COLLEGE OF ENGINEERING TRIVANDRUM
DEPARTMENT OF COMPUTER SCIENCE

APPLICATION SOFTWARE DEVELOPMENT LAB REPORT(CS333)

ADARSH S

REG. NO : TVE17CS007

ROLL NO : 7

S5 CSE

Staff In-charge:

VIPIN VASU A. V

Associate Professor

Department of Computer Science

Contents

Experiment 1

INSTALLATION OF POSTGRESQL

AIM:

To install PostgreSQL

PROCEDURE:

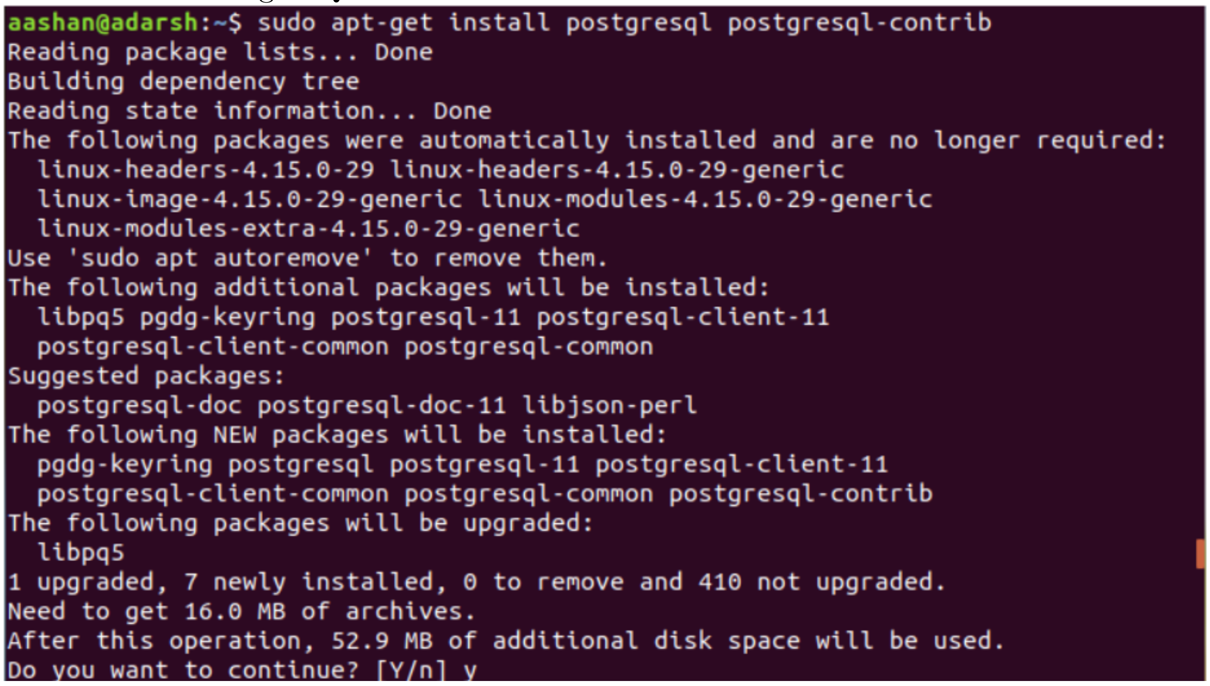
The first step is to update the system using the command:

sudo apt update

Using command:

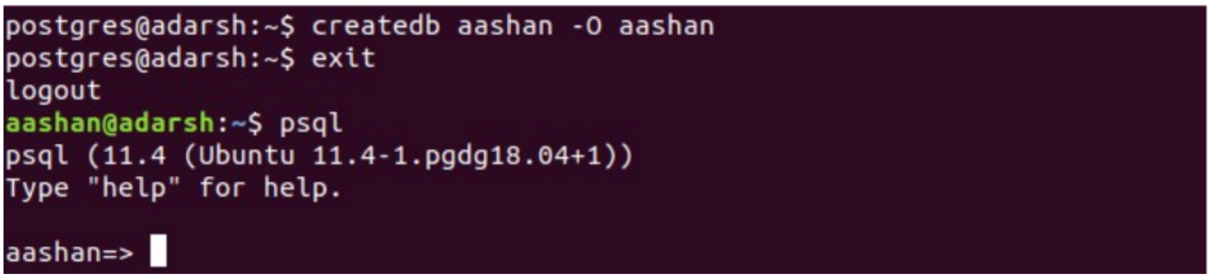
sudo apt-get install postgresql postgresql-contrib

we can install PostgreSQL.



```
aashan@adarsh:~$ sudo apt-get install postgresql postgresql-contrib
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-4.15.0-29 linux-headers-4.15.0-29-generic
  linux-image-4.15.0-29-generic linux-modules-4.15.0-29-generic
  linux-modules-extra-4.15.0-29-generic
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libpq5 pgdg-keyring postgresql-11 postgresql-client-11
  postgresql-client-common postgresql-common
Suggested packages:
  postgresql-doc postgresql-doc-11 libjson-perl
The following NEW packages will be installed:
  pgdg-keyring postgresql postgresql-11 postgresql-client-11
  postgresql-client-common postgresql-common postgresql-contrib
The following packages will be upgraded:
  libpq5
1 upgraded, 7 newly installed, 0 to remove and 410 not upgraded.
Need to get 16.0 MB of archives.
After this operation, 52.9 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

We can create a user by createuser keyword. We can create a database by createdb command



```
postgres@adarsh:~$ createdb aashan -O aashan
postgres@adarsh:~$ exit
logout
aashan@adarsh:~$ psql
psql (11.4 (Ubuntu 11.4-1.pgdg18.04+1))
Type "help" for help.

aashan=> 
```

We can get into postgresql by

sudo su - postgres

or by

psql

RESULT:

PostgreSQL installed in the PC and a database created. Postgresql version: 11.4 ,Ubuntu 18.04

Experiment 2
INTRODUCTION TO SQL

AIM:

Get to know about SQL and its queries

History of SQL

Dr. E. F. Codd published the paper, "A Relational Model of Data for Large Shared Data Banks", in June 1970 in the Association of Computer Machinery (ACM) journal, Communications of the ACM. Codd's model is now accepted as the definitive model for relational database management systems (RDBMS). The language, Structured English Query Language ("SEQUEL") was developed by IBM Corporation, Inc., to use Codd's model. SEQUEL later became SQL (still pronounced "sequel"). In 1979, Relational Software, Inc. (now Oracle Corporation) introduced the first commercially available implementation of SQL. Today, SQL is accepted as the standard RDBMS language.

How SQL Works

The strengths of SQL provide benefits for all types of users, including application programmers, database administrators, managers, and end users. Technically speaking, SQL is a data sub-language. The purpose of SQL is to provide an interface to a relational database such as Oracle, and all SQL statements are instructions to the database. In this SQL differs from general-purpose programming languages like C and BASIC. Among the features of SQL are the following:

1. It processes sets of data as groups rather than as individual units.
2. It provides automatic navigation to the data.
3. It uses statements that are complex and powerful individually, and that therefore stand alone.

Flow-control statements were not part of SQL originally, but they are found in the recently accepted optional part of SQL, ISO/IEC 9075-5: 1996. Flow-control statements are commonly known as "persistent stored modules" (PSM), and Oracle's PL/SQL extension to SQL is similar to PSM. Essentially, SQL lets you work with data at the logical level. You need to be concerned with the implementation details only when you want to manipulate the data. For example, to retrieve a set of rows from a table, you define a condition used to filter the rows. All rows satisfying the condition are retrieved in a single step and can be passed as a unit to the user, to another SQL statement, or to an application. You need not deal with the rows one by one, nor do you have to worry about how they are physically stored or retrieved. All SQL statements use the optimizer, a part

of Oracle that determines the most efficient means of accessing the specified data. Oracle also provides techniques you can use to make the optimizer perform its job better.
SQL provides statements for a variety of tasks, including:
1. Querying data
2. Inserting, updating, and deleting rows in a table
3. Creating, replacing, altering, and dropping objects
4. Controlling access to the database and its objects
5. Guaranteeing database consistency and integrity
SQL unifies all of the above tasks in one consistent language.
Common Language for All Relational Databases
All major relational database management systems support SQL, so you can transfer all skills you have gained with SQL from one database to another. In addition, all programs written in SQL are portable. They can often be moved from one database to another with very little modification.// **S**ummary of SQL Statements
SQL statements are divided into these categories:
1. Data Definition Language (DDL) Statements
2. Data Manipulation Language (DML) Statements
3. Transaction Control Statements (TCL)
4. Session Control Statement
5. System Control Statement
DDL
DDL is short name of Data Definition Language, which deals with database schemas and descriptions, of how the data should reside in the database.
• CREATE – to create database and its objects like (table, index, views, store procedure, function and triggers)
• ALTER – alters the structure of the existing database
• DROP – delete objects from the database
• TRUNCATE – remove all records from a table, including all spaces allocated for the records are removed
• COMMENT – add comments to the data dictionary
• RENAME – rename an object
DML
DML is short name of Data Manipulation Language which deals with data manipulation, and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE etc, and it is used to store, modify, retrieve, delete and update data in database.

• SELECT – retrieve data from the a database
• INSERT – insert data into a table
• UPDATE – updates existing data within a table
• DELETE – Delete all records from a database table
• MERGE – UPSERT operation (insert or update) • CALL – call a PL/SQL or Java subprogram
• EXPLAIN PLAN – interpretation of the data access path
• LOCK TABLE – concurrency Control
Managing Tables
A table is a data structure that holds data in a relational database. A table is composed of rows and columns. A table can represent a single entity that you want to track within your system. This type of a table would represent a list of the employees within your organization, or the orders placed for your company's products. A table can also represent a relationship between two entities. This type of a table could portray the association between employees and their job skills, or the relationship of products to orders. Within the tables, foreign keys are used torepresent relationships.
Creating Tables
To create a table, use the SQL command CREATETABLE. Syntax: CREATE TABLE<TABLENAME>
(<FIELDNAME><DATATYPE><[SIZE]>,.........)
Altering Tables
Alter a table in an Oracle database for any of the following reasons:
1. To add one or more new columns to the table
2. To add one or more integrity constraints to a table
3. To modify an existing column's definition (datatype, length, default value, and NOT-NULL integrity constraint)
4. To modify data block space usage parameters (PCTFREE, PCTUSED)
5. To modify transaction entry settings (INITRANS, MAXTRANS)
6. To modify storage parameters (NEXT, PCTINCREASE, etc.)
7. To enable or disable integrity constraints associated with the table
8. To drop integrity constraints associated with the table
When altering the column definitions of a table, you can only increase the length of an existing column, unless the table has no records. You can also decrease the length of a column in an empty table. For columns of data type CHAR, increasing the length of a column might be a time consuming operation that requires substantial additional storage,

especially if the table contains many rows. This is because the CHAR value in each row must be blank-padded to satisfy the new column length.
If you change the datatype (for example, from VARCHAR2 to CHAR), then the data in the column does not change. However, the length of new CHAR columns might change, due to blank-padding requirements.
Altering a table has the following implications:
1. If a new column is added to a table, then the column is initially null. You can add a column with a NOTNULL constraint to a table only if the table does not contain any rows.
2. If a view or PL/SQL program unit depends on a base table, then the alteration of the base table might affect the dependent object, and always invalidates the dependent object.
Privileges Required to Alter a Table
To alter a table, the table must be contained in your schema, or you must have either the ALTER object privilege for the table or the ALTERANYTABLE system privilege.
Dropping Tables
Use the SQL command DROPTABLE to drop a table. For example, the following statement drops the EMPTAB table:
If the table that you are dropping contains any primary or unique keys referenced by foreign keys to other tables, and if you intend to drop the FOREIGNKEY constraints of the child tables, then include the CASCADE option in the DROPTABLE command.
Datatype
Use the NUMBER datatype to store real numbers in a fixed-point or floating-point format. Numbers using this data type are guaranteed to be portable among different Oracle platforms, and offer up to 38 decimal digits of precision.
For numeric columns you can specify the column as a floating-point number:
Columnname NUMBER Or, you can specify a precision (total number of digits) and scale (number of digits to the right of the decimal point):
Columnname NUMBER (<precision>, <scale>)
Although not required, specifying the precision and scale for numeric fields provides extra integrity checking on input. If a precision is not specified, then the columnstores values as given. Table shows examples of how data would be stored using different scale factors.
Using the DATE Datatype
Use the DATE datatype to store point-in-time values (dates and times) in a table. The DATE datatype stores the century, year, month, day, hours, minutes, and seconds.

fields of seven bytes each, corresponding to century, year, month, day, hour, minute, and second
Date Format
For input and output of dates, the standard Oracle default date format is DD-MON-YY. For example: '13-NOV-92'
To change this default date format on an instance-wide basis, use the NLSDATEFORMAT parameter. To change the format during a session, use the ALTER SESSION statement. To enter dates that are not in the current default date format, use the TODATE function with a format mask.
For example:
TODATE ('November 13, 1992', 'MONTH DD, YYYY')
If the date format DD-MON-YY is used, then YY indicates the year in the 20th century (for example, 31-DEC-92 is December 31, 1992). If you want to indicate years in any century other than the 20 th century, then use a different format mask, as shown above.
Time Format
Time is stored in 24-hour format HH:MM:SS. By default, the time in a date field is 12:00:00 A.M. (midnight) if no time portion is entered. In a time-only entry, the date portion defaults to the first day of the current month. To enter the time portion of a date, use the TODATE function with a format mask indicating the time portion, as in:
INSERT INTO Birthdaystab (bname, bday)
VALUES ('ANNIE',TODATE('13-NOV-92 10:56 A.M.'
,'DD-MON-YY HH:MI A.M.'));
To compare dates that have time data, use the SQL function TRUNC if you want to ignore the time component. Use the SQL function SYSDATE to return the system date and time. The FIXEDDATE initialization parameter allows you to set SYSDATE to a constant; this can be useful for testing.
RESULT:
Understood the basics of SQL and the queries.

Experiment 2

BASIC SQL QUERIES – I

AIM:

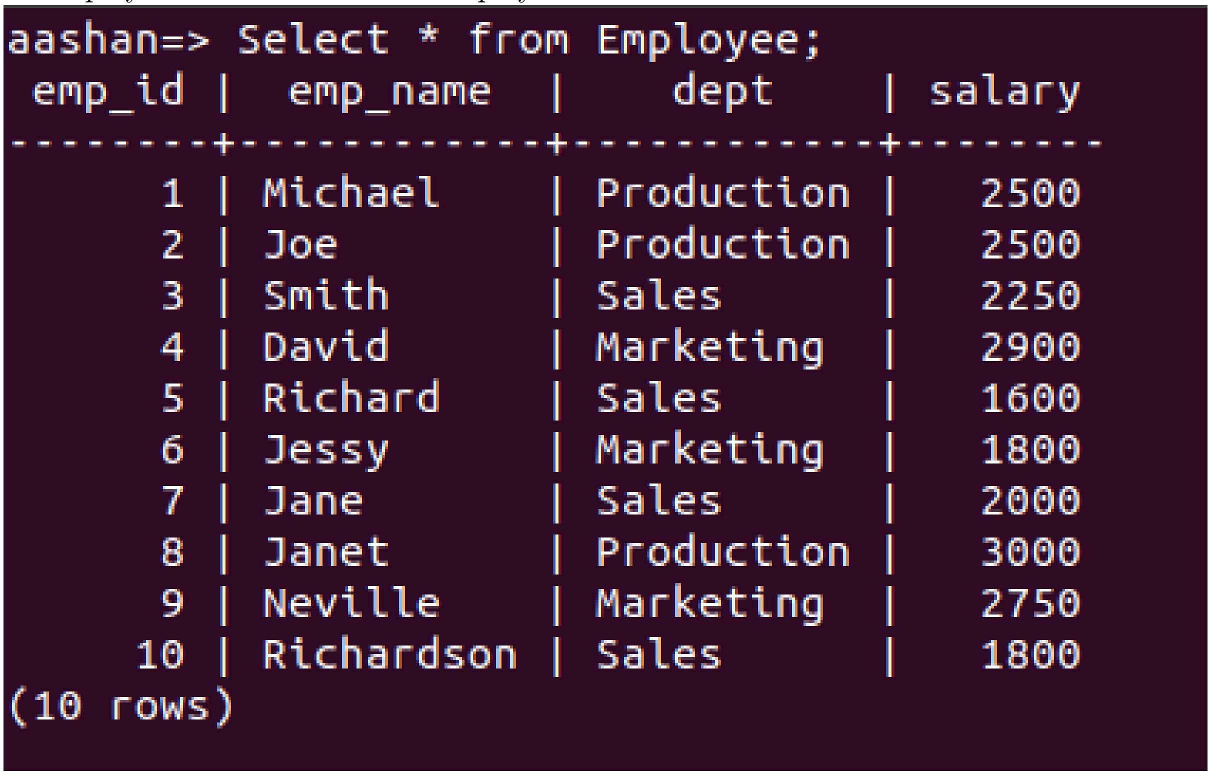
To study the basic sql queries such as

1. SELECT
2. INSERT
3. UPDATE
4. DELETE

QUESTIONS:

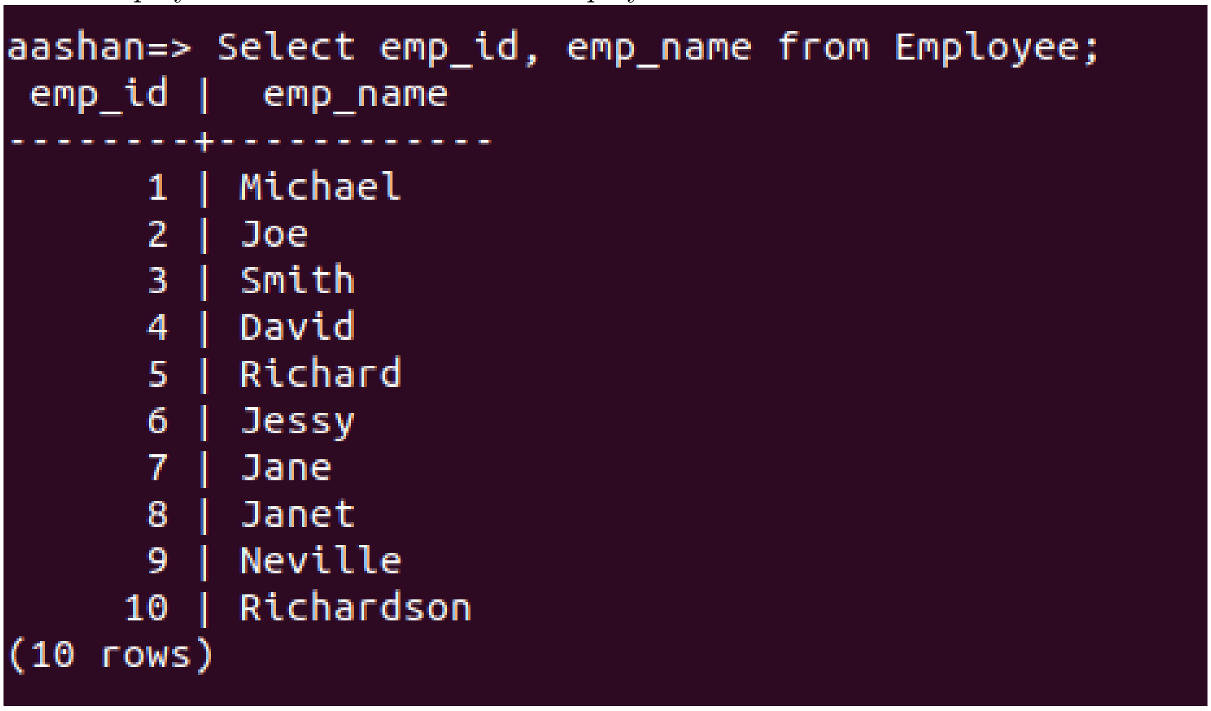
Create a table named Employee and populate the table as shown below.

1. Display the details of all the employees.



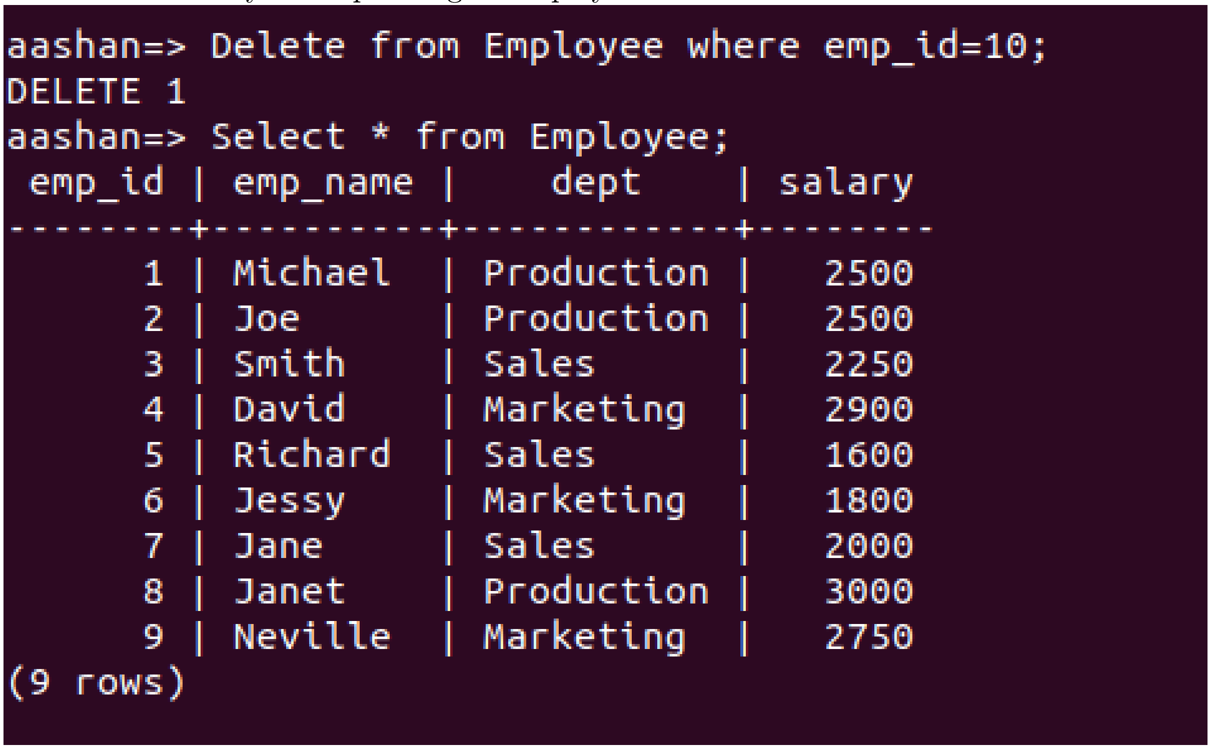
```
aashan=> Select * from Employee;
 emp_id |  emp_name  |    dept    | salary
--------+------------+------------+--------
      1 | Michael    | Production |   2500
      2 | Joe        | Production |   2500
      3 | Smith      | Sales      |   2250
      4 | David      | Marketing  |   2900
      5 | Richard    | Sales      |   1600
      6 | Jessy      | Marketing  |   1800
      7 | Jane       | Sales      |   2000
      8 | Janet      | Production |   3000
      9 | Neville    | Marketing  |   2750
     10 | Richardson | Sales      |   1800
(10 rows)
```

2. Display the names and id's of all employees.



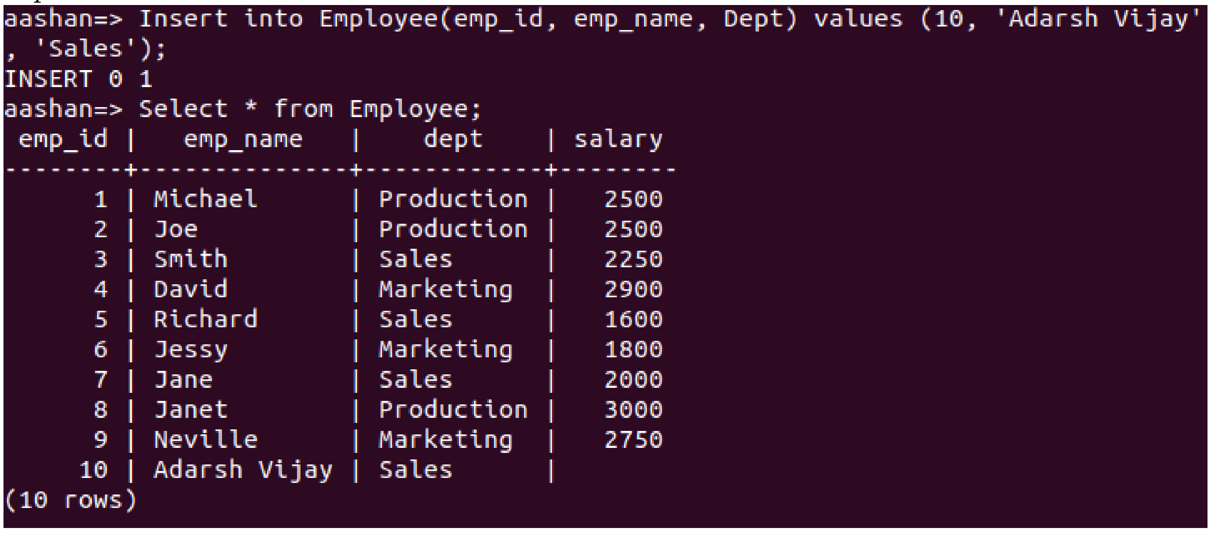
```
aashan=> Select emp_id, emp_name from Employee;
 emp_id |  emp_name
--------+------------
      1 | Michael
      2 | Joe
      3 | Smith
      4 | David
      5 | Richard
      6 | Jessy
      7 | Jane
      8 | Janet
      9 | Neville
     10 | Richardson
(10 rows)
```

3. Delete the entry corresponding to employee id:10.



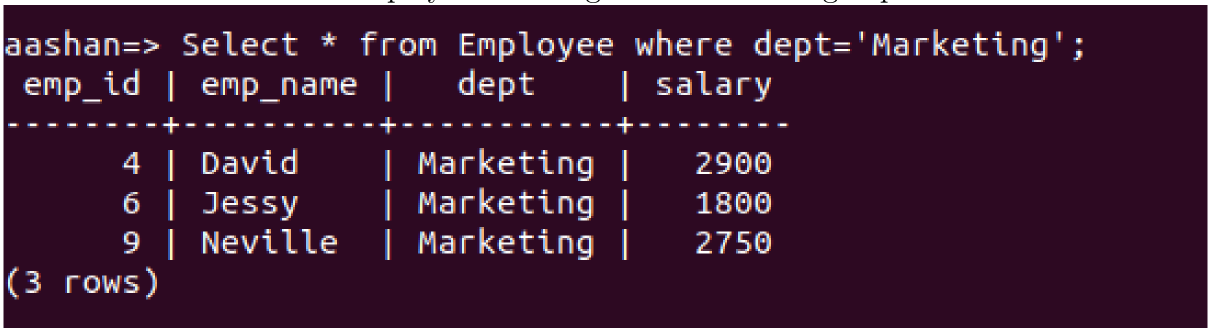
```
aashan=> Delete from Employee where emp_id=10;
DELETE 1
aashan=> Select * from Employee;
 emp_id | emp_name |    dept    | salary
--------+----------+------------+--------
      1 | Michael  | Production |   2500
      2 | Joe      | Production |   2500
      3 | Smith    | Sales      |   2250
      4 | David    | Marketing  |   2900
      5 | Richard  | Sales      |   1600
      6 | Jessy    | Marketing  |   1800
      7 | Jane     | Sales      |   2000
      8 | Janet    | Production |   3000
      9 | Neville  | Marketing  |   2750
(9 rows)
```

4. Insert a new tuple to the table. The salary field of the new employee should be kept NULL.



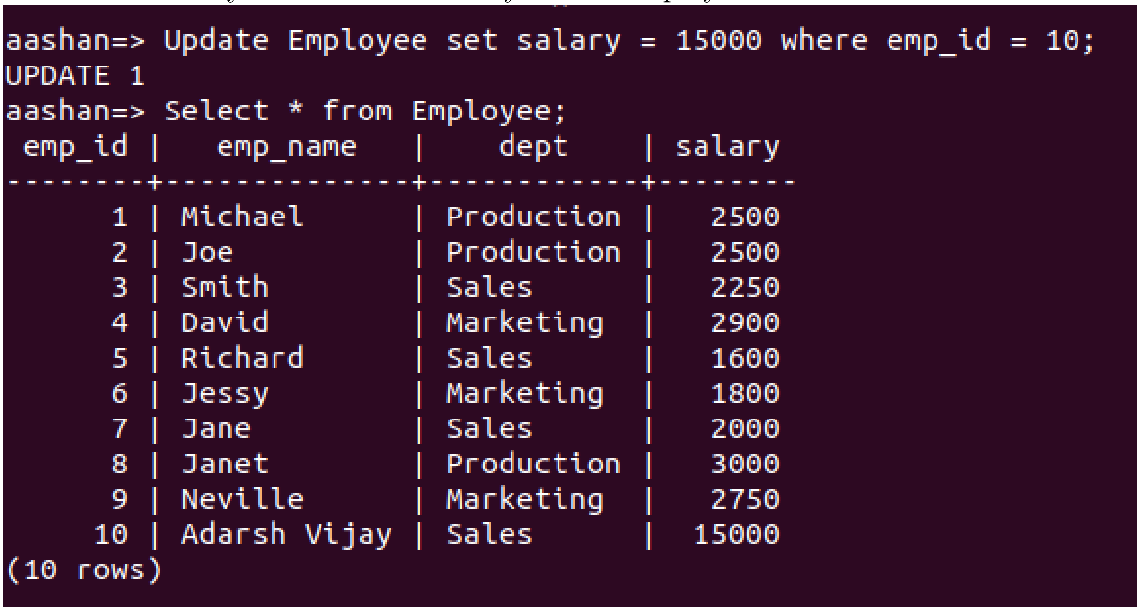
```
aashan=> Insert into Employee(emp_id, emp_name, Dept) values (10, 'Adarsh Vijay'
, 'Sales');
INSERT 0 1
aashan=> Select * from Employee;
 emp_id |   emp_name   |    dept    | salary
--------+--------------+------------+--------
      1 | Michael      | Production |   2500
      2 | Joe          | Production |   2500
      3 | Smith        | Sales      |   2250
      4 | David        | Marketing  |   2900
      5 | Richard      | Sales      |   1600
      6 | Jessy        | Marketing  |   1800
      7 | Jane         | Sales      |   2000
      8 | Janet        | Production |   3000
      9 | Neville      | Marketing  |   2750
     10 | Adarsh Vijay | Sales      |
(10 rows)
```

5. Find the details of all employees working in the marketing department



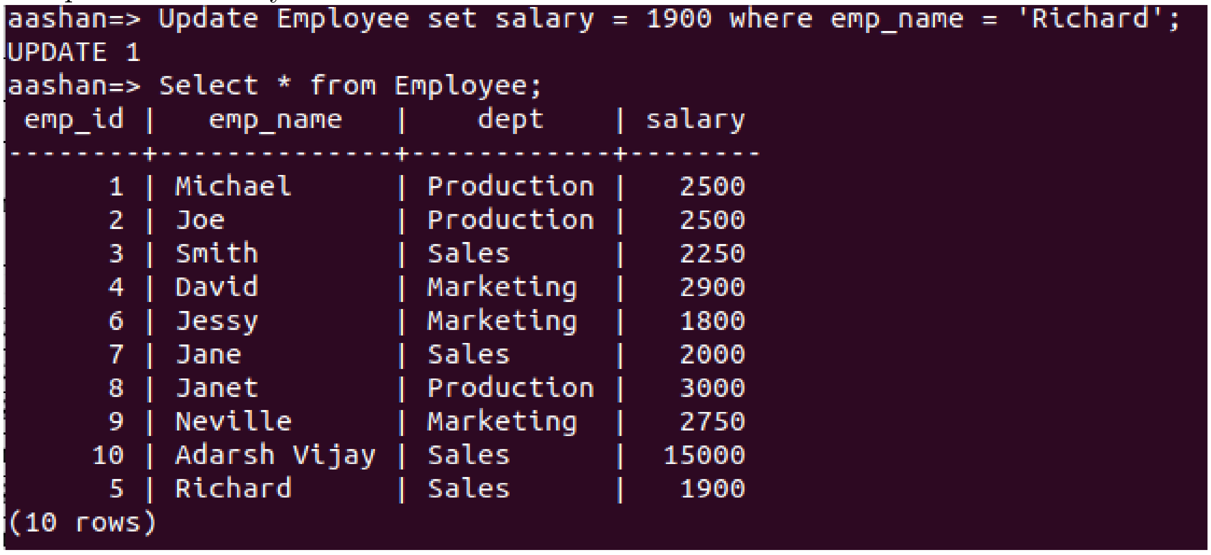
```
aashan=> Select * from Employee where dept='Marketing';
 emp_id | emp_name |    dept    | salary
--------+----------+------------+--------
      4 | David    | Marketing  |   2900
      6 | Jessy    | Marketing  |   1800
      9 | Neville  | Marketing  |   2750
(3 rows)
```

6. Add the salary details of the newly added employee



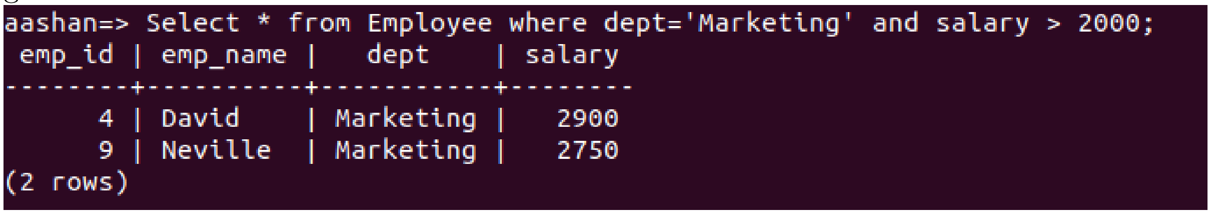
```
aashan=> Update Employee set salary = 15000 where emp_id = 10;
UPDATE 1
aashan=> Select * from Employee;
 emp_id |   emp_name   |    dept    | salary
--------+--------------+------------+--------
      1 | Michael      | Production |   2500
      2 | Joe          | Production |   2500
      3 | Smith        | Sales      |   2250
      4 | David        | Marketing  |   2900
      5 | Richard      | Sales      |   1600
      6 | Jessy        | Marketing  |   1800
      7 | Jane         | Sales      |   2000
      8 | Janet        | Production |   3000
      9 | Neville      | Marketing  |   2750
     10 | Adarsh Vijay | Sales      |  15000
(10 rows)
```

7. Update the salary of Richard to 1900



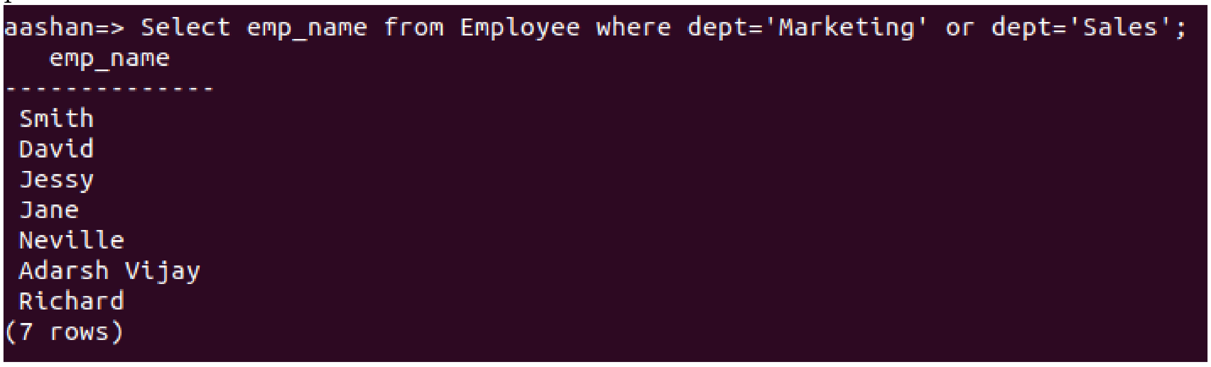
```
aashan=> Update Employee set salary = 1900 where emp_name = 'Richard';
UPDATE 1
aashan=> Select * from Employee;
 emp_id |   emp_name    |    dept    | salary
--------+--------------+------------+--------
      1 | Michael      | Production |   2500
      2 | Joe          | Production |   2500
      3 | Smith        | Sales      |   2250
      4 | David        | Marketing  |   2900
      6 | Jessy        | Marketing  |   1800
      7 | Jane         | Sales      |   2000
      8 | Janet        | Production |   3000
      9 | Neville      | Marketing  |   2750
     10 | Adarsh Vijay | Sales      |  15000
      5 | Richard      | Sales      |   1900
(10 rows)
```

8. Find the details of all employees who are working for marketing and has a salary greater than 2000



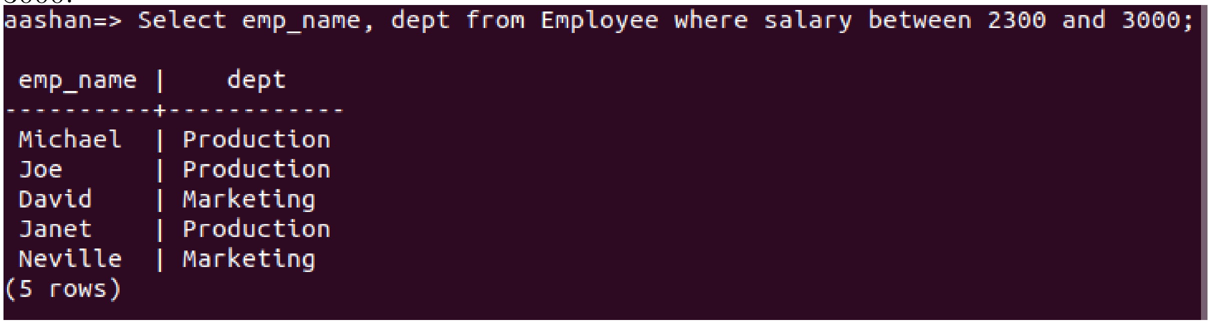
```
aashan=> Select * from Employee where dept='Marketing' and salary > 2000;
 emp_id | emp_name |   dept    | salary
--------+----------+-----------+--------
      4 | David    | Marketing |   2900
      9 | Neville  | Marketing |   2750
(2 rows)
```

9. List the names of all employees working in the sales department and marketing department.



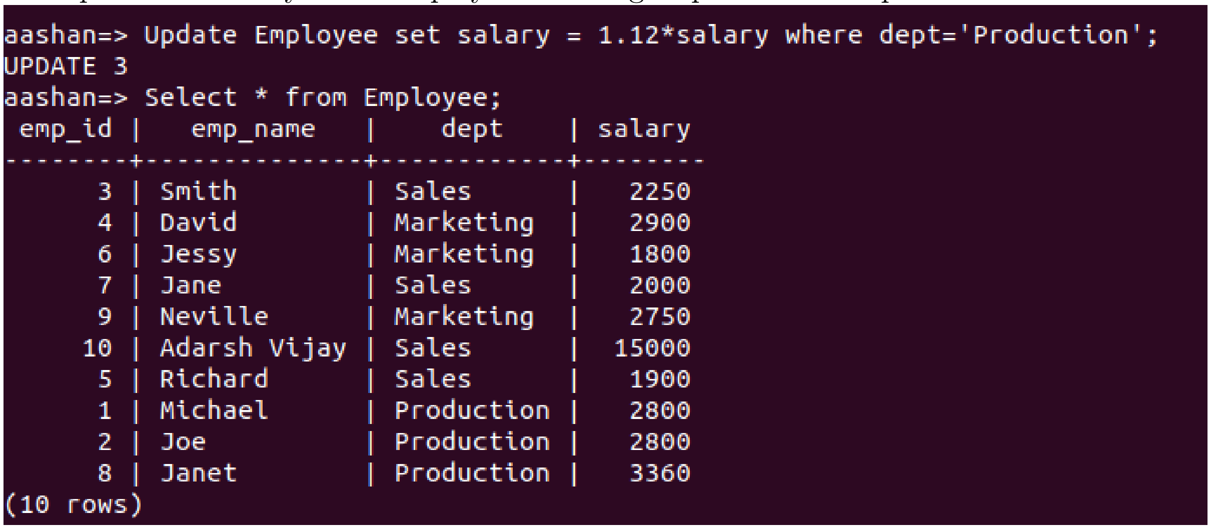
```
aashan=> Select emp_name from Employee where dept='Marketing' or dept='Sales';
   emp_name
--------------
 Smith
 David
 Jessy
 Jane
 Neville
 Adarsh Vijay
 Richard
(7 rows)
```

10. List the names and department of all employees whose salary is between2300 and 3000.



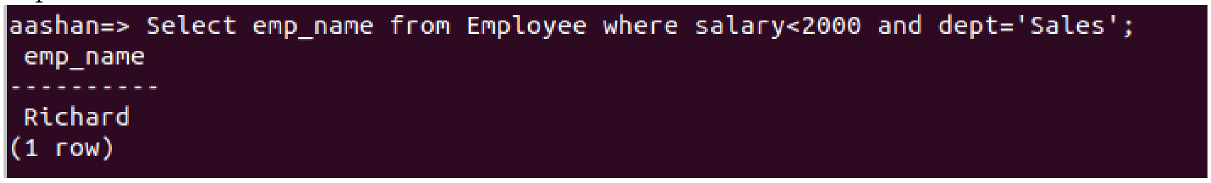
```
aashan=> Select emp_name, dept from Employee where salary between 2300 and 3000;

 emp_name |    dept
----------+------------
 Michael  | Production
 Joe      | Production
 David    | Marketing
 Janet    | Production
 Neville  | Marketing
(5 rows)
```

11. Update the salary of all employees working in production department 12



```
aashan=> Update Employee set salary = 1.12*salary where dept='Production';
UPDATE 3
aashan=> Select * from Employee;
 emp_id |   emp_name   |    dept    | salary
--------+--------------+------------+--------
      3 | Smith        | Sales      |   2250
      4 | David        | Marketing  |   2900
      6 | Jessy        | Marketing  |   1800
      7 | Jane         | Sales      |   2000
      9 | Neville      | Marketing  |   2750
     10 | Adarsh Vijay | Sales      |  15000
      5 | Richard      | Sales      |   1900
      1 | Michael      | Production |   2800
      2 | Joe          | Production |   2800
      8 | Janet        | Production |   3360
(10 rows)
```

12. Display the names of all employees whose salary is less than 2000 or working for sales department.



```
aashan=> Select emp_name from Employee where salary<2000 and dept='Sales';
 emp_name
----------
 Richard
(1 row)
```

RESULT:

Implemented the program for basic SQL queries-1 using Postgresql version: 11.4 ,Ubuntu 18.04 and following output were obtained.

EXPERIMENT 3

BASIC SQL QUERIES – II

AIM:

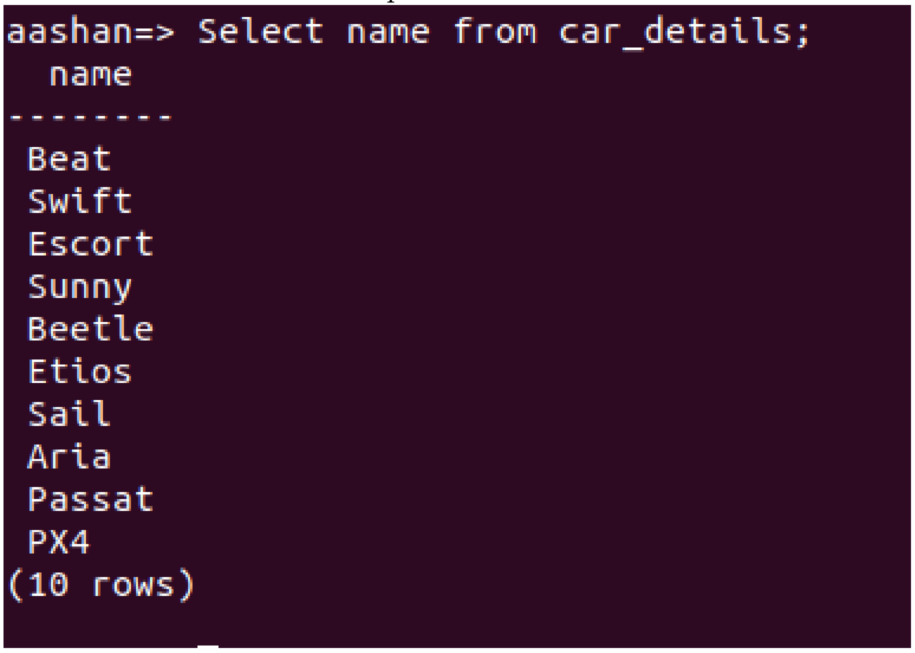
Introduction to SQL statements

1. ALTER
2. RENAME
3. SELECT DISTINCT
4. SQL IN
5. SQL BETWEEN
6. Sql aliases
7. Sql AND
8. Sql OR

QUESTION:

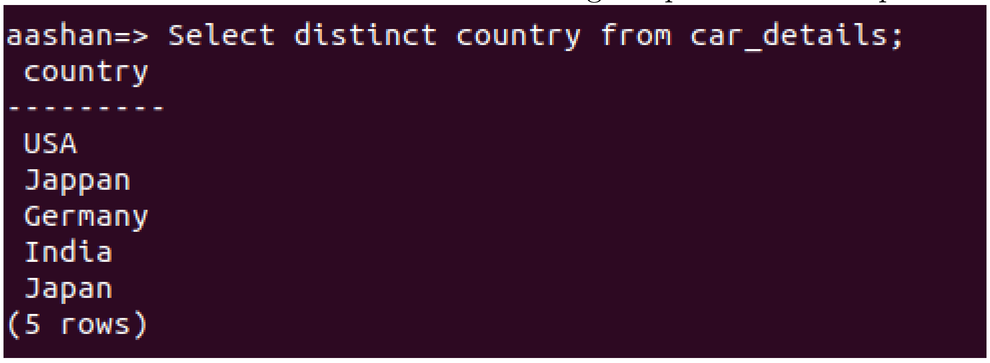
Create a table named cardetails and populate the table as shown below.

1. List the names of all companies as mentioned in the database



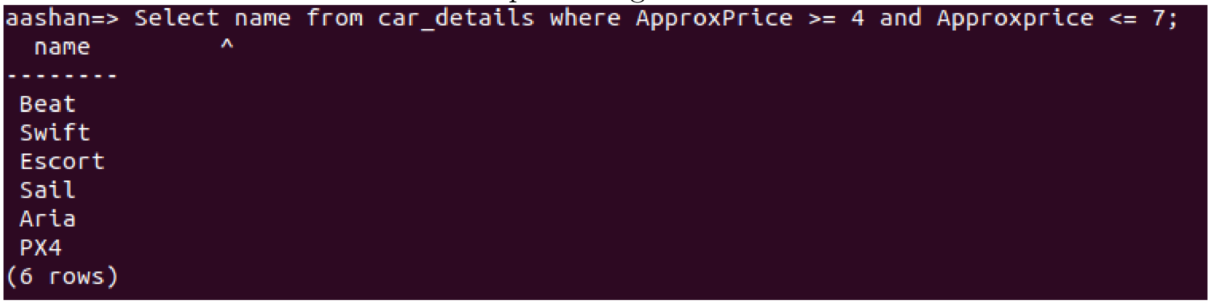
```
aashan=> Select name from car_details;
  name
--------
 Beat
 Swift
 Escort
 Sunny
 Beetle
 Etios
 Sail
 Aria
 Passat
 PX4
(10 rows)
```

2. List the names of all countries having car production companies



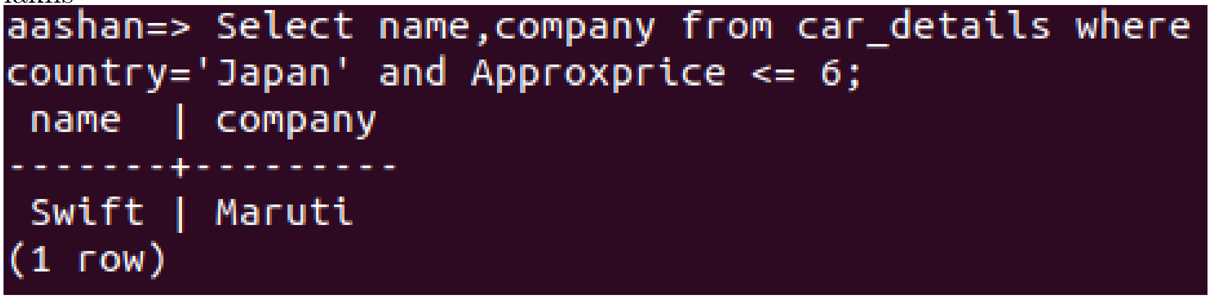
```
aashan=> Select distinct country from car_details;
 country
---------
 USA
 Jappan
 Germany
 India
 Japan
(5 rows)
```

3.List the details of all cars within a price range 4 to 7 lakhs



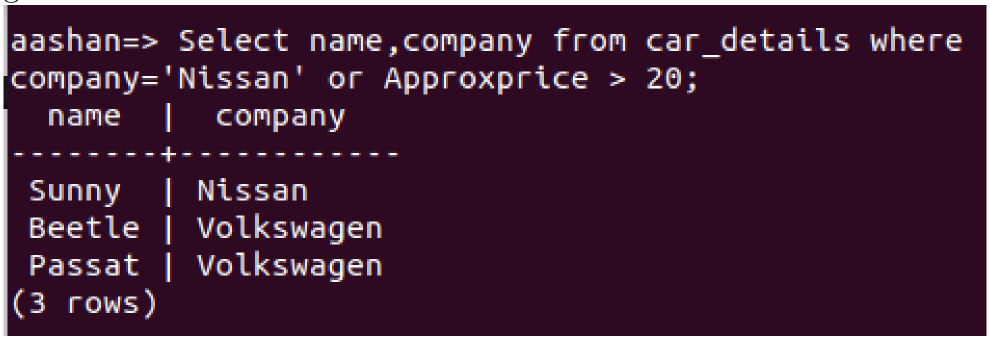
```
aashan=> Select name from car_details where ApproxPrice >= 4 and Approxprice <= 7;
  name         ^
--------
 Beat
 Swift
 Escort
 Sail
 Aria
 PX4
(6 rows)
```

4. List the name and company of all cars originating from Japan and having price<=6 lakhs



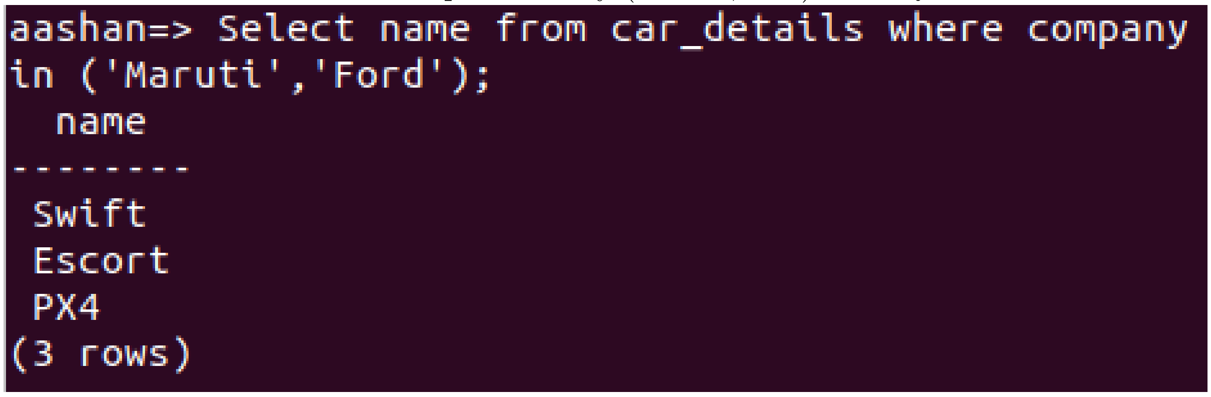
```
aashan=> Select name,company from car_details where
country='Japan' and Approxprice <= 6;
 name  | company
-------+---------
 Swift | Maruti
(1 row)
```

5. List the names and the companies of all cars either from Nissan or having a price greater than 20 lakhs.



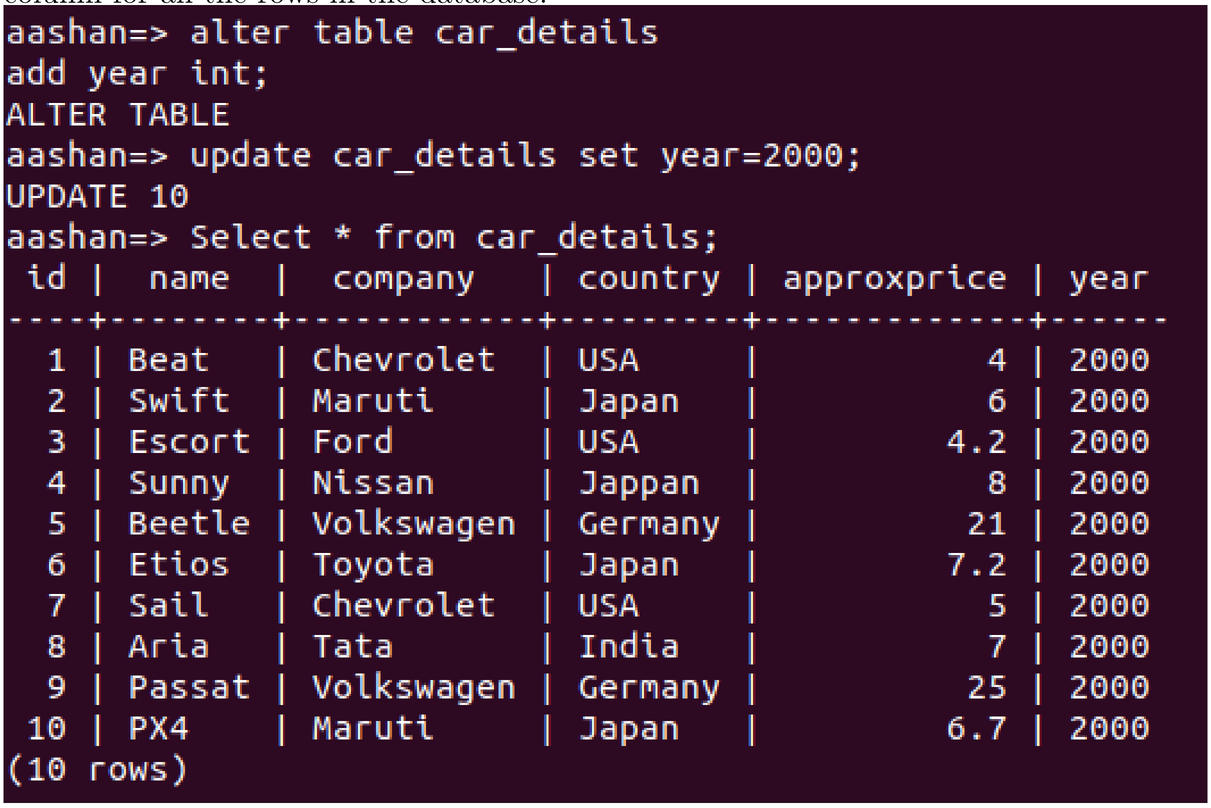
```
aashan=> Select name,company from car_details where
company='Nissan' or Approxprice > 20;
  name  |  company
--------+------------
 Sunny  | Nissan
 Beetle | Volkswagen
 Passat | Volkswagen
(3 rows)
```

6. List the names of all cars produced by (Maruti,Ford).Use SQL IN statement.



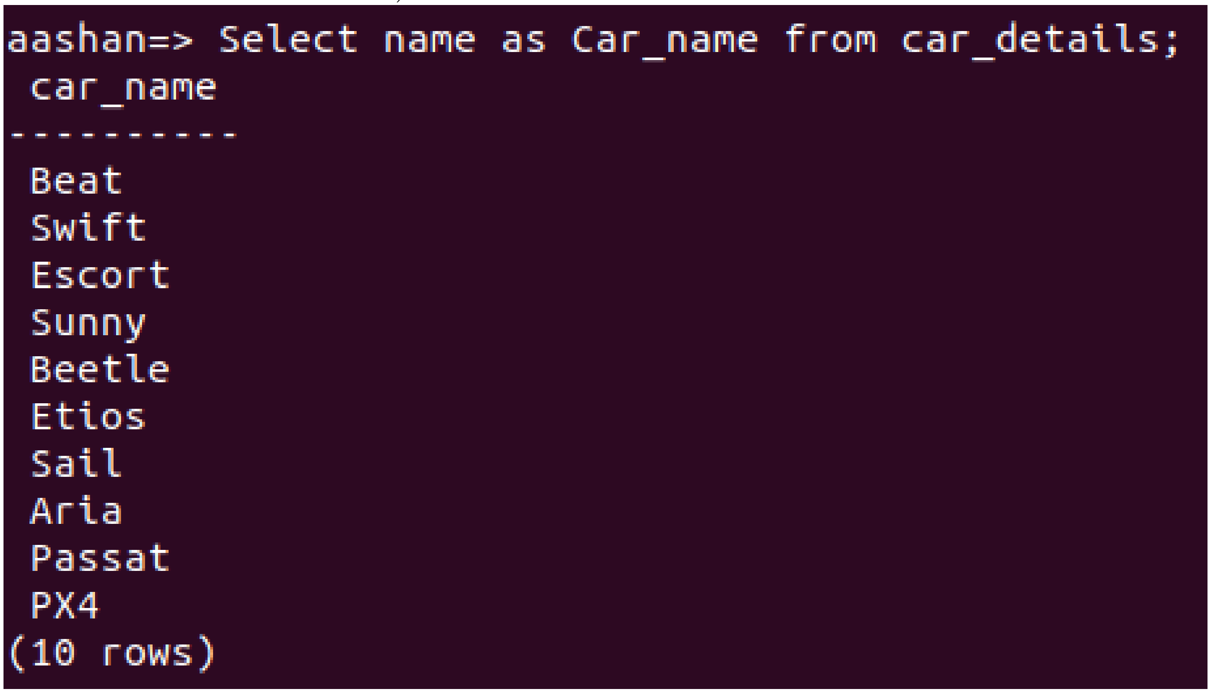
```
aashan=> Select name from car_details where company
in ('Maruti','Ford');
  name
--------
 Swift
 Escort
 PX4
(3 rows)
```

7. Alter the table cars to add a new field year (model release year).Update the year column for all the rows in the database.



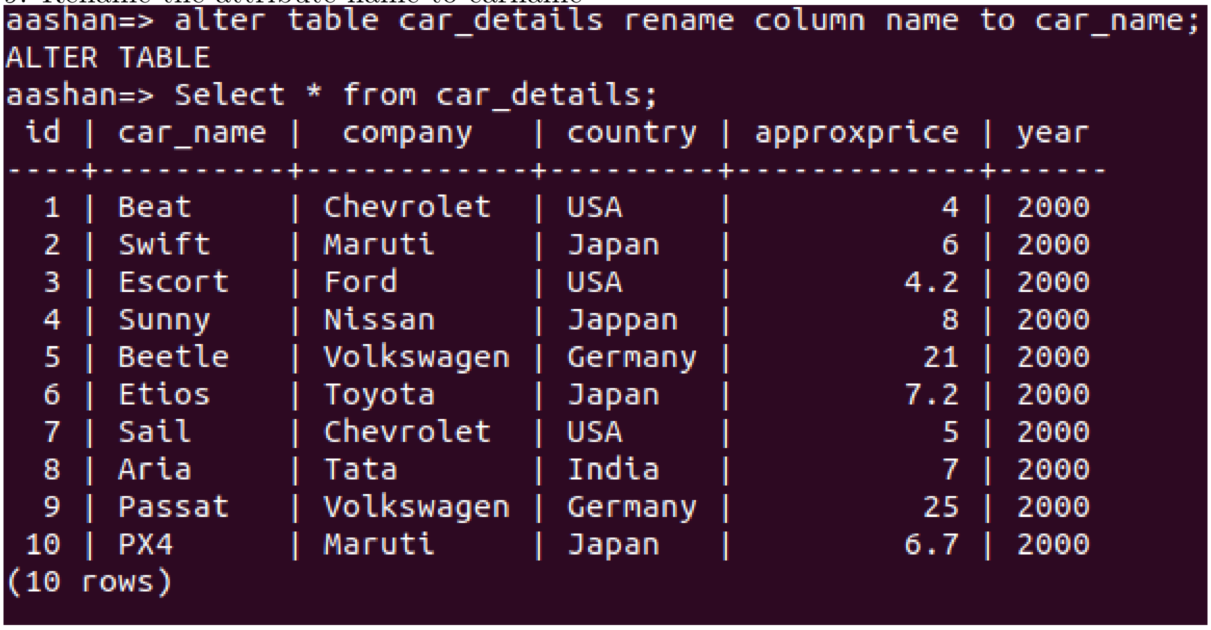
```
aashan=> alter table car_details
add year int;
ALTER TABLE
aashan=> update car_details set year=2000;
UPDATE 10
aashan=> Select * from car_details;
 id |  name  |  company   | country | approxprice | year
----+--------+------------+---------+-------------+------
  1 | Beat   | Chevrolet  | USA     |           4 | 2000
  2 | Swift  | Maruti     | Japan   |           6 | 2000
  3 | Escort | Ford       | USA     |         4.2 | 2000
  4 | Sunny  | Nissan     | Jappan  |           8 | 2000
  5 | Beetle | Volkswagen | Germany |          21 | 2000
  6 | Etios  | Toyota     | Japan   |         7.2 | 2000
  7 | Sail   | Chevrolet  | USA     |           5 | 2000
  8 | Aria   | Tata       | India   |           7 | 2000
  9 | Passat | Volkswagen | Germany |          25 | 2000
 10 | PX4    | Maruti     | Japan   |         6.7 | 2000
(10 rows)
```

8. Display the names of all cars as Carname (while displaying the name attribute should be listed as caraliases)



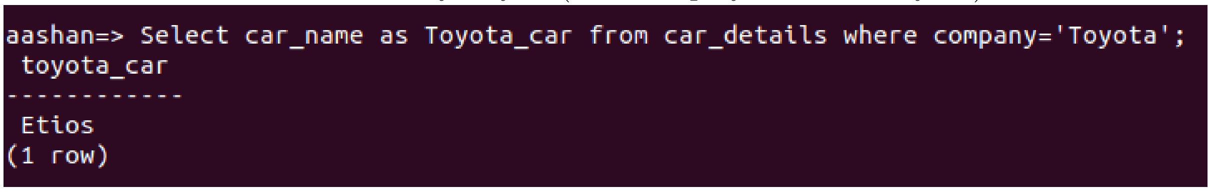
```
aashan=> Select name as Car_name from car_details;
 car_name
----------
 Beat
 Swift
 Escort
 Sunny
 Beetle
 Etios
 Sail
 Aria
 Passat
 PX4
(10 rows)
```

9. Rename the attribute name to carname



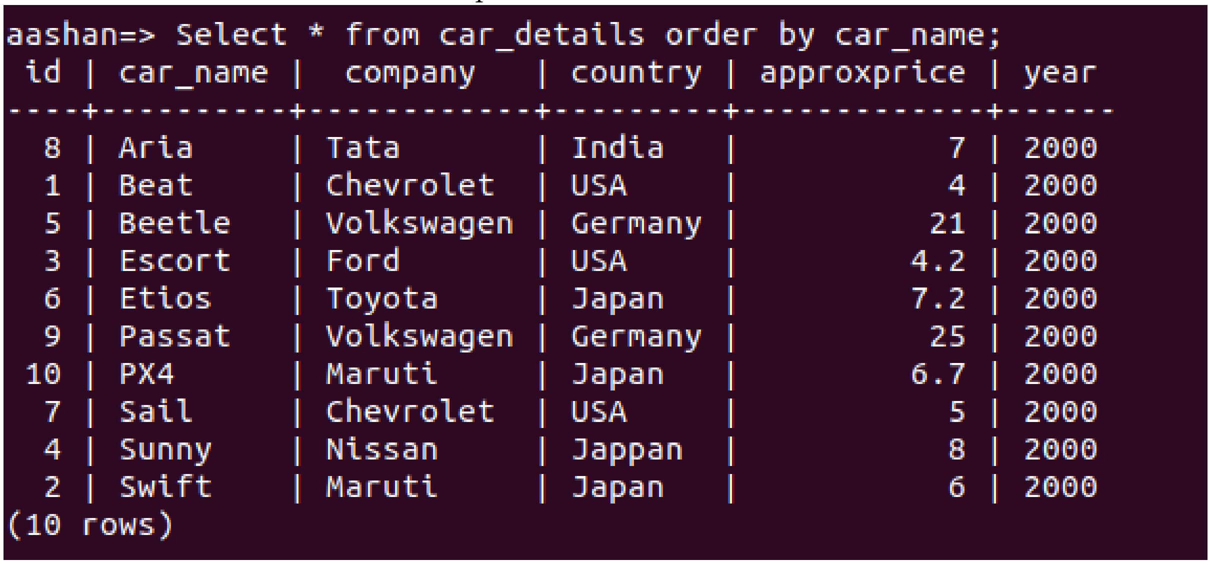
```
aashan=> alter table car_details rename column name to car_name;
ALTER TABLE
aashan=> Select * from car_details;
 id | car_name |  company   | country | approxprice | year
----+----------+------------+---------+-------------+------
  1 | Beat     | Chevrolet  | USA     |           4 | 2000
  2 | Swift    | Maruti     | Japan   |           6 | 2000
  3 | Escort   | Ford       | USA     |         4.2 | 2000
  4 | Sunny    | Nissan     | Jappan  |           8 | 2000
  5 | Beetle   | Volkswagen | Germany |          21 | 2000
  6 | Etios    | Toyota     | Japan   |         7.2 | 2000
  7 | Sail     | Chevrolet  | USA     |           5 | 2000
  8 | Aria     | Tata       | India   |           7 | 2000
  9 | Passat   | Volkswagen | Germany |          25 | 2000
 10 | PX4      | Maruti     | Japan   |         6.7 | 2000
(10 rows)
```

10. List the car manufactured by Toyota(to be displayed as carsToyota)



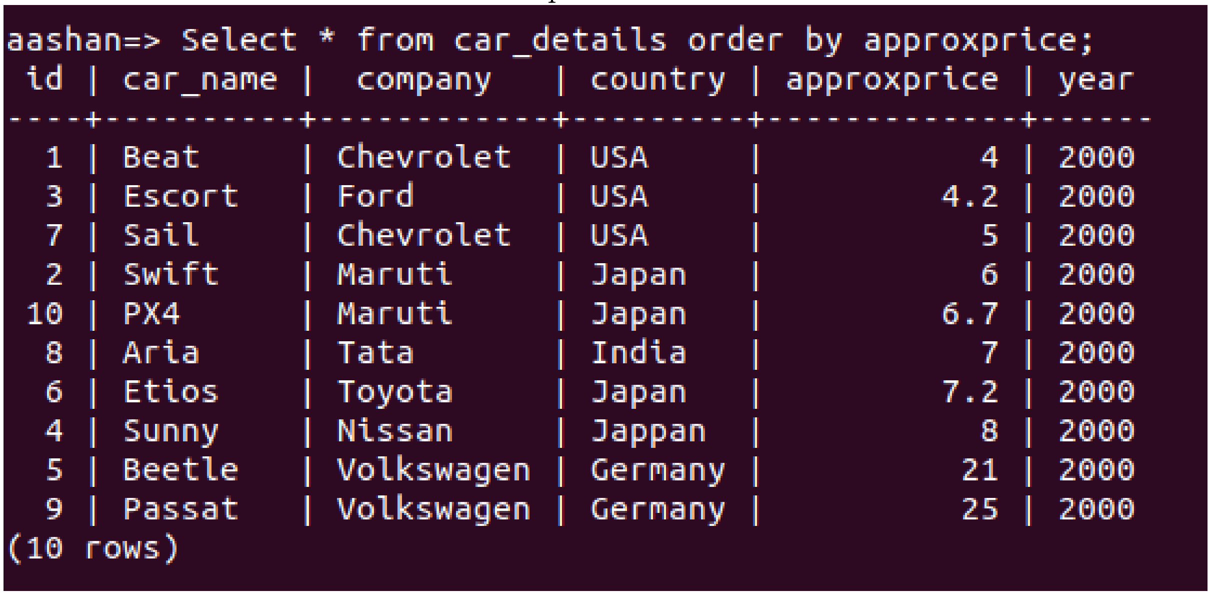
```
aashan=> Select car_name as Toyota_car from car_details where company='Toyota';
 toyota_car
------------
 Etios
(1 row)
```

11. List the details of all cars in alphabetical order



```
aashan=> Select * from car_details order by car_name;
 id | car_name |  company   | country | approxprice | year
----+----------+------------+---------+-------------+------
  8 | Aria     | Tata       | India   |           7 | 2000
  1 | Beat     | Chevrolet  | USA     |           4 | 2000
  5 | Beetle   | Volkswagen | Germany |          21 | 2000
  3 | Escort   | Ford       | USA     |         4.2 | 2000
  6 | Etios    | Toyota     | Japan   |         7.2 | 2000
  9 | Passat   | Volkswagen | Germany |          25 | 2000
 10 | PX4      | Maruti     | Japan   |         6.7 | 2000
  7 | Sail     | Chevrolet  | USA     |           5 | 2000
  4 | Sunny    | Nissan     | Jappan  |           8 | 2000
  2 | Swift    | Maruti     | Japan   |           6 | 2000
(10 rows)
```

12. List the details of all cars from cheapest to costliest.



```
aashan=> Select * from car_details order by approxprice;
 id | car_name |  company   | country | approxprice | year
----+----------+------------+---------+-------------+------
  1 | Beat     | Chevrolet  | USA     |           4 | 2000
  3 | Escort   | Ford       | USA     |         4.2 | 2000
  7 | Sail     | Chevrolet  | USA     |           5 | 2000
  2 | Swift    | Maruti     | Japan   |           6 | 2000
 10 | PX4      | Maruti     | Japan   |         6.7 | 2000
  8 | Aria     | Tata       | India   |           7 | 2000
  6 | Etios    | Toyota     | Japan   |         7.2 | 2000
  4 | Sunny    | Nissan     | Jappan  |           8 | 2000
  5 | Beetle   | Volkswagen | Germany |          21 | 2000
  9 | Passat   | Volkswagen | Germany |          25 | 2000
(10 rows)
```

RESULT:

Implemented the program for basic SQL queries-2 using Postgresql version: 11.4 ,Ubuntu 18.04 and following output were obtained.

Experiment 4

AGGREGATE FUNCTIONS

AIM:

Introduction to Aggregate functions
-AVG() -MAX() -MIN() - COUNT() -SUM()

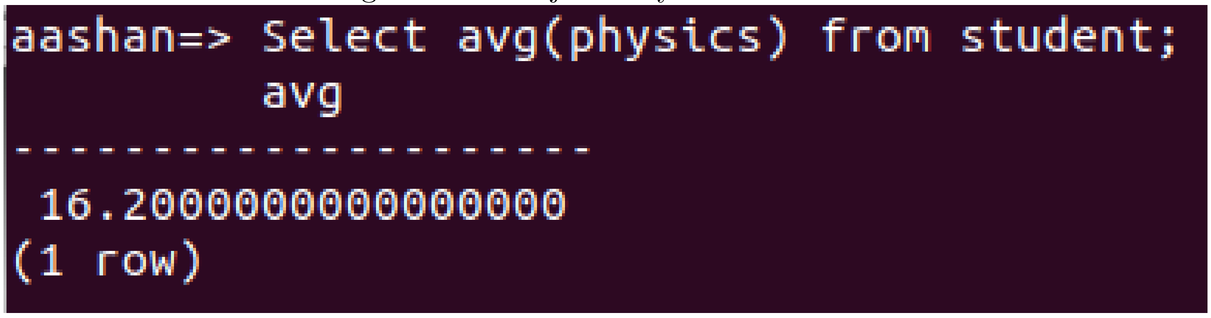
THEORY:

- SUM(fieldname)
  Returns the total sum of the field.
- AVG(fieldname)
  Returns the average of the field.
- COUNT( )
  Count function has three variations:
  - COUNT(*) : returns the number of rows in the table including duplicates and those with null values
  - COUNT(fieldname) : returns the number of rows where field value is not null
  - COUNT (*): returns the total number of rows
- MAX(fieldname)
  Returns the maximum value of the field
- MIN(fieldname)
  Returns the maximum value of the field

QUESTION:

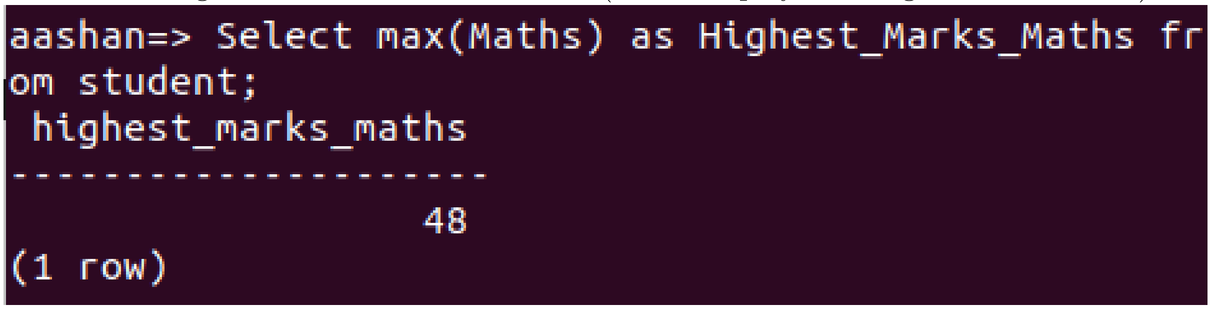
Create a table named student and populate the table as shown in the table. The table contains the marks of 10 students for 3 subjects(Physics, Chemistry, Mathematics).The total marks for physics and chemistry is 25, while for mathematics it is 50.The pass mark for physics and chemistry is 12 and for mathematics it is 25. A student is awarded a 'Pass' if he has passed all the subjects.

1. Find the class average for the subject ‘Physics’.



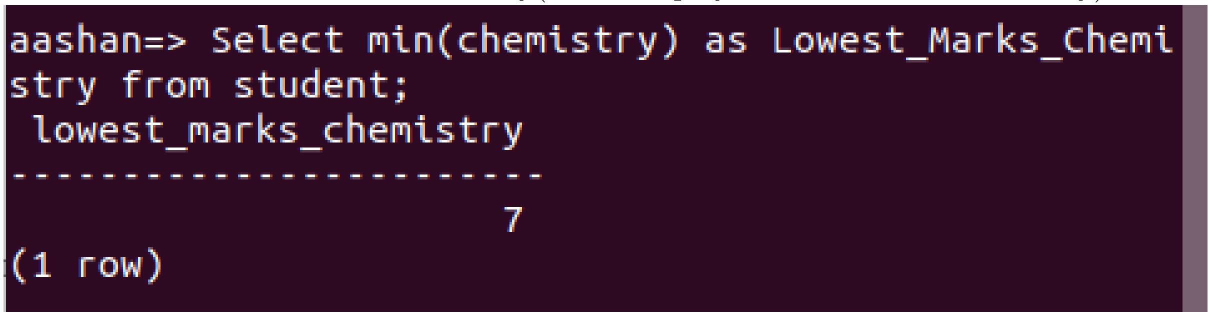
```
aashan=> Select avg(physics) from student;
         avg
---------------------
 16.2000000000000000
(1 row)
```

2. Find the highest marks for mathematics (To be displayed as highestmarksmaths).



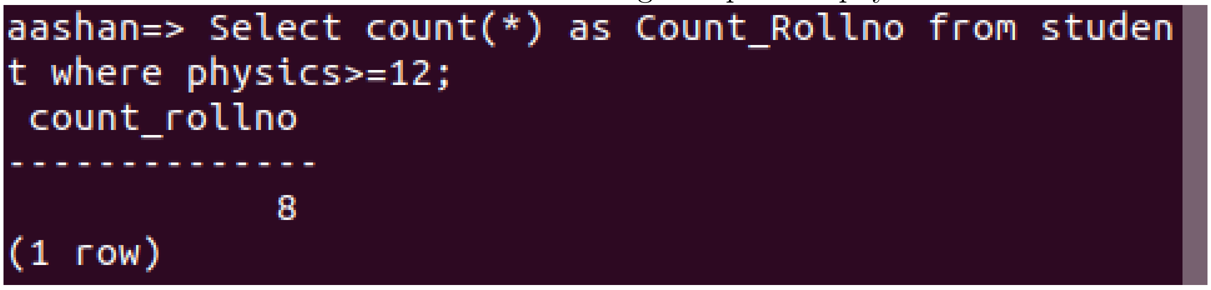
```
aashan=> Select max(Maths) as Highest_Marks_Maths fr
om student;
 highest_marks_maths
---------------------
                  48
(1 row)
```

3. Find the lowest marks for chemistry(To be displayed as lowestmarkchemistry).



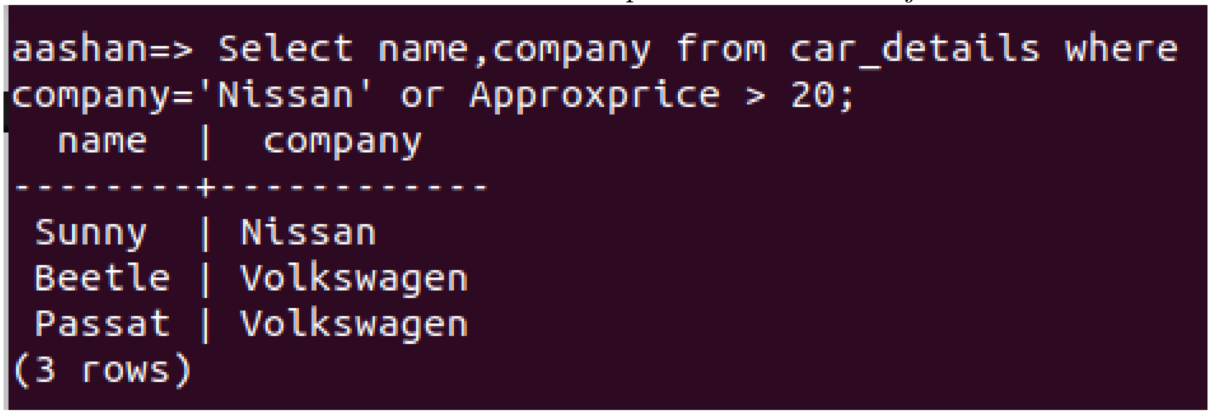
```
aashan=> Select min(chemistry) as Lowest_Marks_Chemi
stry from student;
 lowest_marks_chemistry
------------------------
                      7
(1 row)
```

4. Find the total number of students who has got a ‘pass’ in physics.



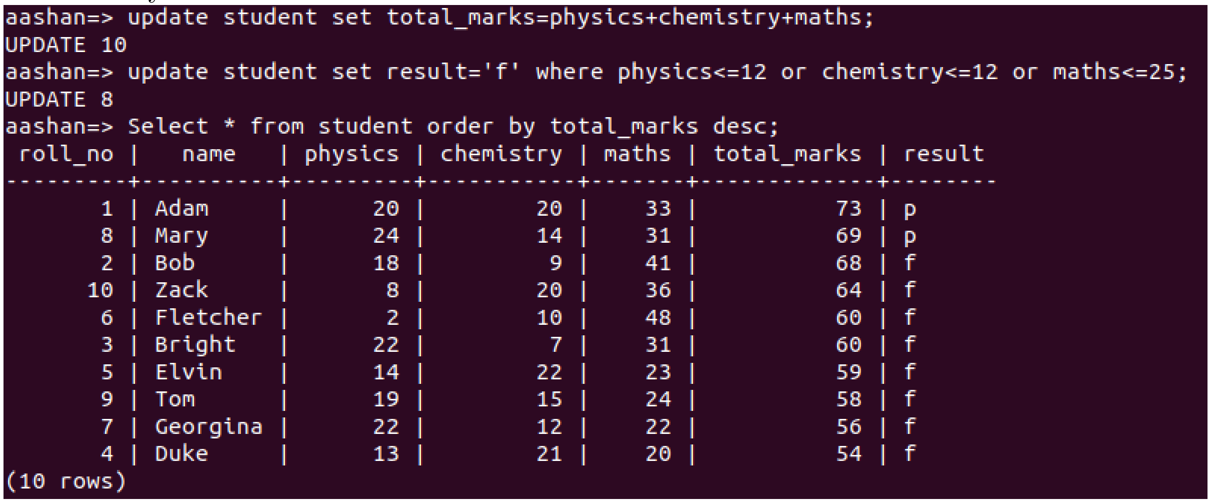
```
aashan=> Select count(*) as Count_Rollno from studen
t where physics>=12;
 count_rollno
--------------
            8
(1 row)
```

5. Generate the list of students who have passed in all the subjects.



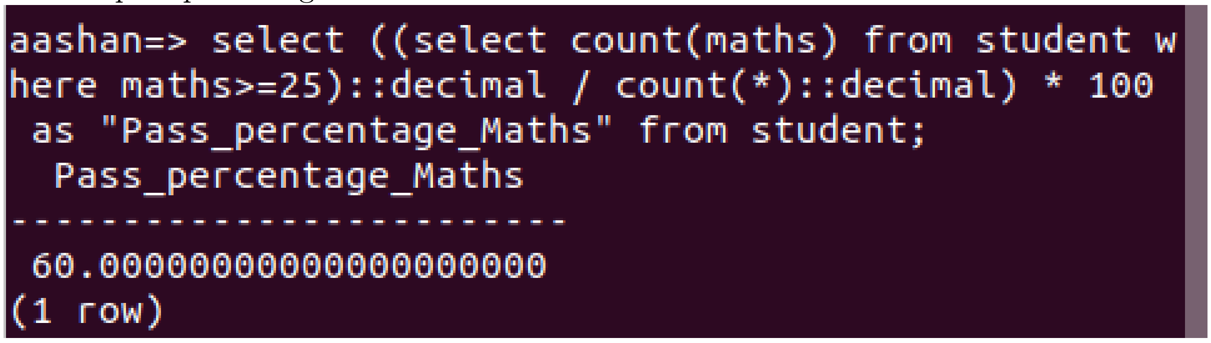
```
aashan=> Select name,company from car_details where
company='Nissan' or Approxprice > 20;
  name   |  company
--------+------------
 Sunny  | Nissan
 Beetle | Volkswagen
 Passat | Volkswagen
(3 rows)
```

6. Generate a rank list for the class.Indicate Pass/Fail. Ranking based on total marks obtained by the students.



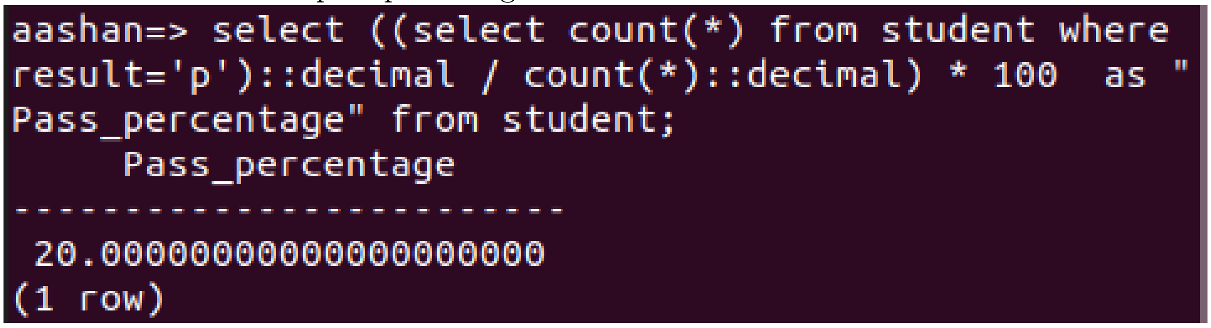
```
aashan=> update student set total_marks=physics+chemistry+maths;
UPDATE 10
aashan=> update student set result='f' where physics<=12 or chemistry<=12 or maths<=25;
UPDATE 8
aashan=> Select * from student order by total_marks desc;
 roll_no |   name    | physics | chemistry | maths | total_marks | result
---------+-----------+---------+-----------+-------+-------------+--------
       1 | Adam      |      20 |        20 |    33 |          73 | p
       8 | Mary      |      24 |        14 |    31 |          69 | p
       2 | Bob       |      18 |         9 |    41 |          68 | f
      10 | Zack      |       8 |        20 |    36 |          64 | f
       6 | Fletcher  |       2 |        10 |    48 |          60 | f
       3 | Bright    |      22 |         7 |    31 |          60 | f
       5 | Elvin     |      14 |        22 |    23 |          59 | f
       9 | Tom       |      19 |        15 |    24 |          58 | f
       7 | Georgina  |      22 |        12 |    22 |          56 | f
       4 | Duke      |      13 |        21 |    20 |          54 | f
(10 rows)
```

7. Find pass percentage of the class for mathematics.



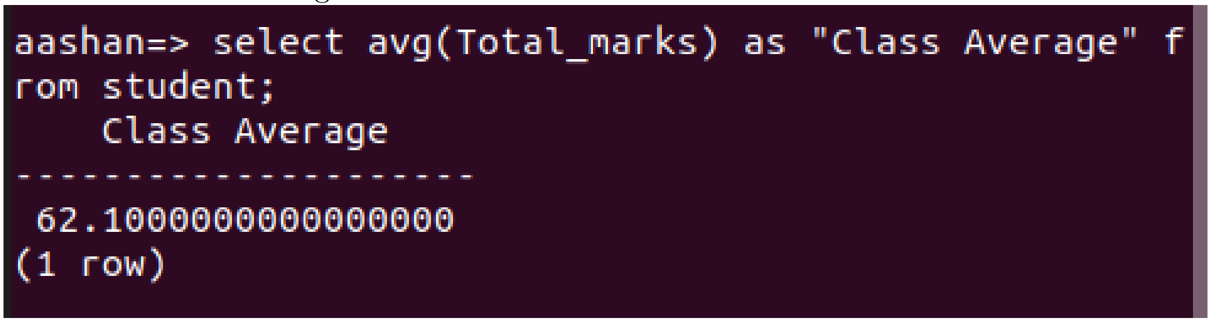
```
aashan=> select ((select count(maths) from student w
here maths>=25)::decimal / count(*)::decimal) * 100
 as "Pass_percentage_Maths" from student;
  Pass_percentage_Maths
-------------------------
 60.0000000000000000000
(1 row)
```

8. Find the overall pass percentage for all class.



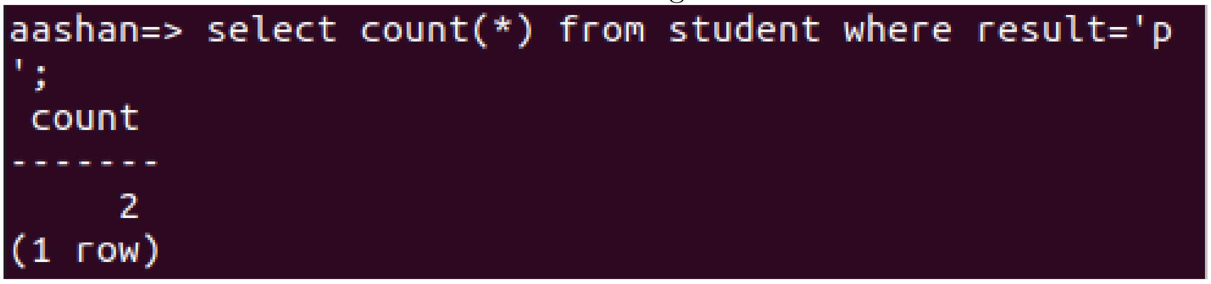
```
aashan=> select ((select count(*) from student where
result='p')::decimal / count(*)::decimal) * 100  as "
Pass_percentage" from student;
     Pass_percentage
-------------------------
 20.00000000000000000000
(1 row)
```

9. Find the class average.



```
aashan=> select avg(Total_marks) as "Class Average" f
rom student;
    Class Average
---------------------
 62.1000000000000000
(1 row)
```

10. Find the total number of students who have got a Pass.



```
aashan=> select count(*) from student where result='p
';
 count
-------
     2
(1 row)
```

RESULT:

Implemented the program for aggregate functions using Postgresql version: 11.4 ,Ubuntu 18.04 and following output were obtained.

EXPERIMENT 5

DATA CONSTRAINTS AND VIEWS

AIM:

To study about various data constraints and views in SQL.

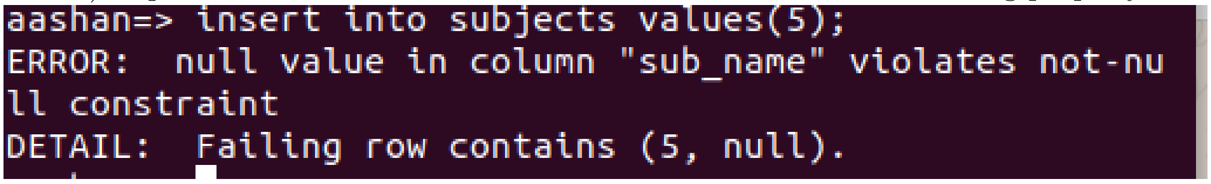
THEORY:

Whenever two tables are related by a common column (or set of columns), define a PRIMARY or UNIQUE key constraint on the column in the parent table, and define a FOREIGNKEY constraint on the column in the child table, to maintain the relationship between the two tables.

When both UNIQUE and NOTNULL constraints are defined on the foreign key, only one row in the child table can reference a parent key value. Because nulls are not allowed in the foreign key, each row in the child table must explicitly reference a value in the parent key.

QUESTIONS:

1. Create the following tables with given constraints a. Create a table named Subjects with the given attributes * Subid(Should not be NULL) * Subname (Should not be NULL) Populate the database. Make sure that all constraints are working properly.



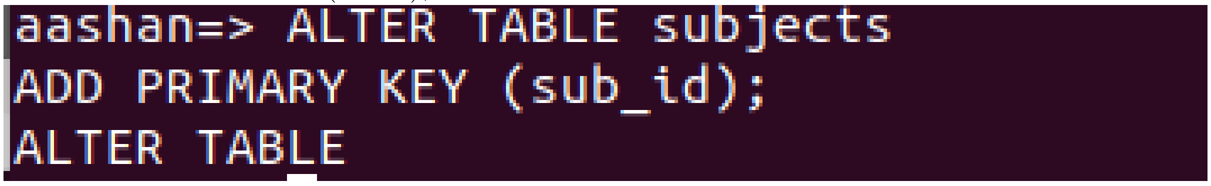
```
aashan=> insert into subjects values(5);
ERROR:  null value in column "sub_name" violates not-nu
ll constraint
DETAIL:  Failing row contains (5, null).
```

In the above figure , we can see that the NOT NULL constraint is working properly .

i) Alter the table to set subid as the primary key.

ALTER TABLE subjects

ADD PRIMARY KEY (subid);



```
aashan=> ALTER TABLE subjects
ADD PRIMARY KEY (sub_id);
ALTER TABLE
```

b. Create a table named Staff with the given attributes -staffid (Should be UNIQUE)

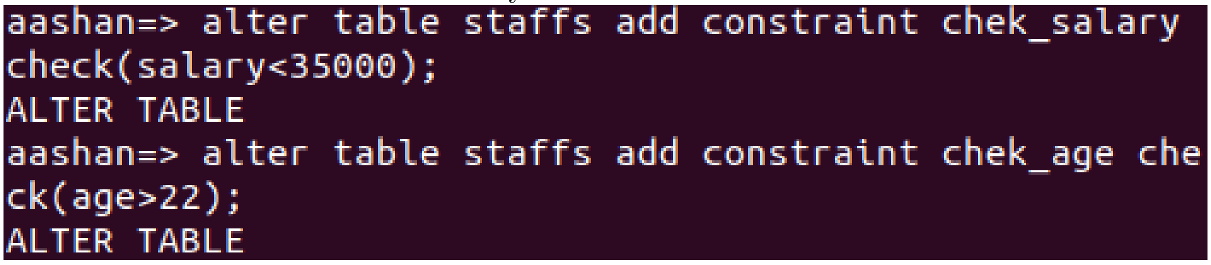
-staffname

-dept

-Age (Greater than 22)

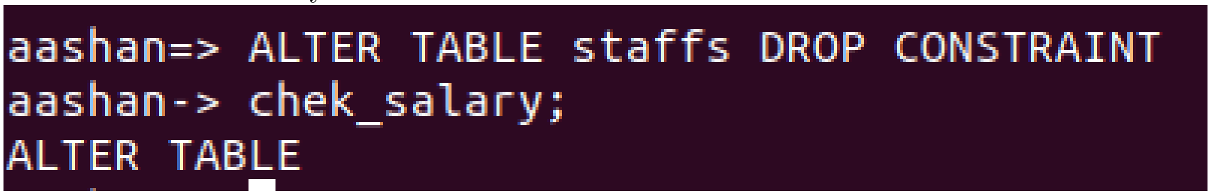
-Salary (Less than 35000)

Populate the database. Make sure that all constraints are working properly.
The staff TABLE has been succesfully created.



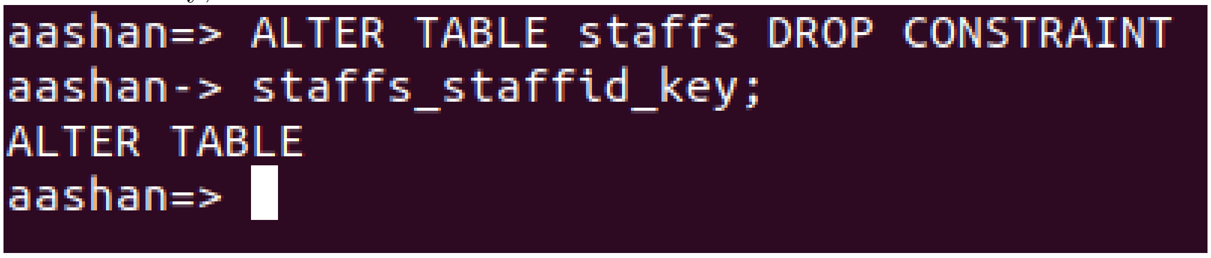
```
aashan=> alter table staffs add constraint chek_salary
check(salary<35000);
ALTER TABLE
aashan=> alter table staffs add constraint chek_age che
ck(age>22);
ALTER TABLE
```

Check constraint is working for ageCheck, constraint is working for salary also. Unique Constraint is also working Properly. Populated Complete Table is given
i)Delete the check constraint imposed on the attribute salary
ALTER TABLE staffs DROP CONSTRAINT chksalary;
We have added a salary of 35000 so that means that the Constraint has been removed



```
aashan=> ALTER TABLE staffs DROP CONSTRAINT
aashan-> chek_salary;
ALTER TABLE
```

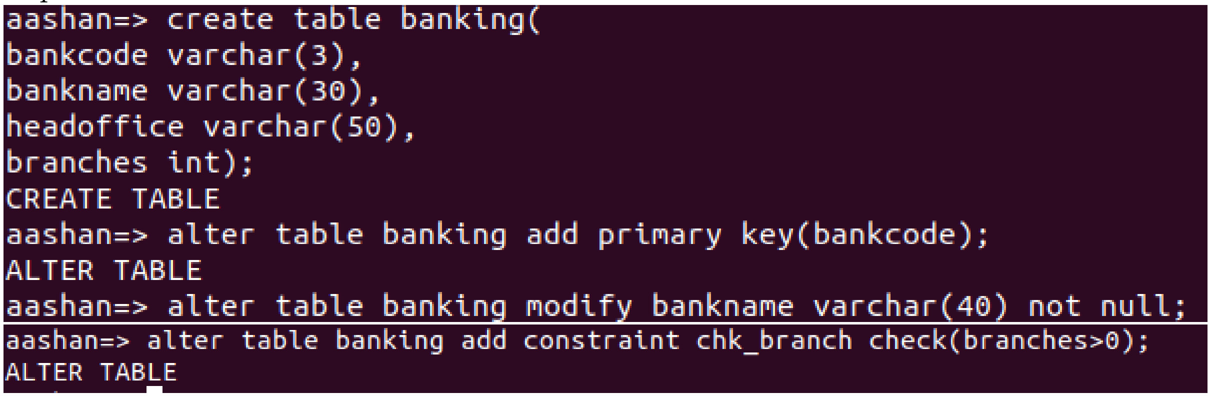
ii)Delete the unique constraint on the attribute staffid
ALTER TABLE staffs DROP CONSTRAINT
staffsstaffidkey;



```
aashan=> ALTER TABLE staffs DROP CONSTRAINT
aashan-> staffs_staffid_key;
ALTER TABLE
aashan=>
```

c. Create a table named Bank with the following attributes
-bankcode (To be set as Primary Key, type=varchar(3))
-bankname (Should not be NULL)
-headoffice-branches (Integer value greater than Zero)
Populate the database. Make sure that all constraints are working properly.All constraints have to be set after creating the table.
TABLE BANKING has been created
Primary Key constraint has been Added .
CHECK Constraint has been Added .
NOT NULL Constraint being Added.

Populated Table.



```
aashan=> create table banking(
bankcode varchar(3),
bankname varchar(30),
headoffice varchar(50),
branches int);
CREATE TABLE
aashan=> alter table banking add primary key(bankcode);
ALTER TABLE
aashan=> alter table banking modify bankname varchar(40) not null;
aashan=> alter table banking add constraint chk_branch check(branches>0);
ALTER TABLE
```

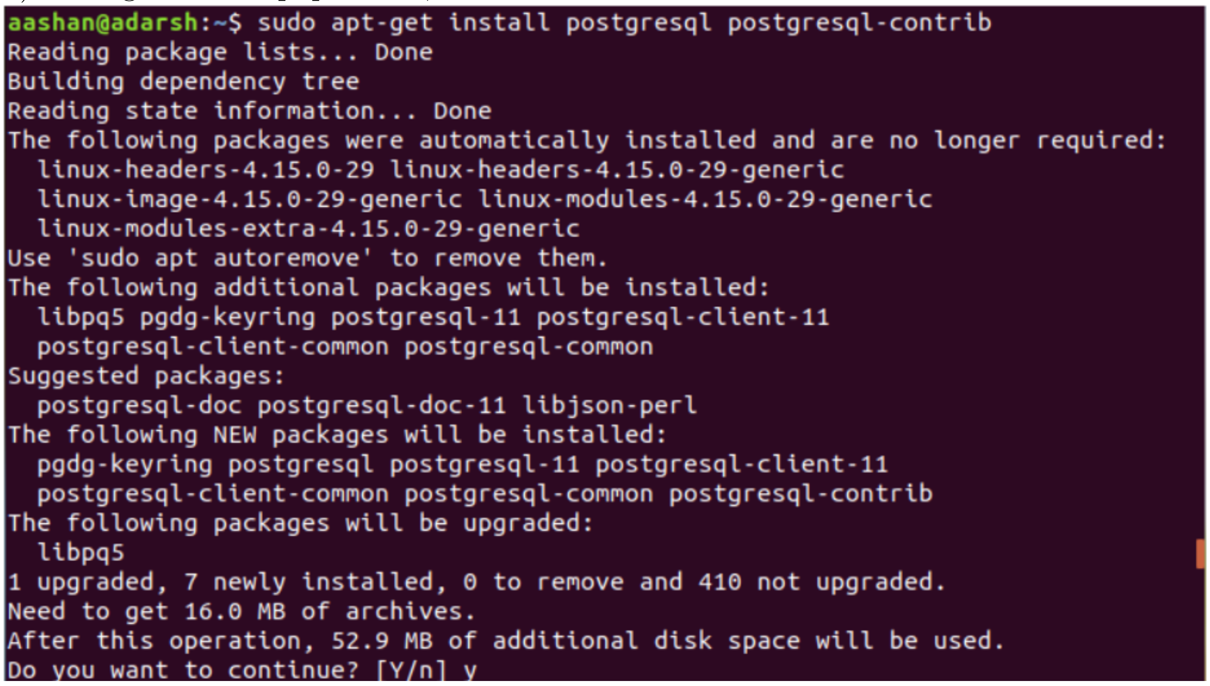
d. Create a table named Branch with the following attributes

-branchid (To be set as Primary Key)

-branchname (Set Default value as ‘New Delhi’)

-bankid (Foreign Key:- Refers to bank code of Bank table)

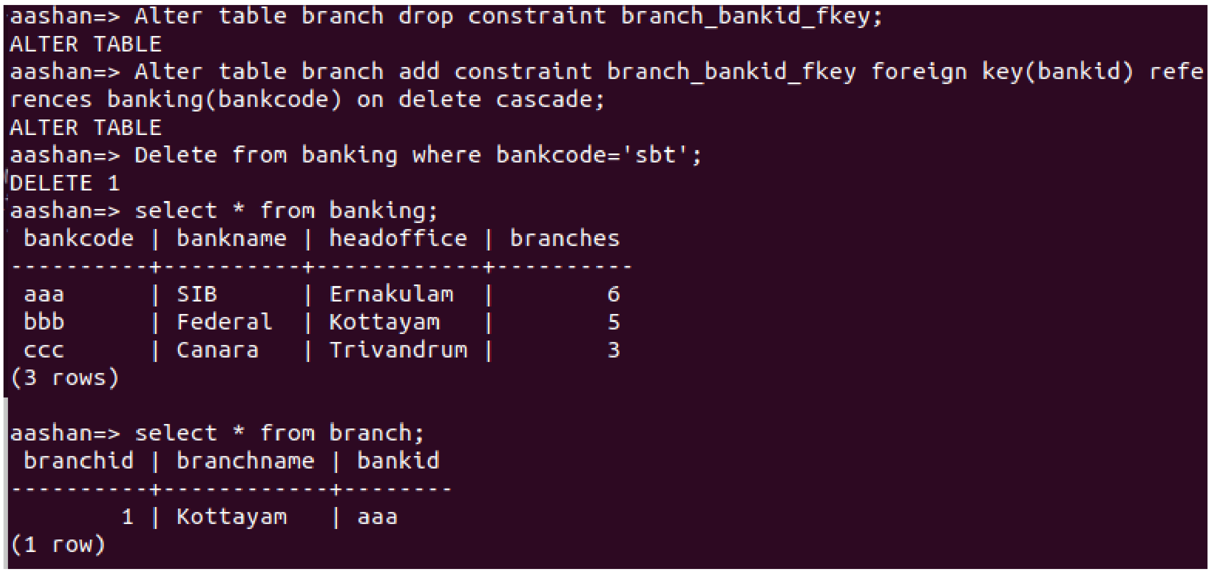
ii) During database population, demonstrate how the DEFAULT Constraint is satisfied.



```
aashan@adarsh:~$ sudo apt-get install postgresql postgresql-contrib
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-4.15.0-29 linux-headers-4.15.0-29-generic
  linux-image-4.15.0-29-generic linux-modules-4.15.0-29-generic
  linux-modules-extra-4.15.0-29-generic
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libpq5 pgdg-keyring postgresql-11 postgresql-client-11
  postgresql-client-common postgresql-common
Suggested packages:
  postgresql-doc postgresql-doc-11 libjson-perl
The following NEW packages will be installed:
  pgdg-keyring postgresql postgresql-11 postgresql-client-11
  postgresql-client-common postgresql-common postgresql-contrib
The following packages will be upgraded:
  libpq5
1 upgraded, 7 newly installed, 0 to remove and 410 not upgraded.
Need to get 16.0 MB of archives.
After this operation, 52.9 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

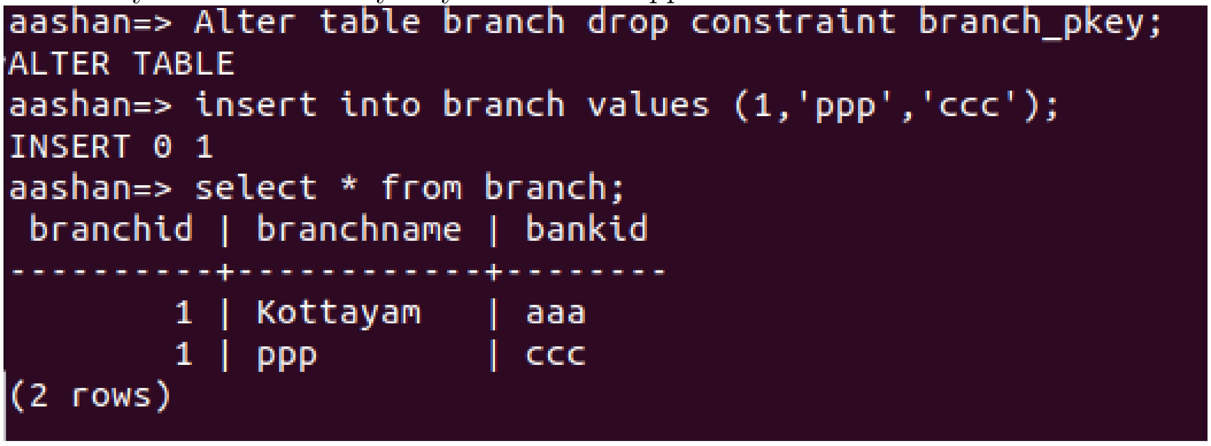
When we gave the Insert Operation without Specifying the branchname , It was Successfully Inserted and when we view the table , we see that the branchname is ‘NEW DELHI ‘ this is Because we used the DEFAULT CONSTRAINT which will set the branchname as ‘NEW DELHI’ if no other value is given to it

iii) Delete the bank with bank code ‘SBT’ and make sure that the corresponding entries are getting deleted from the related tables.



```
aashan=> Alter table branch drop constraint branch_bankid_fkey;
ALTER TABLE
aashan=> Alter table branch add constraint branch_bankid_fkey foreign key(bankid) refe
rences banking(bankcode) on delete cascade;
ALTER TABLE
aashan=> Delete from banking where bankcode='sbt';
DELETE 1
aashan=> select * from banking;
 bankcode | bankname | headoffice | branches
----------+----------+------------+----------
 aaa      | SIB      | Ernakulam  |        6
 bbb      | Federal  | Kottayam   |        5
 ccc      | Canara   | Trivandrum |        3
(3 rows)

aashan=> select * from branch;
 branchid | branchname | bankid
----------+------------+--------
        1 | Kottayam   | aaa
(1 row)
```

WE Tackle this Problem with the help of ON DELETE CASCADING using ALTER TABLE STATEMENT .First we dropped the foreign key using ALTER TABLE BRANCH DROP CONSTRAINT branchbankidfkey
Then we add another Key Constraint
ALTER TABLE BRANCH ADD CONSTRAINT
branchbankidfkey FOREIGN KEY(bankid)
REFERENCES BANKING(bankcode) ON DELETECASCADE;
ON DELETE CASCADE Enables that when the tuple in the parent table is dropped ,the corresponding data in the child tables which reference the parent using a FOREIGN KEY will also get deleted as seen in the Above Figure .
iv) Drop the Primary Key using ALTER command Dropping a Primary Key is done as ALTER TABLE BRANCH DROP CONSTRAINT branchpkey
Here branchpkey is the name by which the primary key of BRANCH can be seen .
we can say that the Primary Key has been Dropped



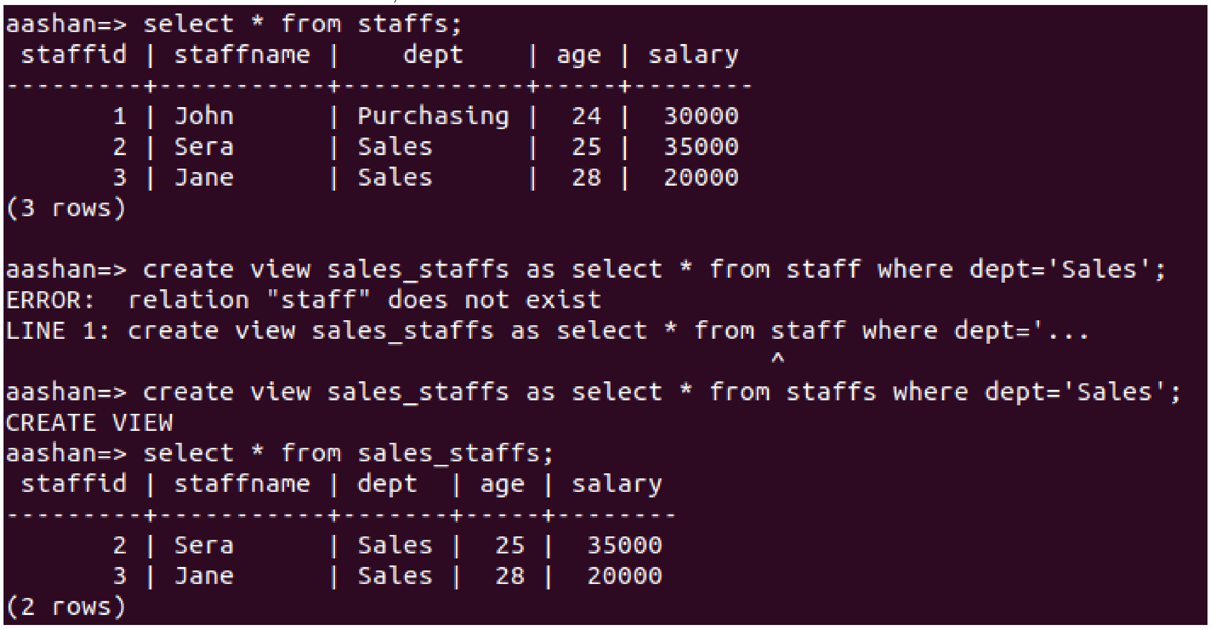
```
aashan=> Alter table branch drop constraint branch_pkey;
ALTER TABLE
aashan=> insert into branch values (1,'ppp','ccc');
INSERT 0 1
aashan=> select * from branch;
 branchid | branchname | bankid
----------+------------+--------
        1 | Kottayam   | aaa
        1 | ppp        | ccc
(2 rows)
```

VIEWS

1. Create a View named salesstaff to hold the details of all staff working in sales Department STAFF TABLE is given below

The VIEW salesstaff is given below

SELECT * FROM salesstaff;



```
aashan=> select * from staffs;
 staffid | staffname |    dept    | age | salary
---------+-----------+------------+-----+--------
       1 | John      | Purchasing |  24 |  30000
       2 | Sera      | Sales      |  25 |  35000
       3 | Jane      | Sales      |  28 |  20000
(3 rows)

aashan=> create view sales_staffs as select * from staff where dept='Sales';
ERROR:  relation "staff" does not exist
LINE 1: create view sales_staffs as select * from staff where dept='...
                                                  ^
aashan=> create view sales_staffs as select * from staffs where dept='Sales';
CREATE VIEW
aashan=> select * from sales_staffs;
 staffid | staffname | dept  | age | salary
---------+-----------+-------+-----+--------
       2 | Sera      | Sales |  25 |  35000
       3 | Jane      | Sales |  28 |  20000
(2 rows)
```

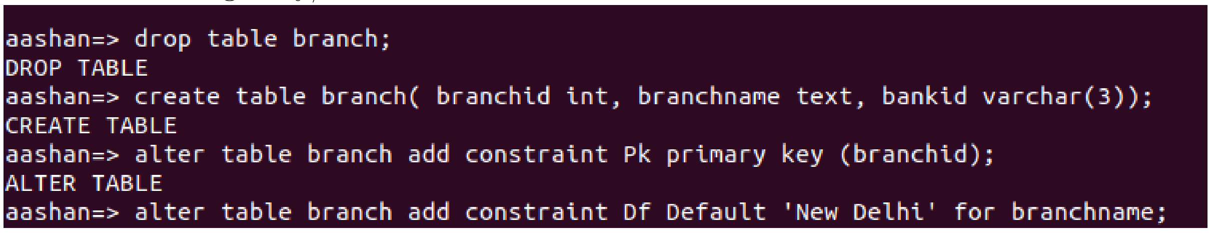
2. Drop table branch. Create another table named branch and name all the constraints as given below:

Constraint name Column Constraint

Pk branchid Primary key

Df branchname Default :'New Delhi'

Fk bankid Foreign key/References



```
aashan=> drop table branch;
DROP TABLE
aashan=> create table branch( branchid int, branchname text, bankid varchar(3));
CREATE TABLE
aashan=> alter table branch add constraint Pk primary key (branchid);
ALTER TABLE
aashan=> alter table branch add constraint Df Default 'New Delhi' for branchname;
```

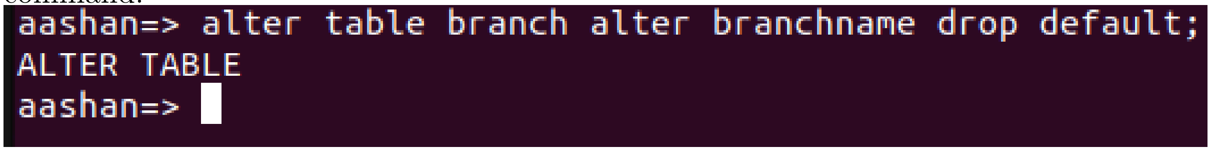
THE table BRANCH is dropped using the Statement

DROP TABLE BRANCH.

Here we Dropped the BRANCH Table and created a new One with the 3 Constraints Specified Outside

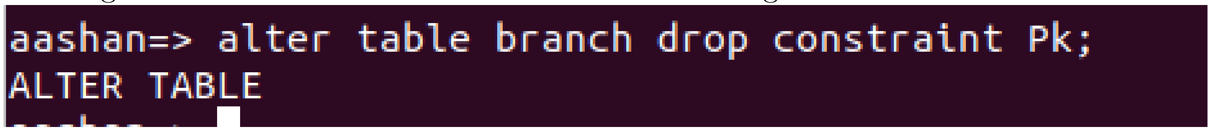
i) Delete the default constraint in the table

To Delete a Constraint , we use the DROP command along with the ALTER TABLE command.



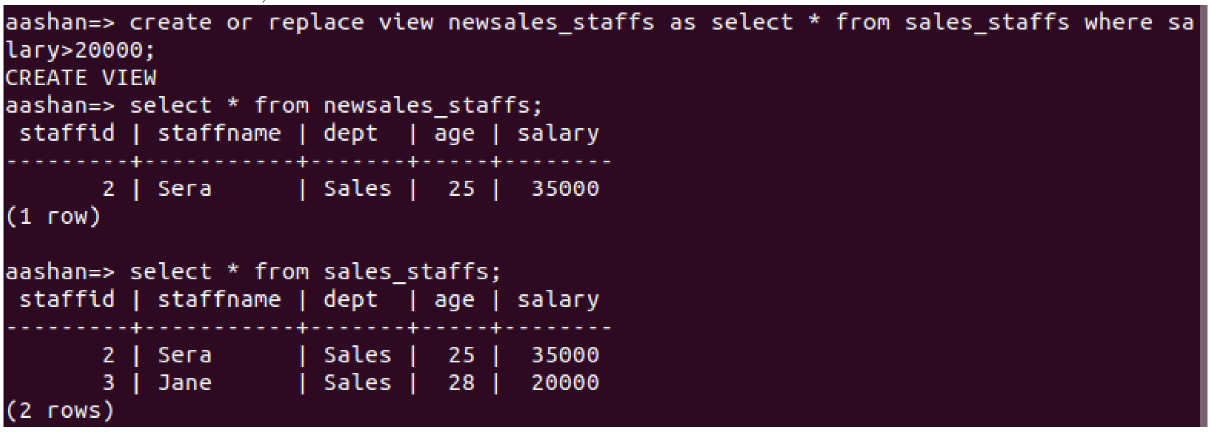
```
aashan=> alter table branch alter branchname drop default;
ALTER TABLE
aashan=>
```

ii) Delete the primary key constraint
Here Again we use the ALTER TABLE command Along with the DROP command .



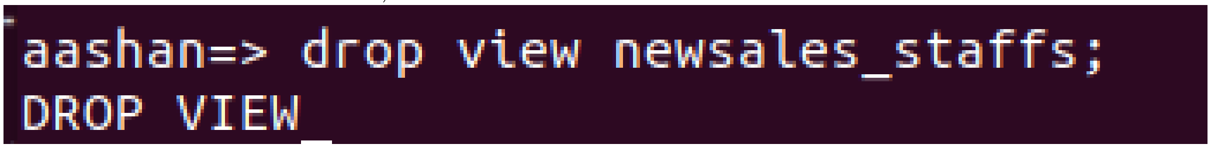
```
aashan=> alter table branch drop constraint Pk;
ALTER TABLE
```

3. Update the view salesstaff to include the details of staff belonging to sales department whose salary is greater than 20000.A view can be updated with the CREATE OR REPLACE VIEW command.
CREATE OR REPLACE VIEW viewname AS
SELECT column1, column2, ...
FROM tablename
WHERE condition;



```
aashan=> create or replace view newsales_staffs as select * from sales_staffs where sa
lary>20000;
CREATE VIEW
aashan=> select * from newsales_staffs;
 staffid | staffname | dept  | age | salary
---------+-----------+-------+-----+--------
       2 | Sera      | Sales |  25 |  35000
(1 row)

aashan=> select * from sales_staffs;
 staffid | staffname | dept  | age | salary
---------+-----------+-------+-----+--------
       2 | Sera      | Sales |  25 |  35000
       3 | Jane      | Sales |  28 |  20000
(2 rows)
```

5. Delete the view salesstaff.
A view is deleted with the DROP VIEW command.
DROP VIEW newsalesstuff;



```
aashan=> drop view newsales_staffs;
DROP VIEW
```

RESULT:
Implemented the program for data-constraint and views using Postgresql version: 11.4 ,Ubuntu 18.04 and following output were obtained.

Experiment 6
STRING FUNCTIONS AND PATTERN MATCHING

AIM:

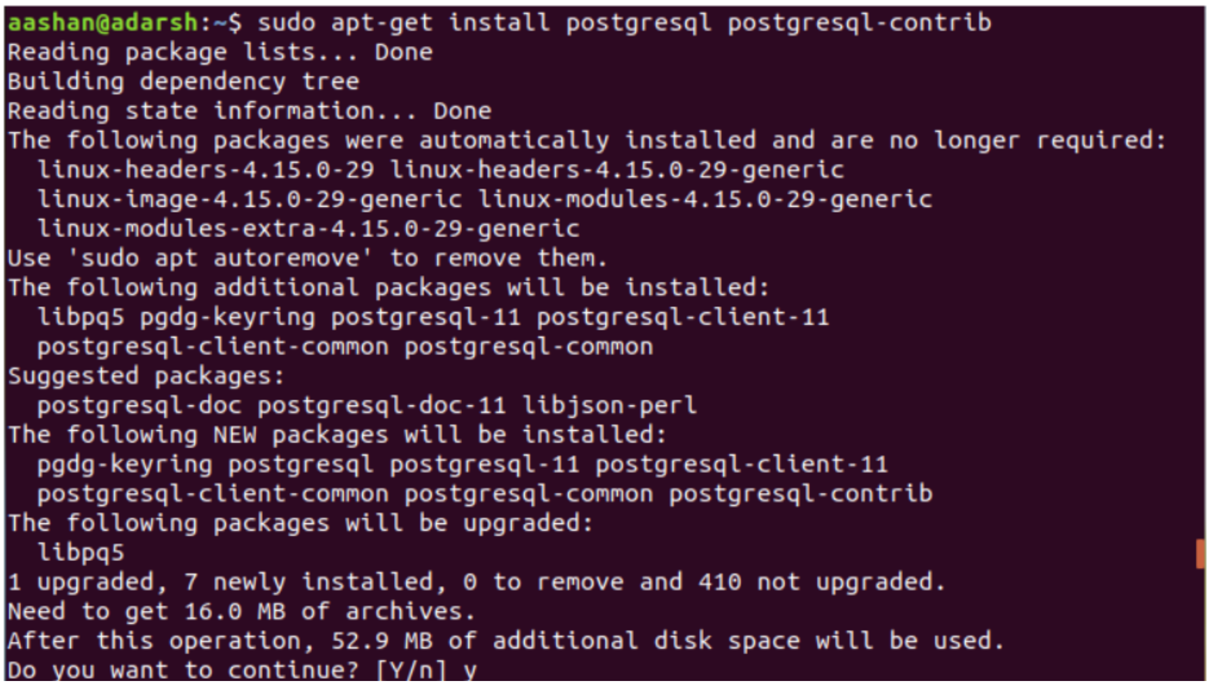
To understand
- SUBSTR - RPAD
- INITCAP - INSTR
- LPAD - CONCAT
- LTRIM - UPPER
- LENGTH - RTRIM
- LOWER - REVERSE

THEORY:

The main string functions are as follows
1.Length() 2.Lower() 3.Upper() 4.Concat()
5.Lpad() 6.Rpad() 7.Rtrim() 8.Substr()
-Length(field name/string)
Gives the length of the string.
-Lower(field name/string)
Gives the content in lowercase letters
-Upper(field name/string)
Gives the content in upper case letters
-Concat (field name/string, field name/string)
Combines the first and second string into single one.
-Lpad(field name/string,length,character)

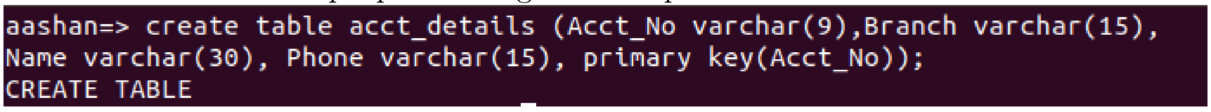
QUESTIONS:

Create a table named acctdetails and populate the table



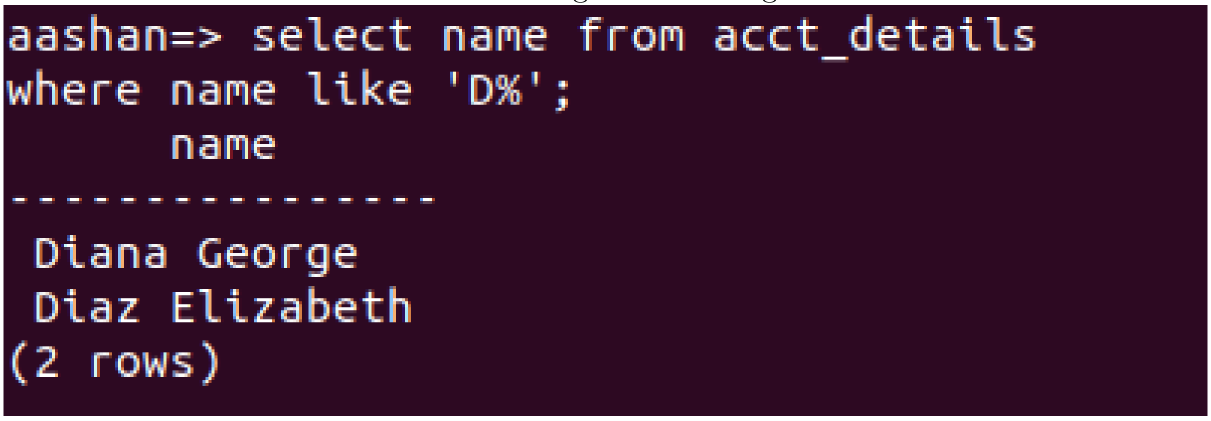
```
aashan@adarsh:~$ sudo apt-get install postgresql postgresql-contrib
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-4.15.0-29 linux-headers-4.15.0-29-generic
  linux-image-4.15.0-29-generic linux-modules-4.15.0-29-generic
  linux-modules-extra-4.15.0-29-generic
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libpq5 pgdg-keyring postgresql-11 postgresql-client-11
  postgresql-client-common postgresql-common
Suggested packages:
  postgresql-doc postgresql-doc-11 libjson-perl
The following NEW packages will be installed:
  pgdg-keyring postgresql postgresql-11 postgresql-client-11
  postgresql-client-common postgresql-common postgresql-contrib
The following packages will be upgraded:
  libpq5
1 upgraded, 7 newly installed, 0 to remove and 410 not upgraded.
Need to get 16.0 MB of archives.
After this operation, 52.9 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

1.Find the names of all people starting on the alphabet 'D'



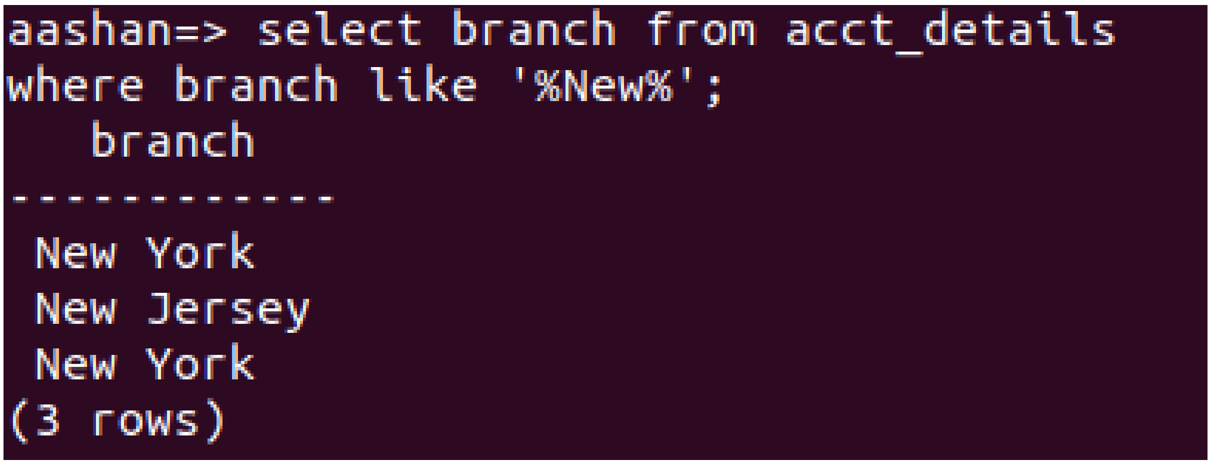
```
aashan=> create table acct_details (Acct_No varchar(9),Branch varchar(15),
Name varchar(30), Phone varchar(15), primary key(Acct_No));
CREATE TABLE
```

2.List the names of all branches containing the substring 'New'



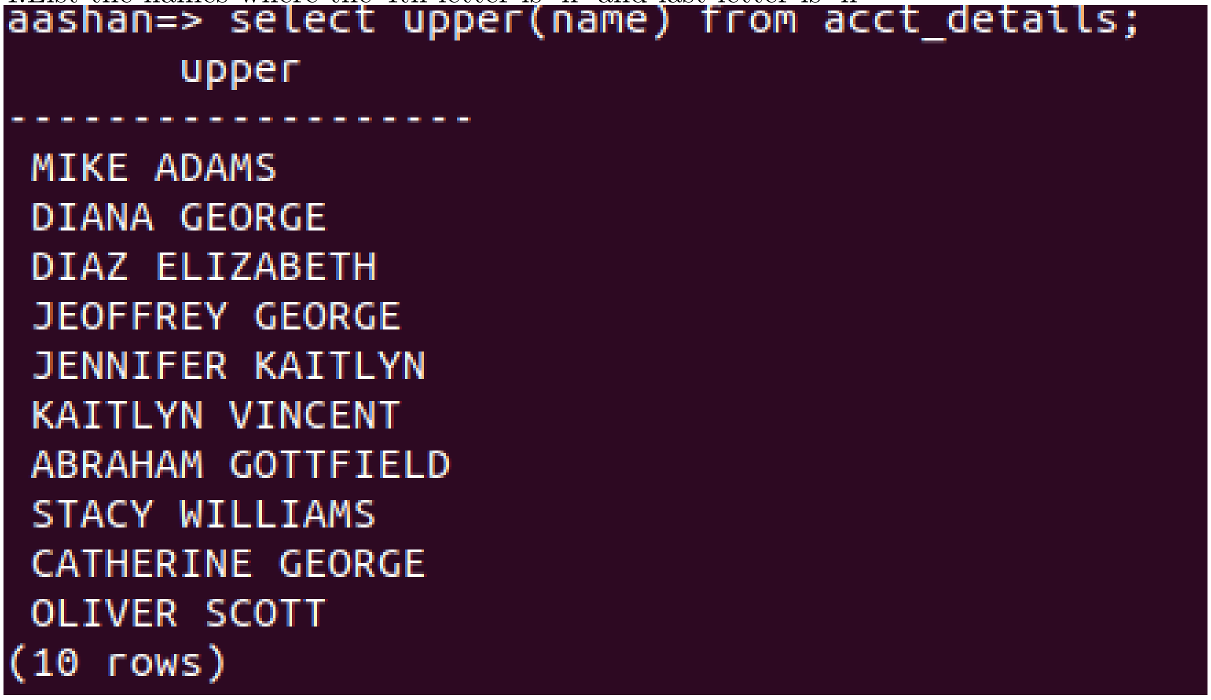
```
aashan=> select name from acct_details
where name like 'D%';
      name
----------------
 Diana George
 Diaz Elizabeth
(2 rows)
```

3.List all the names in Upper Case Format



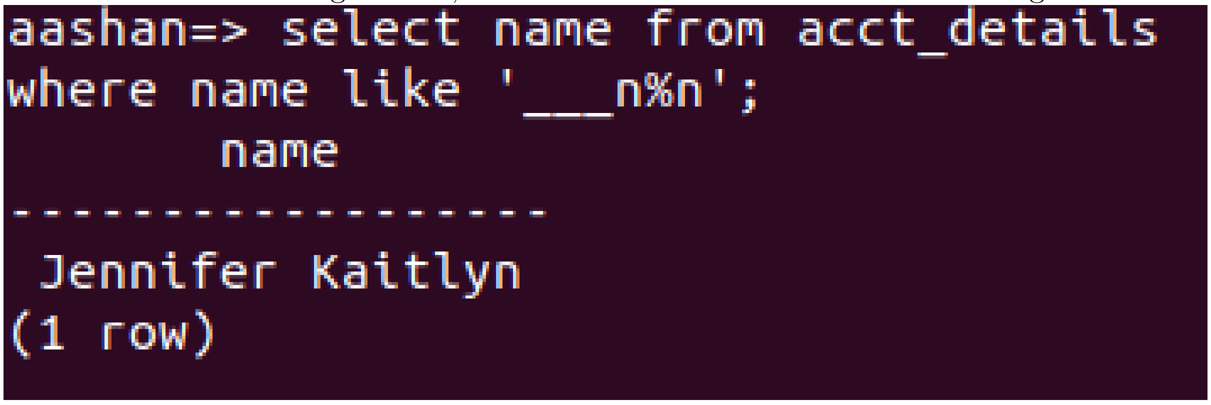
```
aashan=> select branch from acct_details
where branch like '%New%';
   branch
-------------
 New York
 New Jersey
 New York
(3 rows)
```

4.List the names where the 4th letter is ‘n’ and last letter is ‘n’



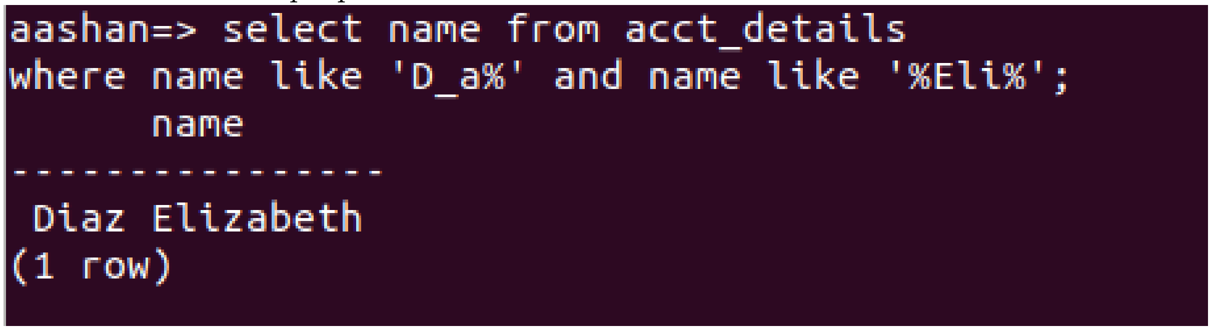
```
aashan=> select upper(name) from acct_details;
       upper
-------------------
 MIKE ADAMS
 DIANA GEORGE
 DIAZ ELIZABETH
 JEOFFREY GEORGE
 JENNIFER KAITLYN
 KAITLYN VINCENT
 ABRAHAM GOTTFIELD
 STACY WILLIAMS
 CATHERINE GEORGE
 OLIVER SCOTT
(10 rows)
```

5.List the names starting on ‘D’ , 3rd letter is ‘a’ and contains the substring‘Eli’



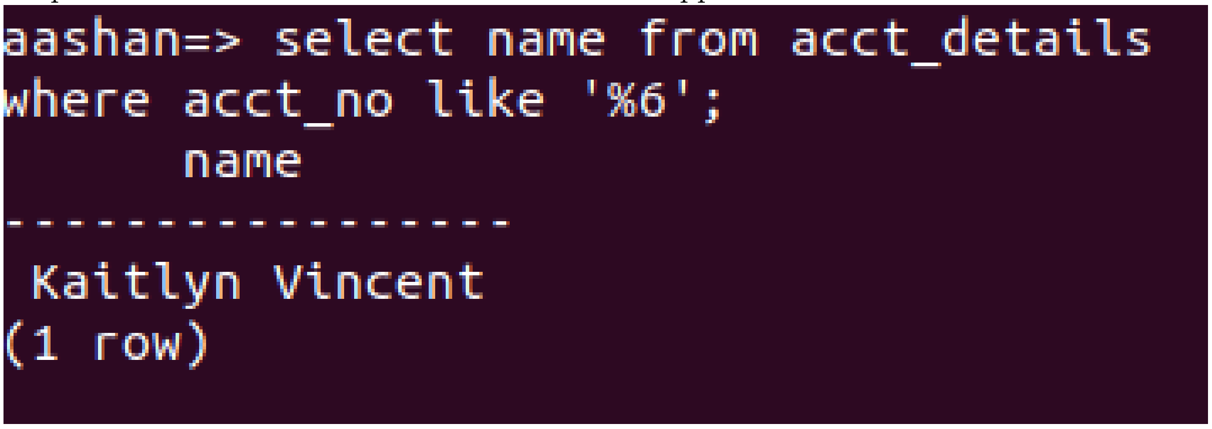
```
aashan=> select name from acct_details
where name like '___n%n';
       name
------------------
 Jennifer Kaitlyn
(1 row)
```

6. List the names of people whose account number ends in '6'



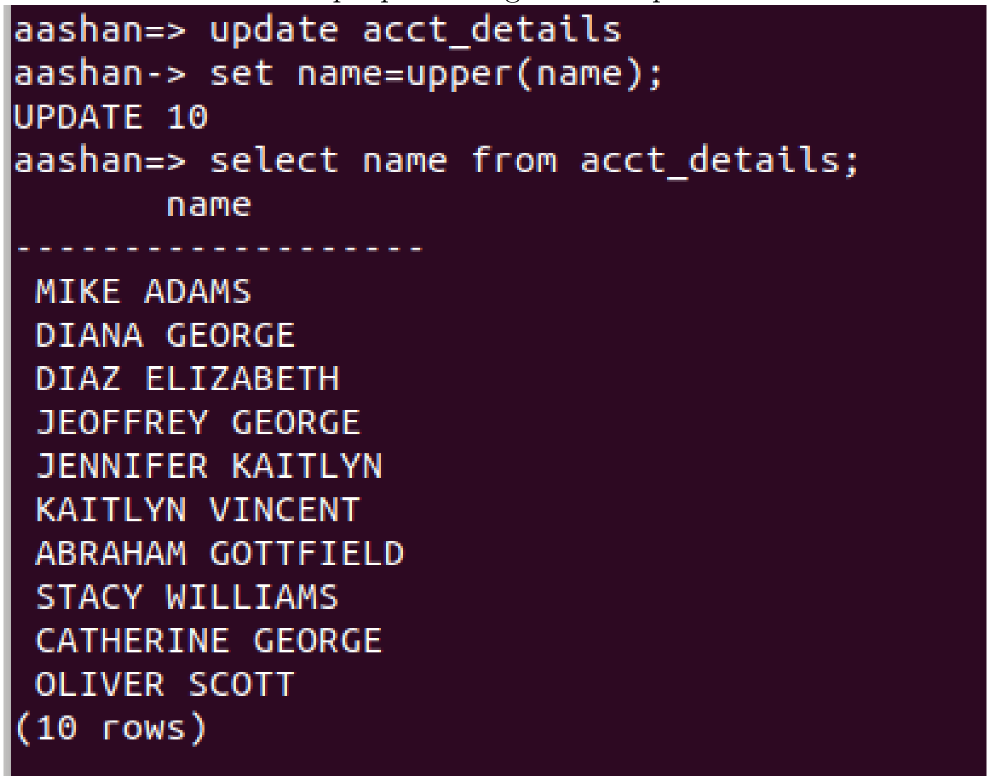
```
aashan=> select name from acct_details
where name like 'D_a%' and name like '%Eli%';
      name
----------------
 Diaz Elizabeth
(1 row)
```

7.Update the table so that all the names are in Upper Case Format



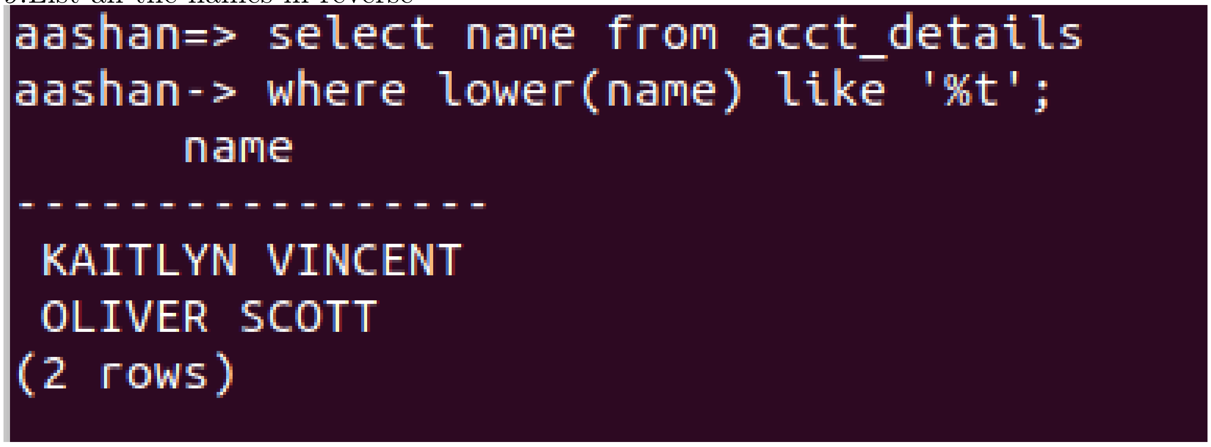
```
aashan=> select name from acct_details
where acct_no like '%6';
      name
-----------------
 Kaitlyn Vincent
(1 row)
```

8. List the names of all people ending on the alphabet 't'



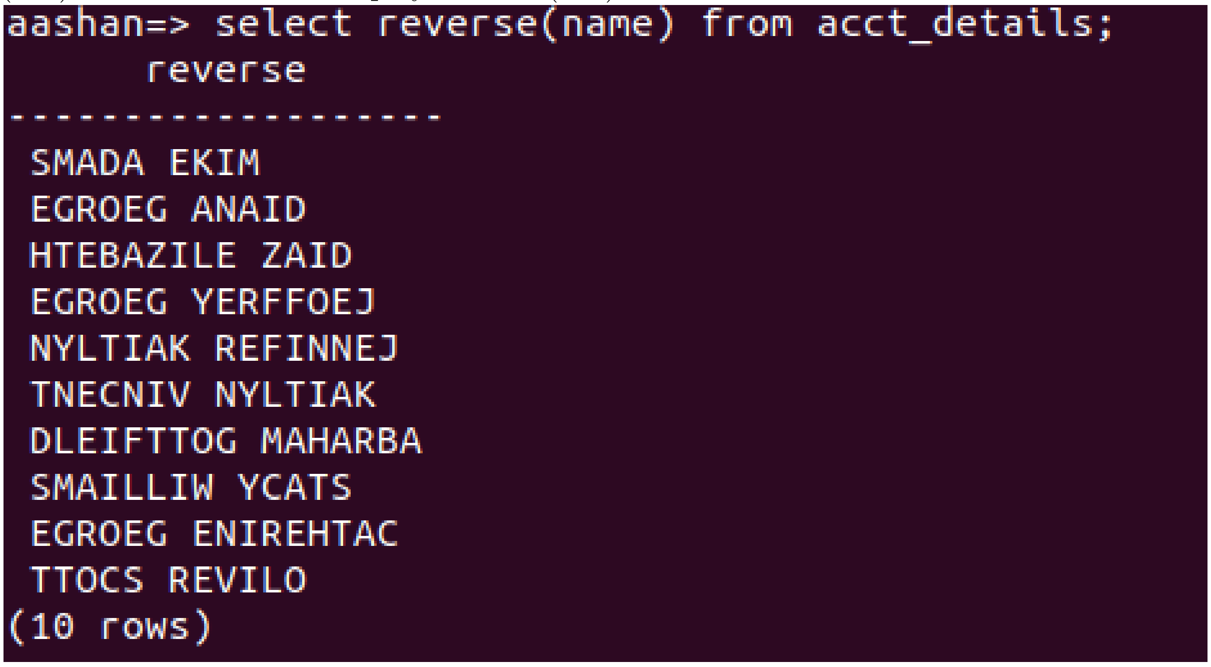
```
aashan=> update acct_details
aashan-> set name=upper(name);
UPDATE 10
aashan=> select name from acct_details;
       name
-------------------
 MIKE ADAMS
 DIANA GEORGE
 DIAZ ELIZABETH
 JEOFFREY GEORGE
 JENNIFER KAITLYN
 KAITLYN VINCENT
 ABRAHAM GOTTFIELD
 STACY WILLIAMS
 CATHERINE GEORGE
 OLIVER SCOTT
(10 rows)
```

9.List all the names in reverse



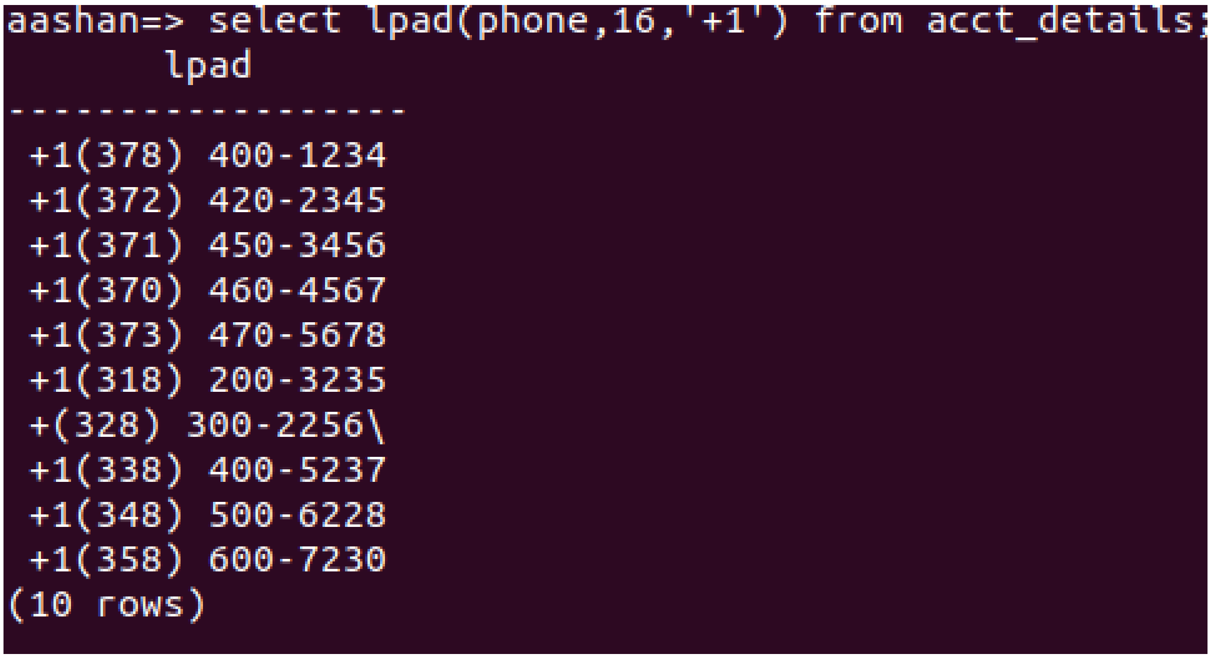
```
aashan=> select name from acct_details
aashan-> where lower(name) like '%t';
      name
------------------
 KAITLYN VINCENT
 OLIVER SCOTT
(2 rows)
```

10. Display all the phone numbers including US Country code (+1). For eg: (378)400-1234should be displayed as +1(378)400-1234. Use LPAD function



```
aashan=> select reverse(name) from acct_details;
      reverse
-------------------
 SMADA EKIM
 EGROEG ANAID
 HTEBAZILE ZAID
 EGROEG YERFFOEJ
 NYLTIAK REFINNEJ
 TNECNIV NYLTIAK
 DLEIFTTOG MAHARBA
 SMAILLIW YCATS
 EGROEG ENIREHTAC
 TTOCS REVILO
(10 rows)
```

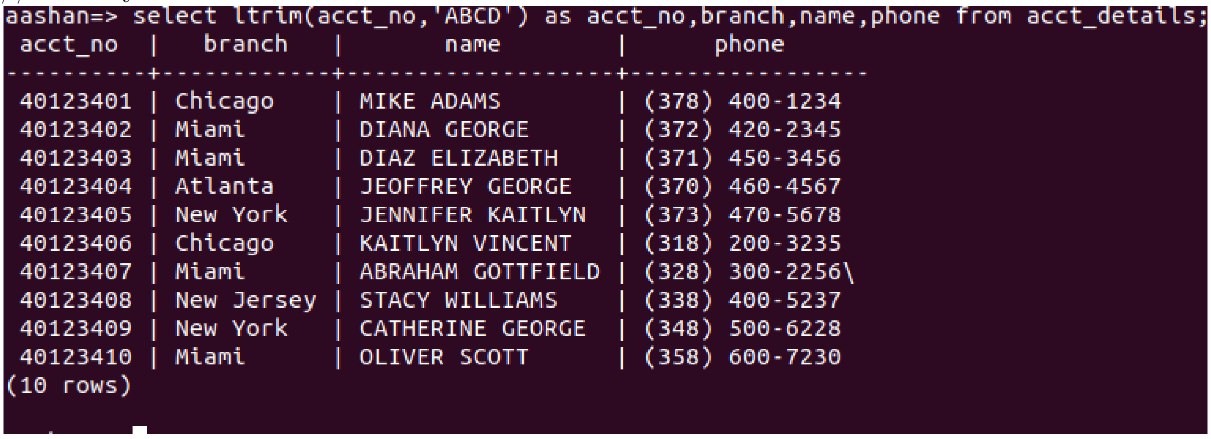
11. Display all the account numbers. The starting alphabet associated with the AccountNo should be removed. Use LTRIM function.



```
aashan=> select lpad(phone,16,'+1') from acct_details;
       lpad
------------------
 +1(378) 400-1234
 +1(372) 420-2345
 +1(371) 450-3456
 +1(370) 460-4567
 +1(373) 470-5678
 +1(318) 200-3235
 +(328) 300-2256\
 +1(338) 400-5237
 +1(348) 500-6228
 +1(358) 600-7230
(10 rows)
```

12. Display the details of all people whose account number starts in '4' and name contains the substring 'Williams'.

//Nobody's account number starts with a number

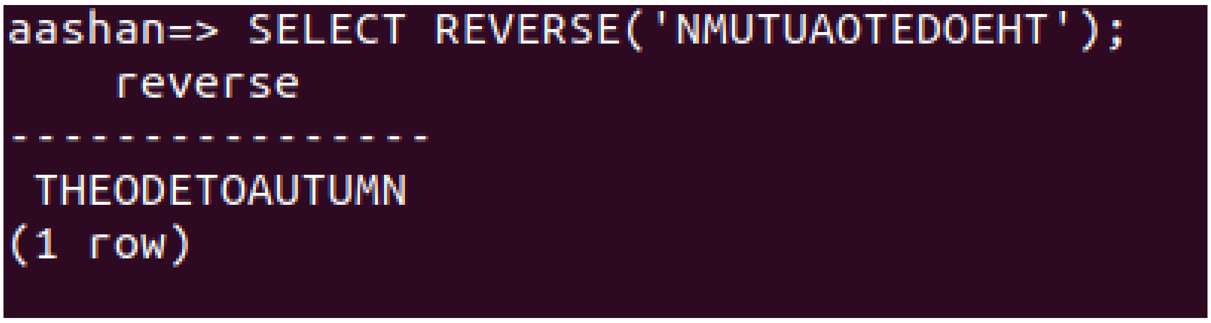


```
aashan=> select ltrim(acct_no,'ABCD') as acct_no,branch,name,phone from acct_details;
 acct_no  |   branch   |       name        |      phone
----------+------------+-------------------+-----------------
 40123401 | Chicago    | MIKE ADAMS        | (378) 400-1234
 40123402 | Miami      | DIANA GEORGE      | (372) 420-2345
 40123403 | Miami      | DIAZ ELIZABETH    | (371) 450-3456
 40123404 | Atlanta    | JEOFFREY GEORGE   | (370) 460-4567
 40123405 | New York   | JENNIFER KAITLYN  | (373) 470-5678
 40123406 | Chicago    | KAITLYN VINCENT   | (318) 200-3235
 40123407 | Miami      | ABRAHAM GOTTFIELD | (328) 300-2256\
 40123408 | New Jersey | STACY WILLIAMS    | (338) 400-5237
 40123409 | New York   | CATHERINE GEORGE  | (348) 500-6228
 40123410 | Miami      | OLIVER SCOTT      | (358) 600-7230
(10 rows)
```

B.Use the system table DUAL for the following questions:

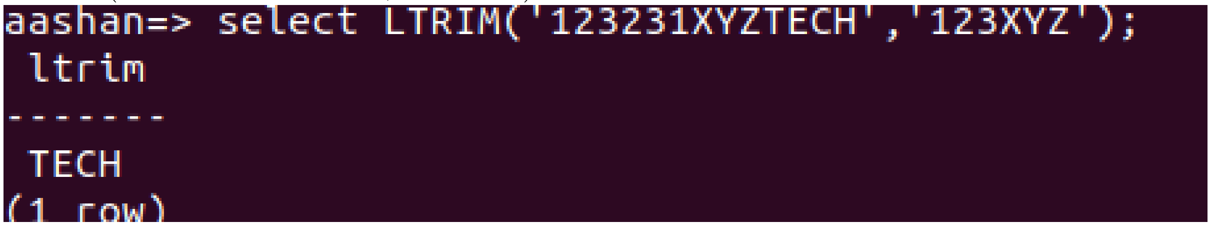
1.Find the reverse of the string ' nmutuAotedOehT'

REVERSE('NMUTUAOTEDOEHT')



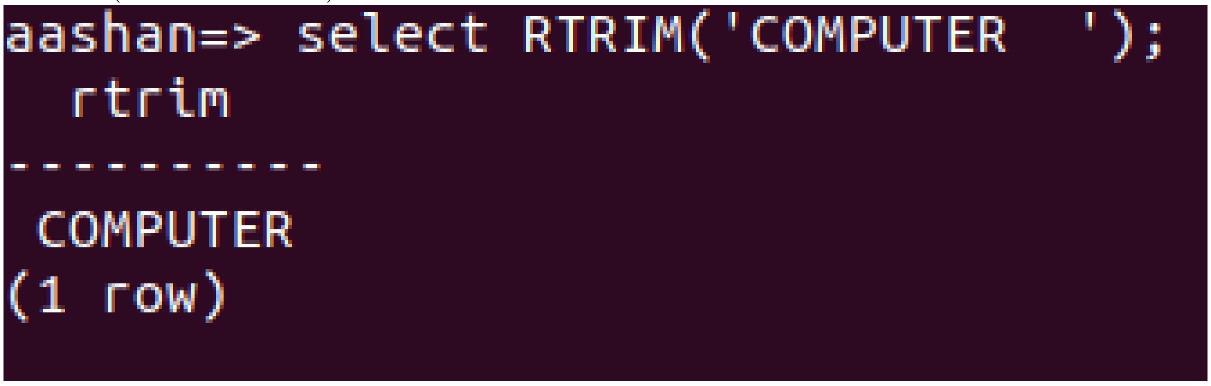
```
aashan=> SELECT REVERSE('NMUTUAOTEDOEHT');
    reverse
----------------
 THEODETOAUTUMN
(1 row)
```

2.Use LTRIM function on ‘123231xyzTech’ so as to obtain the output ‘Tech’
LTRIM(’123231XYZTECH’,’123XYZ’)



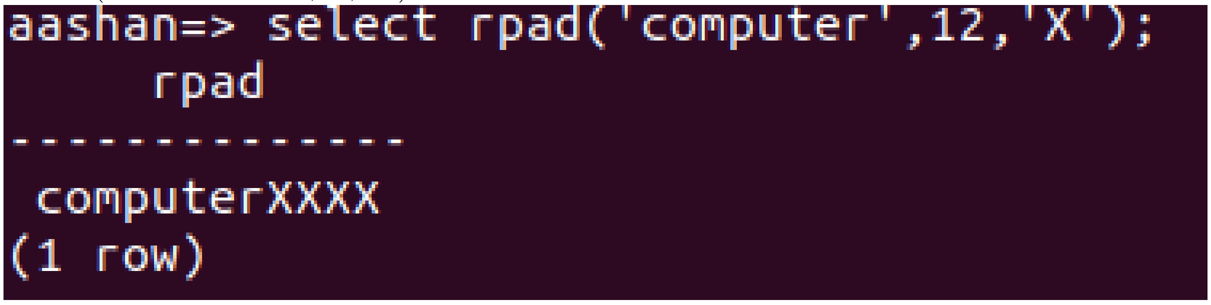
```
aashan=> select LTRIM('123231XYZTECH','123XYZ');
 ltrim
-------
 TECH
(1 row)
```

3.Use RTRIM function on ‘Computer ‘ to remove the trailing spaces.
RTRIM(’COMPUTER’)



```
aashan=> select RTRIM('COMPUTER   ');
  rtrim
----------
 COMPUTER
(1 row)
```

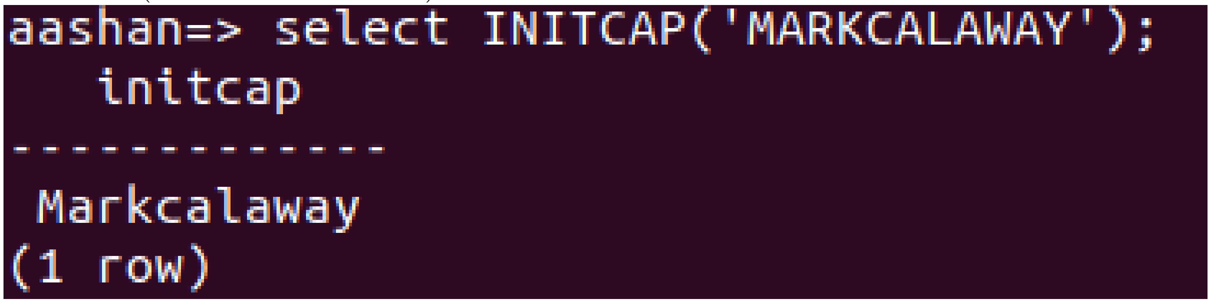
4.Perform RPAD on ‘computer’ to obtain the output as ‘computerXXXX’
RPAD(’COMPUTER’,12,’X’)



```
aashan=> select rpad('computer',12,'X');
     rpad
--------------
 computerXXXX
(1 row)
```

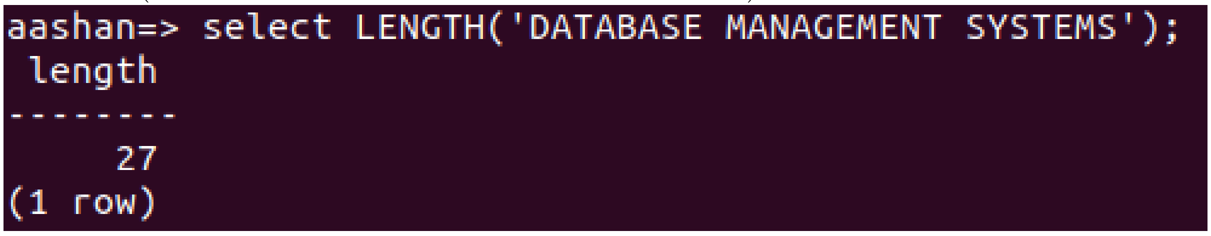
5.Use INSTR function to find the first occurrence of ‘e’ in the string ‘WelcometoKerala’
INSTR(’WELCOME TO KERALA’,’E’,1,1)
//There is no instr() in postgres 6. Perform INITCAP function on ‘mARKcALAwaY’

INITCAP('MARKCALAWAY')



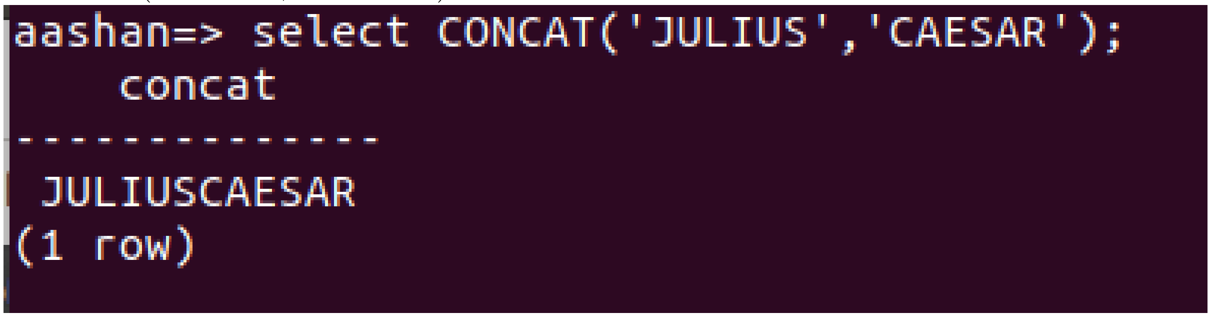
```
aashan=> select INITCAP('MARKCALAWAY');
   initcap
-------------
 Markcalaway
(1 row)
```

7. Find the length of the string ‘Database Management Systems’.
LENGTH('DATABASE MANAGEMENT SYSTEMS')



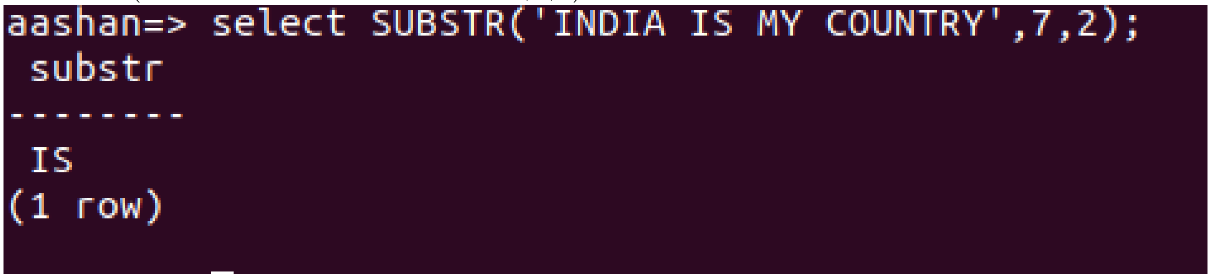
```
aashan=> select LENGTH('DATABASE MANAGEMENT SYSTEMS');
 length
--------
     27
(1 row)
```

8. Concatenate the strings ‘Julius’ and ‘Caesar’
CONCAT('JULIUS','CAESAR')



```
aashan=> select CONCAT('JULIUS','CAESAR');
    concat
-------------
 JULIUSCAESAR
(1 row)
```

9.Use SUBSTR function to retrieve the substring ‘is’ from the string ‘Indiaismycountry’.
SUBSTR('INDIA IS MY COUNTRY',7,2)



```
aashan=> select SUBSTR('INDIA IS MY COUNTRY',7,2);
 substr
--------
 IS
(1 row)
```

10.Use INSTR function to find the second occurrence of ‘k’ from the last.
The string is ‘Making of a King’.

INSTR('MAKING OF A KING','K', -1,2)

//there is no instr() in postgres

RESULT:
Implemented the program for string fucntions and pattern matching using Postgresql version: 11.4 ,Ubuntu 18.04 and following output were obtained.

Experiment 7
JOIN STATEMENTS, SET OPERATIONS, NESTED QUERIES AND GROUPING

AIM:

To get introduced to

- - UNION
- - INTERSECTION
- - MINUS
- - JOIN
- - NESTED QUERIES
- - GROUP BY and HAVING

THEORY:

Joins

A join is a query that combines rows from two or more tables, views, or materialized views ("snapshots"). Oracle performs a join whenever multiple tables appear in the query's FROM clause. The query's select list can select any columns from any of these tables. If any two of these tables have a column name in common, you must qualify all references to these columns throughout the query with table names to avoid ambiguity.

Join Conditions

Most join queries contain WHERE clause conditions that compare two columns, each from a different table. Such a condition is called a join condition.

Equijoins

An equijoin is a join with a join condition containing an equality operator. An equijoin combines rows that have equivalent values for the specified columns.

Self Joins

A self join is a join of a table to itself.

Cartesian Products

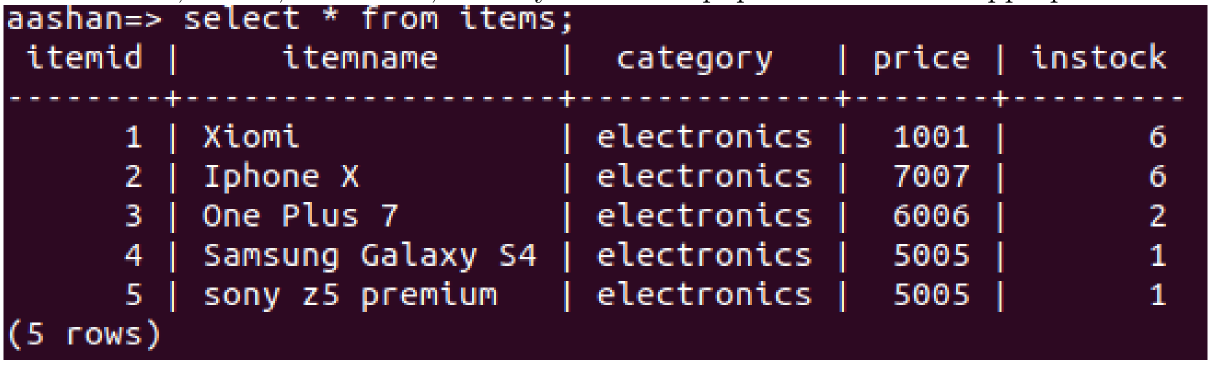
If two tables in a join query have no join condition, Oracle returns their Cartesian product.

Outer Joins

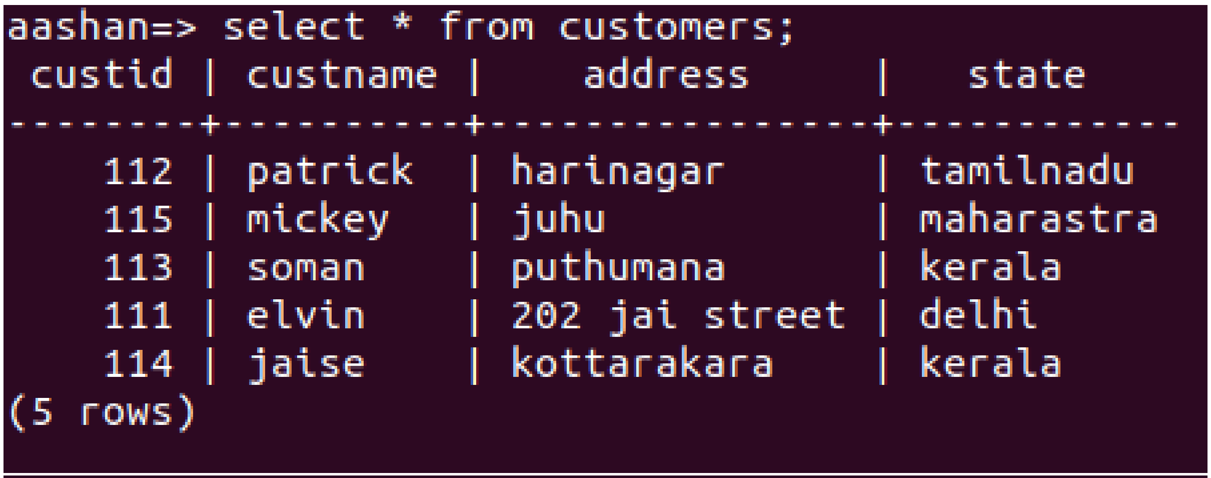
An outer join extends the result of a simple join. An outer join returns all rows that satisfy the join condition and those rows from one table for which no rows from the other satisfy the join condition. Such rows are not returned by a simple join.

Set Operators:

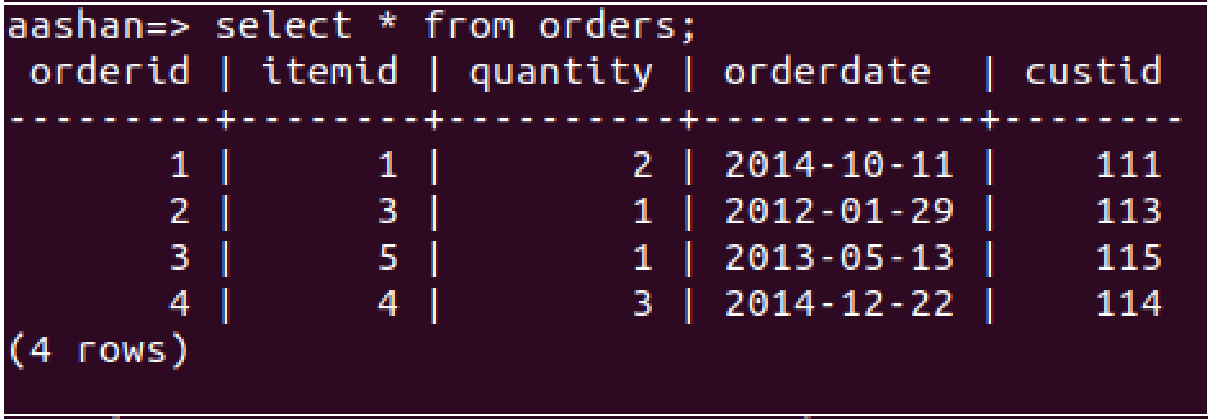
Set operators combine the results of two component queries into a single result. Queries containing set operators are called compound queries. Table lists SQL set operators.
Operator Returns
UNION All - rows selected by either query.
UNION ALL All - rows selected by either query, including all duplicates.
INTERSECT All - distinct rows selected by both queries.
MINUS All - distinct rows selected by the first query but not the second.
All set operators have equal precedence.
NESTED QUERIES
Subquery:
If a sql statement contains another sql statement then the sql statement which is inside another sql statement is called Subquery. It is also known as nested query. The Sql Statement which contains the other sql statement is called Parent Statement.
Nested Subquery:
If a Subquery contains another subquery, then the subquery inside another subquery is called nested subquery.
Correlated Subquery:
If the outcome of a subquery is depends on the value of a column of its parent query table then the Sub query is called Correlated Subquery.
QUESTIONS:
Create Items,Orders,Customers,Delivery tables and populate them with appropriate data.



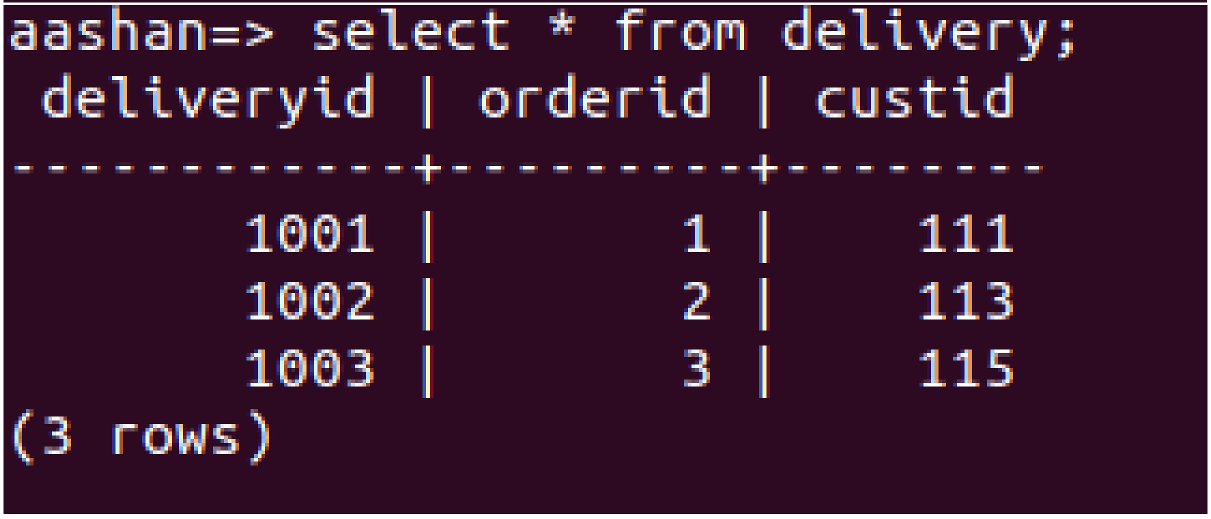
```
aashan=> select * from items;
 itemid |      itemname       |  category   | price | instock
--------+---------------------+-------------+-------+---------
      1 | Xiomi               | electronics |  1001 |       6
      2 | Iphone X            | electronics |  7007 |       6
      3 | One Plus 7          | electronics |  6006 |       2
      4 | Samsung Galaxy S4   | electronics |  5005 |       1
      5 | sony z5 premium     | electronics |  5005 |       1
(5 rows)
```



```
aashan=> select * from customers;
 custid | custname |     address     |   state
--------+----------+-----------------+------------
    112 | patrick  | harinagar       | tamilnadu
    115 | mickey   | juhu            | maharastra
    113 | soman    | puthumana       | kerala
    111 | elvin    | 202 jai street  | delhi
    114 | jaise    | kottarakara     | kerala
(5 rows)
```

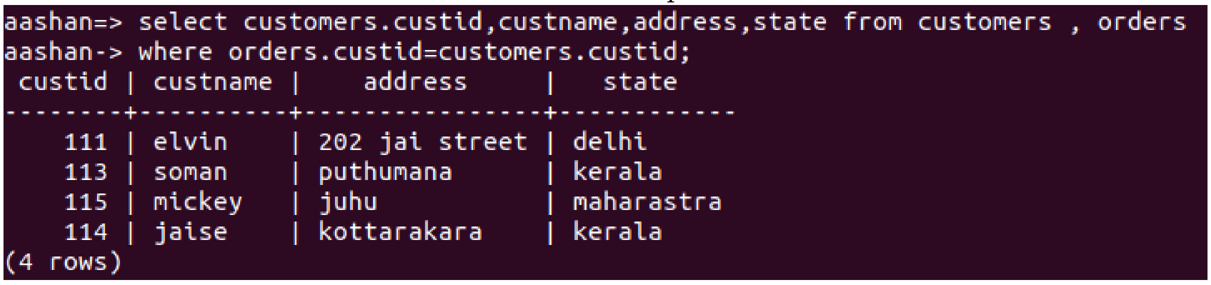


```
aashan=> select * from orders;
 orderid | itemid | quantity | orderdate  | custid
---------+--------+----------+------------+--------
       1 |      1 |        2 | 2014-10-11 |    111
       2 |      3 |        1 | 2012-01-29 |    113
       3 |      5 |        1 | 2013-05-13 |    115
       4 |      4 |        3 | 2014-12-22 |    114
(4 rows)
```



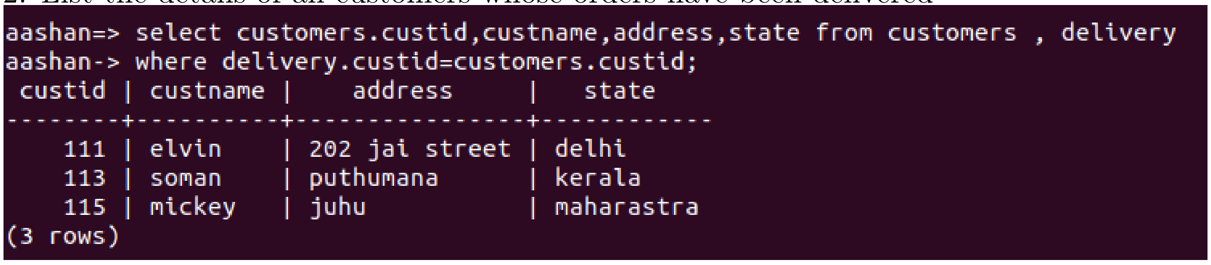
```
aashan=> select * from delivery;
 deliveryid | orderid | custid
------------+---------+--------
       1001 |       1 |    111
       1002 |       2 |    113
       1003 |       3 |    115
(3 rows)
```

1. List the details of all customers who have placed an order



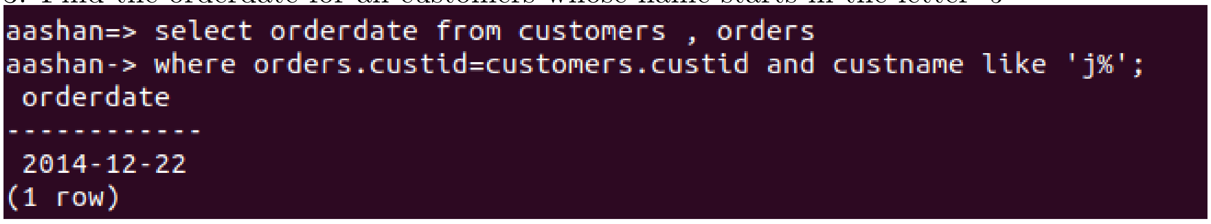
```
aashan=> select customers.custid,custname,address,state from customers , orders
aashan-> where orders.custid=customers.custid;
 custid | custname |    address     |   state
--------+----------+----------------+-----------
    111 | elvin    | 202 jai street | delhi
    113 | soman    | puthumana      | kerala
    115 | mickey   | juhu           | maharastra
    114 | jaise    | kottarakara    | kerala
(4 rows)
```

2. List the details of all customers whose orders have been delivered



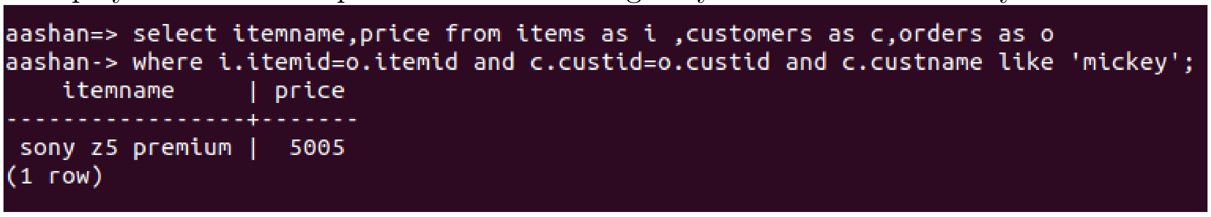
```
aashan=> select customers.custid,custname,address,state from customers , delivery
aashan-> where delivery.custid=customers.custid;
 custid | custname |    address     |   state
--------+----------+----------------+-----------
    111 | elvin    | 202 jai street | delhi
    113 | soman    | puthumana      | kerala
    115 | mickey   | juhu           | maharastra
(3 rows)
```

3. Find the orderdate for all customers whose name starts in the letter ‘J’



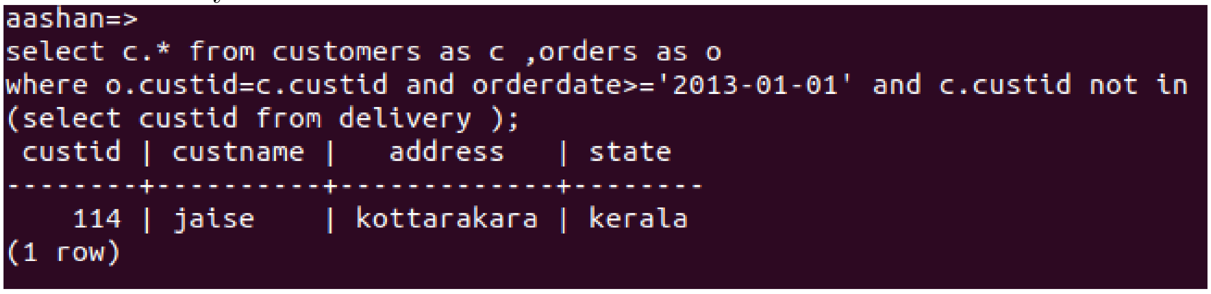
```
aashan=> select orderdate from customers , orders
aashan-> where orders.custid=customers.custid and custname like 'j%';
 orderdate
------------
 2014-12-22
(1 row)
```

4.Display the name and price of all items bought by the customer ‘Mickey’



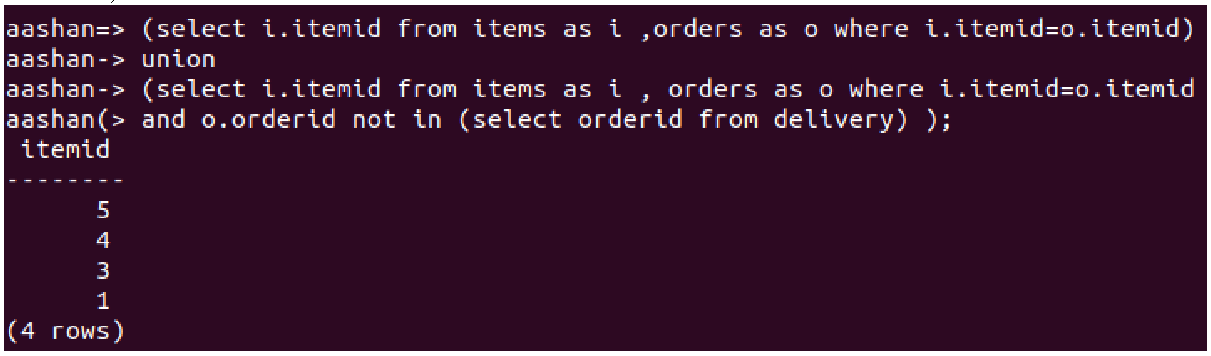
```
aashan=> select itemname,price from items as i ,customers as c,orders as o
aashan-> where i.itemid=o.itemid and c.custid=o.custid and c.custname like 'mickey';
    itemname     | price
-----------------+-------
 sony z5 premium |  5005
(1 row)
```

5. List the details of all customers who have placed an order after January 2013 and not received delivery of items.



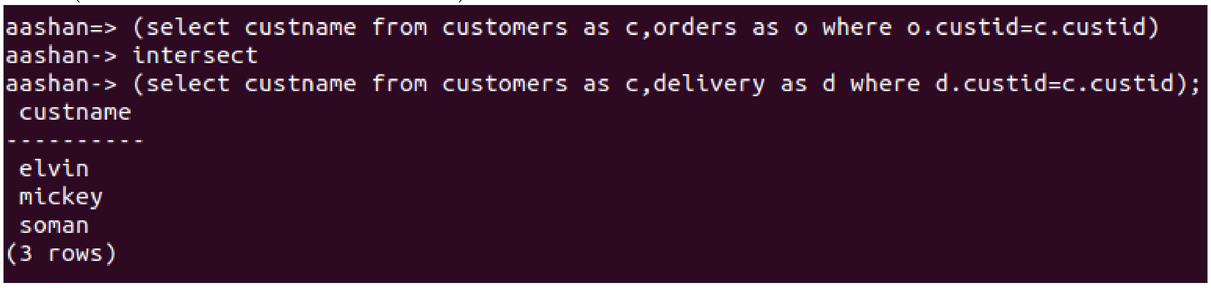
```
aashan=>
select c.* from customers as c ,orders as o
where o.custid=c.custid and orderdate>='2013-01-01' and c.custid not in
(select custid from delivery );
 custid | custname |   address   | state
--------+----------+-------------+--------
    114 | jaise    | kottarakara | kerala
(1 row)
```

6.Find the itemid of items which has either been ordered or not delivered. (Use SET UNION)



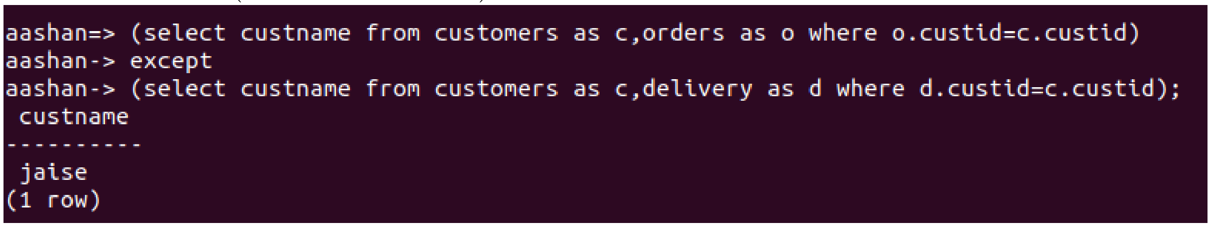
```
aashan=> (select i.itemid from items as i ,orders as o where i.itemid=o.itemid)
aashan-> union
aashan-> (select i.itemid from items as i , orders as o where i.itemid=o.itemid
aashan(> and o.orderid not in (select orderid from delivery) );
 itemid
--------
      5
      4
      3
      1
(4 rows)
```

7.Find the name of all customers who have placed an order and have their orders delivered.(Use SET INTERSECTION)



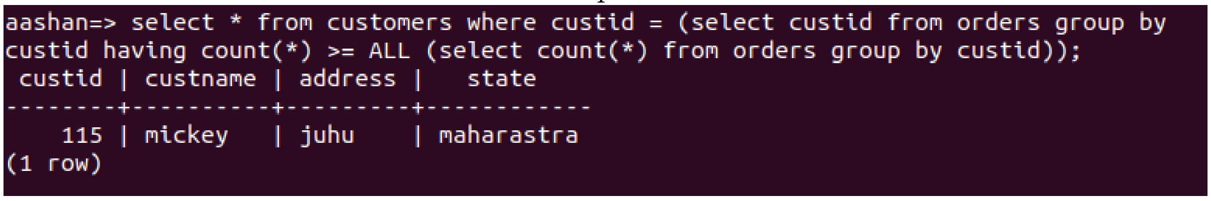
```
aashan=> (select custname from customers as c,orders as o where o.custid=c.custid)
aashan-> intersect
aashan-> (select custname from customers as c,delivery as d where d.custid=c.custid);
 custname
----------
 elvin
 mickey
 soman
(3 rows)
```

8.Find the custname of all customers who have placed an order but not having their ordersdelivered. (Use SET MINUS).



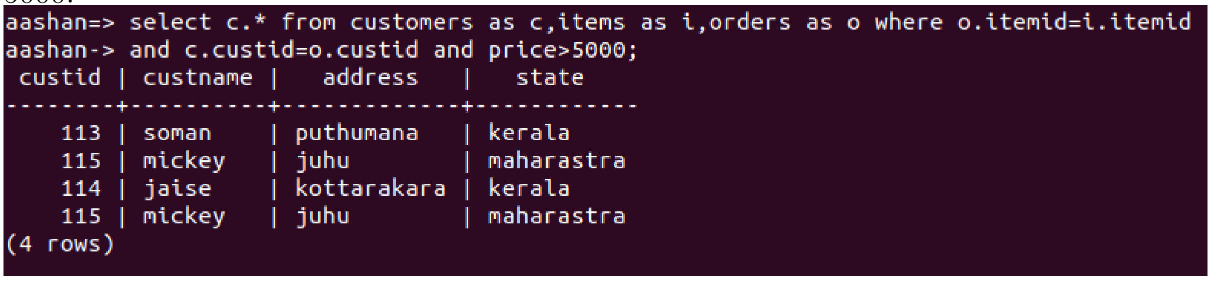
```
aashan=> (select custname from customers as c,orders as o where o.custid=c.custid)
aashan-> except
aashan-> (select custname from customers as c,delivery as d where d.custid=c.custid);
 custname
----------
 jaise
(1 row)
```

9. Find the name of the customer who has placed the most number of orders.



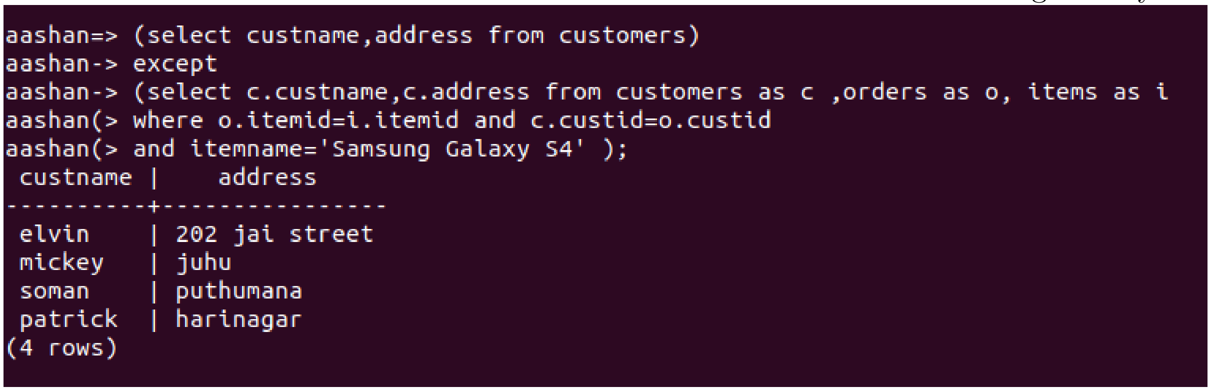
```
aashan=> select * from customers where custid = (select custid from orders group by
custid having count(*) >= ALL (select count(*) from orders group by custid));
 custid | custname | address |   state
--------+----------+---------+-----------
    115 | mickey   | juhu    | maharastra
(1 row)
```

10. Find the details of all customers who have purchased items exceeding a price of 5000.



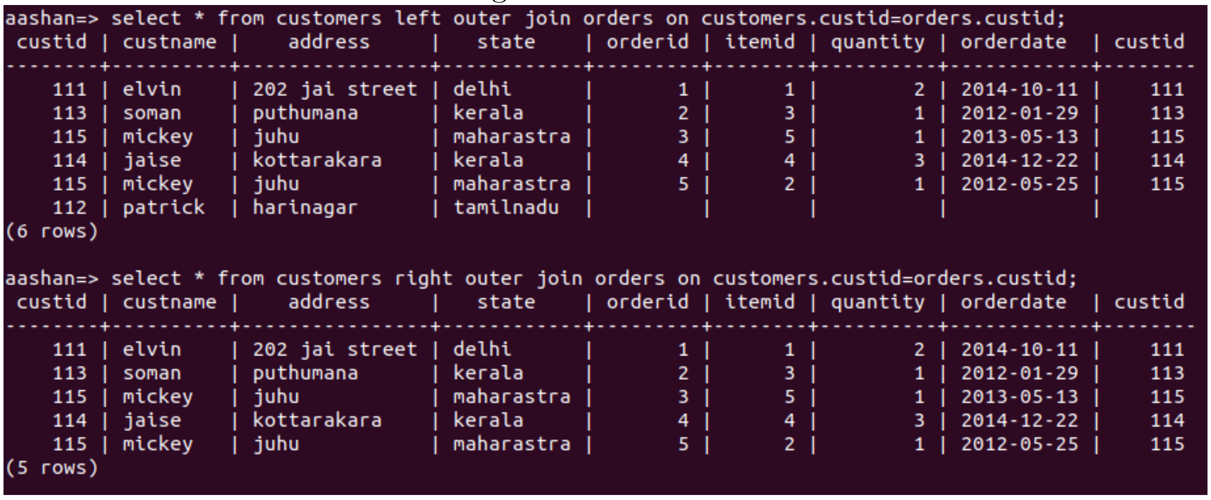
```
aashan=> select c.* from customers as c,items as i,orders as o where o.itemid=i.itemid
aashan-> and c.custid=o.custid and price>5000;
 custid | custname |   address   |   state
--------+----------+-------------+-----------
    113 | soman    | puthumana   | kerala
    115 | mickey   | juhu        | maharastra
    114 | jaise    | kottarakara | kerala
    115 | mickey   | juhu        | maharastra
(4 rows)
```

11.Find the name and address of customers who has not ordered a ‘Samsung Galaxy S4’



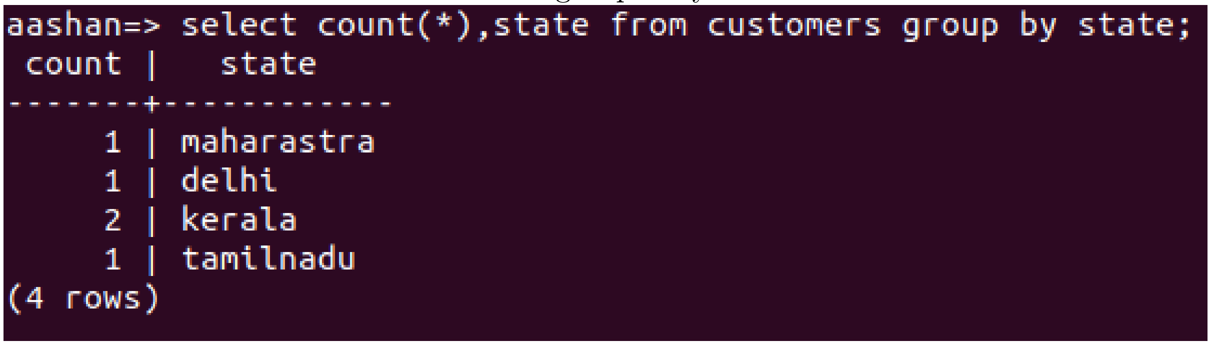
```
aashan=> (select custname,address from customers)
aashan-> except
aashan-> (select c.custname,c.address from customers as c ,orders as o, items as i
aashan(> where o.itemid=i.itemid and c.custid=o.custid
aashan(> and itemname='Samsung Galaxy S4' );
 custname |    address
----------+----------------
 elvin    | 202 jai street
 mickey   | juhu
 soman    | puthumana
 patrick  | harinagar
(4 rows)
```

12. Perform Left Outer Join and Right Outer Join on Customers and Orders Table.



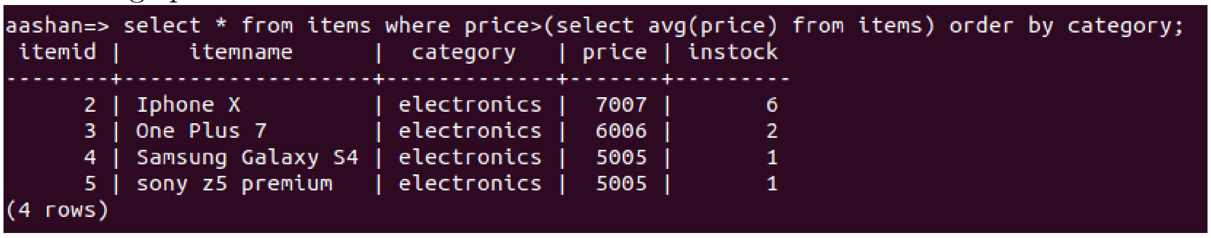
```
aashan=> select * from customers left outer join orders on customers.custid=orders.custid;
 custid | custname |    address     |   state    | orderid | itemid | quantity | orderdate  | custid
--------+----------+----------------+------------+---------+--------+----------+------------+--------
    111 | elvin    | 202 jai street | delhi      |       1 |      1 |        2 | 2014-10-11 |    111
    113 | soman    | puthumana      | kerala     |       2 |      3 |        1 | 2012-01-29 |    113
    115 | mickey   | juhu           | maharastra |       3 |      5 |        1 | 2013-05-13 |    115
    114 | jaise    | kottarakara    | kerala     |       4 |      4 |        3 | 2014-12-22 |    114
    115 | mickey   | juhu           | maharastra |       5 |      2 |        1 | 2012-05-25 |    115
    112 | patrick  | harinagar      | tamilnadu  |         |        |          |            |
(6 rows)

aashan=> select * from customers right outer join orders on customers.custid=orders.custid;
 custid | custname |    address     |   state    | orderid | itemid | quantity | orderdate  | custid
--------+----------+----------------+------------+---------+--------+----------+------------+--------
    111 | elvin    | 202 jai street | delhi      |       1 |      1 |        2 | 2014-10-11 |    111
    113 | soman    | puthumana      | kerala     |       2 |      3 |        1 | 2012-01-29 |    113
    115 | mickey   | juhu           | maharastra |       3 |      5 |        1 | 2013-05-13 |    115
    114 | jaise    | kottarakara    | kerala     |       4 |      4 |        3 | 2014-12-22 |    114
    115 | mickey   | juhu           | maharastra |       5 |      2 |        1 | 2012-05-25 |    115
(5 rows)
```

13. Find the details of all customers grouped by state.



```
aashan=> select count(*),state from customers group by state;
 count |   state
-------+------------
     1 | maharastra
     1 | delhi
     2 | kerala
     1 | tamilnadu
(4 rows)
```

14.Display the details of all items grouped by category and having a price greater than the average price of all items.



```
aashan=> select * from items where price>(select avg(price) from items) order by category;
 itemid |     itemname      |  category   | price | instock
--------+-------------------+-------------+-------+---------
      2 | Iphone X          | electronics |  7007 |       6
      3 | One Plus 7        | electronics |  6006 |       2
      4 | Samsung Galaxy S4 | electronics |  5005 |       1
      5 | sony z5 premium   | electronics |  5005 |       1
(4 rows)
```

RESULT:

Implemented the program for join, set, nested queries and group by clauses using Postgresql version: 11.4 ,Ubuntu 18.04 and following output were obtained.

Experiment 8

PL/SQL AND SEQUENCE

AIM:

To understand the usage of PL/SQL statements and sequences

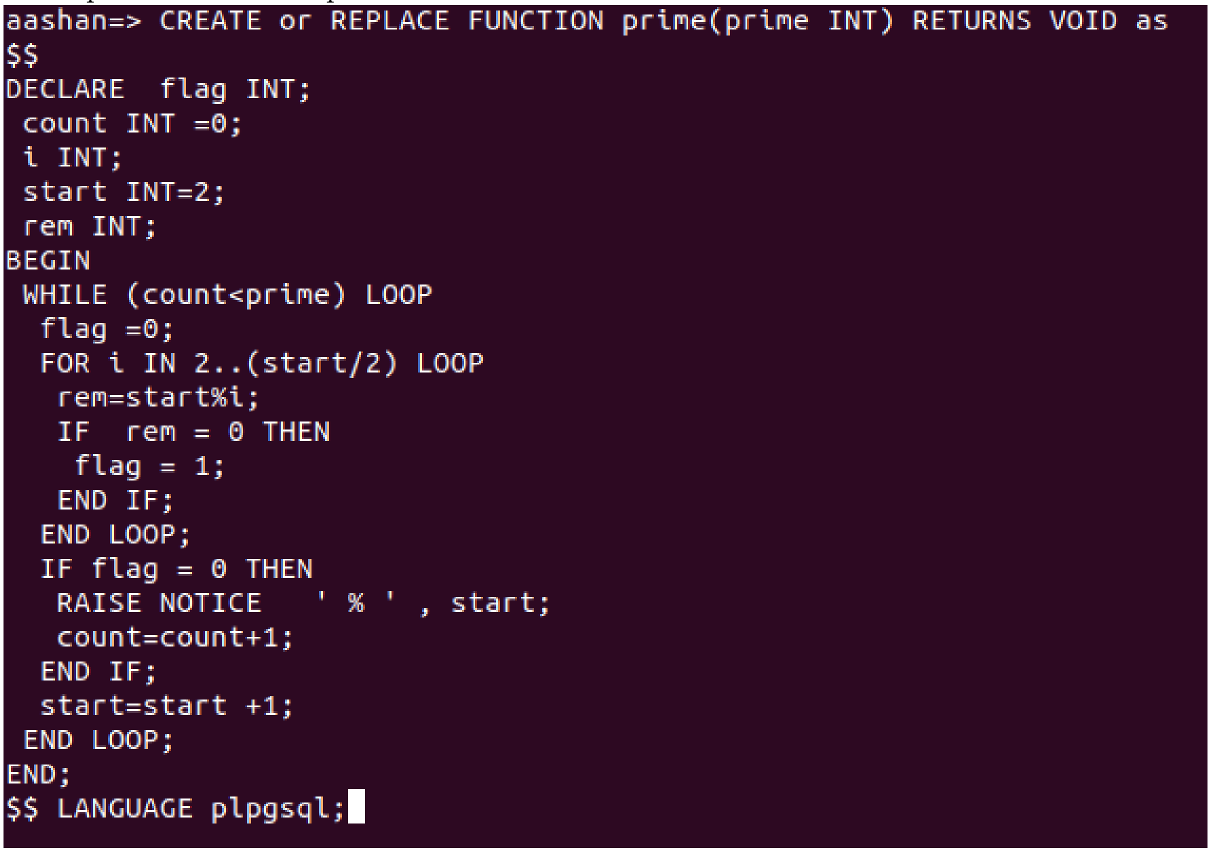
THEORY:

PL/SQL is a block-structured language. That is, the basic units (procedures,functions, and anonymous blocks) that make up a PL/SQL program are logical blocks, which can contain any number of nested sub-blocks. Typically, each logical block corresponds to a problem or subproblem to be solved. Thus, PL/SQL supports the divide-and-conquer approach to problem solving called stepwise refinement.

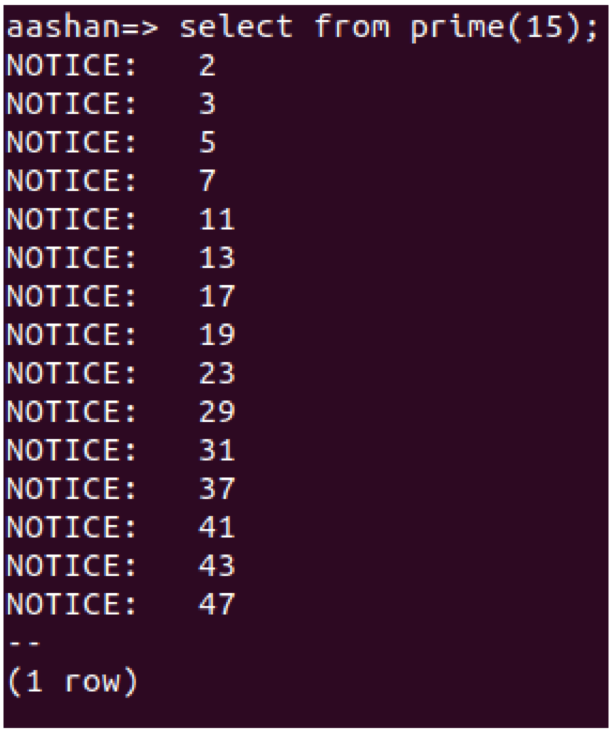
Sequence: The sequence generator generates sequential numbers. Sequence number generation is useful to generate unique primary keys for your data automatically, and to coordinate keys across multiple rows or tables.

QUESTIONS:

1. To print the first ‘n’ prime numbers.

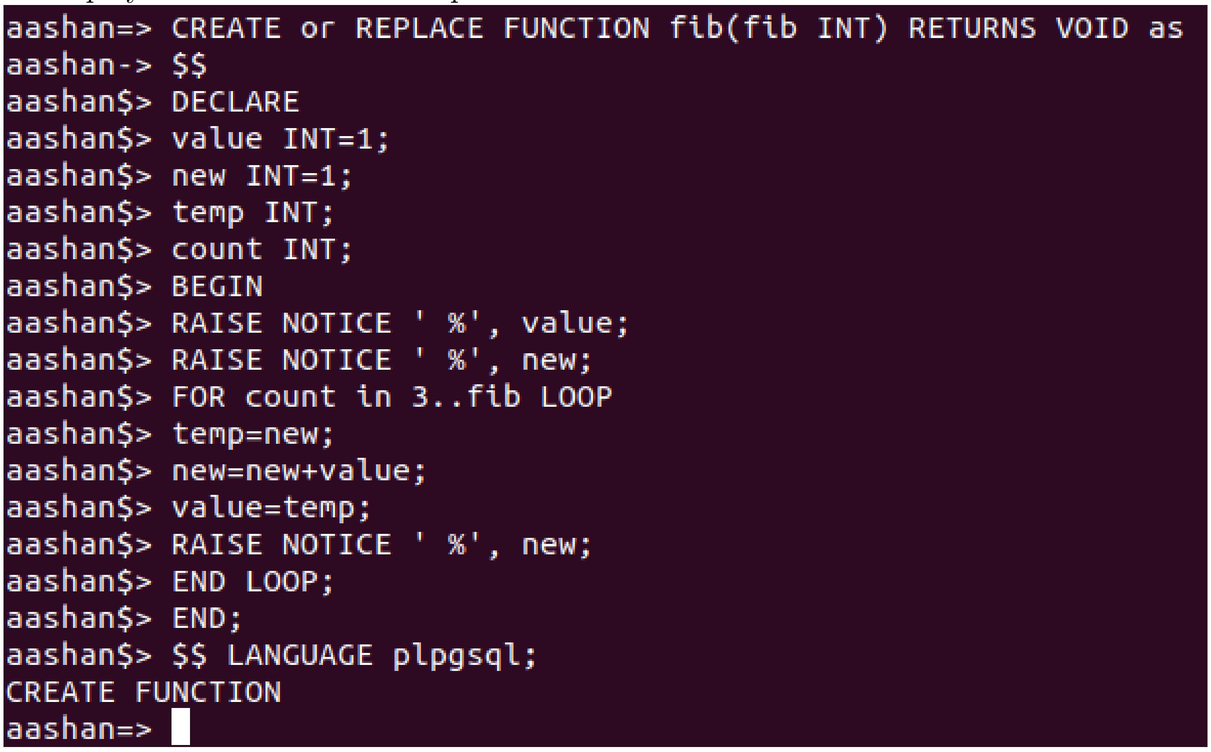


```
aashan=> CREATE or REPLACE FUNCTION prime(prime INT) RETURNS VOID as
$$
DECLARE  flag INT;
 count INT =0;
 i INT;
 start INT=2;
 rem INT;
BEGIN
 WHILE (count<prime) LOOP
  flag =0;
  FOR i IN 2..(start/2) LOOP
   rem=start%i;
   IF  rem = 0 THEN
    flag = 1;
   END IF;
  END LOOP;
  IF flag = 0 THEN
   RAISE NOTICE   ' % ' , start;
   count=count+1;
  END IF;
  start=start +1;
 END LOOP;
END;
$$ LANGUAGE plpgsql;
```

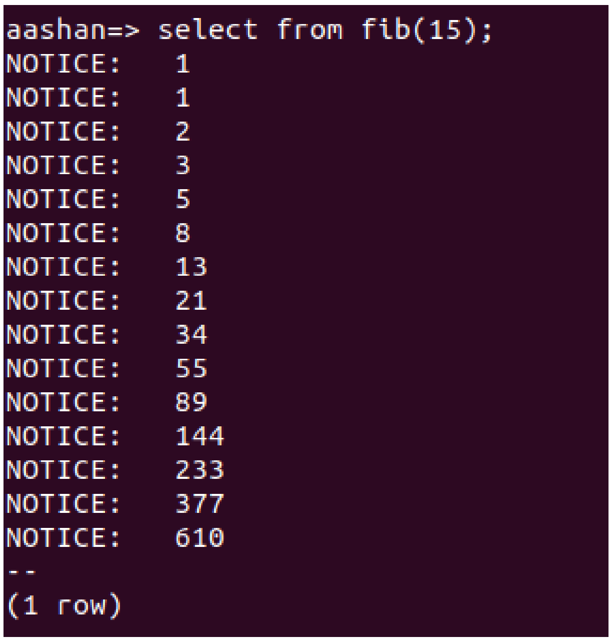


```
aashan=> select from prime(15);
NOTICE:   2
NOTICE:   3
NOTICE:   5
NOTICE:   7
NOTICE:   11
NOTICE:   13
NOTICE:   17
NOTICE:   19
NOTICE:   23
NOTICE:   29
NOTICE:   31
NOTICE:   37
NOTICE:   41
NOTICE:   43
NOTICE:   47
--
(1 row)
```

2. Display the Fibonacci series upto 'n' terms.

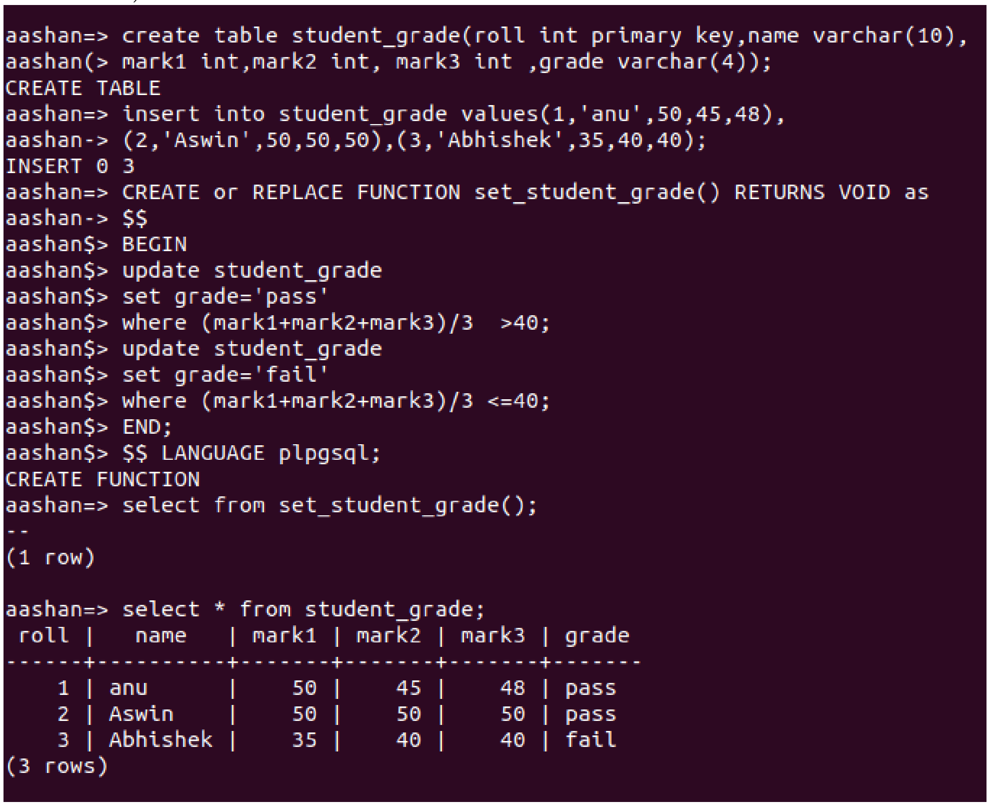


```
aashan=> CREATE or REPLACE FUNCTION fib(fib INT) RETURNS VOID as
aashan-> $$
aashan$> DECLARE
aashan$> value INT=1;
aashan$> new INT=1;
aashan$> temp INT;
aashan$> count INT;
aashan$> BEGIN
aashan$> RAISE NOTICE ' %', value;
aashan$> RAISE NOTICE ' %', new;
aashan$> FOR count in 3..fib LOOP
aashan$> temp=new;
aashan$> new=new+value;
aashan$> value=temp;
aashan$> RAISE NOTICE ' %', new;
aashan$> END LOOP;
aashan$> END;
aashan$> $$ LANGUAGE plpgsql;
CREATE FUNCTION
aashan=>
```



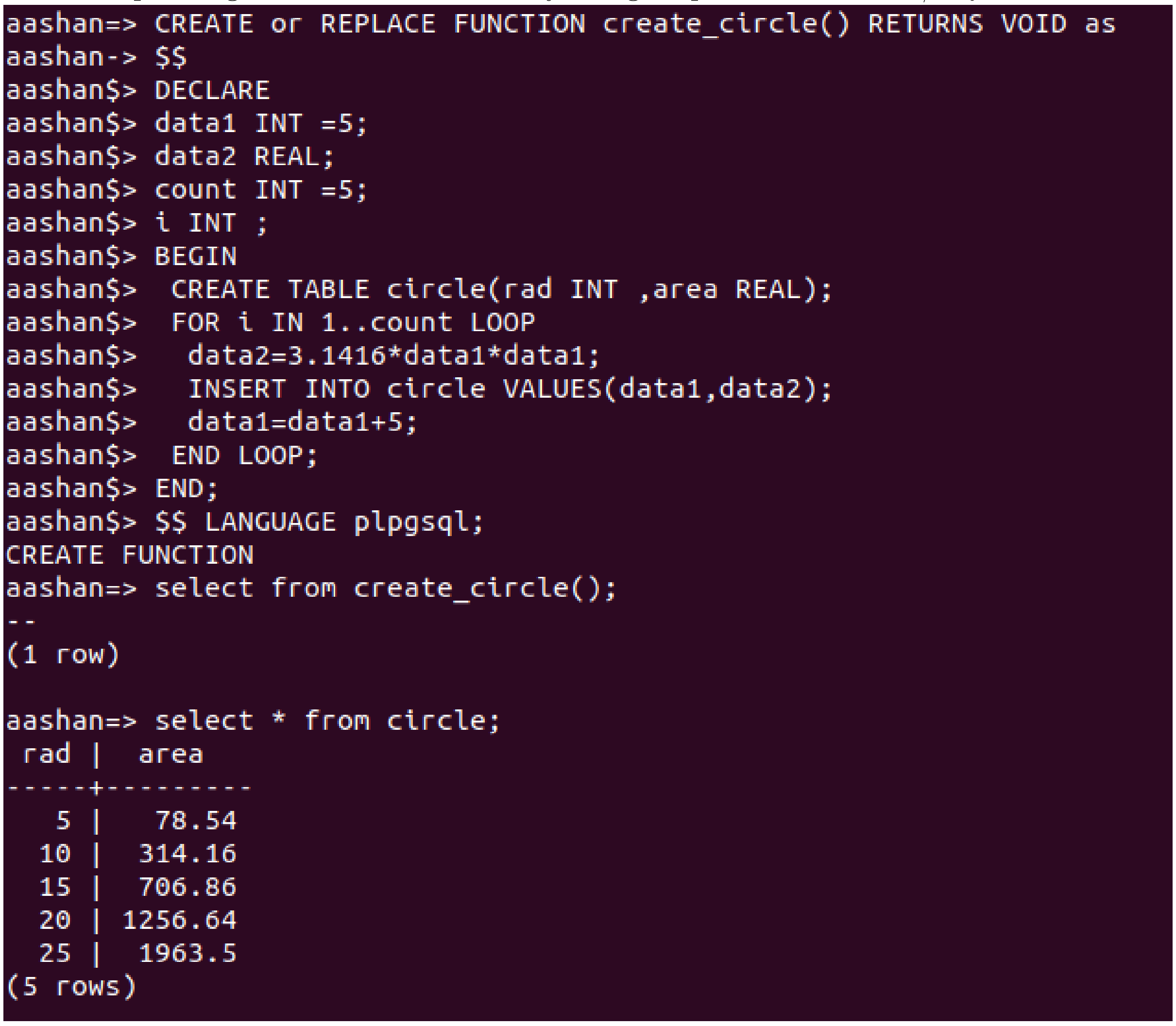
```
aashan=> select from fib(15);
NOTICE:   1
NOTICE:   1
NOTICE:   2
NOTICE:   3
NOTICE:   5
NOTICE:   8
NOTICE:   13
NOTICE:   21
NOTICE:   34
NOTICE:   55
NOTICE:   89
NOTICE:   144
NOTICE:   233
NOTICE:   377
NOTICE:   610
--
(1 row)
```

3. Create a table named studentgrade with the given attributes:
roll, name ,mark1,mark2,mark3, grade.
Read the roll, name and marks from the user. Calculate the grade of the student and insert a tuple into the table using PL/SQL. (Grade= ‘PASS’ if AVG >40, Grade=’FAIL’ otherwise)



```
aashan=> create table student_grade(roll int primary key,name varchar(10),
aashan(> mark1 int,mark2 int, mark3 int ,grade varchar(4));
CREATE TABLE
aashan=> insert into student_grade values(1,'anu',50,45,48),
aashan-> (2,'Aswin',50,50,50),(3,'Abhishek',35,40,40);
INSERT 0 3
aashan=> CREATE or REPLACE FUNCTION set_student_grade() RETURNS VOID as
aashan-> $$
aashan$> BEGIN
aashan$> update student_grade
aashan$> set grade='pass'
aashan$> where (mark1+mark2+mark3)/3  >40;
aashan$> update student_grade
aashan$> set grade='fail'
aashan$> where (mark1+mark2+mark3)/3 <=40;
aashan$> END;
aashan$> $$ LANGUAGE plpgsql;
CREATE FUNCTION
aashan=> select from set_student_grade();
--
(1 row)

aashan=> select * from student_grade;
 roll |   name   | mark1 | mark2 | mark3 | grade
------+----------+-------+-------+-------+-------
    1 | anu      |    50 |    45 |    48 | pass
    2 | Aswin    |    50 |    50 |    50 | pass
    3 | Abhishek |    35 |    40 |    40 | fail
(3 rows)
```

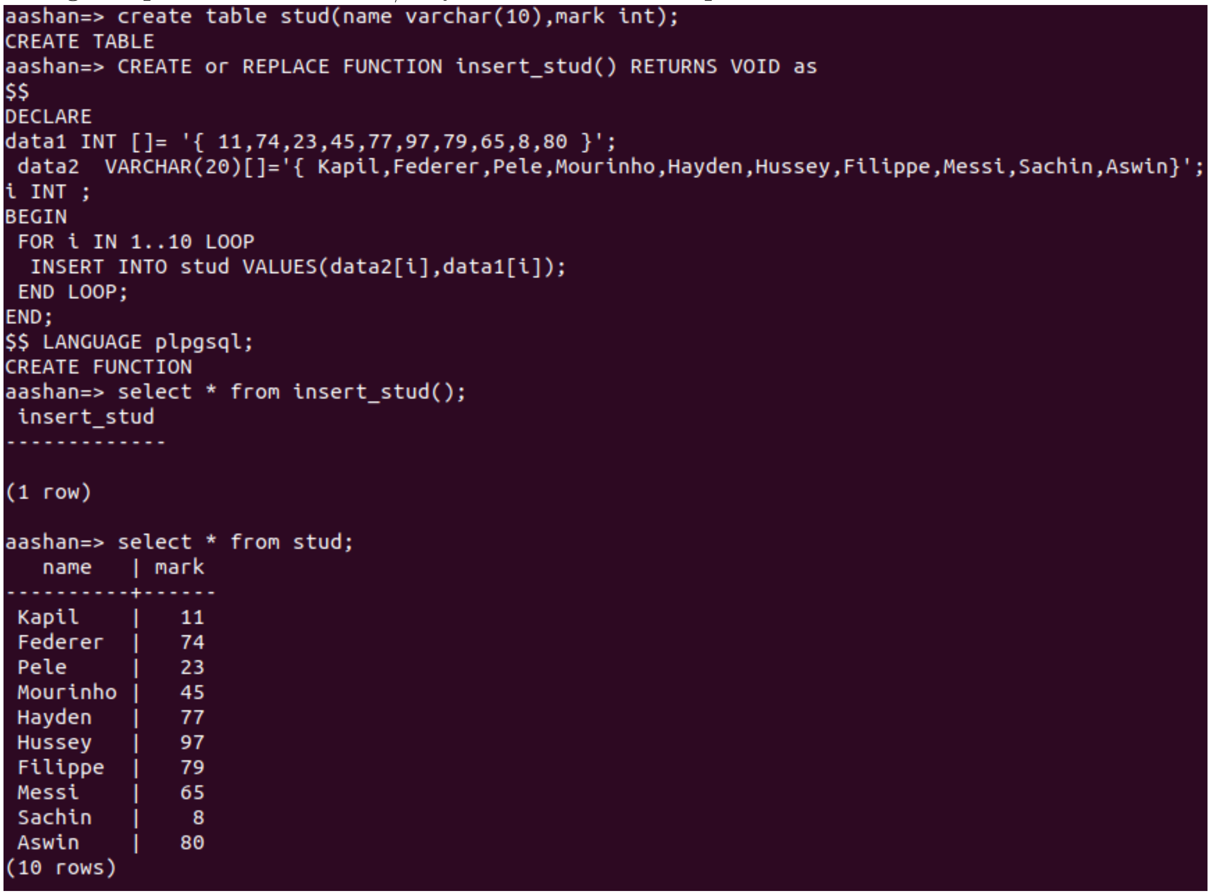
4. Create table circlearea (rad,area). For radius 5,10,15,20,25, find the area and insert the corresponding values into the table by using loop structure in PL/SQL.



```
aashan=> CREATE or REPLACE FUNCTION create_circle() RETURNS VOID as
aashan-> $$
aashan$> DECLARE
aashan$> data1 INT =5;
aashan$> data2 REAL;
aashan$> count INT =5;
aashan$> i INT ;
aashan$> BEGIN
aashan$>  CREATE TABLE circle(rad INT ,area REAL);
aashan$>  FOR i IN 1..count LOOP
aashan$>   data2=3.1416*data1*data1;
aashan$>   INSERT INTO circle VALUES(data1,data2);
aashan$>   data1=data1+5;
aashan$>  END LOOP;
aashan$> END;
aashan$> $$ LANGUAGE plpgsql;
CREATE FUNCTION
aashan=> select from create_circle();
--
(1 row)

aashan=> select * from circle;
 rad |  area
-----+---------
   5 |   78.54
  10 |  314.16
  15 |  706.86
  20 | 1256.64
  25 |  1963.5
(5 rows)
```

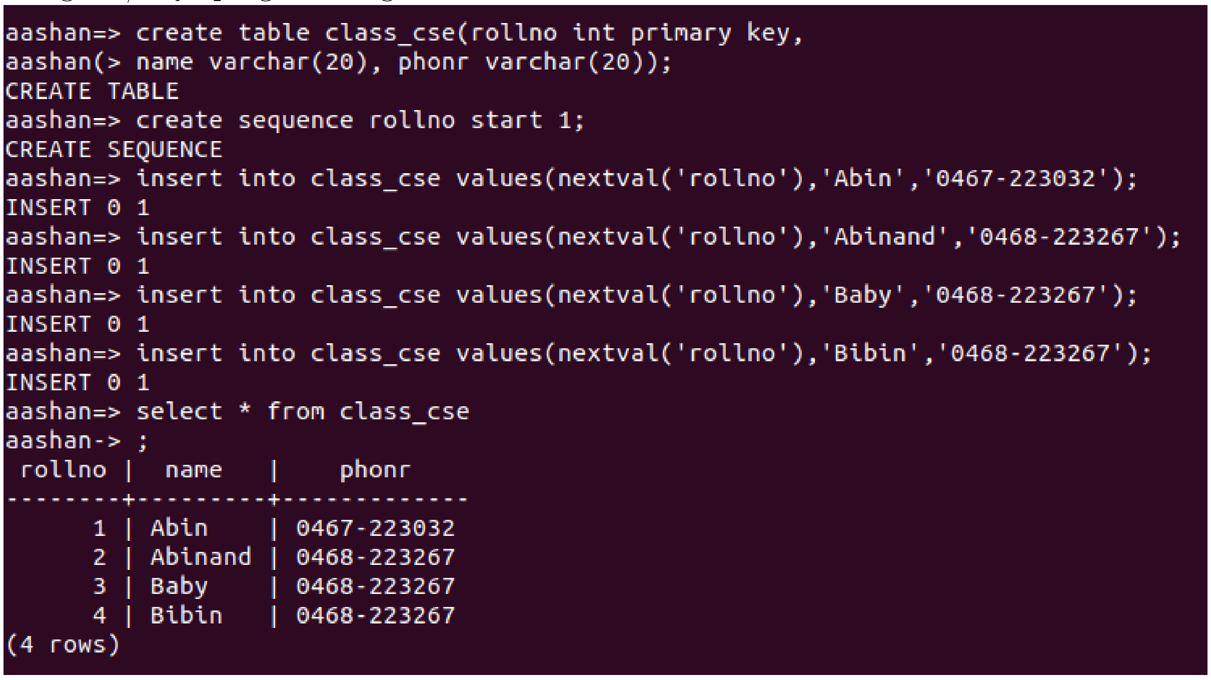
5. Use an array to store the names, marks of 10 students in a class.

Using Loop structures in PL/SQL insert the ten tuples to a table named stud



```
aashan=> create table stud(name varchar(10),mark int);
CREATE TABLE
aashan=> CREATE or REPLACE FUNCTION insert_stud() RETURNS VOID as
$$
DECLARE
data1 INT []= '{ 11,74,23,45,77,97,79,65,8,80 }';
 data2  VARCHAR(20)[]='{ Kapil,Federer,Pele,Mourinho,Hayden,Hussey,Filippe,Messi,Sachin,Aswin}';
i INT ;
BEGIN
 FOR i IN 1..10 LOOP
  INSERT INTO stud VALUES(data2[i],data1[i]);
 END LOOP;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
aashan=> select * from insert_stud();
 insert_stud
-------------

(1 row)

aashan=> select * from stud;
   name   | mark
----------+------
 Kapil    |   11
 Federer  |   74
 Pele     |   23
 Mourinho |   45
 Hayden   |   77
 Hussey   |   97
 Filippe  |   79
 Messi    |   65
 Sachin   |    8
 Aswin    |   80
(10 rows)
```

6. Create a sequence using PL/SQL. Use this sequence to generate the primary key values for a table named classcse with attributes roll,name and phone. Insert some tuples using PL/SQL programming.



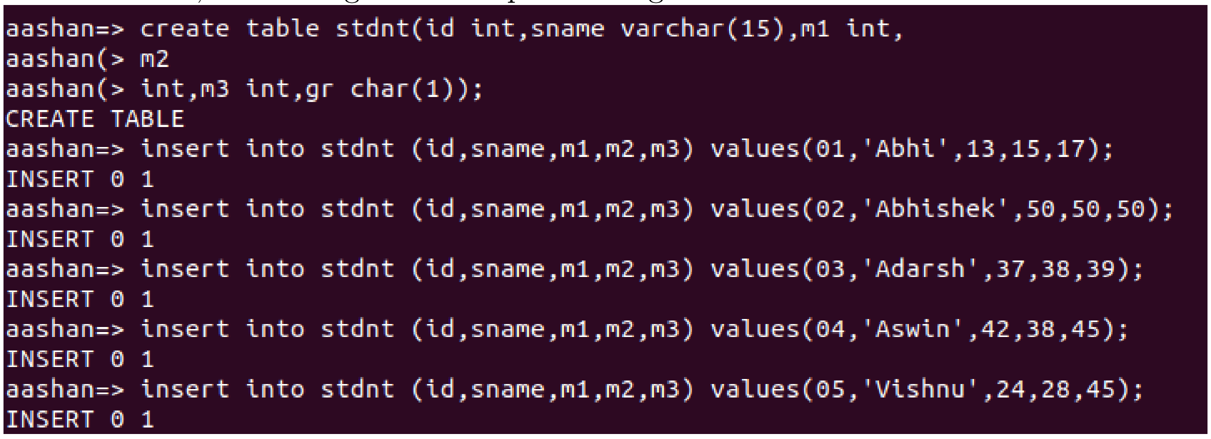
```
aashan=> create table class_cse(rollno int primary key,
aashan(> name varchar(20), phonr varchar(20));
CREATE TABLE
aashan=> create sequence rollno start 1;
CREATE SEQUENCE
aashan=> insert into class_cse values(nextval('rollno'),'Abin','0467-223032');
INSERT 0 1
aashan=> insert into class_cse values(nextval('rollno'),'Abinand','0468-223267');
INSERT 0 1
aashan=> insert into class_cse values(nextval('rollno'),'Baby','0468-223267');
INSERT 0 1
aashan=> insert into class_cse values(nextval('rollno'),'Bibin','0468-223267');
INSERT 0 1
aashan=> select * from class_cse
aashan-> ;
 rollno |  name   |    phonr
--------+---------+-------------
      1 | Abin    | 0467-223032
      2 | Abinand | 0468-223267
      3 | Baby    | 0468-223267
      4 | Bibin   | 0468-223267
(4 rows)
```

RESULT:
Implemented the program for PL/SQL and sequence using Postgresql version: 11.4 ,Ubuntu 18.04 and following output were obtained.

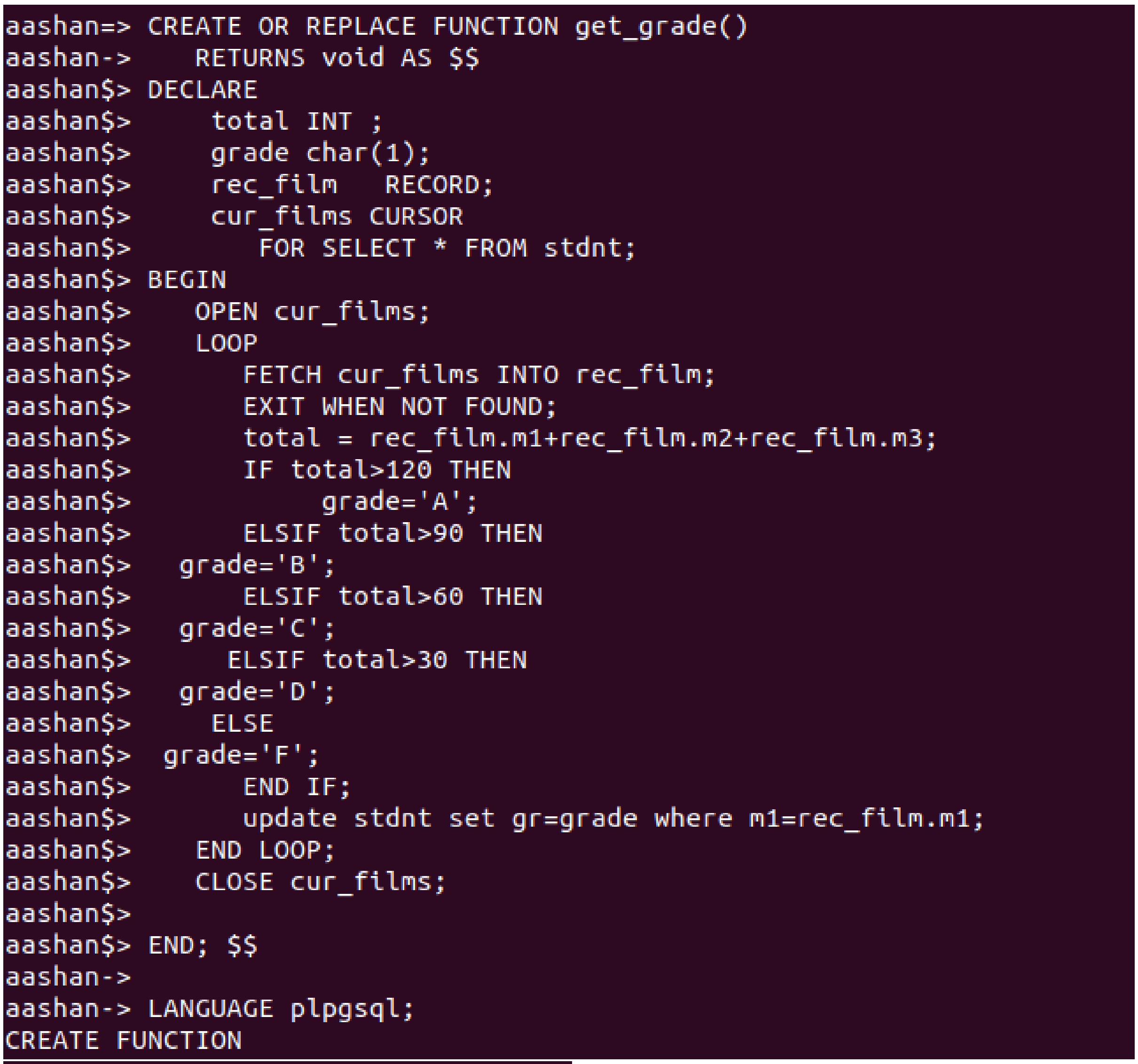
Experiment 9

CURSOR

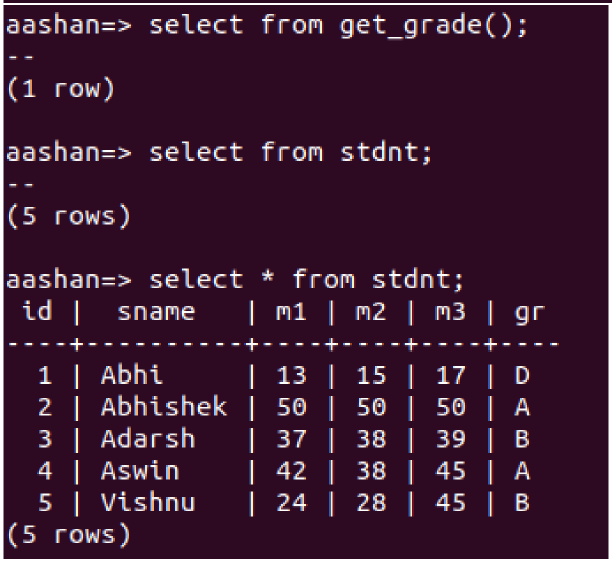
AIM:
To understand the usage of Cursor.
THEORY:
A PL/SQL construct called a cursor lets you name a work area and access its stored information. There are two kinds of cursors: implicit and explicit. PL/SQL implicitly declares a cursor for all SQL data manipulation statements, including queries that return only one row. For queries that return more than one row, you can explicitly declare a cursor to process the rows individually.
QUESTIONS:
1. Create table student (id, name, m1, m2, m3, grade).
Insert 5 tuples into it.
Find the total, calculate grade and update the grade in the table.



```
aashan=> create table stdnt(id int,sname varchar(15),m1 int,
aashan(> m2
aashan(> int,m3 int,gr char(1));
CREATE TABLE
aashan=> insert into stdnt (id,sname,m1,m2,m3) values(01,'Abhi',13,15,17);
INSERT 0 1
aashan=> insert into stdnt (id,sname,m1,m2,m3) values(02,'Abhishek',50,50,50);
INSERT 0 1
aashan=> insert into stdnt (id,sname,m1,m2,m3) values(03,'Adarsh',37,38,39);
INSERT 0 1
aashan=> insert into stdnt (id,sname,m1,m2,m3) values(04,'Aswin',42,38,45);
INSERT 0 1
aashan=> insert into stdnt (id,sname,m1,m2,m3) values(05,'Vishnu',24,28,45);
INSERT 0 1
```

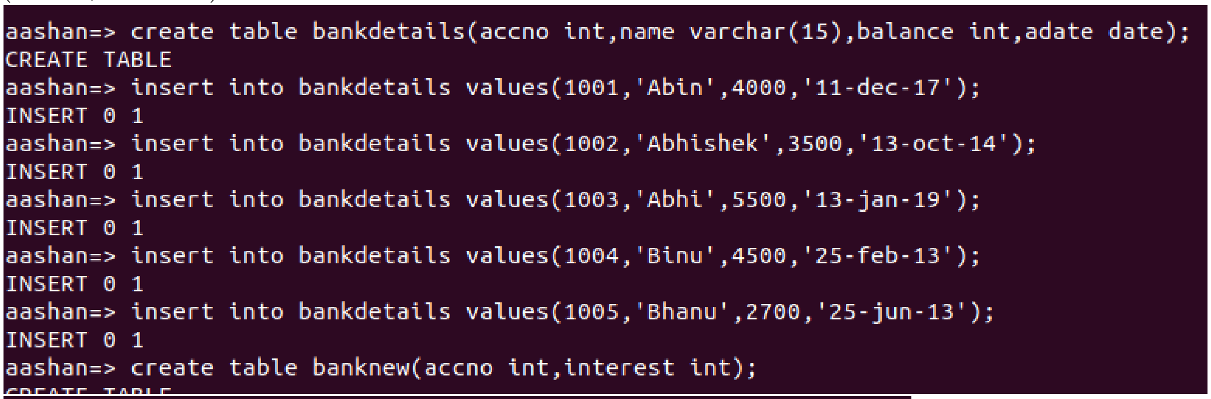


```
aashan=> CREATE OR REPLACE FUNCTION get_grade()
aashan->    RETURNS void AS $$
aashan$> DECLARE
aashan$>     total INT ;
aashan$>     grade char(1);
aashan$>     rec_film   RECORD;
aashan$>     cur_films CURSOR
aashan$>        FOR SELECT * FROM stdnt;
aashan$> BEGIN
aashan$>    OPEN cur_films;
aashan$>    LOOP
aashan$>       FETCH cur_films INTO rec_film;
aashan$>       EXIT WHEN NOT FOUND;
aashan$>       total = rec_film.m1+rec_film.m2+rec_film.m3;
aashan$>       IF total>120 THEN
aashan$>            grade='A';
aashan$>       ELSIF total>90 THEN
aashan$>   grade='B';
aashan$>       ELSIF total>60 THEN
aashan$>   grade='C';
aashan$>      ELSIF total>30 THEN
aashan$>   grade='D';
aashan$>     ELSE
aashan$>  grade='F';
aashan$>       END IF;
aashan$>       update stdnt set gr=grade where m1=rec_film.m1;
aashan$>    END LOOP;
aashan$>    CLOSE cur_films;
aashan$>
aashan$> END; $$
aashan->
aashan-> LANGUAGE plpgsql;
CREATE FUNCTION
```

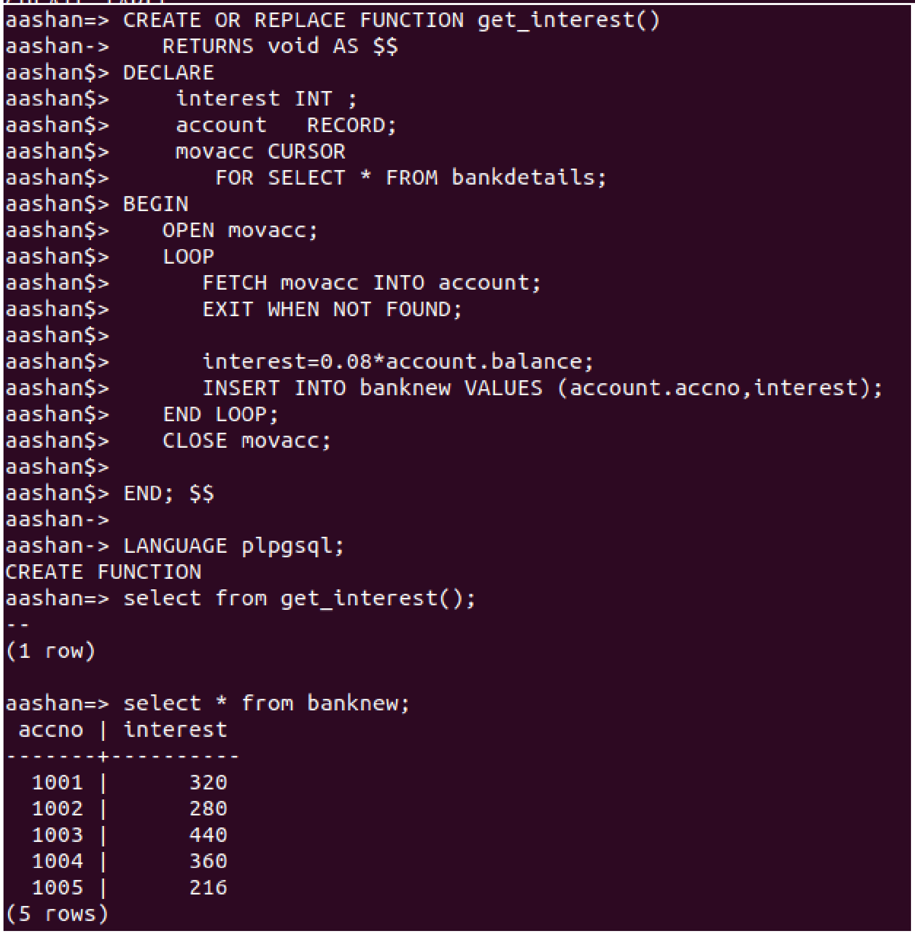


```
aashan=> select from get_grade();
--
(1 row)

aashan=> select from stdnt;
--
(5 rows)

aashan=> select * from stdnt;
 id |   sname   | m1 | m2 | m3 | gr
----+----------+----+----+----+----
  1 | Abhi      | 13 | 15 | 17 | D
  2 | Abhishek | 50 | 50 | 50 | A
  3 | Adarsh    | 37 | 38 | 39 | B
  4 | Aswin     | 42 | 38 | 45 | A
  5 | Vishnu    | 24 | 28 | 45 | B
(5 rows)
```

2. Create bankdetails (accno, name, balance, adate).
Calculate the interest of the amount and insert into a new table with fields.
(accno, interest). Interest= 0.08*balance.

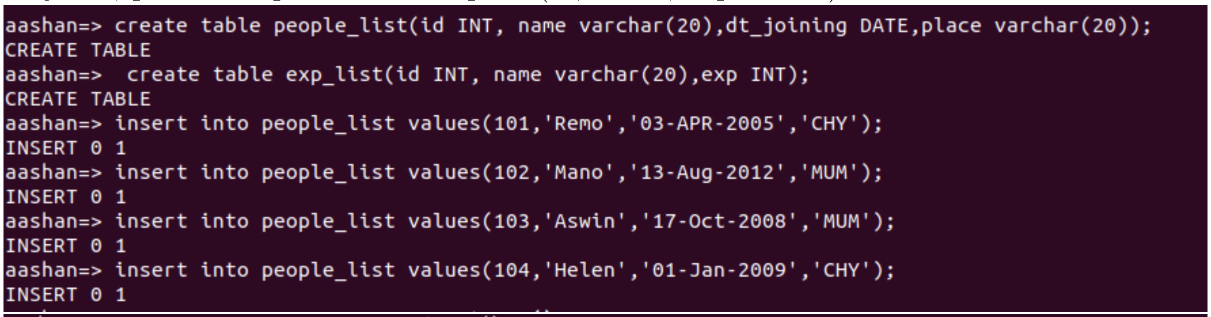


```
aashan=> create table bankdetails(accno int,name varchar(15),balance int,adate date);
CREATE TABLE
aashan=> insert into bankdetails values(1001,'Abin',4000,'11-dec-17');
INSERT 0 1
aashan=> insert into bankdetails values(1002,'Abhishek',3500,'13-oct-14');
INSERT 0 1
aashan=> insert into bankdetails values(1003,'Abhi',5500,'13-jan-19');
INSERT 0 1
aashan=> insert into bankdetails values(1004,'Binu',4500,'25-feb-13');
INSERT 0 1
aashan=> insert into bankdetails values(1005,'Bhanu',2700,'25-jun-13');
INSERT 0 1
aashan=> create table banknew(accno int,interest int);
```

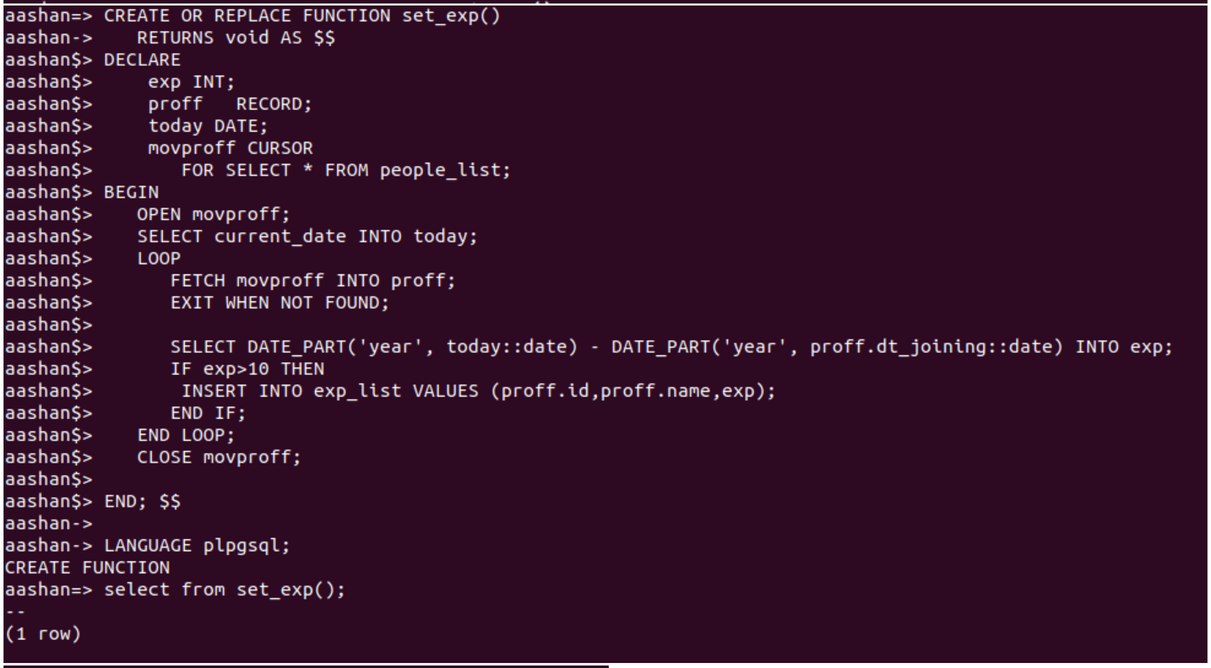


```
aashan=> CREATE OR REPLACE FUNCTION get_interest()
aashan->    RETURNS void AS $$
aashan$> DECLARE
aashan$>     interest INT ;
aashan$>     account   RECORD;
aashan$>     movacc CURSOR
aashan$>        FOR SELECT * FROM bankdetails;
aashan$> BEGIN
aashan$>    OPEN movacc;
aashan$>    LOOP
aashan$>       FETCH movacc INTO account;
aashan$>       EXIT WHEN NOT FOUND;
aashan$>
aashan$>       interest=0.08*account.balance;
aashan$>       INSERT INTO banknew VALUES (account.accno,interest);
aashan$>    END LOOP;
aashan$>    CLOSE movacc;
aashan$>
aashan$> END; $$
aashan->
aashan-> LANGUAGE plpgsql;
CREATE FUNCTION
aashan=> select from get_interest();
--
(1 row)

aashan=> select * from banknew;
 accno | interest
-------+----------
  1001 |      320
  1002 |      280
  1003 |      440
  1004 |      360
  1005 |      216
(5 rows)
```

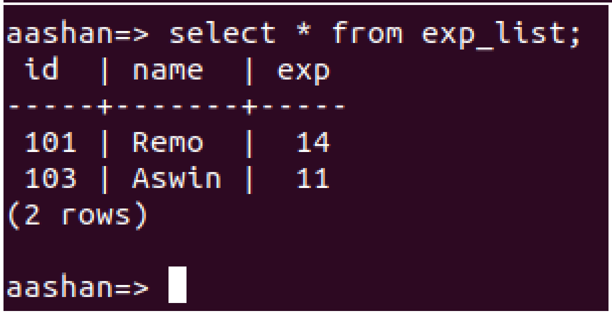
3. Create table peoplelist (id, name, dtjoining, place).If person's experience is above 10 years, put the tuple in table explist (id, name, experience).



```
aashan=> create table people_list(id INT, name varchar(20),dt_joining DATE,place varchar(20));
CREATE TABLE
aashan=>  create table exp_list(id INT, name varchar(20),exp INT);
CREATE TABLE
aashan=> insert into people_list values(101,'Remo','03-APR-2005','CHY');
INSERT 0 1
aashan=> insert into people_list values(102,'Mano','13-Aug-2012','MUM');
INSERT 0 1
aashan=> insert into people_list values(103,'Aswin','17-Oct-2008','MUM');
INSERT 0 1
aashan=> insert into people_list values(104,'Helen','01-Jan-2009','CHY');
INSERT 0 1
```



```
aashan=> CREATE OR REPLACE FUNCTION set_exp()
aashan->    RETURNS void AS $$
aashan$> DECLARE
aashan$>     exp INT;
aashan$>     proff   RECORD;
aashan$>     today DATE;
aashan$>     movproff CURSOR
aashan$>        FOR SELECT * FROM people_list;
aashan$> BEGIN
aashan$>    OPEN movproff;
aashan$>    SELECT current_date INTO today;
aashan$>    LOOP
aashan$>       FETCH movproff INTO proff;
aashan$>       EXIT WHEN NOT FOUND;
aashan$>
aashan$>       SELECT DATE_PART('year', today::date) - DATE_PART('year', proff.dt_joining::date) INTO exp;
aashan$>       IF exp>10 THEN
aashan$>        INSERT INTO exp_list VALUES (proff.id,proff.name,exp);
aashan$>       END IF;
aashan$>    END LOOP;
aashan$>    CLOSE movproff;
aashan$>
aashan$> END; $$
aashan->
aashan-> LANGUAGE plpgsql;
CREATE FUNCTION
aashan=> select from set_exp();
--
(1 row)
```



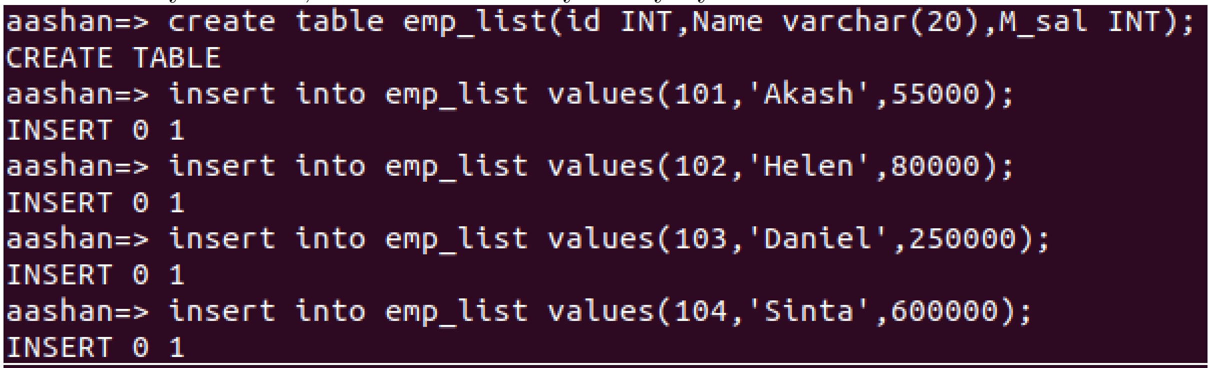
```
aashan=> select * from exp_list;
 id  | name  | exp
-----+-------+-----
 101 | Remo  |  14
 103 | Aswin |  11
(2 rows)

aashan=> 
```

4. Create table employeelist(id,name,monthly salary).
If:
annual salary<60000, increment monthly
salary by 25
between 60000 and

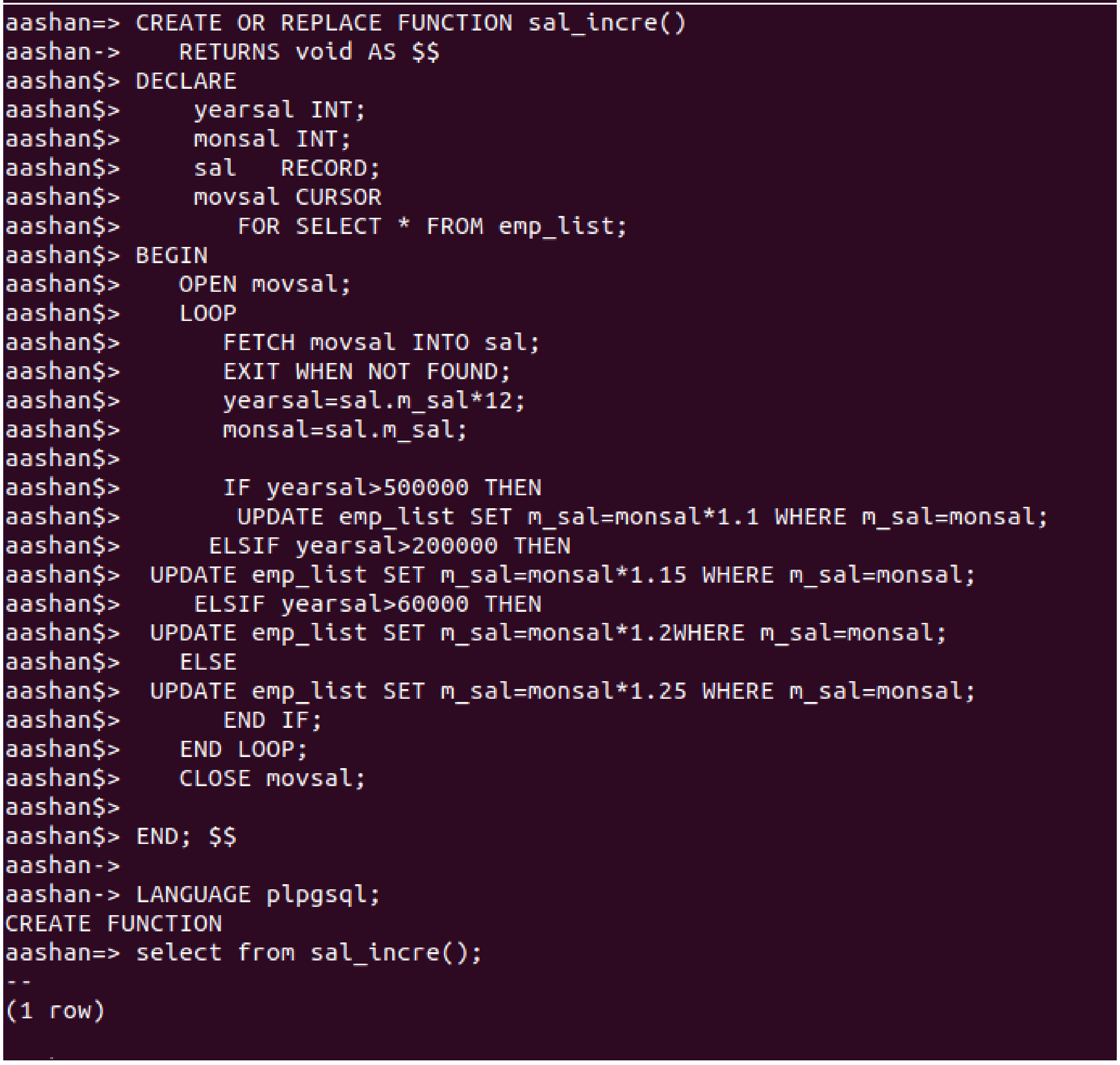
200000, increment by 20

between 200000 and 500000, increment by 15

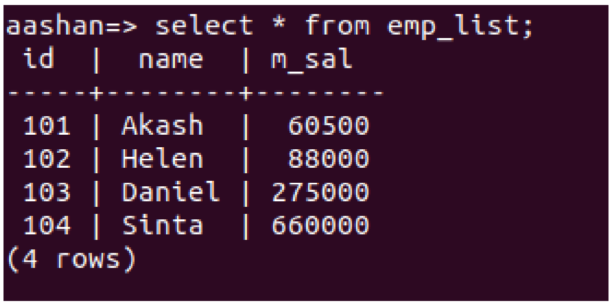
annual salary>500000, increment monthly salary by 10



```
aashan=> create table emp_list(id INT,Name varchar(20),M_sal INT);
CREATE TABLE
aashan=> insert into emp_list values(101,'Akash',55000);
INSERT 0 1
aashan=> insert into emp_list values(102,'Helen',80000);
INSERT 0 1
aashan=> insert into emp_list values(103,'Daniel',250000);
INSERT 0 1
aashan=> insert into emp_list values(104,'Sinta',600000);
INSERT 0 1
```



```
aashan=> CREATE OR REPLACE FUNCTION sal_incre()
aashan->    RETURNS void AS $$
aashan$> DECLARE
aashan$>     yearsal INT;
aashan$>     monsal INT;
aashan$>     sal   RECORD;
aashan$>     movsal CURSOR
aashan$>        FOR SELECT * FROM emp_list;
aashan$> BEGIN
aashan$>    OPEN movsal;
aashan$>    LOOP
aashan$>       FETCH movsal INTO sal;
aashan$>       EXIT WHEN NOT FOUND;
aashan$>       yearsal=sal.m_sal*12;
aashan$>       monsal=sal.m_sal;
aashan$>
aashan$>       IF yearsal>500000 THEN
aashan$>        UPDATE emp_list SET m_sal=monsal*1.1 WHERE m_sal=monsal;
aashan$>      ELSIF yearsal>200000 THEN
aashan$>  UPDATE emp_list SET m_sal=monsal*1.15 WHERE m_sal=monsal;
aashan$>     ELSIF yearsal>60000 THEN
aashan$>  UPDATE emp_list SET m_sal=monsal*1.2WHERE m_sal=monsal;
aashan$>    ELSE
aashan$>  UPDATE emp_list SET m_sal=monsal*1.25 WHERE m_sal=monsal;
aashan$>       END IF;
aashan$>    END LOOP;
aashan$>    CLOSE movsal;
aashan$>
aashan$> END; $$
aashan->
aashan-> LANGUAGE plpgsql;
CREATE FUNCTION
aashan=> select from sal_incre();
--
(1 row)
```



```
aashan=> select * from emp_list;
 id  |  name  | m_sal
-----+--------+--------
 101 | Akash  |  60500
 102 | Helen  |  88000
 103 | Daniel | 275000
 104 | Sinta  | 660000
(4 rows)
```

RESULT:

Implemented the program for cursor using Postgresql version: 11.4 ,Ubuntu 18.04 and following output were obtained.

Experiment 10

TRIGGER AND EXCEPTION HANDLING

AIM:

To understand the usage of triggers and exception handling

THEORY:

Triggers:

Triggers are procedures that are stored in the database and implicitly run, or fired, when something happens.

Exception:

It is used to handle run time errors in program

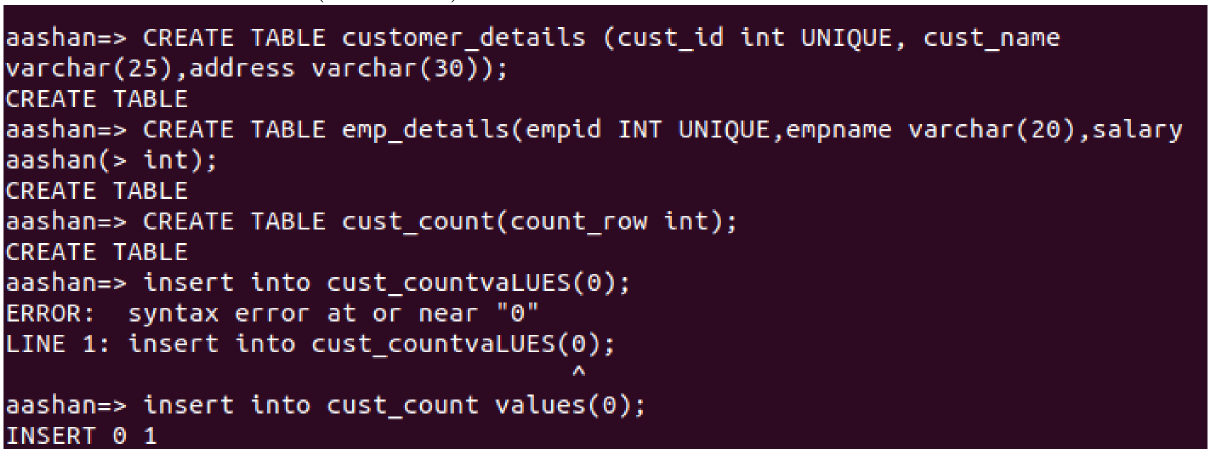
QUESTIONS:

Write PL/SQL programs for the following:

Create a table customerdetails (custid (unique),custname, address).

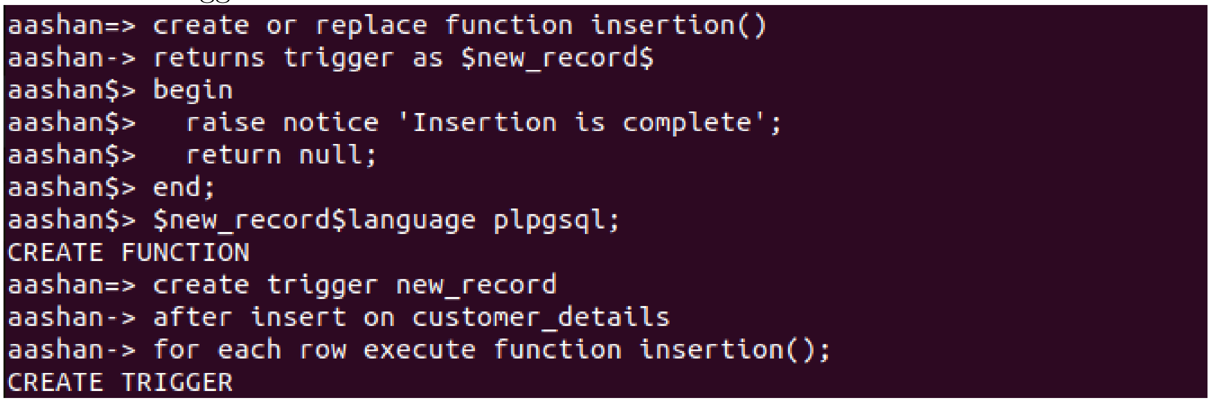
Create a table empdetails(empid(unique), empname,salary)

Create table custcount(countrow)

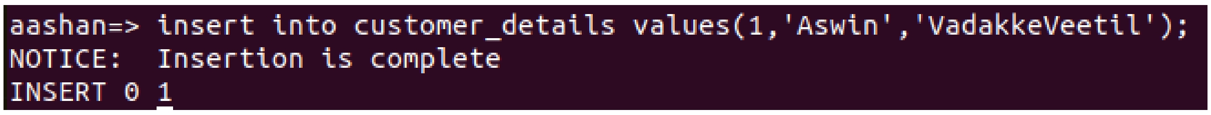


```
aashan=> CREATE TABLE customer_details (cust_id int UNIQUE, cust_name
varchar(25),address varchar(30));
CREATE TABLE
aashan=> CREATE TABLE emp_details(empid INT UNIQUE,empname varchar(20),salary
aashan(> int);
CREATE TABLE
aashan=> CREATE TABLE cust_count(count_row int);
CREATE TABLE
aashan=> insert into cust_countvaLUES(0);
ERROR:  syntax error at or near "0"
LINE 1: insert into cust_countvaLUES(0);
                                     ^
aashan=> insert into cust_count values(0);
INSERT 0 1
```

1. Create a trigger whenever a new record is inserted in the customerdetails table.

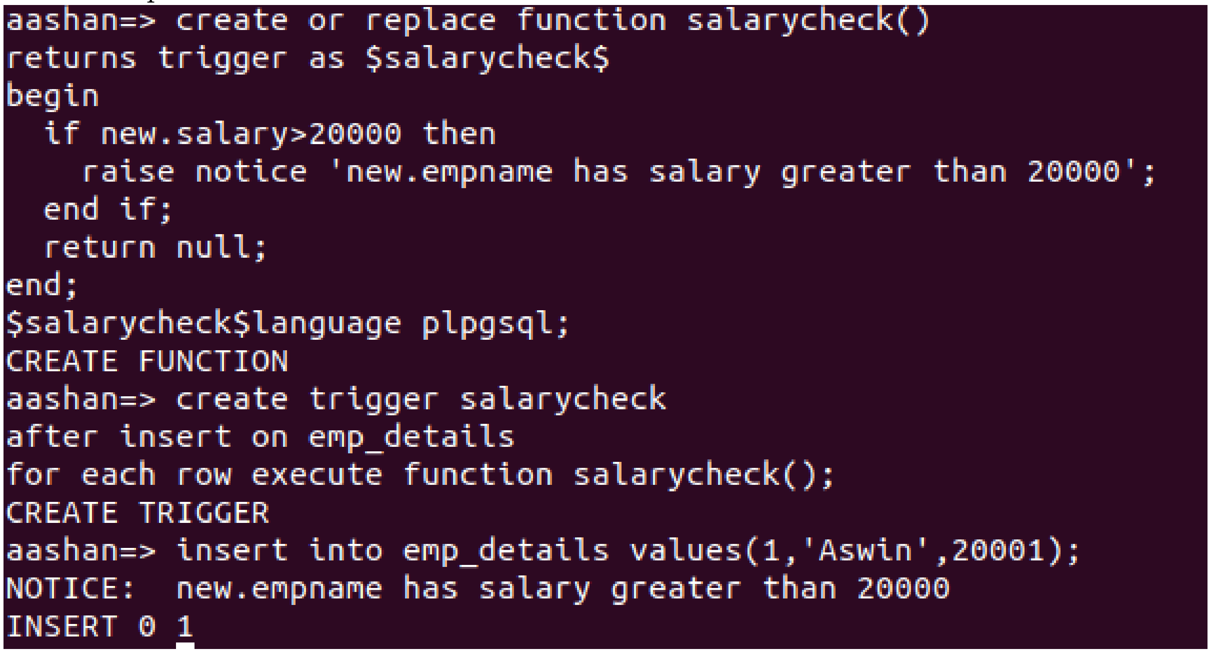


```
aashan=> create or replace function insertion()
aashan-> returns trigger as $new_record$
aashan$> begin
aashan$>   raise notice 'Insertion is complete';
aashan$>   return null;
aashan$> end;
aashan$> $new_record$language plpgsql;
CREATE FUNCTION
aashan=> create trigger new_record
aashan-> after insert on customer_details
aashan-> for each row execute function insertion();
CREATE TRIGGER
```



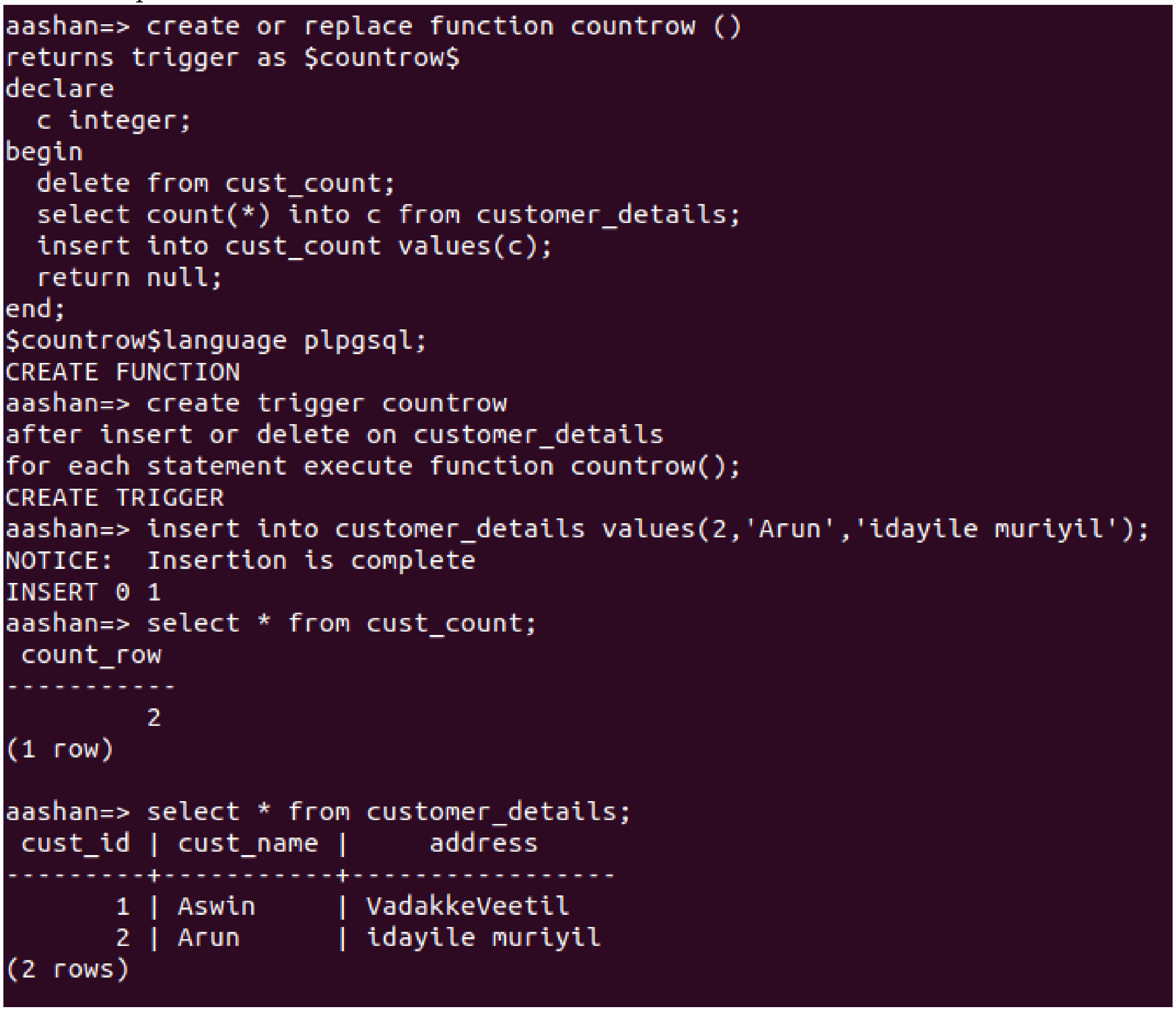
```
aashan=> insert into customer_details values(1,'Aswin','VadakkeVeetil');
NOTICE:  Insertion is complete
INSERT 0 1
```

2. Create a trigger to display a message when a user enters a value >20000 in the salary field of empdetails table.



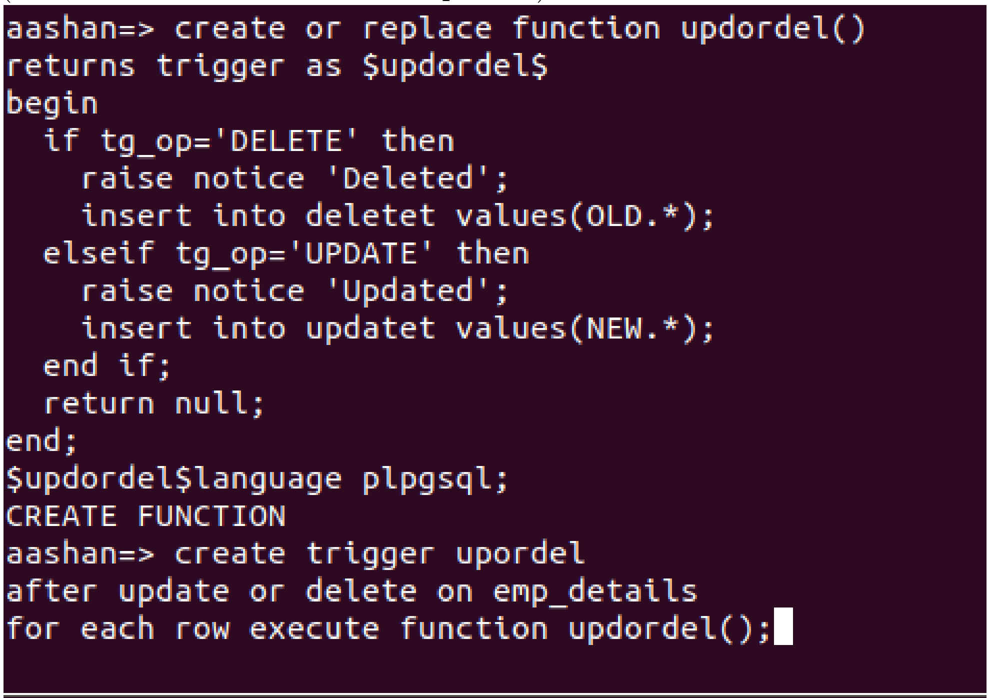
```
aashan=> create or replace function salarycheck()
returns trigger as $salarycheck$
begin
  if new.salary>20000 then
    raise notice 'new.empname has salary greater than 20000';
  end if;
  return null;
end;
$salarycheck$language plpgsql;
CREATE FUNCTION
aashan=> create trigger salarycheck
after insert on emp_details
for each row execute function salarycheck();
CREATE TRIGGER
aashan=> insert into emp_details values(1,'Aswin',20001);
NOTICE:  new.empname has salary greater than 20000
INSERT 0 1
```

3. Create a trigger w.r.t customerdetails table.Increment the value of countrow (in custcount table) whenever a new tuple is inserted and decrement the value of countrow when a tuple is deleted. Initial value of the countrow is set to 0

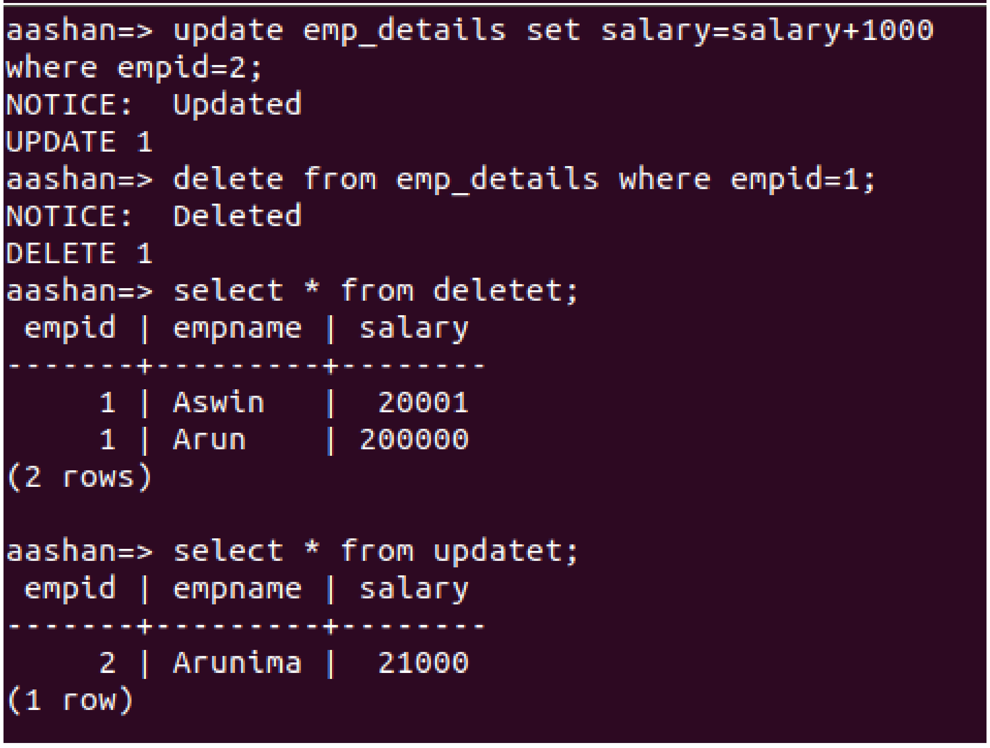


```
aashan=> create or replace function countrow ()
returns trigger as $countrow$
declare
  c integer;
begin
  delete from cust_count;
  select count(*) into c from customer_details;
  insert into cust_count values(c);
  return null;
end;
$countrow$language plpgsql;
CREATE FUNCTION
aashan=> create trigger countrow
after insert or delete on customer_details
for each statement execute function countrow();
CREATE TRIGGER
aashan=> insert into customer_details values(2,'Arun','idayile muriyil');
NOTICE:  Insertion is complete
INSERT 0 1
aashan=> select * from cust_count;
 count_row
-----------
         2
(1 row)

aashan=> select * from customer_details;
 cust_id | cust_name |     address
---------+-----------+-----------------
       1 | Aswin     | VadakkeVeetil
       2 | Arun      | idayile muriyil
(2 rows)
```

4. Create a trigger to insert the deleted rows from empdetails to another table and updated rows to another table.
(Create the tables deleted and updated)

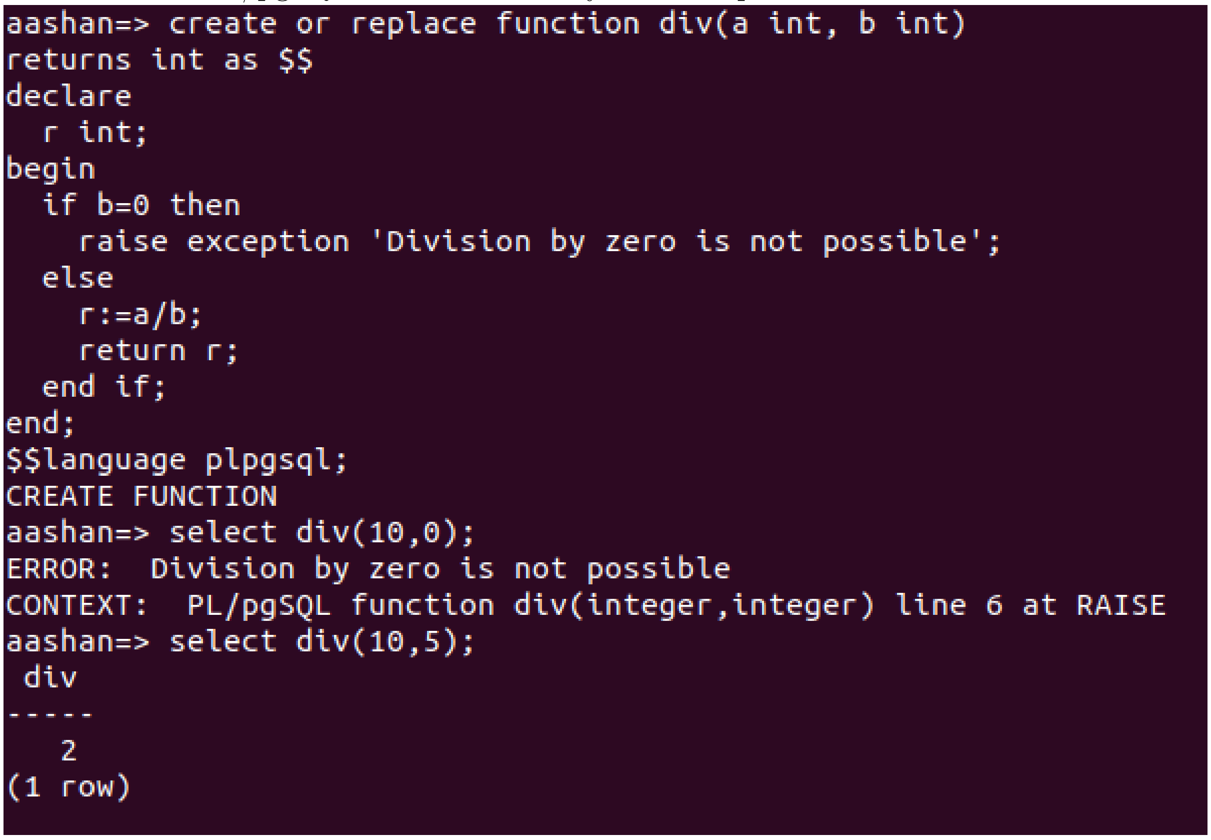


```
aashan=> create or replace function updordel()
returns trigger as $updordel$
begin
  if tg_op='DELETE' then
    raise notice 'Deleted';
    insert into deletet values(OLD.*);
  elseif tg_op='UPDATE' then
    raise notice 'Updated';
    insert into updatet values(NEW.*);
  end if;
  return null;
end;
$updordel$language plpgsql;
CREATE FUNCTION
aashan=> create trigger upordel
after update or delete on emp_details
for each row execute function updordel();
```



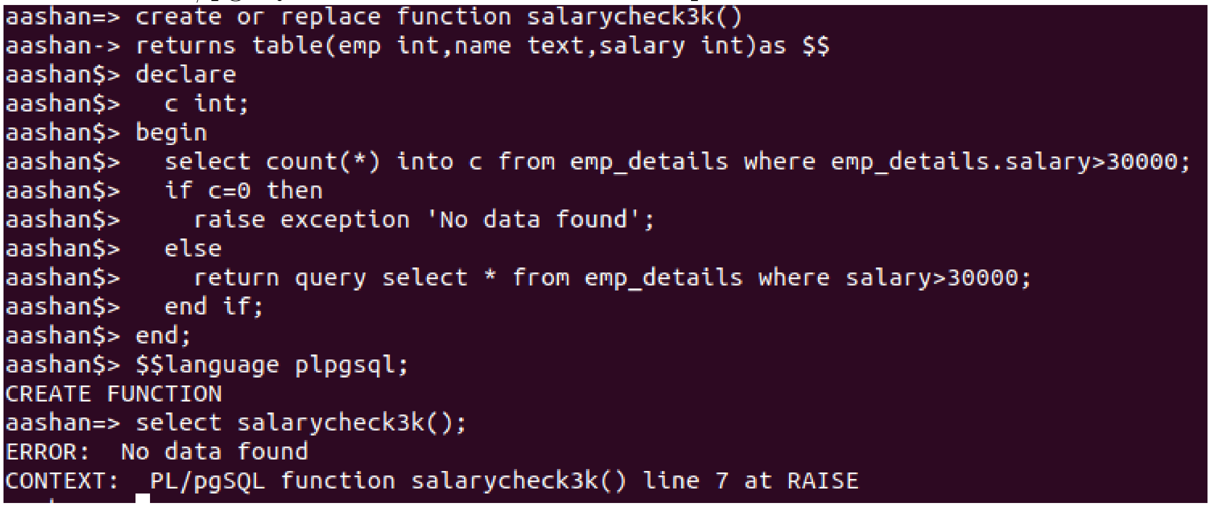
```
aashan=> update emp_details set salary=salary+1000
where empid=2;
NOTICE:  Updated
UPDATE 1
aashan=> delete from emp_details where empid=1;
NOTICE:  Deleted
DELETE 1
aashan=> select * from deletet;
 empid | empname | salary
-------+---------+--------
     1 | Aswin   |  20001
     1 | Arun    | 200000
(2 rows)

aashan=> select * from updatet;
 empid | empname | salary
-------+---------+--------
     2 | Arunima |  21000
(1 row)
```

5. Write a PL/pgSQL to show divide by zero exception.



```
aashan=> create or replace function div(a int, b int)
returns int as $$
declare
  r int;
begin
  if b=0 then
    raise exception 'Division by zero is not possible';
  else
    r:=a/b;
    return r;
  end if;
end;
$$language plpgsql;
CREATE FUNCTION
aashan=> select div(10,0);
ERROR:  Division by zero is not possible
CONTEXT:  PL/pgSQL function div(integer,integer) line 6 at RAISE
aashan=> select div(10,5);
 div
-----
   2
(1 row)
```

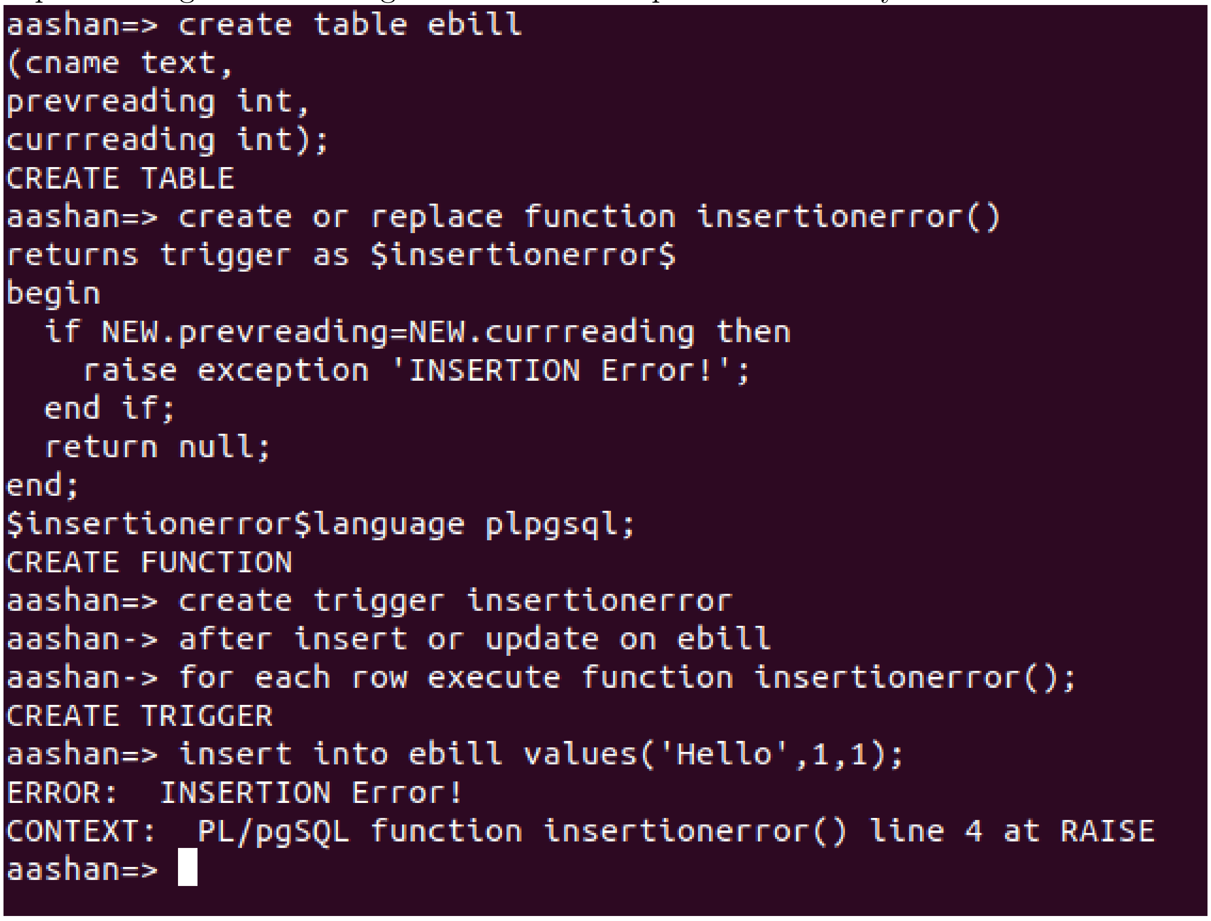
6. Write a PL/pgSQL to show no data found exception.



```
aashan=> create or replace function salarycheck3k()
aashan-> returns table(emp int,name text,salary int)as $$
aashan$> declare
aashan$>   c int;
aashan$> begin
aashan$>   select count(*) into c from emp_details where emp_details.salary>30000;
aashan$>   if c=0 then
aashan$>     raise exception 'No data found';
aashan$>   else
aashan$>     return query select * from emp_details where salary>30000;
aashan$>   end if;
aashan$> end;
aashan$> $$language plpgsql;
CREATE FUNCTION
aashan=> select salarycheck3k();
ERROR:  No data found
CONTEXT:  PL/pgSQL function salarycheck3k() line 7 at RAISE
```

7. Create a table with ebill(cname,prevreading,currreading).

If prevreading = currreading then raise an exception ‘Data Entry Error’.



```
aashan=> create table ebill
(cname text,
prevreading int,
currreading int);
CREATE TABLE
aashan=> create or replace function insertionerror()
returns trigger as $insertionerror$
begin
  if NEW.prevreading=NEW.currreading then
    raise exception 'INSERTION Error!';
  end if;
  return null;
end;
$insertionerror$language plpgsql;
CREATE FUNCTION
aashan=> create trigger insertionerror
aashan-> after insert or update on ebill
aashan-> for each row execute function insertionerror();
CREATE TRIGGER
aashan=> insert into ebill values('Hello',1,1);
ERROR:  INSERTION Error!
CONTEXT:  PL/pgSQL function insertionerror() line 4 at RAISE
aashan=> 
```

RESULT:

Implemented the program for trigger and exception handling using Postgresql version: 11.4 ,Ubuntu 18.04 and following output were obtained.

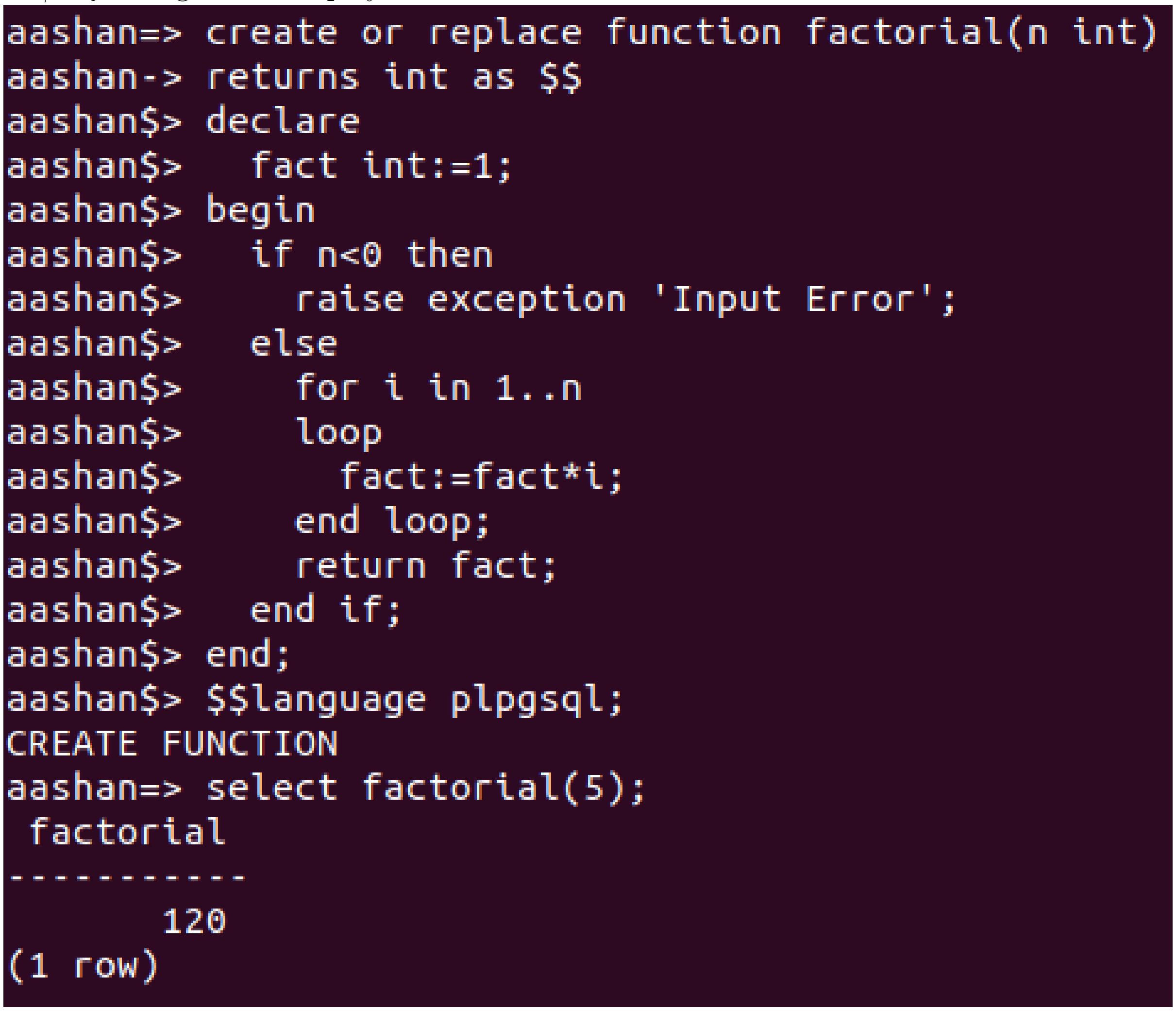
Experiment 11

PROCEDURES, FUNCTIONS AND PACKAGES

AIM:
To understand the usage of procedures, functions and packages.
THEORY:
Functions:
A function is a subprogram that computes a value. Functions and procedures are structured alike, except that functions have a RETURN clause.
Procedures:
A procedure is a subprogram that performs a specific action. A procedure has two parts: the specification (spec for short) and the body.
The procedure spec begins with the keyword PROCEDURE and ends with the procedure name or a parameter list.
Parameter declarations are optional. Procedures that take no parameters are written without parentheses. The procedure body begins with the keyword IS and ends with the keyword END followed by an optional procedure name.
The procedure body has three parts:
a declarative part, an executable part, and an optional exception-handling part.
The declarative part contains local declarations, which are placed between the keywords IS and BEGIN. The keyword DECLARE, which introduces declarations in an anonymous PL/SQL block, is not used. The executable part contains statements, which are placed between the keywords BEGIN and EXCEPTION (or END). At least one statement must appear in the executable part of a procedure. The NULL statement meets this requirement.
The exception-handling part contains exception handlers, which are placed between the keywords EXCEPTION and END.
Packages:
A package is a schema object that groups logically related PL/SQL types, items, and subprograms. Packages usually have two parts, a specification and a body, although sometimes the body is unnecessary. The specification (spec for short) is the interface to your applications; it declares the types, variables, constants, exceptions, cursors, and subprograms available for use. The body fully defines cursors and subprograms, and so implements the spec.

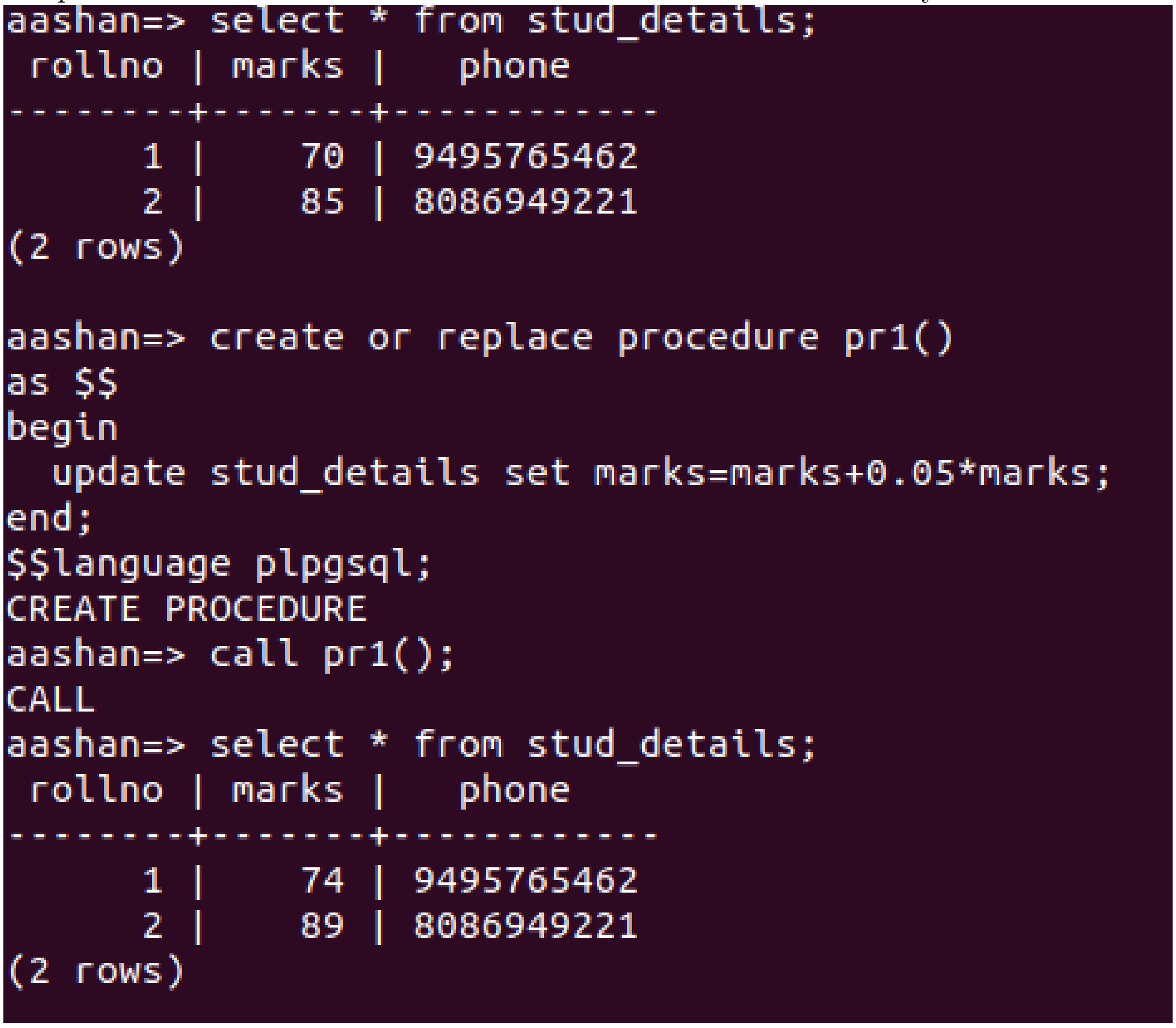
QUESTIONS:

1. Create a function factorial to find the factorial of a number. Use this function in a PL/SQL Program to display the factorial of a number read from the user.



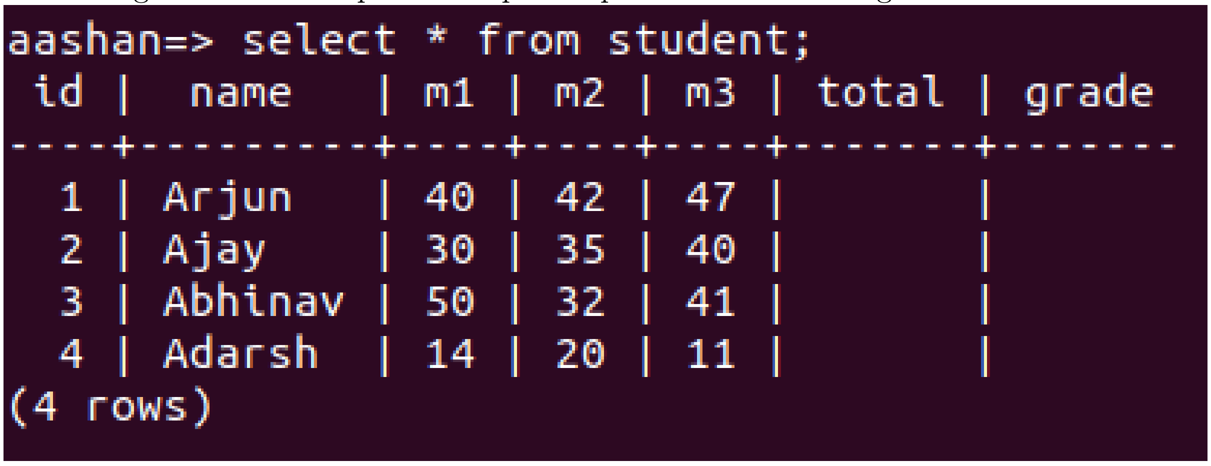
```
aashan=> create or replace function factorial(n int)
aashan-> returns int as $$
aashan$> declare
aashan$>   fact int:=1;
aashan$> begin
aashan$>   if n<0 then
aashan$>     raise exception 'Input Error';
aashan$>   else
aashan$>     for i in 1..n
aashan$>     loop
aashan$>       fact:=fact*i;
aashan$>     end loop;
aashan$>     return fact;
aashan$>   end if;
aashan$> end;
aashan$> $$language plpgsql;
CREATE FUNCTION
aashan=> select factorial(5);
 factorial
-----------
       120
(1 row)
```

2. Create a table studentdetails(roll int,marksint, phone int). Create a procedure pr1 to update all rows in the database. Boost the marks of all students by 5

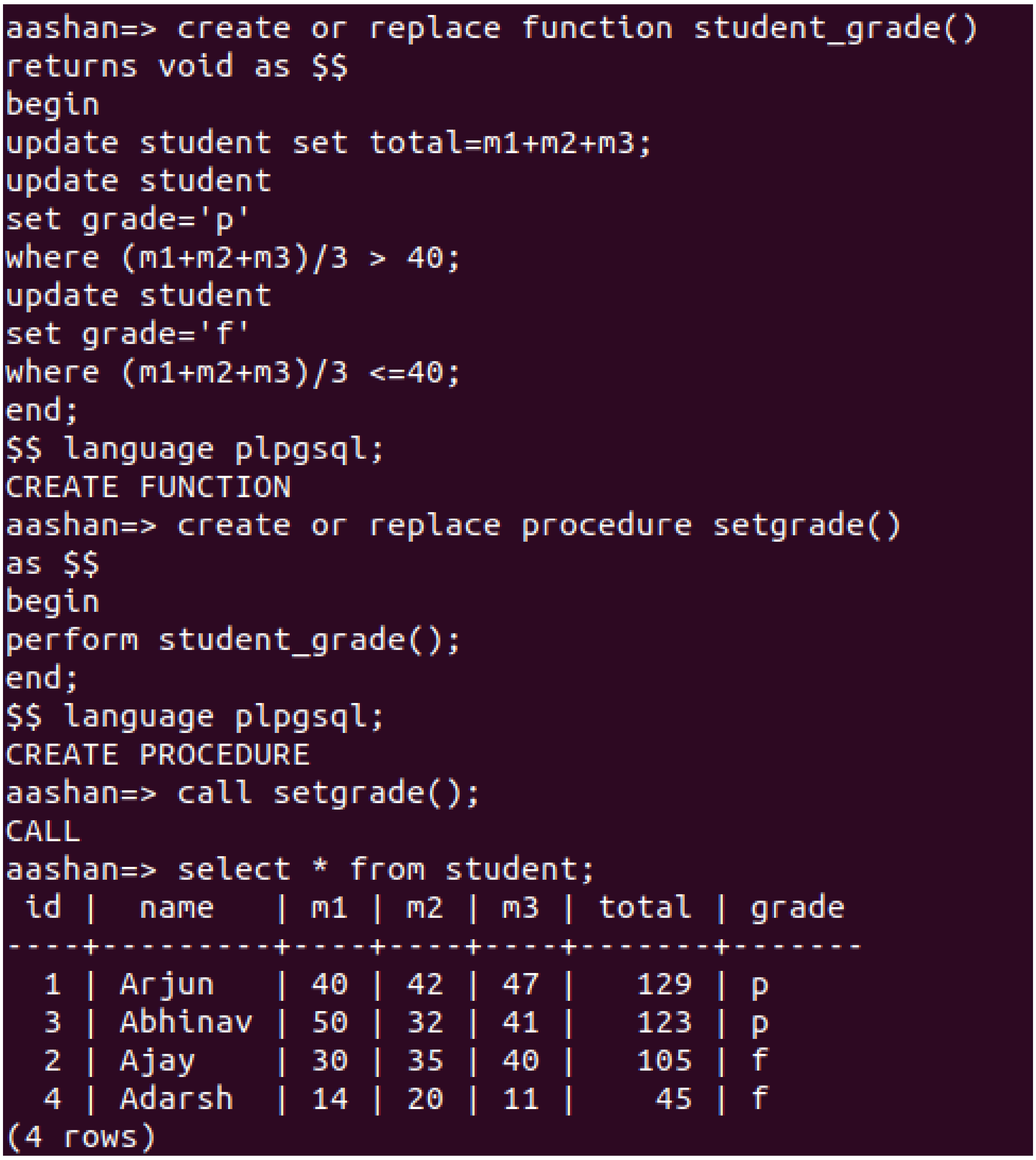


```
aashan=> select * from stud_details;
 rollno | marks |   phone
--------+-------+------------
      1 |    70 | 9495765462
      2 |    85 | 8086949221
(2 rows)

aashan=> create or replace procedure pr1()
as $$
begin
  update stud_details set marks=marks+0.05*marks;
end;
$$language plpgsql;
CREATE PROCEDURE
aashan=> call pr1();
CALL
aashan=> select * from stud_details;
 rollno | marks |   phone
--------+-------+------------
      1 |    74 | 9495765462
      2 |    89 | 8086949221
(2 rows)
```

3. Create table student (id, name, m1, m2, m3, total, grade).Create a function f1 to calculate grade. Create a procedure p1 to update the total and grade.



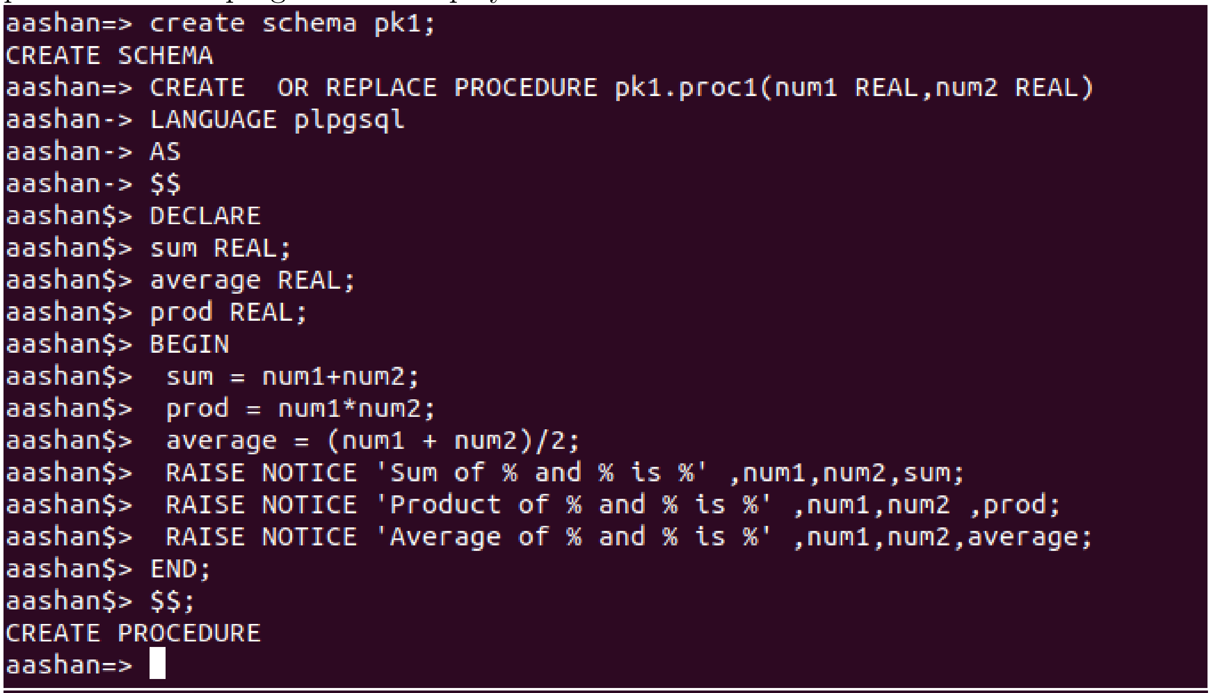
```
aashan=> select * from student;
 id |   name    | m1 | m2 | m3 | total | grade
----+-----------+----+----+----+-------+-------
  1 | Arjun     | 40 | 42 | 47 |       |
  2 | Ajay      | 30 | 35 | 40 |       |
  3 | Abhinav   | 50 | 32 | 41 |       |
  4 | Adarsh    | 14 | 20 | 11 |       |
(4 rows)
```



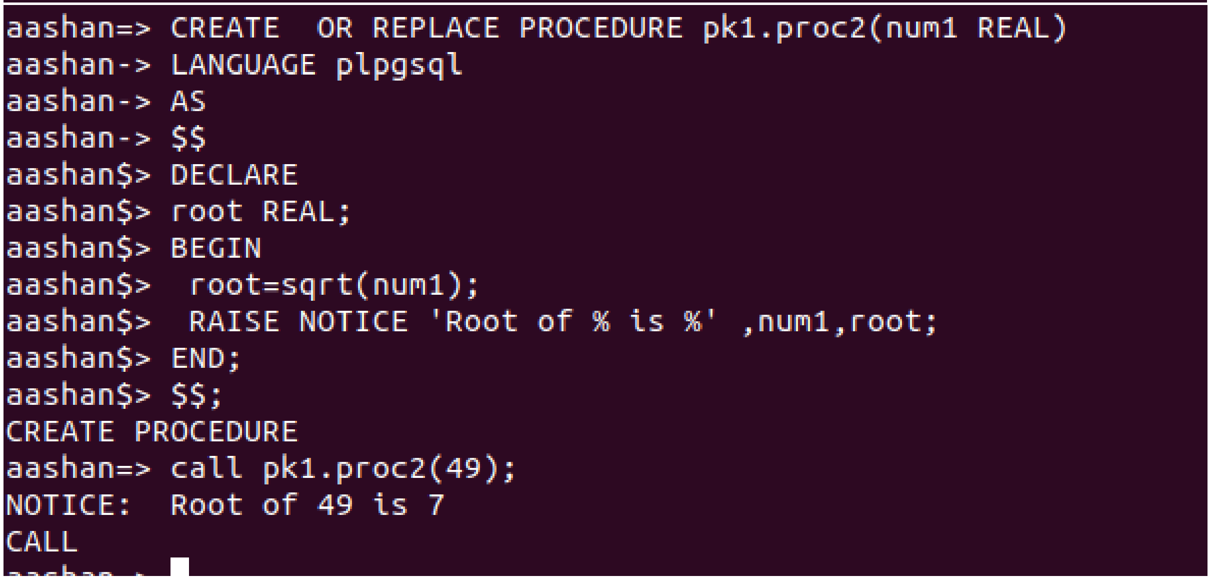
```
aashan=> create or replace function student_grade()
returns void as $$
begin
update student set total=m1+m2+m3;
update student
set grade='p'
where (m1+m2+m3)/3 > 40;
update student
set grade='f'
where (m1+m2+m3)/3 <=40;
end;
$$ language plpgsql;
CREATE FUNCTION
aashan=> create or replace procedure setgrade()
as $$
begin
perform student_grade();
end;
$$ language plpgsql;
CREATE PROCEDURE
aashan=> call setgrade();
CALL
aashan=> select * from student;
 id |  name   | m1 | m2 | m3 | total | grade
----+---------+----+----+----+-------+-------
  1 | Arjun   | 40 | 42 | 47 |   129 | p
  3 | Abhinav | 50 | 32 | 41 |   123 | p
  2 | Ajay    | 30 | 35 | 40 |   105 | f
  4 | Adarsh  | 14 | 20 | 11 |    45 | f
(4 rows)
```

4.Create a package pk1 consisting of the following functions and procedures
- Procedure proc1 to find the sum, average and product of two numbers
- Procedure proc2 to find the square root of a number
- Function named fn11 to check whether a number is even or not
- A function named fn22 to find the sum of 3 numbers

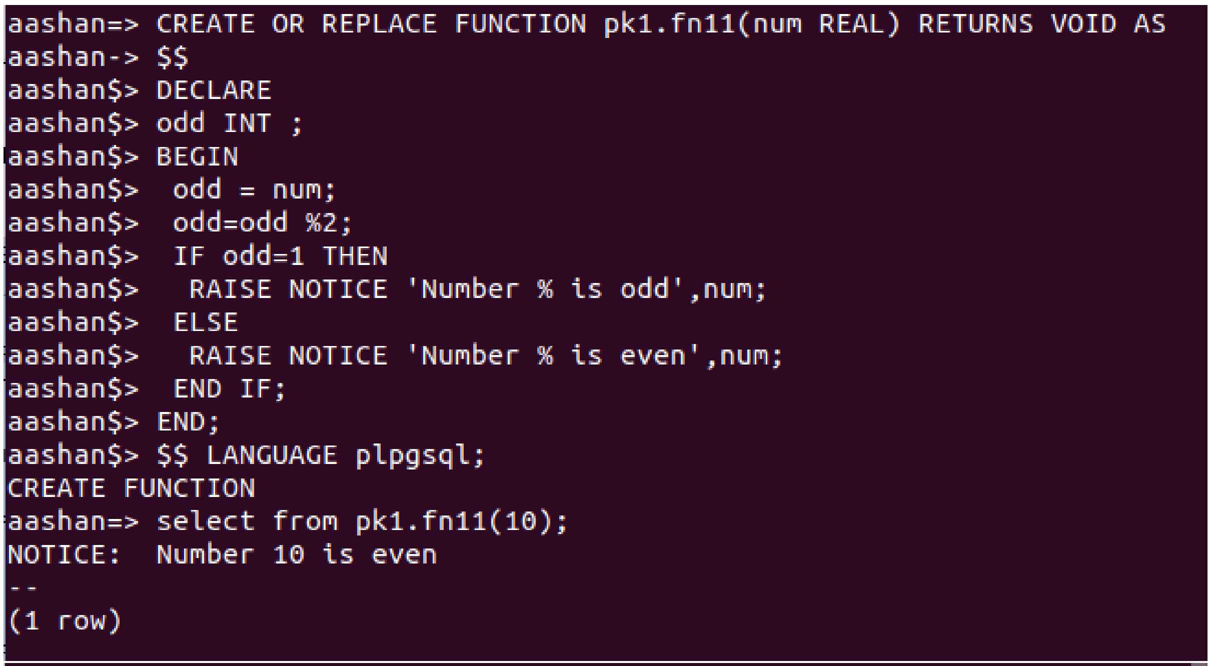
Use this package in a PL/SQL program. Call the functions f11, f22 and procedures pro1, pro2 within the program and display their results.



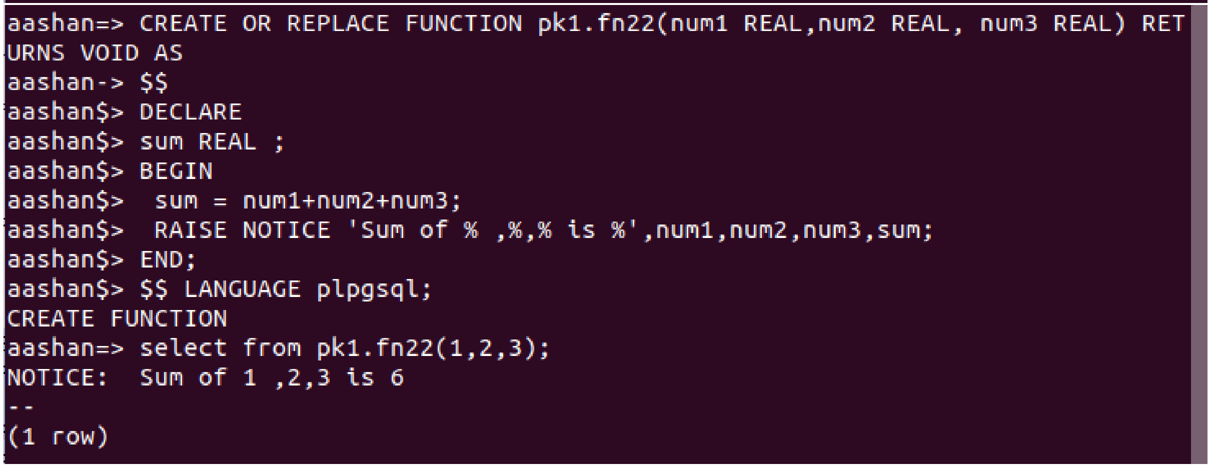
```
aashan=> create schema pk1;
CREATE SCHEMA
aashan=> CREATE  OR REPLACE PROCEDURE pk1.proc1(num1 REAL,num2 REAL)
aashan-> LANGUAGE plpgsql
aashan-> AS
aashan-> $$
aashan$> DECLARE
aashan$> sum REAL;
aashan$> average REAL;
aashan$> prod REAL;
aashan$> BEGIN
aashan$>  sum = num1+num2;
aashan$>  prod = num1*num2;
aashan$>  average = (num1 + num2)/2;
aashan$>  RAISE NOTICE 'Sum of % and % is %' ,num1,num2,sum;
aashan$>  RAISE NOTICE 'Product of % and % is %' ,num1,num2 ,prod;
aashan$>  RAISE NOTICE 'Average of % and % is %' ,num1,num2,average;
aashan$> END;
aashan$> $$;
CREATE PROCEDURE
aashan=> 
```



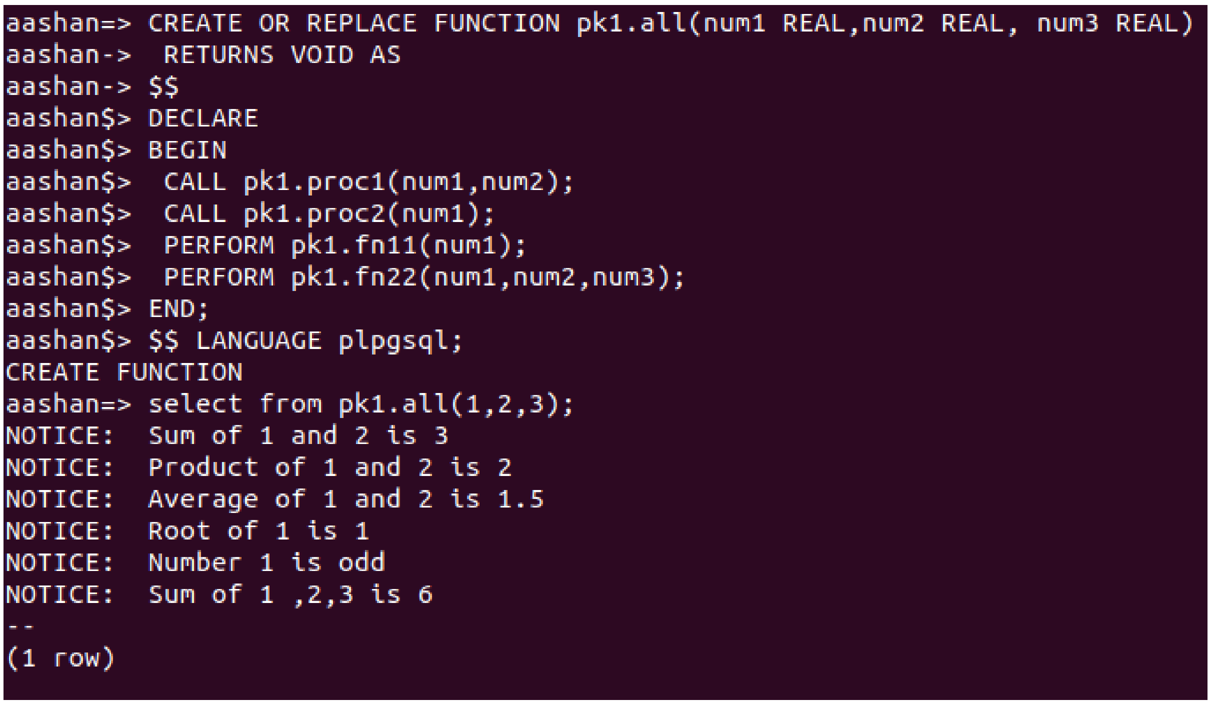
```
aashan=> CREATE  OR REPLACE PROCEDURE pk1.proc2(num1 REAL)
aashan-> LANGUAGE plpgsql
aashan-> AS
aashan-> $$
aashan$> DECLARE
aashan$> root REAL;
aashan$> BEGIN
aashan$>  root=sqrt(num1);
aashan$>  RAISE NOTICE 'Root of % is %' ,num1,root;
aashan$> END;
aashan$> $$;
CREATE PROCEDURE
aashan=> call pk1.proc2(49);
NOTICE:  Root of 49 is 7
CALL
```



```
aashan=> CREATE OR REPLACE FUNCTION pk1.fn11(num REAL) RETURNS VOID AS
aashan-> $$
aashan$> DECLARE
aashan$> odd INT ;
aashan$> BEGIN
aashan$>  odd = num;
aashan$>  odd=odd %2;
aashan$>  IF odd=1 THEN
aashan$>   RAISE NOTICE 'Number % is odd',num;
aashan$>  ELSE
aashan$>   RAISE NOTICE 'Number % is even',num;
aashan$>  END IF;
aashan$> END;
aashan$> $$ LANGUAGE plpgsql;
CREATE FUNCTION
aashan=> select from pk1.fn11(10);
NOTICE:  Number 10 is even
--
(1 row)
```



```
aashan=> CREATE OR REPLACE FUNCTION pk1.fn22(num1 REAL,num2 REAL, num3 REAL) RET
URNS VOID AS
aashan-> $$
aashan$> DECLARE
aashan$> sum REAL ;
aashan$> BEGIN
aashan$>  sum = num1+num2+num3;
aashan$>  RAISE NOTICE 'Sum of % ,%,% is %',num1,num2,num3,sum;
aashan$> END;
aashan$> $$ LANGUAGE plpgsql;
CREATE FUNCTION
aashan=> select from pk1.fn22(1,2,3);
NOTICE:  Sum of 1 ,2,3 is 6
--
(1 row)
```



```
aashan=> CREATE OR REPLACE FUNCTION pk1.all(num1 REAL,num2 REAL, num3 REAL)
aashan->  RETURNS VOID AS
aashan-> $$
aashan$> DECLARE
aashan$> BEGIN
aashan$>  CALL pk1.proc1(num1,num2);
aashan$>  CALL pk1.proc2(num1);
aashan$>  PERFORM pk1.fn11(num1);
aashan$>  PERFORM pk1.fn22(num1,num2,num3);
aashan$> END;
aashan$> $$ LANGUAGE plpgsql;
CREATE FUNCTION
aashan=> select from pk1.all(1,2,3);
NOTICE:  Sum of 1 and 2 is 3
NOTICE:  Product of 1 and 2 is 2
NOTICE:  Average of 1 and 2 is 1.5
NOTICE:  Root of 1 is 1
NOTICE:  Number 1 is odd
NOTICE:  Sum of 1 ,2,3 is 6
--
(1 row)
```

Result:

Implemented program for Procedures, functions and packages using Postgresql version: 11.4 ,Ubuntu 18.04 and following output were obtained.