College Of Engineering Trivandrum

Application Software Development Lab
CS333

Abhishek Manoharan
S5 CSE Roll No:2

TVE17CS002

Supervisors: Vipin Vasu A V , Remya Krishnan

A report submitted in partial fulfilment of ASD lab work
in Semester 5

in the

Department of Computer Science

October 2, 2019

Contents

List of Figures

1 POSTGRESQL INSTALLATION

Cycle 1

Exp No 0

1.1 Step 1 Update system and install dependencies

It is recommended to update your current system packages if it is a new server instance.

sudo apt update && sudo apt -y upgrade

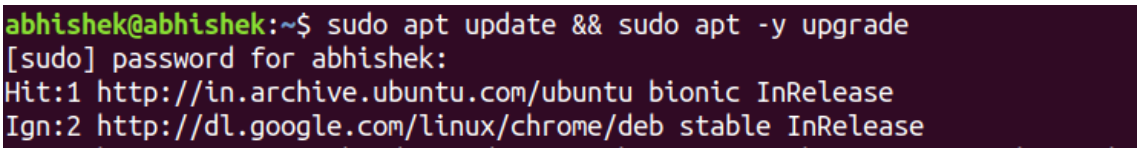

Figure 1.1: *Update and Upgrade*

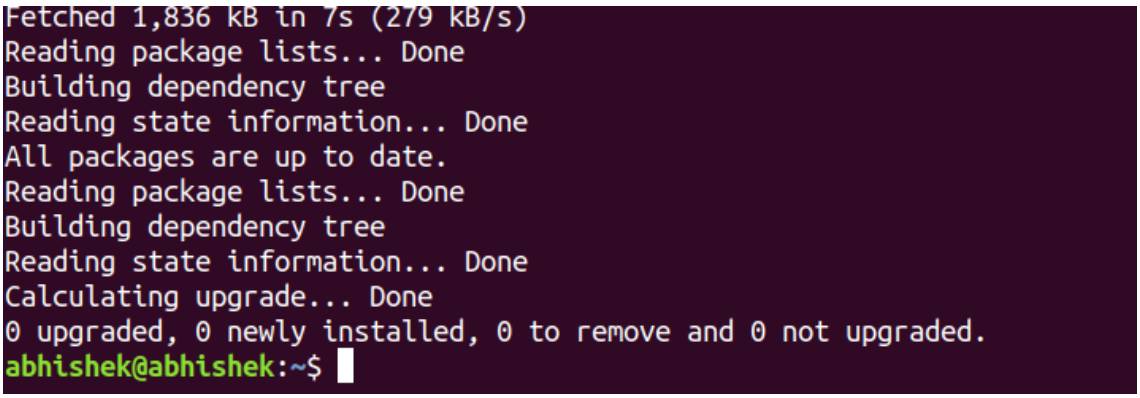

Figure 1.2: *Update and upgrade*

install vim and wget if not already installed.

```
sudo apt install -y wget vim
```

```
abhishek@abhishek:~$ sudo apt install -y wget vim
Reading package lists... Done
Building dependency tree
Reading state information... Done
vim is already the newest version (2:8.0.1453-1ubuntu1.1).
wget is already the newest version (1.19.4-1ubuntu2.2).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
abhishek@abhishek:~$
```

Figure 1.3: *Installing vim and wget*

1.2 Step 2: Add PostgreSQL 11 APT repository

Before adding repository content to your Ubuntu 18.04 , We need to import the repository signing key:
wget –quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc — sudo apt-key add -

```
abhishek@abhishek:~$ wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
[sudo] password for abhishek:
OK
abhishek@abhishek:~$
```

Figure 1.4: *Adding repo*

After importing GPG key, add repository contents to your Ubuntu 18.04 system:
RELEASE=$(lsb_release -cs)
echo "deb http://apt.postgresql.org/pub/repos/apt/ $RELEASE"-pgdg main — sudo tee /etc/apt/sources.list.d/pgdg.list

```
abhishek@abhishek:~$ RELEASE=$(lsb_release -cs)
abhishek@abhishek:~$ echo "deb http://apt.postgresql.org/pub/repos/apt/ ${RELEASE}"-pgdg main | sudo tee  /etc/apt/sources.list.d/pgdg.list
deb http://apt.postgresql.org/pub/repos/apt/ bionic-pgdg main
abhishek@abhishek:~$
```

Figure 1.5: *Adding repo content*

Verify repository file contents
cat /etc/apt/sources.list.d/pgdg.list

```
abhishek@abhishek:~$ cat /etc/apt/sources.list.d/pgdg.list
deb http://apt.postgresql.org/pub/repos/apt/ bionic-pgdg main
abhishek@abhishek:~$
```

Figure 1.6: *Verifying repo content*

1.3 Step 3: Install PostgreSQL 11 on Ubuntu 18.04

The last installation step is for PostgreSQL 11 packages. Run the following commands to install PostgreSQL 11 on Ubuntu 18.04

sudo apt update

sudo apt -y install postgresql-11

```
abhishek@abhishek:~$ sudo apt -y install postgresql-11
Reading package lists... Done
Building dependency tree
Reading state information... Done
postgresql-11 is already the newest version (11.4-1.pgdg18.04+1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
abhishek@abhishek:~$
```

Figure 1.7: *Installing Postgres*

1.4 Step 4: Set PostgreSQL admin users password and do testing

Set a password for the default admin user

sudo su - postgres

psql -c "alter user postgres with password 'your password'"

```
abhishek@abhishek:~$ sudo su - postgres
postgres@abhishek:~$ psql -c "alter user postgres with password 'password'"
ALTER ROLE
postgres@abhishek:~$ psql -c "alter user asdlab with password 'password'"
ALTER ROLE
postgres@abhishek:~$
```

Figure 1.8: *Setting Password*

1.5 Result

Successfully installed postgreSQL 11.5 in Ubuntu 18.04.

2 INTRODUCTION TO SQL

Cycle 1

Exp No 1

2.1 Aim

Understand the basics of SQL

2.2 Introduction

SQL is a standard language for accessing and manipulating databases.

2.3 What is SQL

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

2.4 What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases

- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

2.5 SQL is a Standard - BUT....

Although SQL is an ANSI/ISO standard, there are different versions of the SQL language.

However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

2.6 RDBMS

RDBMS stands for Relational Database Management System.

RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

2.7 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 theACM. 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.

2.8 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 sublanguage. 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.

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. Summary 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

2.9 Managing Tables

A table is the 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 could 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 to represent relationships.

2.10 Creating Tables

To create a table, use the SQL command CREATETABLE.
Syntax:

CREATE TABLE 'TABLE NAME'('FIELD NAME ' 'DATA TYPE' 'SIZE',.....)

2.11 Altering Tables

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 NOTNULL 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

2.12 Data Types

2.13 String Data Types

String Datatypes

The following are the **String Datatypes** in PostgreSQL:

Data Type Syntax	Explanation
char(*size*)	Where ***size*** is the number of characters to store. Fixed-length strings. Space padded on right to equal ***size*** characters.
character(*size*)	Where ***size*** is the number of characters to store. Fixed-length strings. Space padded on right to equal ***size*** characters.
varchar(*size*)	Where ***size*** is the number of characters to store. Variable-length string.
character varying(*size*)	Where ***size*** is the number of characters to store. Variable-length string.
text	Variable-length string.

Figure 2.1: *String Data Types*

2.14 Numerical Data Types

Numeric Datatypes

The following are the **Numeric Datatypes** in PostgreSQL:

Data Type Syntax	Explanation
bit(*size*)	Fixed-length bit string Where ***size*** is the length of the bit string.
varbit(*size*) bit varying(*size*)	Variable-length bit string Where ***size*** is the length of the bit string.
smallint	Equivalent to int2. 2-byte signed integer.
int	Equivalent to int4. 4-byte signed integer.
integer	Equivalent to int4. 4-byte signed integer.
bigint	Big integer value which is equivalent to int8. 8-byte signed integer.
smallserial	Small auto-incrementing integer value which is equivalent to serial2. 2-byte signed integer that is auto-incrementing.
serial	Auto-incrementing integer value which is equivalent to serial4. 4-byte signed integer that is auto-incrementing.
bigserial	Big auto-incrementing integer value which is equivalent to serial8. 8-byte signed integer that is auto-incrementing.
numeric(*m*,*d*)	Where ***m*** is the total digits and ***d*** is the number of digits after the decimal.
double precision	8 byte, double precision, floating-point number
real	4-byte, single precision, floating-point number
money	Currency value.
bool	Logical boolean data type - true or false
boolean	Logical boolean data type - true or false

Figure 2.2: *Numerical Data Types*

2.15 Date/Time Data Types

Date/Time Datatypes

The following are the **Date/Time Datatypes** in PostgreSQL:

Data Type Syntax	Explanation
date	Displayed as 'YYYY-MM-DD'.
timestamp	Displayed as 'YYYY-MM-DD HH:MM:SS'.
timestamp without time zone	Displayed as 'YYYY-MM-DD HH:MM:SS'.
timestamp with time zone	Displayed as 'YYYY-MM-DD HH:MM:SS-TZ'. Equivalent to timestamptz.
time	Displayed as 'HH:MM:SS' with no time zone.
time without time zone	Displayed as 'HH:MM:SS' with no time zone.
time with time zone	Displayed as 'HH:MM:SS-TZ' with time zone. Equivalent to timetz.

Figure 2.3: *Date/Time Data Types*

2.16 Result

Understood the basics of SQL.

3 BASIC SQL QUERIES I

Cycle 1

Exp No 2

3.1 Aim

To study the basic sql queries such as

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

3.2 Theory

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

SELECT column1, column2, ...
FROM table_name;

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:
SELECT * FROM table_name;

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax
It is possible to write the INSERT INTO statement in two ways. The first way specifies both the column names and the values to be inserted:

INSERT INTO table_name (column1, column2, column3, ...)VALUES (value1, value2, value3, ...);

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT INTO syntax would be as follows:

INSERT INTO table_name VALUES (value1, value2, value3, ...);

The SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;

The SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

DELETE Syntax

DELETE FROM table_name WHERE condition;

3.3 Questions

Create table named employee and populate it.

```
create table Employee(Emp_id INT NOT NULL,Emp_name VARCHAR(10) NOT NULL,Dept
VARCHAR(20) NOT NULL,Salary INT ,PRIMARY KEY(Emp_id) );


insert into Employee values('1' ,'Micheal', 'Production', '2500');
insert into Employee values('2', 'Joe', 'Production' ,'2500');
insert into Employee values('3', 'Smith', 'Sales' ,'2250');
insert into Employee values('4', 'David, 'Marketing' ,'2900');
insert into Employee values('5', 'Richard', 'Sales' ,'1600');
insert into Employee values('6', 'Jessy', 'Marketing' ,'1800');
insert into Employee values('7', 'Jane', 'Sales' ,'2000');
insert into Employee values('8', 'Janet', 'Production' ,'3000');
insert into Employee values('9', 'Neville', 'Marketing' ,'2750');
insert into Employee values('10', 'Richardson', 'Sales' ,'1800');
```

1. Display the details of all the employees.

```
select * from Employee;
```

```
asdlab=# select * from Employee;
 emp_id |  emp_name  |    dept    | salary
--------+------------+------------+--------
      2 | Joe        | Production |   2500
      3 | Smith      | Sales      |   2250
      5 | Richard    | Sales      |   1600
      6 | Jessy      | Marketing  |   1800
      7 | Jane       | Sales      |   2000
      8 | Janet      | Production |   3000
      9 | Neville    | Marketing  |   2750
     10 | Richardson | Sales      |   1800
(8 rows)
```

Figure 3.1: *Employee Table*

2. Display the names and ids of all employees.

```
select emp_id,emp_name from Employee;
```

```
asdlab=# select emp_id,emp_name from Employee;
 emp_id |  emp_name
--------+------------
      2 | Joe
      3 | Smith
      5 | Richard
      6 | Jessy
      7 | Jane
      8 | Janet
      9 | Neville
     10 | Richardson
(8 rows)

asdlab=#
```

Figure 3.2: *Emp id and emp name Table*

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

```
delete from Employee where emp_id=10;
```

```
asdlab=# delete from Employee where emp_id=10;
DELETE 1
asdlab=# select * from Employee;
 emp_id | emp_name |    dept    | salary
--------+----------+------------+--------
      2 | Joe      | Production |   2500
      3 | Smith    | Sales      |   2250
      5 | Richard  | Sales      |   1600
      6 | Jessy    | Marketing  |   1800
      7 | Jane     | Sales      |   2000
      8 | Janet    | Production |   3000
      9 | Neville  | Marketing  |   2750
(7 rows)

asdlab=#
```

Figure 3.3: *Employee Table after deletion*

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

```
insert into Employee values('10', 'Abhi', 'Production');
```

```
asdlab=# insert into Employee values('10', 'Abhi', 'Production' );
INSERT 0 1
asdlab=# select * from Employee;
 emp_id | emp_name |    dept    | salary
--------+----------+------------+--------
      2 | Joe      | Production |   2500
      3 | Smith    | Sales      |   2250
      5 | Richard  | Sales      |   1600
      6 | Jessy    | Marketing  |   1800
      7 | Jane     | Sales      |   2000
      8 | Janet    | Production |   3000
      9 | Neville  | Marketing  |   2750
     10 | Abhi     | Production |
(8 rows)

asdlab=#
```

Figure 3.4: *Employee Table after deletion*

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

```
select * from Employee where Dept='Marketing';
```

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

asdlab=#
```

Figure 3.5: *Employees working in marketing*

6. Add the salary details of the newly added employee.

```
update Employee set salary='1000' where emp_id=10;
```

```
asdlab=# update Employee set salary='1000' where emp_id=10;
UPDATE 1
asdlab=# select * from Employee;
 emp_id | emp_name |    dept    | salary
--------+----------+------------+--------
      2 | Joe      | Production |   2500
      3 | Smith    | Sales      |   2250
      5 | Richard  | Sales      |   1600
      6 | Jessy    | Marketing  |   1800
      7 | Jane     | Sales      |   2000
      8 | Janet    | Production |   3000
      9 | Neville  | Marketing  |   2750
      1 | Micheal  | Production |   2500
      4 | David    | Marketing  |   2900
     10 | Abhi     | Production |   1000
(10 rows)
```

Figure 3.6: *After setting salary of new employee*

7. Update the salary of Richard to 1900.

```
update Employee set salary='1900' where emp_name='Richard';
```

```
asdlab=# update Employee set salary='1900' where emp_name='Richard';
UPDATE 1
asdlab=# select * from Employee;
 emp_id | emp_name |    dept    | salary
--------+----------+------------+--------
      2 | Joe      | Production |   2500
      3 | Smith    | Sales      |   2250
      6 | Jessy    | Marketing  |   1800
      7 | Jane     | Sales      |   2000
      8 | Janet    | Production |   3000
      9 | Neville  | Marketing  |   2750
      1 | Micheal  | Production |   2500
      4 | David    | Marketing  |   2900
     10 | Abhi     | Production |   1000
      5 | Richard  | Sales      |   1900
(10 rows)
```

Figure 3.7: *After updating richard's salary*

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

```
select * from Employee where Dept='Marketing' and salary >'2000';
```

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

asdlab=#
```

Figure 3.8: *Marketing and above 2000*

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

```
select emp_name from Employee where Dept='Marketing' or Dept='Sales';
```

```
asdlab=# select emp_name from Employee where Dept='Marketing' or Dept='Sales';
 emp_name
----------
 Smith
 Jessy
 Jane
 Neville
 David
 Richard
(6 rows)
```

Figure 3.9: *Employees in sales or marketing*

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

```
select emp_name,Dept from Employee where salary>'2300' and salary<'3000'
;
```

```
asdlab=# select emp_name,Dept from Employee where salary>'2300' and salary<'3000' ;
 emp_name |    dept
----------+------------
 Joe      | Production
 Neville  | Marketing
 Micheal  | Production
 David    | Marketing
(4 rows)

asdlab=#
```

Figure 3.10: *Salary in between 2300 , 3000*

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

```
update Employee set salary=salary*1.2 ;
```

```
asdlab=#  update Employee set salary=salary*1.2 ;
UPDATE 10
asdlab=# select * from Employee;
 emp_id | emp_name |    dept    | salary
--------+----------+------------+--------
      2 | Joe      | Production |   3000
      3 | Smith    | Sales      |   2700
      6 | Jessy    | Marketing  |   2160
      7 | Jane     | Sales      |   2400
      8 | Janet    | Production |   3600
      9 | Neville  | Marketing  |   3300
      1 | Micheal  | Production |   3000
      4 | David    | Marketing  |   3480
     10 | Abhi     | Production |   1200
      5 | Richard  | Sales      |   2280
(10 rows)

asdlab=#
```

Figure 3.11: *Updating everyone's salary*

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

```
select emp_name from Employee where salary<2000 or Dept='Sales';
```

```
asdlab=# select emp_name from Employee where salary<2000 or Dept='Sales';
 emp_name
----------
 Smith
 Jane
 Abhi
 Richard
(4 rows)

asdlab=#
```

Figure 3.12: *Sales and less than 2000*

3.4 Result

Implemented the program for basic SQL queries using Postgresql (Ubuntu 11.5-1.pgdg18.04+1) in ubuntu 18.04 and following output were obtained.

4 BASIC SQL QUERIES II

Cycle 1

Exp No 3

4.1 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

4.2 Questions

Create a table named car_details and populate the table.

```
create table car_details(ID int not null,Name Varchar(10),company varchar(10),
country varchar(10),ApproxPrice real);
```

```
insert into car_details values ('1' , 'Beat','Chevrolet' ,'USA','4');
insert into car_details values ('2' , 'Swift','Maruti' ,'Japan','6');
insert into car_details values ('3' , 'Escort','Ford' ,'USA','4.2');
insert into car_details values ('4' , 'Sunny','Nissan' ,'Japan','8');
insert into car_details values ('5' , 'Etios','Toyoto' ,'Japan','7.2');
insert into car_details values ('6' , 'Beetle','Volkswagen' ,'Germany','21');
insert into car_details values ('7' , 'Sail','Chevrolet' ,'USA','5');
insert into car_details values ('8' , 'Aria','Tata' ,'India','7');
insert into car_details values ('9' , 'Passat','Volkswagen' ,'Germany','25');
insert into car_details values ('10' , 'SX4','Maruti' ,'Japan','6.7');
```

0. Display the details of all cars

```
select * from car_details;
```

```
asdlab=# select * from car_details;
 id |  name   |  company   | country | approxprice
----+---------+------------+---------+-------------
  1 | Beat    | Chevrolet  | USA     |           4
  2 | Swift   | Maruti     | Japan   |           6
  3 | Escort  | Ford       | USA     |         4.2
  4 | Sunny   | Nissan     | Japan   |           8
  5 | Etios   | Toyoto     | Japan   |         7.2
  6 | Beetle  | Volkswagen | Germany |          21
  7 | Sail    | Chevrolet  | USA     |           5
  8 | Aria    | Tata       | India   |           7
  9 | Passat  | Volkswagen | Germany |          25
 10 | SX4     | Maruti     | Japan   |         6.7
(10 rows)

asdlab=#
```

Figure 4.1: *Car_details Table*

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

```
select distinct company from car_details;
```

```
asdlab=# select distinct company from car_details;
  company
------------
 Ford
 Toyoto
 Maruti
 Chevrolet
 Tata
 Nissan
 Volkswagen
(7 rows)

asdlab=#
```

Figure 4.2: *Company names*

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

```
select distinct country from car_details;
```

```
asdlab=# select distinct country from car_details;
 country
---------
 USA
 Germany
 India
 Japan
(4 rows)

asdlab=#
```

Figure 4.3: *Country Names*

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

```
select * from car_details where approxprice>='4' and approxprice<=7;
```

```
asdlab=# select * from car_details where approxprice>='4' and approxprice<=7;
 id |  name   |  company   | country | approxprice
----+---------+------------+---------+-------------
  1 | Beat    | Chevrolet | USA     |           4
  2 | Swift   | Maruti     | Japan   |           6
  3 | Escort | Ford       | USA     |         4.2
  7 | Sail    | Chevrolet | USA     |           5
  8 | Aria    | Tata       | India   |           7
 10 | SX4     | Maruti     | Japan   |         6.7
(6 rows)

asdlab=#
```

Figure 4.4: *Cars in between 4 lakh and 7 lakh*

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

```
select name,company from car_details where country='Japan' and approxprice<='6';
```

```
asdlab=# select name,company from car_details where country='Japan' and approxprice<='6';
 name  | company
-------+---------
 Swift | Maruti
(1 row)

asdlab=#
```

Figure 4.5: *cars in japan below 6 lakhs*

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

```
select name,company from car_details where company='Nissan' or approxprice>'20';
```

```
asdlab=# select name,company from car_details where company='Nissan' or approxprice>'20';
  name   |  company
---------+------------
 Sunny   | Nissan
 Beetle | Volkswagen
 Passat | Volkswagen
(3 rows)

asdlab=#
```

Figure 4.6: *Nissan or price greater than 20 lakhs*

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

```
select name from car_details where company in ('Maruti' , 'Ford');
```

```
asdlab=# select name from car_details where company in ('Maruti' , 'Ford');
  name
--------
 Swift
 Escort
 SX4
(3 rows)

asdlab=#
```

Figure 4.7: *Maruti or Ford*

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

```
alter table car_details add year int;
update car_details set year='2015';
```

```
asdlab=# alter table car_details add year int;
ALTER TABLE
asdlab=# update car_details set year='2015';
UPDATE 10
asdlab=# select * from car_details ;
 id |  name   |  company   | country | approxprice | year
----+---------+------------+---------+-------------+------
  1 | Beat    | Chevrolet  | USA     |           4 | 2015
  2 | Swift   | Maruti     | Japan   |           6 | 2015
  3 | Escort  | Ford       | USA     |         4.2 | 2015
  4 | Sunny   | Nissan     | Japan   |           8 | 2015
  5 | Etios   | Toyoto     | Japan   |         7.2 | 2015
  6 | Beetle  | Volkswagen | Germany |          21 | 2015
  7 | Sail    | Chevrolet  | USA     |           5 | 2015
  8 | Aria    | Tata       | India   |           7 | 2015
  9 | Passat  | Volkswagen | Germany |          25 | 2015
 10 | SX4     | Maruti     | Japan   |         6.7 | 2015
(10 rows)

asdlab#
```

Figure 4.8: *Adding Year Column*

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

```
select name as Car_name from car_details ;
```

```
asdlab=#  select name as Car_name from car_details ;
 car_name
----------
 Beat
 Swift
 Escort
 Sunny
 Etios
 Beetle
 Sail
 Aria
 Passat
 SX4
(10 rows)

asdlab=#
```

Figure 4.9: *Selecting car names as car_names*

9 Rename the attribute name to car_name.

```
alter table car_details rename name to car_name;
```

```
asdlab=# alter table car_details rename name to c
ALTER TABLE
asdlab=# \d+ car_details
                                                  Table "
   Column    |         Type          | Collation
-------------+-----------------------+-----------
 id          | integer               |
 car_name    | character varying(10) |
 company     | character varying(10) |
 country     | character varying(10) |
 approxprice | real                  |
 year        | integer               |

asdlab=#
```

Figure 4.10: *Rename car name*

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

```
select car_name as cars_toyoto from car_details where company='Toyoto';
```

```
asdlab=# select car_name as cars_toyoto from car_details where company='Toyoto';
 cars_toyoto
-------------
 Etios
(1 row)

asdlab=#
```

Figure 4.11: *Cars of Toyoto*

11 List the details of all cars in alphabetical order.

```
select * from car_details order by car_name;
```

```
asdlab=# select * from car_details order by car_name;
 id | car_name |  company   | country | approxprice | year
----+----------+------------+---------+-------------+------
  8 | Aria     | Tata       | India   |           7 | 2015
  1 | Beat     | Chevrolet  | USA     |           4 | 2015
  6 | Beetle   | Volkswagen | Germany |          21 | 2015
  3 | Escort   | Ford       | USA     |         4.2 | 2015
  5 | Etios    | Toyoto     | Japan   |         7.2 | 2015
  9 | Passat   | Volkswagen | Germany |          25 | 2015
  7 | Sail     | Chevrolet  | USA     |           5 | 2015
  4 | Sunny    | Nissan     | Japan   |           8 | 2015
  2 | Swift    | Maruti     | Japan   |           6 | 2015
 10 | SX4      | Maruti     | Japan   |         6.7 | 2015
(10 rows)

asdlab=#
```

Figure 4.12: *Car details in alphabetical order*

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

```
select * from car_details order by approxprice;
```

```
asdlab=# select * from car_details order by approxprice;
 id | car_name |  company   | country | approxprice | year
----+----------+------------+---------+-------------+------
  1 | Beat     | Chevrolet  | USA     |           4 | 2015
  3 | Escort   | Ford       | USA     |         4.2 | 2015
  7 | Sail     | Chevrolet  | USA     |           5 | 2015
  2 | Swift    | Maruti     | Japan   |           6 | 2015
 10 | SX4      | Maruti     | Japan   |         6.7 | 2015
  8 | Aria     | Tata       | India   |           7 | 2015
  5 | Etios    | Toyoto     | Japan   |         7.2 | 2015
  4 | Sunny    | Nissan     | Japan   |           8 | 2015
  6 | Beetle   | Volkswagen | Germany |          21 | 2015
  9 | Passat   | Volkswagen | Germany |          25 | 2015
(10 rows)

asdlab=#
```

Figure 4.13: *Car details in cheapest to costliest*

4.3 Result

Implemented the program for basic SQL queries using Postgresql (Ubuntu 11.5-1.pgdg18.04+1) in ubuntu 18.04 and following output were obtained.

5 Introduction to Aggregate functions

Cycle 1

Exp No 4

5.1 Aim

Introduction to Aggregate functions

- 1. AVG()
- 2. MAX()
- 3. MIN()
- 4. SUM()
- 5. COUNT()

5.2 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 (All): returns the total number of rows. It is same like count(*)
- **Max(fieldname)** Returns the maximum value of the field
- **Min(fieldname)** Returns the maximum value of the field

5.3 Questions

Create a table named student and populate 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.

```
create table student(rollno int not null,name varchar(10),physics int,chemistry
int , maths int,primary key(rollno));
```

```
insert into student values('1','Adam','20','20','33');
insert into student values('2','Bob','18','9','41');
insert into student values('3','Bright','22','7','31');
insert into student values('4','Duke','13','21','20');
insert into student values('5','Elvin','14','22','23');
insert into student values('6','Fetcher','2','10','48');
insert into student values('7','Georgina','22','12','22');
insert into student values('8','Mary','24','14','31');
insert into student values('9','Tom','19','15','24');
insert into student values('10','Zack','8','20','36');
```

0. Display the Table

```
select * from student;
```

```
asdlab=# select * from student;
 rollno |   name   | physics | chemistry | maths
--------+----------+---------+-----------+-------
      1 | Adam     |      20 |        20 |    33
      2 | Bob      |      18 |         9 |    41
      3 | Bright   |      22 |         7 |    31
      4 | Duke     |      13 |        21 |    20
      5 | Elvin    |      14 |        22 |    23
      6 | Fetcher  |       2 |        10 |    48
      7 | Georgina |      22 |        12 |    22
      8 | Mary     |      24 |        14 |    31
      9 | Tom      |      19 |        15 |    24
     10 | Zack     |       8 |        20 |    36
(10 rows)

asdlab#
```

Figure 5.1: *Student Table*

1 Find the class average for the subject Physics

```
select avg(physics) from student;
```

```
asdlab=# select avg(physics) from student;
         avg
---------------------
 16.2000000000000000
(1 row)

asdlab=#
```

Figure 5.2: *Average for physics*

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

```
select max(maths) as highest_marks_maths from student;
```

```
asdlab=# select max(maths) as highest_marks_maths from student;
 highest_marks_maths
---------------------
                  48
(1 row)

asdlab=#
```

Figure 5.3: *Max maths*

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

```
select min(chemistry) as lowest_marks_chemistry from student;
```

```
asdlab=# select min(chemistry) as lowest_marks_chemistry from student;
 lowest_marks_chemistry
------------------------
                      7
(1 row)

asdlab=#
```

Figure 5.4: *Minimum for chemistry*

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

```
select count(*) from student where physics>='12';
```

```
asdlab=# select count(*) from student where physics>='12';
 count
-------
     8
(1 row)

asdlab=#
```

Figure 5.5: *Physics Pass*

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

```
select * from student where physics>='12' and chemistry>='12' and maths>='25';
```

```
asdlab=# select * from student where physics>='12' and chemistry>='12' and maths>='25';
 rollno | name | physics | chemistry | maths
--------+------+---------+-----------+-------
      1 | Adam |      20 |        20 |    33
      8 | Mary |      24 |        14 |    31
(2 rows)

asdlab=#
```

Figure 5.6: *All Pass*

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

```
alter table student add column total int ;
alter table student add column p_or_f char(1) ;
update student set total=physics+chemistry+maths;
update student set p_or_f='p' where physics>=12 and chemistry>=12 and maths>=25;
update student set p_or_f='f' where physics<12 or chemistry<12 or maths<25;
select * from student order by total desc;
```

```
asdlab=# alter table student add column total int ;
ALTER TABLE
asdlab=# alter table student add column p_or_f char(1) ;
ALTER TABLE
asdlab=# update student set total=physics+chemistry+maths;
UPDATE 10
asdlab=#  update student set p_or_f='p' where physics>=12 and chemistry>=12 and maths>=25;
UPDATE 2
asdlab=# update student set p_or_f='f' where physics<12 or chemistry<12 or maths<25;
UPDATE 8
asdlab=# select * from student order by total desc;
 rollno |   name   | physics | chemistry | maths | total | p_or_f
--------+----------+---------+-----------+-------+-------+--------
      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 | Fetcher  |       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)

asdlab=#
```

Figure 5.7: *Rank list*
labelinsert

7. Find pass percentage of the class for mathematics.

```
select count(rollno)*100/(select count(rollno) from student) as maths_pass_percentage
from student where maths>='25';
```

```
asdlab=# select count(rollno)*100/(sel
 maths_pass_percentage
-----------------------
                    60
(1 row)

asdlab=#
```

Figure 5.8: *Maths Pass percent*

8 Find the overall pass percentage for all class.

```
select count(rollno)*100/(select count(rollno) from student) as pass_percentage
from student where p_or_f='p';
```

```
asdlab=# select count(rollno)*100/
 pass_percentage
------------------
                20
(1 row)

asdlab=#
```

Figure 5.9: *Overall Pass percentage*

9 Find the class average.

```
select avg(total) from student;
```

```
asdlab=# select avg(total) from student;
         avg
--------------------
 62.1000000000000000
(1 row)

asdlab=#
```

Figure 5.10: *Class average*

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

```
select count(*) from student where p_or_f='p';
```

```
asdlab=# select count(*) from student where p_or_f='p';
 count
-------
     2
(1 row)

asdlab=#
```

Figure 5.11: *Total Pass students*

5.4 Result

Implemented the program for Aggregate functions in Postgresql 11.5 (Ubuntu 11.5-1.pgdg18.04+1) in ubuntu 18.04 and following output were obtained.

6 DATA CONSTRAINTS AND VIEWS

Cycle 1

Exp No 5

6.1 Aim

To study about various data constraints and views in SQL.

6.2 Theory

SQL Constraints
SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table. The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Uniquely identifies a row/record in another table
- CHECK - Ensures that all values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column when no value is specified
- INDEX - Used to create and retrieve data from the database very quickly

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

6.3 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.

```
create table Subjects(sub_id int not null,sub_name varchar(10) not null);
```

```
insert into subjects values( '1', "Maths");
insert into subjects values( '2', 'Physics');
insert into subjects values( '3','Chemistry');
insert into subjects values( '4','English');
```

0. Display the Table

```
select * from subjects;
```

```
asdlab=# select * from subjects;
 sub_id | sub_name
--------+----------
      2 | Physics
      3 | Chemistry
      4 | English
      1 | Maths
(4 rows)

asdlab=#
```

Figure 6.1: *Subjects Table*

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

```
alter table subjects add primary key(sub_id);
```

```
asdlab=# alter table subjects add primary key(sub_id);
ALTER TABLE
asdlab=#  insert into subjects values( '1', 'Zoology');
ERROR:  duplicate key value violates unique constraint "subjects_pkey"
DETAIL:  Key (sub_id)=(1) already exists.
asdlab=#
```

Figure 6.2: *subid as primary key*

b. Create a table named Staff with the given attributes

- **-staffid (Should be UNIQUE)**
- **-staffname**
- **-dept**
- **-Age (Greater than 22)**
- **-Salary (Less than 35000)**

```
create table Staff(staff_id int not null unique,staff_name varchar(10),
dept varchar(10),age int ,salary int,check(age>22),check(salary<35000));

insert into staff values('1','John','Purchasing','24','30000');
insert into staff values('2','Sera','Sales','25','20000');
insert into staff values('3','Jane','Sales','28','25000');
```

```
asdlab=# select * from staff;
 staff_id | staff_name |    dept    | age | salary
----------+------------+------------+-----+--------
        1 | John       | Purchasing |  24 |  30000
        2 | Sera       | Sales      |  25 |  20000
        3 | Jane       | Sales      |  28 |  25000
(3 rows)

asdlab=#
```

Figure 6.3: *Staff Table*

i)Delete the check constraint imposed on the attribute salary)

```
alter table staff drop constraint staff_salary_check;
```

ii)Delete the unique constraint on the attribute staffid.

```
alter table staff drop constraint staff_staff_id_key;
```

```
asdlab=# alter table staff drop constraint staff_salary_check;
ALTER TABLE
asdlab=# alter table staff  drop constraint staff_staff_id_key;
ALTER TABLE
asdlab=#
```

Figure 6.4: *Dropping both constraints*

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)

```
create table Bank(bankcode varchar(3),bank_name varchar(10),
headoffice varchar(10),branchoffice int );

alter table bank add constraint primarykey primary key (bankcode);
alter table bank add constraint branchoffice_check check(branchoffice>0);
insert into bank values('AAA','SIB','Ernakulam','6');
insert into bank values('BBB','Federal','Kottayam','5');
insert into bank values('CCC','Canara','Trivandrum','3');
```

```
asdlab=# select * from bank;
 bankcode | bank_name | headoffice | branchoffice
----------+-----------+------------+--------------
 AAA      | SIB       | Ernakulam  |            6
 BBB      | Federal   | Kottayam   |            5
 CCC      | Canara    | Trivandrum |            3
(3 rows)

asdlab=# \d+ bank
                                                  Table
    Column    |         Type          | Collation |
--------------+-----------------------+-----------+
 bankcode     | character varying(3)  |           |
 bank_name    | character varying(10) |           |
 headoffice   | character varying(10) |           |
 branchoffice | integer               |           |
Indexes:
    "primarykey" PRIMARY KEY, btree (bankcode)
Check constraints:
    "branchoffice_check" CHECK (branchoffice > 0)

asdlab=#
```

Figure 6.5: *Bank 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)

```
create table Branch(branchid int ,branchname varchar(10) default 'New
Delhi',
bankid varchar(3),primary key(branchid) );

alter table Branch add constraint branch_bankid_fkey foreign key(bankid)
references bank(bankcode) on update cascade on delete cascade;
```

```
insert into Branch values('1','Kottayam','CCC');
insert into Branch(branchid,bankid) values('2','AAA');
insert into bank values('SBT','Indian','Delhi','7');
insert into Branch values('5','Calicut','SBT');
```

```
                                        Table "public.branch"
   Column   |         Type          | Collation | Nullable |           Default            | Storage  | Stat
tion
------------+-----------------------+-----------+----------+------------------------------+----------+-----
-----
 branchid   | integer               |           | not null |                              | plain    |
 branchname | character varying(10) |           |          | 'New Delhi'::character varying | extended |
 bankid     | character varying(3)  |           |          |                              | extended |
Indexes:
    "branch_pkey" PRIMARY KEY, btree (branchid)
Foreign-key constraints:
    "branch_bankid_fkey" FOREIGN KEY (bankid) REFERENCES bank(bankcode) ON UPDATE CASCADE ON DELETE CASCADE
```

Figure 6.6: *Shema of branch*

```
asdlab=# select * from branch;
 branchid | branchname | bankid
----------+------------+--------
        1 | Kottayam   | CCC
        2 | New Delhi  | AAA
        5 | Calicut    | SBT
(3 rows)
```

Figure 6.7: *Branch Table*

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

```
delete from bank where bankcode='SBT';
```

```
asdlab=# delete from bank where bankcode='SBT';
DELETE 1
asdlab=# select * from bank;
 bankcode | bank_name | headoffice | branchoffice
----------+-----------+------------+--------------
 AAA      | SIB       | Ernakulam  |            6
 BBB      | Federal   | Kottayam   |            5
 CCC      | Canara    | Trivandrum |            3
(3 rows)

asdlab=# select * from branch;
 branchid | branchname | bankid
----------+------------+--------
        1 | Kottayam   | CCC
        2 | New Delhi  | AAA
(2 rows)
```

Figure 6.8: *Deleting SBT*

iv) Drop the Primary Key using ALTER command

```
alter table branch drop constraint branch_pkey;
insert into branch values('1','PPP','CCC');
```

```
asdlab=# alter table branch drop constraint  branch_pkey;
ALTER TABLE
asdlab=# insert into branch values('1','PPP','CCC');
INSERT 0 1
asdlab=# select * from branch;
 branchid | branchname | bankid
----------+------------+--------
        1 | Kottayam   | CCC
        2 | New Delhi  | AAA
        1 | PPP        | CCC
(3 rows)

asdlab=#
```

Figure 6.9: *Dropping Primary key*

2. Create a View named sales_staff to hold the details of all staff working in sales Department

```
create view sales_staff as select * from staff where dept='Sales';
select * from sales_staff;
```

```
asdlab=# create view sales_staff as select * from staff where dept='Sales';
CREATE VIEW
asdlab=# select * from sales_staff;
 staff_id | staff_name | dept  | age | salary
----------+------------+-------+-----+--------
        2 | Sera       | Sales |  25 |  20000
        3 | Jane       | Sales |  28 |  25000
(2 rows)

asdlab=#
```

Figure 6.10: *view of sales staff*

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

- Constraint name Column Constraint
- -Pk branch_id Primary key
- Df branch_name Default :New Delhi
- Fk bankid Foreign key/References

```
 drop table Branch;
create table Branch(branchid int ,branchname varchar(10)
constraint Df default 'New Delhi' ,bankid varchar(3),
constraint pk primary key(branchid),
constraint Fk foreign key(bankid) references bank(bankcode) on update cascade
on delete cascade);
```

i) Delete the default constraint in the table

```
alter table branch alter branchname drop default;
```

ii) Delete the primary key constraint

```
alter table branch drop constraint Pk;
```

```
asdlab=# drop table Branch;
DROP TABLE
asdlab=# create table Branch(branchid int ,branchname varchar(10) constraint Df default 'New D
int pk primary key(branchid),constraint Fk foreign key(bankid) references bank(bankcode) on up
CREATE TABLE
asdlab=# alter table branch alter branchname drop default;
ALTER TABLE
asdlab=# alter table branch drop constraint Pk ;
ALTER TABLE
asdlab=# \d+ branch
                                        Table "public.branch"
   Column   |         Type          | Collation | Nullable | Default | Storage  | Stats target
------------+-----------------------+-----------+----------+---------+----------+--------------
 branchid   | integer               |           | not null |         | plain    |
 branchname | character varying(10) |           |          |         | extended |
 bankid     | character varying(3)  |           |          |         | extended |
Foreign-key constraints:
    "fk" FOREIGN KEY (bankid) REFERENCES bank(bankcode) ON UPDATE CASCADE ON DELETE CASCADE

asdlab#
```

Figure 6.11: *New Branch Table schema*

4. Update the view sales_staff to include the details of staff belonging to sales department whose salary is greater than 20000.

```
create or replace view sales_staff as(select * from staff
where salary>'20000' and dept='Sales');
```

view.png

```
asdlab=# create or replace view sales_staff as(select * from staff where salary>'20000' and dept='Sales');
CREATE VIEW
asdlab=# select * from sales_staff;
 staff_id | staff_name | dept  | age | salary
----------+------------+-------+-----+--------
        3 | Jane       | Sales |  28 |  25000
(1 row)

asdlab=#
```

Figure 6.12: *new view of sales staff*

5. **Delete the view sales_staff.**

```
drop view sales_staff;
```

```
asdlab=# drop view sales_staff;
DROP VIEW
asdlab=# select * from sales_staff;
ERROR:  relation "sales_staff" does not exist
LINE 1: select * from sales_staff;
                      ^
asdlab=#
```

Figure 6.13: *Deleting The view*

6.4 Result

Implemented the programs using Data constrains and views in Postgresql 11.5 (Ubuntu 11.5-1.pgdg18.04+1) in ubuntu 18.04 and following output were obtained.

7 STRING FUNCTIONS AND PATTERN MATCHING

Cycle 1

Exp No 6

7.1 Aim

Study String Functions & Pattern Matching

- SUBSTR - RPAD - LPAD
- LTRIM - UPPER - RTRIM
- LOWER - INITCAP - CONCAT
- LENGTH - REVERSE -POSITIION

7.2 Thoery

The main string functions are as follows:

Length()	*Length(field name/string)*	Gives the length of the string.
Lower()	*Lower(field name/string*	Gives the content in lowercase letters
Upper()	*Upper(field name/string)*	Gives the content in upper case letters
Concat()	*Concat (field name/string, field name/string)*	Combines the first and second string .
Lpad()	*Lpad(field name/string,length,character)*	Returns char1, left-padded to length n with the sequence of characters in char2
Rpad()	*Rpad(field name/string,length,character)*	Returns char1, right-padded to length n with char2
Rtrim()	*Rtrim(field name/string,substring)*	Returns char, with all the rightmost characters that appear in set removed
Instr()	*Instr(field name/string,substring,n,m)*	Searches char1 beginning with its nth character for the nth occurrence of char2 and returns the position of the character in char1 that is the first character of this occurrence.
Substr()	*Substr(field name/string,n,m)*	Returns a portion of char, beginning at character m, n characters long.

7.3 Questions

A. Create a table named acct_details and populate the table .

```
create table acc_details(Acct_no char(9) primary key,Branch varchar(10),
name varchar(20),phone int);


insert into acc_details values('A40123401','Chicago' ,'Mike Adams' ,'(378)400-1234');
insert into acc_details values('A40123402','Miami','Diana George','(372)420-2345');
insert into acc_details values('B40123403','Miami','Diaz Elizabeth','(371)450-3456');
insert into acc_details values('B40123404','Atlanta','Jeoffrey George','(370)460-4567'
insert into acc_details values('B40123405','New York','Jennifer Kaitlyn','(373)470-567
insert into acc_details values('C40123406','Chicago','Kaitlyn Vincent','(318)200-3235'
insert into acc_details values('C40123407','Miami','Abraham Gottfield','(328)300-2256'
insert into acc_details values('C50123408','New Jersey','Stacy Williams','(338)400-523
insert into acc_details values('D50123409','New York','Catherine George','(348)500-622
insert into acc_details values('D50123410','Miami','Oliver Scott','(358)600-7230');
```

0. Display the Table

```
select * from acc_details;
```

```
postgres=# select * from acc_details;
  acct_no  |   branch   |       name        |     phone
-----------+------------+-------------------+---------------
 A40123401 | Chicago    | Mike Adams        | (378)400-1234
 A40123402 | Miami      | Diana George      | (372)420-2345
 B40123403 | Miami      | Diaz Elizabeth    | (371)450-3456
 B40123404 | Atlanta    | Jeoffrey George   | (370)460-4567
 B40123405 | New York   | Jennifer Kaitlyn  | (373)470-5678
 C40123406 | Chicago    | Kaitlyn Vincent   | (318)200-3235
 C40123407 | Miami      | Abraham Gottfield | (328)300-2256
 C50123408 | New Jersey | Stacy Williams    | (338)400-5237
 D50123409 | New York   | Catherine George  | (348)500-6228
 D50123410 | Miami      | Oliver Scott      | (358)600-7230
(10 rows)

postgres=#
```

Figure 7.1: *acc_details Table*

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

```
select name from acc_details where name like 'D%';
```

```
postgres=# select name from acc_details where name like 'D%';
      name
---------------
 Diana George
 Diaz Elizabeth
(2 rows)

postgres=#
```

Figure 7.2: *People with name starting in D*

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

```
select branch from acc_details where branch like '%New%';
```

```
postgres=# select branch from acc_details where branch like '%New%';
   branch
------------
 New York
 New Jersey
 New York
(3 rows)

postgres=#
```

Figure 7.3: *Branch name containing new*

3. List all the names in Upper Case Format

```
select upper(name) from acc_details;
```

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

postgres=#
```

Figure 7.4: *Names in Upper case*

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

```
select name from acc_details where name like '___n%n';
```

```
postgres=# select name from acc_details where  name like '___n%n';
       name
------------------
 Jennifer Kaitlyn
(1 row)

postgres=#
```

Figure 7.5: *Fourth and last 'n'*

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

```
select name from acc_details where name like 'D_a%' and name like '%Eli%';
```

```
postgres=# select name from acc_details where  name like 'D_a%' and name like '%Eli%';
      name
---------------
 Diaz Elizabeth
(1 row)

postgres=#
```

Figure 7.6: *D__a and substring eli*

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

```
select name from acc_details where acct_no like '%6';
```

```
postgres=# select name from acc_details where  acct_no like '%6';
      name
-----------------
 Kaitlyn Vincent
(1 row)

postgres=#
```

Figure 7.7: *Account number ends in 6*

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

```
update acc_details set name=upper(name);
```

```
postgres=# update acc_details set name=upper(name);
UPDATE 10
postgres=# select * from acc_details;
  acct_no  |   branch   |       name        |     phone
-----------+------------+-------------------+---------------
 A40123401 | Chicago    | MIKE ADAMS        | (378)400-1234
 A40123402 | Miami      | DIANA GEORGE      | (372)420-2345
 B40123403 | Miami      | DIAZ ELIZABETH    | (371)450-3456
 B40123404 | Atlanta    | JEOFFREY GEORGE   | (370)460-4567
 B40123405 | New York   | JENNIFER KAITLYN  | (373)470-5678
 C40123406 | Chicago    | KAITLYN VINCENT   | (318)200-3235
 C40123407 | Miami      | ABRAHAM GOTTFIELD | (328)300-2256
 C50123408 | New Jersey | STACY WILLIAMS    | (338)400-5237
 D50123409 | New York   | CATHERINE GEORGE  | (348)500-6228
 D50123410 | Miami      | OLIVER SCOTT      | (358)600-7230
(10 rows)

postgres=#
```

Figure 7.8: *Updating name into upper case*

8. **List the names of all people ending on the alphabet t;**

```
select name from acc_details where lower(name) like '%t'
```

```
postgres=# select name from acc_details where lower(name) like '%t';
      name
----------------
 KAITLYN VINCENT
 OLIVER SCOTT
(2 rows)

postgres=#
```

Figure 7.9: *Name ends with t*

9. List all the names in reverse

```
select reverse(name) as reverse_name from acc_details;
```

```
postgres=# select reverse(name) as REVERSE_name from acc_details;
   reverse_name
------------------
 SMADA EKIM
 EGROEG ANAID
 HTEBAZILE ZAID
 EGROEG YERFFOEJ
 NYLTIAK REFINNEJ
 TNECNIV NYLTIAK
 DLEIFTTOG MAHARBA
 SMAILLIW YCATS
 EGROEG ENIREHTAC
 TTOCS REVILO
(10 rows)

postgres=#
```

Figure 7.10: *Reverse the name*

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

```
select lpad(phone,15,'+1') from acc_details;
```

```
postgres=# select lpad(phone,15,'+1') from acc_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
 +1(328)300-2256
 +1(338)400-5237
 +1(348)500-6228
 +1(358)600-7230
(10 rows)

postgres=#
```

Figure 7.11: *Phone number including +1*

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

```
select ltrim(acct_no,'ABCD') as acct_no,name,branch,phone from acc_details;
```

```
postgres=# select ltrim(acct_no,'ABCD') as acct_no,name,branch,phone from acc_details;
 acct_no  |       name        |   branch   |     phone
----------+-------------------+------------+---------------
 40123401 | MIKE ADAMS        | Chicago    | (378)400-1234
 40123402 | DIANA GEORGE      | Miami      | (372)420-2345
 40123403 | DIAZ ELIZABETH    | Miami      | (371)450-3456
 40123404 | JEOFFREY GEORGE   | Atlanta    | (370)460-4567
 40123405 | JENNIFER KAITLYN  | New York   | (373)470-5678
 40123406 | KAITLYN VINCENT   | Chicago    | (318)200-3235
 40123407 | ABRAHAM GOTTFIELD | Miami      | (328)300-2256
 50123408 | STACY WILLIAMS    | New Jersey | (338)400-5237
 50123409 | CATHERINE GEORGE  | New York   | (348)500-6228
 50123410 | OLIVER SCOTT      | Miami      | (358)600-7230
(10 rows)

postgres=#
```

Figure 7.12: *Removing first alphabets from acc no*

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

```
select * from acc_details where upper(name) like '%WILLIAMS%' or acct_no
like '_4%';
```

```
postgres=# select * from acc_details where  upper(name) like '%WILLIAMS%' or acct_no like '_4%';
  acct_no  |   branch   |       name        |     phone
-----------+------------+-------------------+---------------
 A40123401 | Chicago    | MIKE ADAMS        | (378)400-1234
 A40123402 | Miami      | DIANA GEORGE      | (372)420-2345
 B40123403 | Miami      | DIAZ ELIZABETH    | (371)450-3456
 B40123404 | Atlanta    | JEOFFREY GEORGE   | (370)460-4567
 B40123405 | New York   | JENNIFER KAITLYN  | (373)470-5678
 C40123406 | Chicago    | KAITLYN VINCENT   | (318)200-3235
 C40123407 | Miami      | ABRAHAM GOTTFIELD | (328)300-2256
 C50123408 | New Jersey | STACY WILLIAMS    | (338)400-5237
(8 rows)

postgres=#
```

Figure 7.13: *name contain williams or acc no starts in 4*

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

1. Find the reverse of the string nmutuAotedOehT.

```
SELECT REVERSE('NMUTUAOTEDOEHT');
```

```
postgres=# select REVERSE('NMUTUAOTEDOEHT');
    reverse
----------------
 THEODETOAUTUMN
(1 row)

postgres=#
```

Figure 7.14: *Function Reverse*

2. Use LTRIM function on 123231xyzTech so as to obtain the output Tech

```
SELECT LTRIM('123231XYZTECH','123XYZ');
```

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

postgres=#
```

Figure 7.15: *Function ltrim*

.3. Use RTRIM function on Computer to remove the trailing spaces.

```
SELECT RTRIM('COMPUTER');
```

```
postgres=# SELECT RTRIM('COMPUTER          ');
  rtrim
----------
 COMPUTER
(1 row)

postgres=#
```

Figure 7.16: *Function rtrim*

4. Perform RPAD on computer to obtain the output as computerXXXX

```
SELECT RPAD('COMPUTER',12,'X');
```

```
postgres=# SELECT RPAD('COMPUTER',12,'X');
     rpad
--------------
 COMPUTERXXXX
(1 row)

postgres=#
```

Figure 7.17: *Function rpad*

5. Use POSITION function to find the first occurrence of e in the string Welcome to Kerala.

```
SELECT POSITION('E' in 'WELCOMETOKERALA');
```

```
postgres=# SELECT POSITION('E' in 'WELCOMETOKERALA');
 position
----------
        2
(1 row)

postgres=#
```

Figure 7.18: *Function position*

6. Perform INITCAP function on mARKcALAwaY.

```
SELECT INITCAP('MARK CALAWAY');
```

```
postgres=# select INITCAP('MARK CALAWAY');
   initcap
-------------
 Mark Calaway
(1 row)

postgres=#
```

Figure 7.19: *Function initcap*

7. Find the length of the string Database Management Systems..

```
SELECT LENGTH('DATABASE MANAGEMENT SYSTEMS');
```

```
postgres=# SELECT LENGTH('DATABASE MANAGEMENT SYSTEMS');
 length
--------
     27
(1 row)

postgres=#
```

Figure 7.20: *Function Length*

8. Concatenate the strings Julius and Caesar.

```
SELECT CONCAT('JULIUS','CAESAR');
```

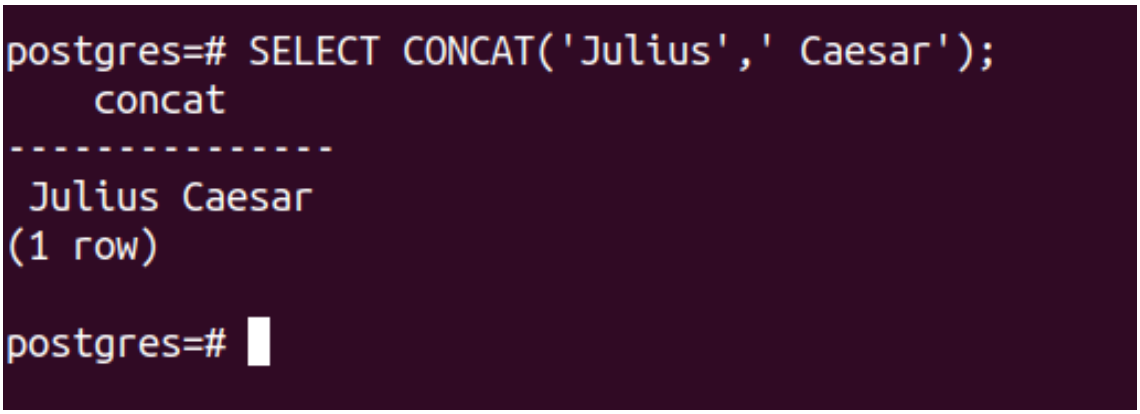

Figure 7.21: *Function concat*

9. Use SUBSTR function to retrieve the substring is from the string India is my country.

```
SELECT SUBSTR('INDIA IS MY COUNTRY',7,2);
```

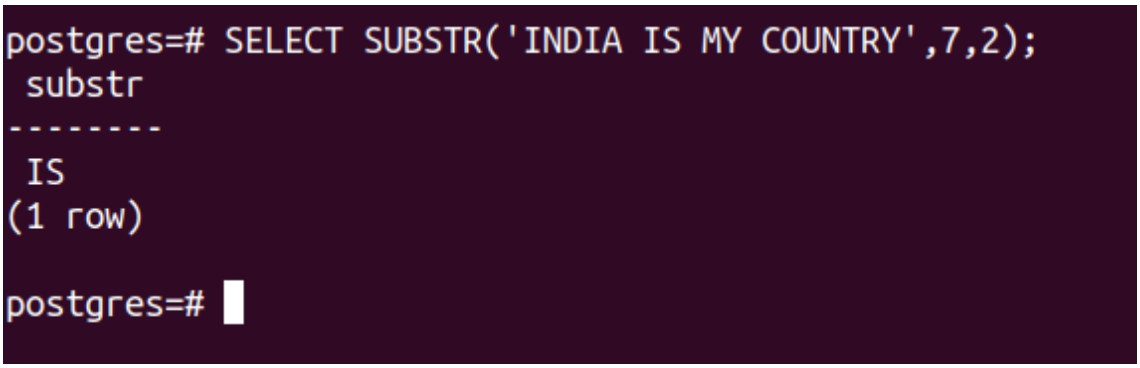

Figure 7.22: *Function substr*

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

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

7.4 Result

Implemented the programs using String Functions in Postgresql 11.5 (Ubuntu 11.5-1.pgdg18.04+1) in ubuntu 18.04 and following output were obtained.

8 JOIN STATEMENTS, SET OPERATIONS, NESTED QUERIES AND GROUPING

Cycle 1

Exp No 7

8.1 Aim

To get introduced to

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

8.2 Theory

SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

SQL Set Operation

The SQL Set operation is used to combine the two or more SQL SELECT statements. Types of Set Operation

- Union
- Intersect
- Minus(Except)

The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns. GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

8.3 Questions

Create Items,Orders,Customers,Delivery tables and populate them with appropriate data.

create table items(
itemid int not null,
itemname varchar(50) not null,
category varchar(20) not null,
price int not null,
instock int ,
constraint checkstock check(instock >0),
constraint pkey primary key(itemid));

```
insert into items values(5,'sony z5 premium','electronics',5005,1);
insert into items values(4,'Samsung Galaxy S4','electronics',5005,1);
insert into items values(3,'One Plus 7','electronics',6006,2);
insert into items values(2,'Iphone X','electronics',7007,6);
insert into items values(1,'Xiomi','electronics',1001,6);
```

0. Display the details items table

```
select * from items;
```

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

asdlab=#
```

Figure 8.1: *items Table*

create table customers(
custid int not null,
custname varchar(20),
address varchar(50) not null,
state varchar(10) not null,
primary key(custid));

```
insert into customers values(111,'elvin','202 jai street','delhi');
insert into customers values(113,'soman','puthumana','kerala');
insert into customers values(115,'mickey','juhu','maharastra');
insert into customers values(112,'patrick','harinagar','tamilnadu');
insert into customers values(114,'jaise','kottarakara','kerala');
```

0. Display the details customers table

```
select * from customers;
```

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

asdlab=#
```

Figure 8.2: *customers Table*

create table orders(
orderid int not null,
itemid int ,
quantity int not null,
orderdate date ,custid int
primary key(orderid),
foreign key(itemid) references items(itemid) on update cascade on delete cascade ,
foreign key(custid) references customers(custid) on update cascade on delete cascade));

```
insert into orders values(1,1,2,'2014-10-11',111);
insert into orders values(2,3,1,'2012-01-29',113);
insert into orders values(3,5,1,'2013-05-13',115);
insert into orders values(4,4,3,'2014-12-22',114);
```

0. Display the details orders table

```
select * from orders;
```

```
asdlab=# 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)

asdlab=#
```

Figure 8.3: *orders Table*

create table delivery(
deliveryid int not null,
orderid int ,custid int ,
primary key(deliveryid),
foreign key(orderid) references orders(orderid) on update cascade on delete cascade,
foreign key(custid) references customers(custid) on update cascade on delete cascade);

```
insert into delivery values(1001,1,111);
insert into delivery values(1002,2,113);
insert into delivery values(1003,3,115);
```

0. Display the details Delivery table

```
select * from delivery;
```

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

asdlab=#
```

Figure 8.4: *delivery Table*

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

```
select customers.custid,custname,address,state from customers , orders
where orders.custid=customers.custid;
```

```
asdlab=# select customers.custid,custname,address,state from customers , orders 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)

asdlab=#
```

Figure 8.5: *Customers who placed an order*

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

```
select customers.custid,custname,address,state from customers , delivery
where delivery.custid=customers.custid;
```

```
asdlab=# select customers.custid,custname,address,state from customers , delivery 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)

asdlab=#
```

Figure 8.6: *Customers whose orders are delivered*

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

```
select orderdate from customers , orders
where orders.custid=customers.custid and custname like 'j%';
```

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

asdlab=#
```

Figure 8.7: *orderdate with customers name starts in J*

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

```
select itemname,price from items as i ,customers as c,orders as o
where i.itemid=o.itemid and c.custid=o.custid and c.custname like'mickey';
```

```
asdlab=# select itemname,price from items as i ,customers as c
custname like'mickey';
    itemname     | price
-----------------+-------
 sony z5 premium |  5005
(1 row)

asdlab=#
```

Figure 8.8: *Mickey's Order*

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

```
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 );
```

```
asdlab=# select c.* from customers as c ,orders as o where o.custid=c.custid
 custid | custname |   address   | state
--------+----------+-------------+--------
    114 | jaise    | kottarakara | kerala
(1 row)

asdlab=#
```

Figure 8.9: *Undelivered Order aftter 2013 january*

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

```
(select i.itemid from items as i ,orders as o where i.itemid=o.itemid)
union
(select i.itemid from items as i , orders as o where i.itemid=o.itemid
and o.orderid not in (select orderid from delivery) );
```

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

asdlab=#
```

Figure 8.10: *Either ordered or not delivered*

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

```
(select custname from customers as c,orders as o where o.custid=c.custid)
intersect
(select custname from customers as c,delivery as d where d.custid=c.custid);
```

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

asdlab=#
```

Figure 8.11: *Ordered and delivered*

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

```
(select custname from customers as c,orders as o where o.custid=c.custid)
except
(select custname from customers as c,delivery as d where d.custid=c.custid);
```

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

asdlab=#
```

Figure 8.12: *Ordered but not delivered*

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

```
insert into orders values(5,2,1,'2012-05-25',115);

select * from customers where custid=(select custid from orders
```

```
group by custid order by count(*) desc LIMIT 1);
```

```
asdlab=# select * from customers where custid=(select custid from orders  group by custid order by  count(*) desc LIMIT 1);
 custid | custname | address |   state
--------+----------+---------+-----------
    115 | mickey   | juhu    | maharastra
(1 row)

asdlab=#
```

Figure 8.13: *Customer who placed most number of orders*

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

```
select c.* from customers as c,items as i,orders as o where o.itemid=i.itemid
and c.custid=o.custid and price>5000;
```

```
asdlab=# select c.* from customers as c,items as i,orders as o where
 custid | custname |   address   |   state
--------+----------+-------------+------------
    113 | soman    | puthumana   | kerala
    115 | mickey   | juhu        | maharastra
    114 | jaise    | kottarakara | kerala
(3 rows)

asdlab=#
```

Figure 8.14: *More than 5000*

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

```
(select custname,address from customers)
except
(select c.custname,c.address from customers as c ,orders as o, items as i
where o.itemid=i.itemid and c.custid=o.custid
and itemname='Samsung Galaxy S4' );
```

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

Figure 8.15: *Customers not ordered galaxy s4*

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

```
select * from customers left outer join orders on customers.custid=orders.custid;
select * from customers right outer join orders on customers.custid=orders.custid;
```

```
asdlab=# 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
    112 | patrick  | harinagar      | tamilnadu  |         |        |          |            |
(5 rows)

asdlab=#
```

Figure 8.16: *Left outer join*

```
asdlab=# 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
(4 rows)
```

Figure 8.17: *Right outer join*

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

```
select count(*),state from customers group by state;
```

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

asdlab=#
```

Figure 8.18: *Grouped by state*

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

```
select * from items where price in (select price from items group by price
having price>(select avg(price) from items group by category));
```

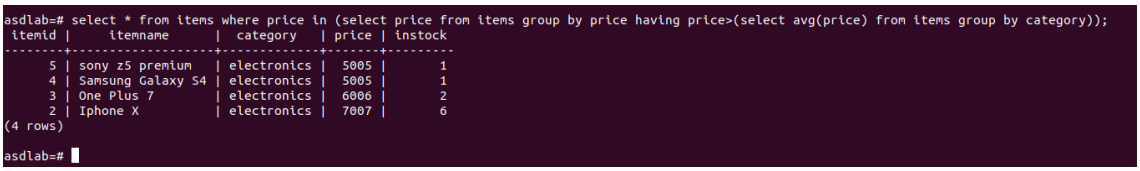

Figure 8.19: *price ¿ Average price*

8.4 Result

Implemented the programs using Join operation,set operation ,grouping and nested queries in Postgresql 11.5 (Ubuntu 11.5-1.pgdg18.04+1) in Ubuntu 18.04 and following output were obtained.

9 PL/PGSQL AND SEQUENCE

Cycle 2

Exp No 8

9.1 Aim

To study the basic pl/pgsql and sequence queries.

9.2 Theory

PL/pgSQL is a loadable procedural language for the PostgreSQL database system. The design goals of PL/pgSQL were to create a loadable procedural language that

- can be used to create functions and trigger procedures,
- adds control structures to the SQL language,
- can perform complex computations,
- inherits all user-defined types, functions, and operators,
- can be defined to be trusted by the server, is easy to use.

Functions created with PL/pgSQL can be used anywhere that built-in functions could be used. For example, it is possible to create complex conditional computation functions and later use them to define operators or use them in index expressions.

In PostgreSQL 9.0 and later, PL/pgSQL is installed by default.
However it is still a loadable module, so especially security-conscious administrators could choose to remove it.

9.3 Questions

Write PL/SQL programs for the following:

9.3.1 To print the first n prime numbers..

Code

```
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;
```

Output

```
select from prime(5);
```

```
postgres=# 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;
CREATE FUNCTION
postgres=# select from prime(5);
NOTICE:   2
NOTICE:   3
NOTICE:   5
NOTICE:   7
NOTICE:   11
--
(1 row)
```

Figure 9.1: *N prime numbers*

9.3.2 Display the Fibonacci series upto n terms

Code

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

Output

```
select * from fib(10);
```

```
postgres=# CREATE or REPLACE FUNCTION fib(fib INT) RETURNS VOID as
postgres-# $$
postgres$# DECLARE
postgres$# value INT=1;
postgres$# new INT=1;
postgres$# temp INT;
postgres$# count INT =1;
postgres$# BEGIN
postgres$# RAISE NOTICE ' %', value;
postgres$# RAISE NOTICE ' %', new;
postgres$# FOR count in 3..fib LOOP
postgres$# temp=new;
postgres$# new=new+value;
postgres$# value=temp;
postgres$# RAISE NOTICE ' %', new;
postgres$# END LOOP;
postgres$# END;
postgres$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# select from fib(10);
NOTICE:   1
NOTICE:   1
NOTICE:   2
NOTICE:   3
NOTICE:   5
NOTICE:   8
NOTICE:   13
NOTICE:   21
NOTICE:   34
NOTICE:   55
--
(1 row)
```

Figure 9.2: *N terms in Fibonacci sequence*

9.3.3 Assigning Grade

**Create a table named student_grade 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)**

create table student_grade(roll int primary key,name varchar(10),mark1 int,mark2 int, mark3 int ,grade varchar(4));

```
insert into student_grade values(1,'anu',50,45,48),(2,'manu',50,50,50),
(3,'ramu',35,40,40);

select * from student_grade;
```

```
asdlab=# select * from student_grade;
 roll | name | mark1 | mark2 | mark3 | grade
------+------+-------+-------+-------+-------
    1 | anu  |    50 |    45 |    48 |
    2 | manu |    50 |    50 |    50 |
    3 | ramu |    35 |    40 |    40 |
(3 rows)
```

Figure 9.3: *student_grade Table*

Code

```
CREATE or REPLACE FUNCTION set_student_grade() RETURNS VOID as
$$
BEGIN
update student_grade
set grade='pass'
where (mark1+mark2+mark3)/3  >40;
update student_grade
set grade='fail'
where (mark1+mark2+mark3)/3 <=40;
END;
$$ LANGUAGE plpgsql;
```

Output

```
select from set_student_grade;
select * from student_grade;
```

```
asdlab=# CREATE or REPLACE FUNCTION set_student_grade() RETURNS VOID as
asdlab-# $$
asdlab$# BEGIN
asdlab$# update student_grade
asdlab$# set grade='pass'
asdlab$# where (mark1+mark2+mark3)/3  >40;
asdlab$# update student_grade
asdlab$# set grade='fail'
asdlab$# where (mark1+mark2+mark3)/3 <=40;
asdlab$# END;
asdlab$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=# select set_student_grade();
 set_student_grade
-------------------

(1 row)

asdlab=# select * from student_grade;
 roll | name | mark1 | mark2 | mark3 | grade
------+------+-------+-------+-------+-------
    1 | anu  |    50 |    45 |    48 | pass
    2 | manu |    50 |    50 |    50 | pass
    3 | ramu |    35 |    40 |    40 | fail
(3 rows)

asdlab=#
```

Figure 9.4: *After setting Grade*

9.3.4 Circle and Area

Create table circle_area (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.

Code

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

Output

```
select from create_circle();
select * from circle;
```

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

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

asdlab=#
```

Figure 9.5: *Circle*

9.3.5 Insertion using Array

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

```
create table stud(name varchar(10),mark int);
```

Code

```
CREATE or REPLACE FUNCTION insert_stud() RETURNS VOID as
$$
DECLARE
data1 INT []= '{ 25,76,43,45,67,57,97,56,89,8 }';
 data2  VARCHAR(20)[]='{ ARUN,AMAL,PETER,JOSE,ANNIE,MARY,JOSEPH,MARK,MIDHUN,KEVIN}';
i INT ;
BEGIN
FOR i IN 1..10 LOOP
INSERT INTO stud VALUES(data2[i],data1[i]);
END LOOP;
END;
$$ LANGUAGE plpgsql;
```

Output

```
select from insert_stud();
select * from stud;
```

```
asdlab=# CREATE or REPLACE FUNCTION insert_stud() RETURNS VOID as
asdlab-# $$
asdlab$# DECLARE
asdlab$# data1 INT []= '{ 25,76,43,45,67,57,97,56,89,8 }';
asdlab$#  data2  VARCHAR(20)[]='{ ARUN,AMAL,PETER,JOSE,ANNIE,MARY,JOSEPH,MARK,MIDHUN,KEVIN}';
asdlab$# i INT ;
asdlab$# BEGIN
asdlab$#  FOR i IN 1..10 LOOP
asdlab$#   INSERT INTO stud VALUES(data2[i],data1[i]);
asdlab$#  END LOOP;
asdlab$# END;
asdlab$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=# select insert_stud();
 insert_stud
-------------

(1 row)

asdlab=# select * from stud;
  name   | mark
---------+------
 ARUN    |   25
 AMAL    |   76
 PETER   |   43
 JOSE    |   45
 ANNIE   |   67
 MARY    |   57
 JOSEPH  |   97
 MARK    |   56
 MIDHUN  |   89
 KEVIN   |    8
(10 rows)

asdlab#
```

Figure 9.6: *Array insertion*

9.3.6 Insertion using Array

A SEQUENCE USING PL/SQL. USE THIS SEQUENCE TO GENERATE THE PRIMARY KEY VALUES FOR A TABLE NAMED CLASS_CSE WITH ATTRIBUTES ROLL,NAME AND PHONE. INSERT SOME TUPLES USING PL/SQL PROGRAMMING.

```
Create table class_cse (roll int primary key,name varchar(10),phone varchar(15));
```

Code

```
CREATE  SEQUENCE csekey
START 101;

CREATE or REPLACE FUNCTION class_cse() RETURNS VOID as
$$
DECLARE
data1 VARCHAR []= '{ 0482-239091,0484-234562 ,0485-11234,0489-43617,0481-23145}';
 data2  VARCHAR(20)[]='{ ARUN,AMAL,PETER,JOSE,ANNIE }';
i INT ;
j INT;
BEGIN
FOR i IN 1..5 LOOP
INSERT INTO class_cse VALUES(nextval('csekey'),data2[i],data1[i]);
END LOOP;
END;
$$ LANGUAGE plpgsql;
```

Output

```
select from class_cse();
select * from class_cse;
```

```
asdlab=# CREATE  SEQUENCE csekey
asdlab-# START 101;
CREATE SEQUENCE
asdlab=#
asdlab=# CREATE or REPLACE FUNCTION class_cse() RETURNS VOID as
asdlab-# $$
asdlab$# DECLARE
asdlab$# data1 VARCHAR []= '{ 0482-239091,0484-234562 ,0485-11234,0489-43617,0481-23145}';
asdlab$#  data2  VARCHAR(20)[]='{ ARUN,AMAL,PETER,JOSE,ANNIE }';
asdlab$# i INT ;
asdlab$# j INT;
asdlab$# BEGIN
asdlab$#  FOR i IN 1..5 LOOP
asdlab$#   INSERT INTO class_cse VALUES(nextval('csekey'),data2[i],data1[i]);
asdlab$#  END LOOP;
asdlab$# END;
asdlab$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=# select class_cse();
 class_cse
-----------

(1 row)

asdlab=# select * from class_cse;
 roll | name  |    phone
------+-------+-------------
  101 | ARUN  | 0482-239091
  102 | AMAL  | 0484-234562
  103 | PETER | 0485-11234
  104 | JOSE  | 0489-43617
  105 | ANNIE | 0481-23145
(5 rows)

asdlab=#
```

Figure 9.7: *table class cse*

9.4 Result

Implemented the PL/PGSQL programs in Postgresql 11.5 (Ubuntu 11.5-1.pgdg18.04+1) in Ubuntu 18.04 and following output were obtained.

10 CURSOR

Cycle 2

Exp No 9

10.1 Aim

To study the use and implementation of cursors in PL/SQL.

10.2 Theory

A PL/PGSQL cursor allows us to encapsulate a query and process each individual row at a time. We use cursors when we want to divide a large result set into parts and process each part individually. If we process it at once, we may have a memory overflow error.

10.3 Questions

10.3.1 Grade calculation

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.

Table Creation

create table stdnt(id int,sname varchar(15),m1 int,m2 int,m3 int,gr char(1));

```
insert into stdnt (id,sname,m1,m2,m3) values(88,'anu',39,67,92);
insert into stdnt (id,sname,m1,m2,m3) values(10,'jan',58,61,29);
```

```
insert into stdnt (id,sname,m1,m2,m3) values(30,'karuna',87,79,77);
insert into stdnt (id,sname,m1,m2,m3) values(29,'jossy',39,80,45);


select * from stdnt;
```

```
asdlab=# select * from stdnt;
 id | sname  | m1 | m2 | m3 | gr
----+--------+----+----+----+----
 10 | jan    | 58 | 61 | 29 |
 30 | karuna | 87 | 79 | 77 |
 88 | anu    | 39 | 67 | 92 |
 29 | jossy  | 39 | 80 | 45 |
(4 rows)

asdlab=#
```

Figure 10.1: *stdnt Table*

Code

```
CREATE OR REPLACE FUNCTION get_grade()
   RETURNS void AS $$
DECLARE
    total INT ;
    grade char(1);
    rec_film   RECORD;
    cur_films CURSOR
       FOR SELECT * FROM stdnt;
BEGIN
   OPEN cur_films;
   LOOP
      FETCH cur_films INTO rec_film;
      EXIT WHEN NOT FOUND;
      total = rec_film.m1+rec_film.m2+rec_film.m3;
      IF total>240 THEN
          grade='A';
      ELSIF total>180 THEN
        grade='B';
```

```
      ELSIF total>120 THEN
        grade='C';
     ELSIF total>60 THEN
        grade='D';
    ELSE
           grade='F';
      END IF;
      update stdnt set gr=grade where m1=rec_film.m1;
   END LOOP;
   CLOSE cur_films;
END; $$
LANGUAGE plpgsql;
```

Output

```
select get_grade();
select * from stdnt;
```

```
asdlab=# CREATE OR REPLACE FUNCTION get_grade()
   RETURNS void AS $$
DECLARE
    total INT ;
    grade char(1);
    rec_film   RECORD;
    cur_films CURSOR
       FOR SELECT * FROM stdnt;
BEGIN
   OPEN cur_films;
   LOOP
      FETCH cur_films INTO rec_film;
      EXIT WHEN NOT FOUND;
      total = rec_film.m1+rec_film.m2+rec_film.m3;
      IF total>240 THEN
           grade='A';
      ELSIF total>180 THEN
  grade='B';
      ELSIF total>120 THEN
  grade='C';
     ELSIF total>60 THEN
  grade='D';
    ELSE
 grade='F';
      END IF;
      update stdnt set gr=grade where m1=rec_film.m1;
   END LOOP;
   CLOSE cur_films;

END; $$

LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=# select from get_grade();
--
(1 row)

asdlab=# select * from stdnt;
 id | sname  | m1 | m2 | m3 | gr
----+--------+----+----+----+----
 10 | jan    | 58 | 61 | 29 | C
 30 | karuna | 87 | 79 | 77 | A
 88 | anu    | 39 | 67 | 92 | C
 29 | jossy  | 39 | 80 | 45 | C
(4 rows)
```

Figure 10.2: *Assigned Grades*

10.3.2 Interest Calculation

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

Table Creation

create table bankdetails(accno int,name varchar(15),balance int,adate date);
create table banknew(accno int,interest int);

```
insert into bankdetails values(1001,'aby',3005,'10-oct-15');
insert into bankdetails values(1002,'alan',4000,'05-may-95');
insert into bankdetails values(1003,'amal',5000,'16-mar-92');
insert into bankdetails values(1004,'jeffin',3500,'01-apr-50');
insert into bankdetails values(1005,'majo',6600,'01-jan-01');
```

```
select * from bankdetails;
```

```
asdlab=# select * from bankdetails;
 accno |  name  | balance |   adate
-------+--------+---------+------------
  1001 | aby    |    3005 | 2015-10-10
  1005 | majo   |    6600 | 2001-01-01
  1002 | alan   |    4000 | 1995-05-05
  1003 | amal   |    5000 | 1992-03-16
  1004 | jeffin |    3500 | 2050-04-01
(5 rows)
```

Figure 10.3: *bankdetails Table*

Code

```
CREATE OR REPLACE FUNCTION get_interest()
   RETURNS void AS $$
DECLARE
    interest INT ;
    account   RECORD;
    movacc CURSOR
       FOR SELECT * FROM bankdetails;
BEGIN
   OPEN movacc;
   LOOP
      FETCH movacc INTO account;
      EXIT WHEN NOT FOUND;

      interest=0.08*account.balance;
      INSERT INTO banknew VALUES (account.accno,interest);
   END LOOP;
   CLOSE movacc;

END; $$

LANGUAGE plpgsql;
```

Output

```
select get_interest();
select * from banknew;
```

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

asdlab=# select * from banknew;
 accno | interest
-------+----------
  1001 |      240
  1005 |      528
  1002 |      320
  1003 |      400
  1004 |      280
(5 rows)

asdlab=#
```

Figure 10.4: *banknew table*

10.3.3 Finding Experienced People

Create table people_list (id, name, dt_joining, place).If persons experience is above 10 years, put the tuple in table exp_list (id, name, experience).)

Table creation

create table people_list(id INT, name varchar(20),dt_joining DATE,place varchar(20));
create table exp_list(id INT, name varchar(20),exp INT);

```
insert into people_list values(101,'Robert','03-APR-2005','CHY');
insert into people_list values(102,'Mathew','07-JUN-2008','CHY');
insert into people_list values(103,'Luffy','15-APR-2003','FSN');
insert into people_list values(104,'Lucci','13-AUG-2009','KTM');
insert into people_list values(105,'Law','14-APR-2005','WTC');
insert into people_list values(101,'Vivi','21-SEP-2010','ABA');
```

```
select * from people_list;
```

```
asdlab=# select * from people_list;
 id  |  name  | dt_joining | place
-----+--------+------------+-------
 101 | Robert | 2005-04-03 | CHY
 102 | Mathew | 2008-06-07 | CHY
 103 | Luffy  | 2003-04-15 | FSN
 104 | Lucci  | 2009-08-13 | KTM
 105 | Law    | 2005-04-14 | WTC
 101 | Vivi   | 2010-09-21 | ABA
(6 rows)

asdlab=#
```

Figure 10.5: *people_list Table*

Code

```
CREATE OR REPLACE FUNCTION set_exp()
   RETURNS void AS $$
DECLARE
    exp INT;
    proff   RECORD;
    today DATE;
    movproff CURSOR
       FOR SELECT * FROM people_list;
BEGIN
   OPEN movproff;
   SELECT current_date INTO today;
   LOOP
      FETCH movproff INTO proff;
      EXIT WHEN NOT FOUND;

      SELECT DATE_PART('year', today::date) - DATE_PART('year', proff.dt_joining::date
      IF exp>10 THEN
       INSERT INTO exp_list VALUES (proff.id,proff.name,exp);
      END IF;
   END LOOP;
   CLOSE movproff;

END; $$

LANGUAGE plpgsql;
```

Output

```
select from set_exp;
select * from exp_list;
```

```
asdlab=# CREATE OR REPLACE FUNCTION put_exp()
   RETURNS void AS $$
DECLARE
    exp INT;
    proff   RECORD;
    today DATE;
    movproff CURSOR
       FOR SELECT * FROM people_list;
BEGIN
   OPEN movproff;
   SELECT current_date INTO today;
   LOOP
      FETCH movproff INTO proff;
      EXIT WHEN NOT FOUND;

      SELECT DATE_PART('year', today::date) - DATE_PART('year', proff.dt_joining::date) INTO exp;
      IF exp>10 THEN
       INSERT INTO exp_list VALUES (proff.id,proff.name,exp);
      END IF;
   END LOOP;
   CLOSE movproff;

END; $$

LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=# select put_exp();
 put_exp
---------

(1 row)

asdlab=# select * from exp_list;
 id  |  name  | exp
-----+--------+-----
 101 | Robert |  14
 102 | Mathew |  11
 103 | Luffy  |  16
 105 | Law    |  14
(4 rows)
```

Figure 10.6: *more than 10 experienced*

10.3.4 Salary Increment

Create table employee_list(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%.

Table creation

create table emp_list(id INT,Name varchar(20),M_sal INT);

```
   insert into emp_list values(101,'Mathew',55000);
insert into emp_list values(102,'Jose',80000);
insert into emp_list values(103,'John',250000);
insert into emp_list values(104,'Ann',600000);

   select * from emp_list;
```

```
asdlab=# select * from emp_list;
 id  |  name  | m_sal
-----+--------+--------
 101 | Mathew |  55000
 102 | Jose   |  80000
 103 | John   | 250000
 104 | Ann    | 600000
(4 rows)

asdlab=#
```

Figure 10.7: *emp_list Table*

Code

```
CREATE OR REPLACE FUNCTION sal_incre()
   RETURNS void AS $$
DECLARE
    yearsal INT;
    monsal INT;
    sal   RECORD;
    movsal CURSOR
       FOR SELECT * FROM emp_list;
BEGIN
   OPEN movsal;
   LOOP
      FETCH movsal INTO sal;
      EXIT WHEN NOT FOUND;
      yearsal=sal.m_sal*12;
      monsal=sal.m_sal;

        IF yearsal>500000 THEN
           UPDATE emp_list SET m_sal=monsal*1.1 WHERE m_sal=monsal;
        ELSIF yearsal>200000 THEN
        UPDATE emp_list SET m_sal=monsal*1.15 WHERE m_sal=monsal;
        ELSIF yearsal>60000 THEN
         UPDATE emp_list SET m_sal=monsal*1.2WHERE m_sal=monsal;
        ELSE
         UPDATE emp_list SET m_sal=monsal*1.25 WHERE m_sal=monsal;
     END IF;
   END LOOP;
   CLOSE movsal;

END; $$

LANGUAGE plpgsql;
```

Output

```
select from sal_incre();
select * from emp_list;
```

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

Figure 10.8: *salary increment function*

```
asdlab=# select * from emp_list;
 id  |  name  | m_sal
-----+--------+--------
 101 | Mathew |  60500
 102 | Jose   |  88000
 103 | John   | 275000
 104 | Ann    | 660000
(4 rows)

asdlab=#
```

Figure 10.9: *emp_list table updated*

10.4 Result

Implemented the PL/PGSQL CURSOR programs in Postgresql 11.5 (Ubuntu 11.5-1.pgdg18.04+1
in Ubuntu 18.04 and following output were obtained.

11 TRIGGER AND EXCEPTION HANDLING

Cycle 2

Exp No 10

11.1 Aim

To study PL/SQL trigger and exception handling.

11.2 Theory

Triggers are procedures that are stored in the database and implicitly run, or fired, when something happens
Exceptions are used to handle run time errors in program

11.3 Questions

11.3.1 Trigger whenever data is inserted

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

Table Creation

CREATE TABLE customer_details (cust_id int UNIQUE,cust_name varchar(25),address varchar(30));

Code

```
CREATE OR REPLACE FUNCTION cust_det_insert() RETURNS TRIGGER AS
$cust_det_insert$
    BEGIN
     RAISE NOTICE 'A row is inserted';
RETURN NEW;
    END;
$cust_det_insert$
LANGUAGE plpgsql;
CREATE TRIGGER cust_det_insert
AFTER INSERT ON customer_details
    FOR EACH STATEMENT EXECUTE PROCEDURE cust_det_insert();
```

Output

```
INSERT INTO customer_details VALUES(1,'John','Ezhaparambbil');
```

```
asdlab=# delete from customer_details;
DELETE 1
asdlab=# CREATE OR REPLACE FUNCTION cust_det_insert() RETURNS TRIGGER AS
asdlab-# $cust_det_insert$
asdlab$#     BEGIN
asdlab$#       RAISE NOTICE 'A row is inserted';
asdlab$#   RETURN NEW;
asdlab$#     END;
asdlab$# $cust_det_insert$
asdlab-# LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=# CREATE TRIGGER cust_det_insert
asdlab-# AFTER INSERT ON customer_details
asdlab-#     FOR EACH STATEMENT EXECUTE PROCEDURE cust_det_insert();
ERROR:  trigger "cust_det_insert" for relation "customer_details" already exists
asdlab=# INSERT INTO customer_details VALUES(1,'John','Ezhaparambbil');
NOTICE:  A row is inserted
INSERT 0 1
asdlab=#
```

Figure 11.1: *Data is inserted*

11.3.2 Message when salary >20000

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

Table Creation

CREATE TABLE emp_details(empid INT UNIQUE,empname varchar(20),salary int);

Code

```
CREATE OR REPLACE FUNCTION emp_sal_check() RETURNS trigger AS $emp_sal$
    BEGIN
        IF NEW.salary >20000 THEN
           RAISE NOTICE 'Employee % has salary greater than 20000 ',NEW.empname;
        END IF;
        RETURN NEW;
    END;
$emp_sal$ LANGUAGE plpgsql;

CREATE TRIGGER emp_sal AFTER INSERT OR UPDATE ON emp_details
    FOR EACH ROW EXECUTE PROCEDURE emp_sal_check();
```

Output

```
INSERT INTO emp_details VALUES(1,'John',25000);
```

```
asdlab=# CREATE OR REPLACE FUNCTION emp_sal_check() RETURNS trigger AS $emp_sal$
asdlab$#     BEGIN
asdlab$#         IF NEW.salary >20000 THEN
asdlab$#            RAISE NOTICE 'Employee % has salary greater than 20000 ',NEW.empname;
asdlab$#         END IF;
asdlab$#         RETURN NEW;
asdlab$#     END;
asdlab$# $emp_sal$ LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=#
asdlab=# CREATE TRIGGER emp_sal AFTER INSERT OR UPDATE ON emp_details
asdlab-#     FOR EACH ROW EXECUTE PROCEDURE emp_sal_check();
CREATE TRIGGER
asdlab=# INSERT INTO emp_details VALUES(1,'John',25000);
NOTICE:  Employee John has salary greater than 20000
INSERT 0 1
asdlab=#
```

Figure 11.2: *Salary >20000*

11.3.3 Row count

Create a trigger w.r.tcustomer_detailstable.
Increment the value of count_row (in cust_count table) whenever a new tuple is inserted and decrement the value of count_row when a tuple is deleted.

Initial value of the count_row is set to 0.

Table creation

CREATE TABLE cust_count(count_row int);

```
insert into cust_count VALUES(0);
```

Code

```
CREATE OR REPLACE FUNCTION cust_count() RETURNS trigger AS $cust_count$
DECLARE
count INT;
BEGIN
SELECT * FROM cust_count INTO count;
  IF (TG_OP = 'DELETE') THEN
IF count !=0 THEN
UPDATE  cust_count SET count_row=count_row-1;
END IF;
ELSIF (TG_OP = 'INSERT') THEN
UPDATE  cust_count SET count_row=count_row+1;
END IF;
RETURN NEW;
END;
$cust_count$ LANGUAGE plpgsql;

CREATE TRIGGER cust_count_change
AFTER INSERT OR DELETE ON customer_details
    FOR EACH ROW EXECUTE PROCEDURE cust_count();
```

Output

```
asdlab=# CREATE OR REPLACE FUNCTION cust_count() RETURNS trigger AS $cust_count$
asdlab$# DECLARE
asdlab$#  count INT;
asdlab$# BEGIN
asdlab$#  SELECT * FROM cust_count INTO count;
asdlab$#   IF (TG_OP = 'DELETE') THEN
asdlab$#   IF count !=0 THEN
asdlab$#    UPDATE  cust_count SET count_row=count_row-1;
asdlab$#   END IF;
asdlab$#  ELSIF (TG_OP = 'INSERT') THEN
asdlab$#   UPDATE  cust_count SET count_row=count_row+1;
asdlab$#  END IF;
asdlab$#  RETURN NEW;
asdlab$# END;
asdlab$# $cust_count$ LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=#
asdlab=# CREATE TRIGGER cust_count_change
asdlab-# AFTER INSERT OR DELETE ON customer_details
asdlab-#     FOR EACH ROW EXECUTE PROCEDURE cust_count();
ERROR:  trigger "cust_count_change" for relation "customer_details" already exists
asdlab=# INSERT INTO customer_details VALUES(1,'John','Ezhaparambbil');
NOTICE:  A row is inserted
INSERT 0 1
asdlab=# INSERT INTO customer_details VALUES(2,'Pretty','Thenganachalil');
NOTICE:  A row is inserted
INSERT 0 1
asdlab=# select * from cust_count;
 count_row
-----------
         2
(1 row)

asdlab=# delete from customer_details where cust_id=1;
DELETE 1
asdlab=# select * from cust_count;
 count_row
-----------
         1
(1 row)

asdlab=#
```

Figure 11.3: *Row count*

11.3.4 Deletion and Updating

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

Table creation

CREATE TABLE deleted(empid INT ,empname varchar(20),salary int);
CREATE TABLE updated(empid INT,empname varchar(20),salary int);

Code

```
CREATE OR REPLACE FUNCTION del_upd() RETURNS trigger AS $del_upd$
BEGIN
  IF (TG_OP = 'DELETE') THEN
INSERT INTO deleted VALUES(OLD.empid,OLD.empname,OLD.salary);
ELSIF (TG_OP = 'UPDATE') THEN
     INSERT INTO updated VALUES(OLD.empid,OLD.empname,OLD.salary);
END IF;
RETURN NEW;
END;
$del_upd$ LANGUAGE plpgsql;

CREATE TRIGGER del_upd
AFTER UPDATE OR DELETE ON emp_details
    FOR EACH ROW EXECUTE PROCEDURE del_upd();
```

Output

```
asdlab=# CREATE TRIGGER del_upd
asdlab-# AFTER UPDATE OR DELETE ON emp_details
asdlab-# ^C
asdlab=# CREATE OR REPLACE FUNCTION del_upd() RETURNS trigger AS $del_upd$
asdlab$# BEGIN
asdlab$#   IF (TG_OP = 'DELETE') THEN
asdlab$#    INSERT INTO deleted VALUES(OLD.empid,OLD.empname,OLD.salary);
asdlab$#  ELSIF (TG_OP = 'UPDATE') THEN
asdlab$#       INSERT INTO updated VALUES(OLD.empid,OLD.empname,OLD.salary);
asdlab$#  END IF;
asdlab$#  RETURN NEW;
asdlab$# END;
asdlab$# $del_upd$ LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=#
asdlab=# CREATE TRIGGER del_upd
asdlab-# AFTER UPDATE OR DELETE ON emp_details
asdlab-#     FOR EACH ROW EXECUTE PROCEDURE del_upd();
CREATE TRIGGER
asdlab=# UPDATE emp_details SET salary=salary+20000 WHERE empid=1;
NOTICE:  Employee John has salary greater than 20000
UPDATE 1
asdlab=# select * from updated;
 empid | empname | salary
-------+---------+--------
     1 | John    |  25000
(1 row)

asdlab=# DELETE FROM emp_details where empid=2;
DELETE 1
asdlab=# select * from deleted;
 empid | empname  | salary
-------+----------+--------
     2 | prageesh |  20000
(1 row)

asdlab=#
```

Figure 11.4: *Deleted and Updated table*

11.3.5 Divide zero Exception

Write a PL/SQL to show divide by zero exception

Code

```
CREATE OR REPLACE FUNCTION div(a INT,b INT) RETURNS INT as
$$
DECLARE
result INT;
BEGIN
IF b=0 THEN
RAISE EXCEPTION  'DVIDE BY ZERO  ' ;
ELSE
result=a/b;
RETURN result ;
END IF;
END;
$$
LANGUAGE plpgsql;
```

Output

```
asdlab=# select * from div(6,0);
ERROR:  DEVIDE BY ZERO
CONTEXT:  PL/pgSQL function div(integer,integer) line 6 at RAISE
asdlab=# CREATE OR REPLACE FUNCTION div(a INT,b INT) RETURNS INT as
asdlab-# $$
asdlab$# DECLARE
asdlab$#  result INT;
asdlab$# BEGIN
asdlab$#  IF b=0 THEN
asdlab$#   RAISE EXCEPTION  'DVIDE BY ZERO  ' ;
asdlab$#  ELSE
asdlab$#   result=a/b;
asdlab$#   RETURN result ;
asdlab$#  END IF;
asdlab$# END;
asdlab$# $$
asdlab-# LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=# select * from div(6,2);
 div
-----
   3
(1 row)

asdlab=# select * from div(6,0);
ERROR:  DVIDE BY ZERO
CONTEXT:  PL/pgSQL function div(integer,integer) line 6 at RAISE
asdlab=#
```

Figure 11.5: *Divide By Zero*

11.3.6 No Data Found Exception

Write a PL/SQL to show no data found exception

Code

```
CREATE OR REPLACE FUNCTION get_the_sal(id INT) RETURNS INT as
$$
DECLARE
result INT;
BEGIN
SELECT salary INTO result FROM emp_details WHERE empid=id;
IF RESULT IS NULL THEN
RAISE EXCEPTION 'NO DATA FOUND' ;
ELSE
RETURN result;
END IF;
END;
$$
LANGUAGE plpgsql;
```

Output

```
asdlab=# insert into emp_details values(2,'prageesh',20000);
INSERT 0 1
asdlab=# CREATE OR REPLACE FUNCTION get_the_sal(id INT) RETURNS INT as
asdlab-# $$
asdlab$# DECLARE
asdlab$#  result INT;
asdlab$# BEGIN
asdlab$#  SELECT salary INTO result FROM emp_details WHERE empid=id;
asdlab$#  IF RESULT IS NULL THEN
asdlab$#   RAISE EXCEPTION 'NO DATA FOUND' ;
asdlab$#  ELSE
asdlab$#   RETURN result;
asdlab$#  END IF;
asdlab$# END;
asdlab$# $$
asdlab-# LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=#
asdlab=# select * from get_the_sal(1);
 get_the_sal
-------------
       45000
(1 row)

asdlab=# select * from get_the_sal(2);
 get_the_sal
-------------
       20000
(1 row)

asdlab=# select * from get_the_sal(3);
ERROR:  NO DATA FOUND
CONTEXT:  PL/pgSQL function get_the_sal(integer) line 7 at RAISE
asdlab=#
```

Figure 11.6: *No Data Found*

11.3.7 Wrong Ebill

Create a table with ebill(cname,prevreading,currreading). If prevreading = currreading then raise an exception Data Entry Error.

Table creation

CREATE TABLE ebill(cname varchar(20),preread int,curread int);

Code

```
CREATE OR REPLACE FUNCTION check_reading() RETURNS TRIGGER AS
$checkread$
BEGIN
IF NEW.preread=NEW.curread THEN
RAISE EXCEPTION 'DATA ENTRY ERROR A % B %' ,NEW.preread,NEW.curread;
ELSE
RAISE NOTICE 'STATEMENT PROCESSED'  ;
END IF;
RETURN NEW;
END;
$checkread$
LANGUAGE plpgsql;

CREATE TRIGGER check_reading
BEFORE INSERT ON ebill
FOR EACH ROW EXECUTE PROCEDURE check_reading();
```

Output

```
asdlab=# CREATE OR REPLACE FUNCTION check_reading() RETURNS TRIGGER AS
asdlab-# $checkread$
asdlab$# BEGIN
asdlab$#  IF NEW.preread=NEW.curread THEN
asdlab$# RAISE EXCEPTION 'DATA ENTRY ERROR A % B %' ,NEW.preread,NEW.curread;
asdlab$#  ELSE
asdlab$#   RAISE NOTICE 'STATEMENT PROCESSED'  ;
asdlab$#  END IF;
asdlab$#  RETURN NEW;
asdlab$# END;
asdlab$# $checkread$
asdlab-# LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=#
asdlab=# CREATE TRIGGER check_reading
asdlab-# BEFORE INSERT ON ebill
asdlab-# FOR EACH ROW EXECUTE PROCEDURE check_reading();
ERROR:  trigger "check_reading" for relation "ebill" already exists
asdlab=# INSERT INTO ebill VALUES('devi',100,100);
ERROR:  DATA ENTRY ERROR A 100 B 100
CONTEXT:  PL/pgSQL function check_reading() line 4 at RAISE
asdlab=# INSERT INTO ebill VALUES('devi',100,110);
NOTICE:  STATEMENT PROCESSED
INSERT 0 1
asdlab=#
```

Figure 11.7: *Incorrect Reading*

11.4 Result

Implemented the PL/PGSQL programs for trigger and exception in Postgresql 11.5 (Ubuntu 11.5-1.pgdg18.04+1) in Ubuntu 18.04 and following output were obtained.

12 PROCEDURES , FUNCTIONS & SCHEMA

Cycle 2

Exp No 11

12.1 Aim

To study PL/PGSQL Procedures and functions.

12.2 Theory

FUNCTIONS

A function is a subprogram that computes a value. Functions and procedures are structured alike, except that functions have a RETURN clause. You write functions using the syntax

```
FUNCTION name [(parameter[, parameter, ...])]
    RETURN datatype IS [local declarations]
BEGIN
    executable statements
[EXCEPTION
    exception handlers]
END
[name];
```

PROCEDURES

A procedure is a subprogram that performs a specific action. You write procedures using the syntax

```
  PROCEDURE name [(parameter[,parameter, ...])] IS [local declarations]
BEGIN
```

```
executable statements
[EXCEPTION
    exception handlers]
END
[name];
```

12.3 Questions

12.3.1 Factorial of a number

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

Code

```
CREATE OR REPLACE FUNCTION fact(fact INT) RETURNS INT AS
$$
DECLARE
    count INT = 1;
    result INT = 1;
BEGIN
    FOR count IN 1..fact LOOP
    result = result* count;
    END LOOP;
RETURN result;
END;
$$ LANGUAGE plpgsql;
```

Output

```
SELECT * FROM fact(n);
```

```
asdlab=# CREATE OR REPLACE FUNCTION fact(fact INT) RETURNS INT AS
asdlab-# $$
asdlab$# DECLARE
asdlab$# count INT = 1;
asdlab$# result INT = 1;
asdlab$# BEGIN
asdlab$# FOR count IN 1..fact LOOP
asdlab$# result = result* count;
asdlab$# END LOOP;
asdlab$# RETURN result;
asdlab$# END;
asdlab$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=# select * from fact(0);
 fact
------
    1
(1 row)

asdlab=# select * from fact(6);
 fact
------
  720
(1 row)
```

Figure 12.1: *Factorial of a number*

12.3.2 Boost Marks

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

Table Creation

```
TABLE student_details(roll int,marks int, phone int);

INSERT INTO student_details VALUES(1,70,9496947423);
INSERT INTO student_details VALUES(2,85,9495941358);
INSERT INTO student_details VALUES(3,78,8281865009);
```

Code

```
CREATE OR REPLACE PROCEDURE boost()
```

```
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE student_details SET marks=marks*1.05;
END;
$$;
```

Output

```
CALL boost();
```

```
asdlab=# CREATE OR REPLACE PROCEDURE boost()
asdlab-# LANGUAGE plpgsql
asdlab-# AS $$
asdlab$# BEGIN
asdlab$# UPDATE student_details SET marks=marks*1.05;
asdlab$# END;
asdlab$# $$;
CREATE PROCEDURE
asdlab=# CALL boost();
CALL
asdlab=# select * from student_details;
 roll | marks |   phone
------+-------+------------
    1 |    74 | 9496947712
    2 |    89 | 9495941120
    3 |    82 | 8281865216
(3 rows)

asdlab=#
```

Figure 12.2: *Boost Marks*

12.3.3 Finding Total and grade

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.

Table creation

CREATE TABLE studentmark(id int, name varchar(10), m1 int, m2 int, m3 int, total int, grade varchar(1));

Code

```
CREATE OR REPLACE FUNCTION insert_stud(id INT ,name varchar(20),m1 INT, m2 INT, m3 INT
RETURNS VOID AS
$$
DECLARE
total INT;
grade CHAR;
BEGIN
total=m1+m2+m3;
INSERT INTO studentmark VALUES(id,name,m1,m2,m3,total);
IF  total >=240 THEN
grade='A';
ELSIF total >=180 THEN
grade='B';
ELSIF total>=120 THEN
grade='C';
ELSIF  total>=60 THEN
grade = 'D' ;
ELSE
grade ='F';
END IF;
CALL insert_grade(id,grade);
END;
$$
LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE insert_grade(sid INT ,sgrade CHAR)
LANGUAGE plpgsql
AS $$
BEGIN
UPDATE studentmark SET grade=sgrade WHERE id=sid;
END;
$$;
```

Output

```
asdlab=# CREATE OR REPLACE FUNCTION insert_stud(id INT ,name varchar(20),m1 INT, m2 INT, m3 INT)
asdlab-# RETURNS VOID AS
asdlab-# $$
asdlab$# DECLARE
asdlab$#  total INT;
asdlab$#  grade CHAR;
asdlab$# BEGIN
asdlab$#  total=m1+m2+m3;
asdlab$#  INSERT INTO studentmark VALUES(id,name,m1,m2,m3,total);
asdlab$#  IF  total >=240 THEN
asdlab$#   grade='A';
asdlab$#  ELSIF total >=180 THEN
asdlab$#   grade='B';
asdlab$#  ELSIF total>=120 THEN
asdlab$#   grade='C';
asdlab$#  ELSIF  total>=60 THEN
asdlab$#   grade = 'D' ;
asdlab$#  ELSE
asdlab$#   grade ='F';
asdlab$#  END IF;
asdlab$#  CALL insert_grade(id,grade);
asdlab$# END;
asdlab$# $$
asdlab-# LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=# CREATE OR REPLACE PROCEDURE insert_grade(sid INT ,sgrade CHAR)
asdlab-# LANGUAGE plpgsql
asdlab-# AS $$
asdlab$# BEGIN
asdlab$#   UPDATE studentmark SET grade=sgrade WHERE id=sid;
asdlab$# END;
asdlab$# $$;
CREATE PROCEDURE
asdlab=# select from insert_stud(1,'raju',90,90,90);
--
(1 row)

asdlab=# select * from studentmark;
 id | name | m1 | m2 | m3 | total | grade
----+------+----+----+----+-------+-------
  1 | raju | 90 | 90 | 90 |   270 | A
(1 row)

asdlab=#
```

Figure 12.3: *Total Marks and Grade*

12.3.4 Schema

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

Code

```
CREATE SCHEMA pk1;

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

CREATE  OR REPLACE PROCEDURE pk1.proc2(num1 REAL)
LANGUAGE plpgsql
AS
$$
DECLARE
root REAL;
BEGIN
root=sqrt(num1);
RAISE NOTICE 'Root of % is %' ,num1,root;
END;
$$;
```

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

CREATE OR REPLACE FUNCTION pk1.fn22(num1 REAL,num2 REAL, num3 REAL) RETURNS VOID AS
$$
DECLARE
sum REAL ;
BEGIN
sum = num1+num2+num3;
RAISE NOTICE 'Sum of % ,%,% is %',num1,num2,num3,sum;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION pk1.all(num1 REAL,num2 REAL, num3 REAL)
 RETURNS VOID AS
$$
DECLARE
BEGIN
CALL pk1.proc1(num1,num2);
CALL pk1.proc2(num1);
PERFORM pk1.fn11(num1);
PERFORM pk1.fn22(num1,num2,num3);
END;
$$ LANGUAGE plpgsql;
```

Output

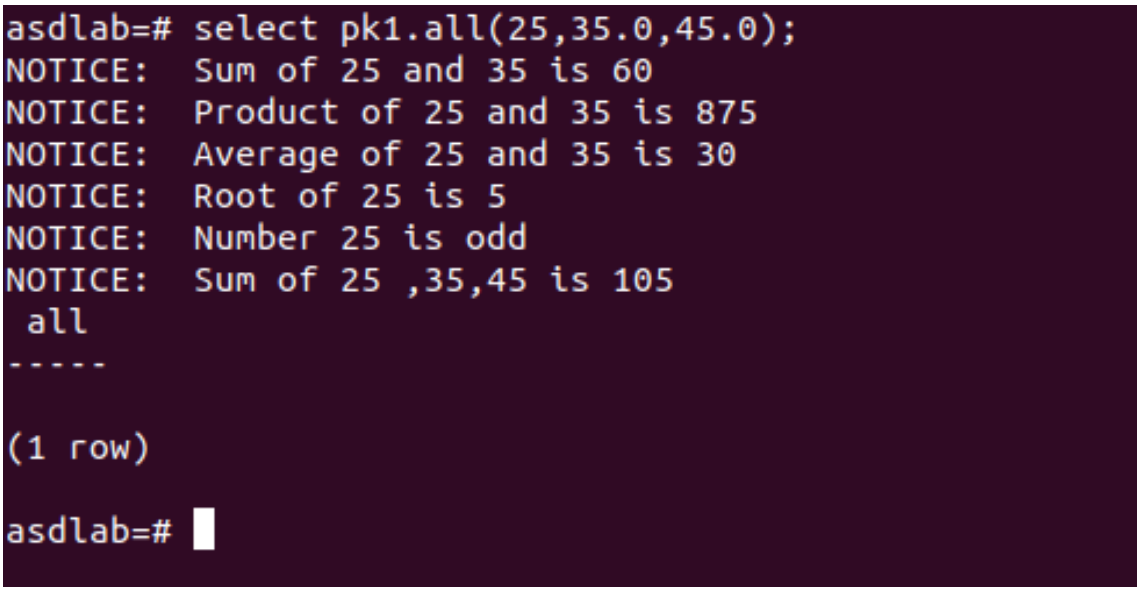

Figure 12.4: *Functions and Procedure called together*

12.4 Result

Implemented the PL/PGSQL programs using procedures , functions and schema in Postgresql 11.5 (Ubuntu 11.5-1.pgdg18.04+1) in Ubuntu 18.04 and following output were obtained.