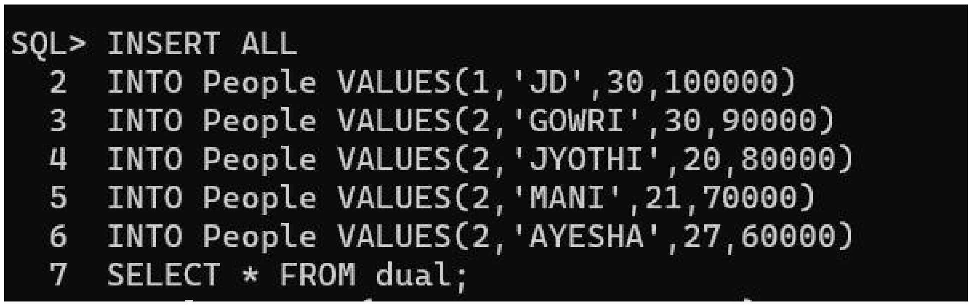


```
SQL-CSE530>DECLARE
total_rows number(2);
BEGIN
UPDATE instructor
SET salary = salary + 5000;
IF sql%notfound THEN
dbms_output.put_line('no instructors updated');
ELSIF sql%found THEN
total_rows := sql%rowcount;
dbms_output.put_line( total_rows || ' instructors updated ');
END IF;
END;
/
no instructors updated

PL/SQL procedure successfully completed.
```

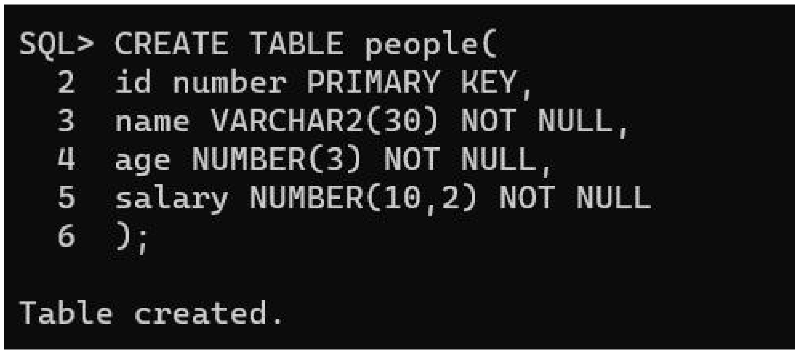
# Experiment-16

Aim :To write PL/SQL program to implement Cursor on table.

Table Creation :



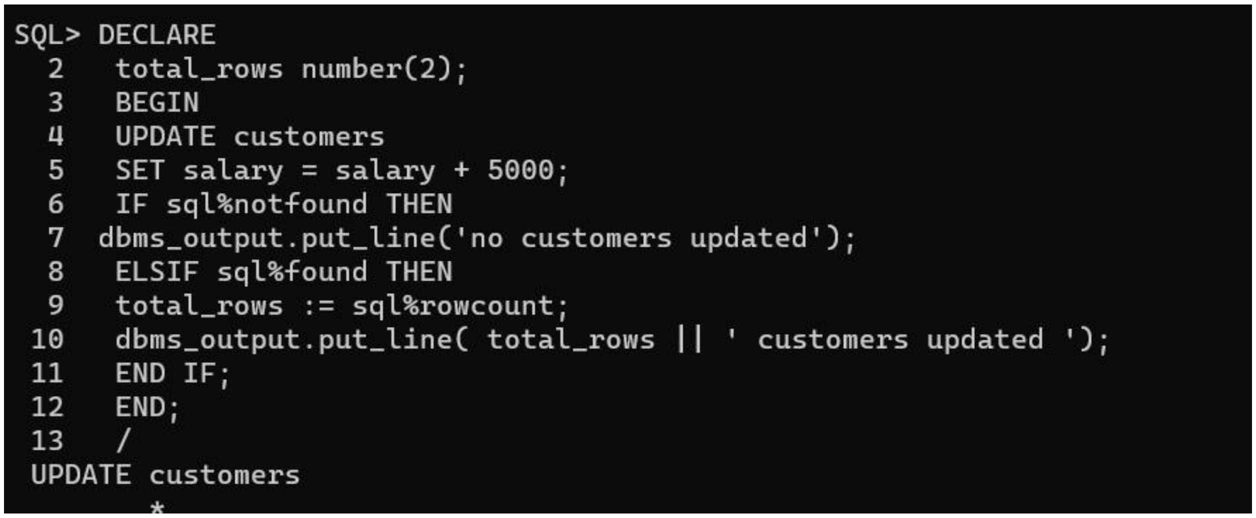
```
SQL> INSERT ALL
INTO People VALUES(1,'JD',30,100000)
INTO People VALUES(2,'GOWRI',30,90000)
INTO People VALUES(2,'JYOTHI',20,80000)
INTO People VALUES(2,'MANI',21,70000)
INTO People VALUES(2,'AYESHA',27,60000)
SELECT * FROM dual;
```



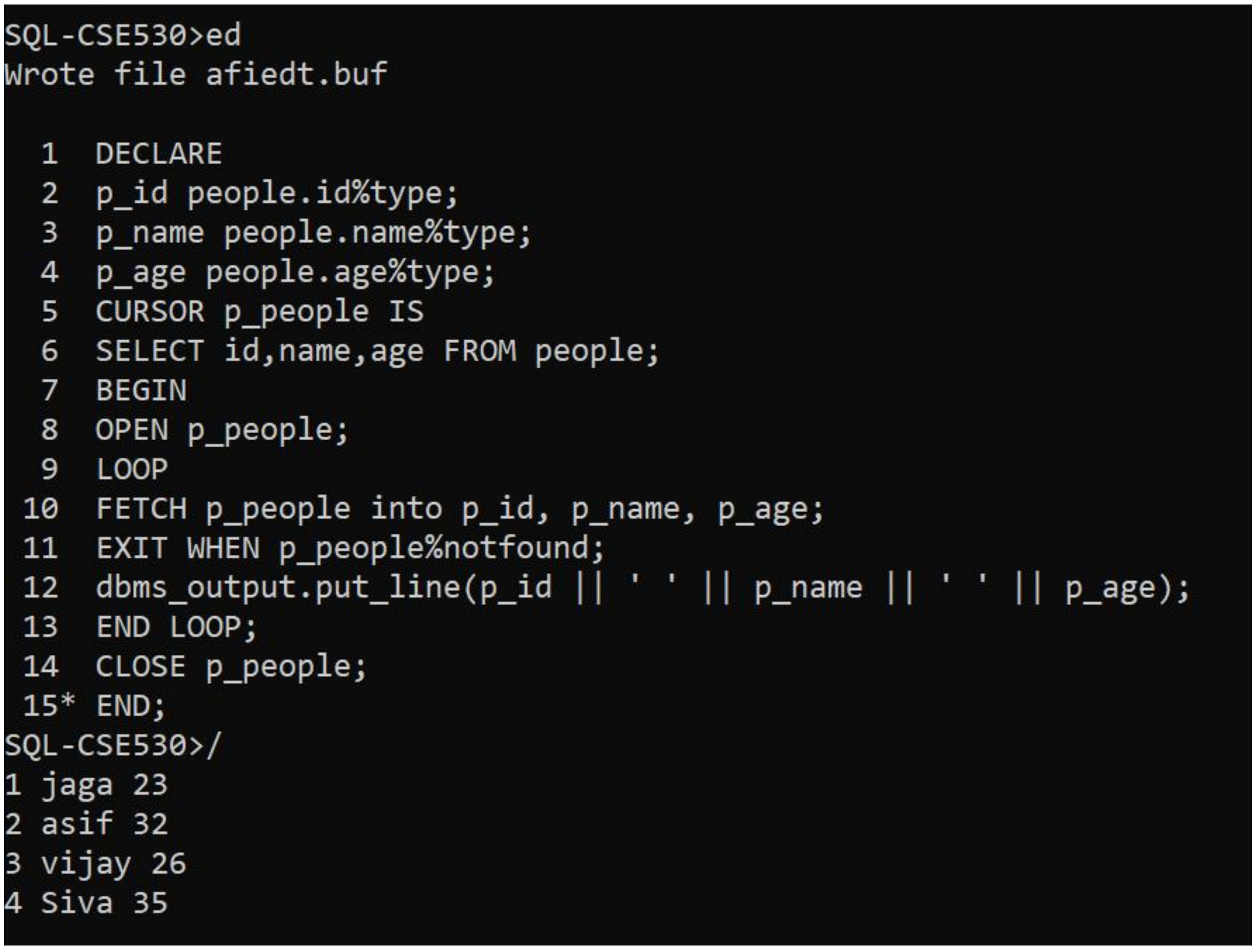
```
SQL> CREATE TABLE people(
id number PRIMARY KEY,
name VARCHAR2(30) NOT NULL,
age NUMBER(3) NOT NULL,
salary NUMBER(10,2) NOT NULL
);

Table created.
```

## Create update procedure

## Create procedure:



```
SQL> DECLARE
 total_rows number(2);
 BEGIN
 UPDATE customers
 SET salary = salary + 5000;
 IF sql%notfound THEN
dbms_output.put_line('no customers updated');
 ELSIF sql%found THEN
 total_rows := sql%rowcount;
 dbms_output.put_line( total_rows || ' customers updated ');
 END IF;
 END;
 /
UPDATE customers
       *
```



```
SQL-CSE530>ed
Wrote file afiedt.buf

DECLARE
p_id people.id%type;
p_name people.name%type;
p_age people.age%type;
CURSOR p_people IS
SELECT id,name,age FROM people;
BEGIN
OPEN p_people;
LOOP
FETCH p_people into p_id, p_name, p_age;
EXIT WHEN p_people%notfound;
dbms_output.put_line(p_id || ' ' || p_name || ' ' || p_age);
END LOOP;
CLOSE p_people;
END;
SQL-CSE530>/
1 jaga 23
2 asif 32
3 vijay 26
4 Siva 35
```