

PROGRAMMING LAB-IV

RDBMS(USING POSTGRES SQL)

PROGRAM NO:1

CUSTOMER INFORMATION**AIM:**

Create a table customer (cust_no varchar (5), cust_name varchar (15),
age number, phone varchar (10))

- insert 5 records and display it
- add new field d_birth with date datatype
- create another table cust_phone with fields cust_name and phone from customer table
- remove the field age
- change the size of the cust_name to 25
- delete all the records from the table
- rename the table cutomer to cust
- drop the table

Queries:

```
postgres=# create table customer(cust_no varchar(5),cust_name
varchar(15),age in
t,phone varchar(15));
CREATE TABLE
```

```
postgres=# \d customer;
```

Column	Type	Collation	Nullable	Default
cust_no	character varying(5)			
cust_name	character varying(15)			
age	integer			
phone	character varying(15)			

b)add new field d_birth with date datatype

```
postgres=# insert into customer values('C001','A',10,'98765');
INSERT 0 1
postgres=# insert into customer values('C002','B',11,'43210');
```

INSERT 0 1

```
postgres=# insert into customer values('C003','C',12,'12345');
```

INSERT 0 1

```
postgres=# insert into customer values('C004','D',13,'67890');
```

INSERT 0 1

```
postgres=# insert into customer values('C005','E',14,'24680');
```

INSERT 0 1

```
postgres=# select * from customer;
```

```
cust_no | cust_name | age | phone
```

```
-----+-----+-----+-----
```

```
C001   | A         | 10 | 98765
```

```
C002   | B         | 11 | 43210
```

```
C003   | C         | 12 | 12345
```

```
C004   | D         | 13 | 67890
```

```
C005   | E         | 14 | 24680
```

(5 rows)

b)add new field d_birth with date datatype

```
postgres=# alter table customer add d_birth date;
```

ALTER TABLE

```
postgres=# \d customer;
```

```
postgres=# \d customer;
```

Column	Type	Collation	Nullable	Default
cust_no	character varying(5)			
cust_name	character varying(15)			
age	integer			
phone	character varying(15)			
d_birth	date			

c)create another table cust_phone with fields cust_name and phone from customer table

```
postgres=# create table cust_phone as (select cust_name,phone from customer);
```

SELECT 5

```
postgres=# select * from cust_phone;
cust_name | phone
```

```
-----+-----
A         | 98765
B         | 43210
C         | 12345
D         | 67890
E         | 24680
(5 rows)
```

d):remove the field age

```
postgres=# alter table customer drop column age;
ALTER TABLE
```

Table "public.customer"				
Column	Type	Collation	Nullable	Default
cust_no	character varying(5)			
cust_name	character varying(15)			
phone	character varying(15)			
d_birth	date			

e)change the size of the cust_name to 25

```
postgres=# alter table customer alter column cust_name type varchar(25);
ALTER TABLE
```

```
postgres=# \d customer;
```

Table "public.customer"				
Column	Type	Collation	Nullable	Default
cust_no	character varying(5)			
cust_name	character varying(25)			
phone	character varying(15)			
d_birth	date			

g)delete all the records from the table

```
postgres=# delete from customer;
```

DELETE 5

```
postgres=# select * from customer;
cust_no | cust_name | phone | d_birth
```

```
-----+-----+-----+-----
```

```
(0 rows)
```

g)rename the table cutomer to cust

```
postgres=# alter table customer rename to cust;
ALTER TABLE
```

```
postgres=# \d customer;
Did not find any relation named "customer".
postgres=# \d cust;
```

Table "public.customer"				
Column	Type	Collation	Nullable	Default
cust_no	character varying(5)			
cust_name	character varying(25)			
phone	character varying(15)			
d_birth	date			

i. drop the table

```
postgres=# drop table cust;
DROP TABLE
postgres=# \d cust;
Did not find any relation named "cust".
```

```
postgres=# drop table cust_phone;
DROP TABLE
postgres=# \d cust_phone;
Did not find any relation named "cust_phone".
```

PROGRAM NO:2

SALESMAN INFORMATION**AIM:**

Create a table sale_man (salesman_no primary key, s_name not null,
place, phone unique)

Create table sales_order (order_no
primary key order_date
not null

salesman_no foreign key references salesman_no in
sales_man del_type values should be either P or F (check
constraints)

order_status values should be 'Inprocess','Fullfilled','Backorder',
'Cancelled' (check constraints))

a)Insert few records in both tables

b>Delete primary key from sales_man table

c>Delete Foreign key and Check constraints from sales_order table

d)Add primary key in sales_man using ALTER TABLE

e)Add foreign key and CHECK constraints in sales_order table using
ALTER TABLE

QUERIES:

postgres=# create table sales_man(salesman_no varchar(5)

primarykey,s_name varchar(20) not null,place varchar(20),phone varchar(15)
unique);

CREATE TABLE

postgres=# \d sales_man;

Column	Type	Collation	Nullable	Default
salesman_no	character varying(5)		not null	
s_name	character varying(20)		not null	
place	character varying(20)			
phone	character varying(15)			

Indexes:

"sales_man_pkey" PRIMARY KEY, btree (salesman_no)

"sales_man_phone_key" UNIQUE CONSTRAINT, btree (phone)

Referenced by:

TABLE "sales_order" CONSTRAINT "sales_order_salesman_no_fkey"
salesman_no) REFERENCES sales_man(salesman_no)

```
postgres=# create table sales_order(order_no varchar(5) primary key,
order_date date not null,salesman_no VARCHAR(5) references
sales_man(salesman_no),del_type varchar(5) check(del_type in
('P','F')),order_status varchar(15) constraint chk_order check(order_status in
('inprocess','fullfilled','backorder','cancelled')));
CREATE TABLE
```

```
postgres=# \d sales_order;
```

```
postgres=# \d sales_man;
          table "public.sales_man"
   Column |          Type          | Collation | Nullable | Default
-----+-----+-----+-----+-----
salesman_no | character varying(5) |           | not null |
s_name      | character varying(20) |           | not null |
place       | character varying(20) |           |          |
phone       | character varying(15) |           |          |
Indexes:
    "sales_man_pkey" PRIMARY KEY, btree (salesman_no)
    "sales_man_phone_key" UNIQUE CONSTRAINT, btree (phone)
Referenced by:
    TABLE "sales_order" CONSTRAINT "sales_order_salesman_no_fkey" FOREIGN KEY (salesman_no) REFERENCES sales_man(salesman_no)
```

a)Insert few records in both tables

```
postgres=# insert into sales_man values('S1','A','Calicut',2468);
```

```
INSERT 0 1
```

```
postgres=# insert into sales_man values('S2','B','Koyilandy',1357);
```

```
INSERT 0 1
```

```
postgres=#
```

```
postgres=# insert into sales_man values('S3','C','Kochi',3579);
```

```
INSERT 0 1
```

```
postgres=# insert into sales_man values('S4','D','Kottayam',4681);
```

```
INSERT 0 1
```

```
postgres=# insert into sales_man values('S5','E','Palakkad',5791);
```

```
INSERT 0 1
```

```
postgres=# select * from sales_man;
```

```
salesman_no | s_name | place | phone
-----+-----+-----+-----
S1          | A      | Calicut | 2468
```

```

S2      | B      | Koyilandy | 1357
S3      | C      | Kochi     | 3579
S4      | D      | Kottayam  | 4681
S5      | E      | Palakkad  | 5791

```

(5 rows)

```

postgres=# insert into sales_order values('1','4-JAN-2020','S1','P','inprocess');
INSERT 0 1

```

```

postgres=# insert into sales_order values('2','15-JAN-2020','S2','F','fullfilled');
INSERT 0 1

```

```

postgres=# insert into sales_order values('3','20-JAN-2020','S3','P','fullfilled');
INSERT 0 1

```

```

postgres=# insert into sales_order values('4','4-NOV-2020','S4','F','cancelled');
INSERT 0 1

```

```

postgres=# insert into sales_order values('5','10-SEP-2020','S5','P','inprocess');
INSERT 0 1

```

order_no	order_date	salesman_no	del_type	order_status
1	2020-01-04	S1	P	inprocess
2	2020-01-15	S2	F	fullfilled
3	2020-01-20	S3	P	fullfilled
4	2020-11-04	S4	F	cancelled
5	2020-09-10	S5	P	inprocess

(5 rows)

c) Delete Foreign key constraints from sales_order table

```

postgres=# alter table sales_order drop constraint
sales_order_salesman_no_fkey;

```

ALTER TABLE

```

Table "public.sales_order"
Column | Type | Collation | Nullable | Default
-----+-----+-----+-----+-----
order_no | character varying(5) | | not null |
order_date | date | | not null |
salesman_no | character varying(5) | | |
del_type | character varying(5) | | |
order_status | character varying(15) | | |
Indexes:
    "sales_order_pkey" PRIMARY KEY, btree (order_no)
Check constraints:
    "chk_order" CHECK (order_status::text = ANY (ARRAY['inprocess'::character varying, 'fullfilled'::character varying, 'backorder'::character varying, 'cancelled'::character varying]::text[]))
    "sales_order_del_type_check" CHECK (del_type::text = ANY (ARRAY['P'::character varying, 'F'::character varying]::text[]))

```


d) Delete primary key from sales_man table

```
postgres=# alter table sales_man drop constraint sales_man_pkey ;
ALTER TABLE
postgres=# \d sales_man;
```

Column	Type	Collation	Nullable	Default
salesman_no	character varying(5)		not null	
s_name	character varying(20)		not null	
place	character varying(20)			
phone	character varying(15)			

Indexes:
 "sales_man_phone_key" UNIQUE CONSTRAINT, btree (phone)

d) Delete Check constraints from sales_order table

```
postgres=# alter table sales_order drop constraint chk_order;
ALTER TABLE
postgres=# alter table sales_order drop constraint
sales_order_del_type_check;
ALTER TABLE
postgres=# \d sales_order;
```

Column	Type	Collation	Nullable	Default
order_no	character varying(5)		not null	
order_date	date		not null	
salesman_no	character varying(5)			
del_type	character varying(5)			
order_status	character varying(15)			

Indexes:
 "sales_order_pkey" PRIMARY KEY, btree (order_no)

e) Add primary key in sales_man using ALTER TABLE

```
postgres=# alter table sales_man add primary key(salesman_no);
ALTER TABLE
```

Column	Type	Collation	Nullable	Default
salesman_no	character varying(5)		not null	
s_name	character varying(20)		not null	
place	character varying(20)			
phone	character varying(15)			

Indexes:
 "sales_man_pkey" PRIMARY KEY, btree (salesman_no)
 "sales_man_phone_key" UNIQUE CONSTRAINT, btree (phone)

f)Add foreign key and CHECK constraints in sales_order table using ALTER TABLE

```
postgres=# alter table sales_order add constraint fk_sno foreign
key(salesman_no) references sales_man(salesman_no);
ALTER TABLE
```

```
postgres=# alter table sales_order add constraint chk_type check (del_type
in('P','F'));
ALTER TABLE
```

```
postgres=# alter table sales_order add constraint chk_order check
(order_status in('inprocess','fullfilled','backorder','cancelled'));
ALTER TABLE
postgres=# \d sales_order;
```

Table "public.sales_order"				
Column	Type	Collation	Nullable	Default
order_no	character varying(5)		not null	
order_date	date		not null	
salesman_no	character varying(5)			
del_type	character varying(5)			
order_status	character varying(15)			

Indexes:

"sales_order_pkey" PRIMARY KEY, btree (order_no)

Check constraints:

"chk_order" CHECK (order_status::text = ANY (ARRAY['inprocess'::character varying, 'fullfilled'::character varying, 'backorder'::character varying, 'cancelled'::character varying]::text[]))

"chk_type" CHECK (del_type::text = ANY (ARRAY['P'::character varying, 'F'::character varying]::text[]))

Foreign-key constraints:

"fk_sno" FOREIGN KEY (salesman_no) REFERENCES sales_man(salesman_no)

PROGRAM NO:3

HOSPITAL INFORMATION**AIM:**

1.

Create a table Hospital with the fields

(doctorid,doctorname,department,qualification,experience).

Write the queries to perform the following.

- a) Insert 5 records
- b) Display the details of Doctors
- c) Display the details of doctors who have the qualification 'MD'
- d) Display all doctors who have more than 5 years experience but do not have the qualification 'MD'
- e) Display the doctors in 'Skin' department
- f) update the experience of doctor with doctored='D003' to 5
- g) Delete the doctor with DoctorID='D005'

QUERIES:

```
postgres=# create table hospital(doc_id varchar(5),doc_name varchar(10),dept
varchar(15),qualification varchar(15),experience number(5));
```

CREATE TABLE

```
postgres=# \d hospital;
```

Column	Type	Collation	Nullable	Default
doc_id	character varying(5)			
doc_name	character varying(10)			
dept	character varying(15)			
qualification	character varying(15)			
experience	integer			

a) Insert 5 records

```
postgres=# insert into hospital values('D001','A','skin','MD',1);
```

INSERT 0 1

```
postgres=# insert into hospital values('D002','B','skin','MD',5);
```

INSERT 0 1

```
postgres=# insert into hospital values('D003','C','ortho','MD',8);
```

```
INSERT 0 1
```

```
postgres=# insert into hospital values('D004','D','gync','MD',9);
```

```
INSERT 0 1
```

```
postgres=# insert into hospital values('D005','E','skin','MBBS',10);
```

```
INSERT 0 1
```

b) Display the details of Doctors

```
postgres=# select * from hospital;
```

```
postgres=# select * from hospital;
 doc_id | doc_name | dept  | qualification | experience
-----+-----+-----+-----+-----
 D001   | A        | skin  | MD            | 1
 D002   | B        | skin  | MD            | 5
 D003   | C        | ortho | MD            | 8
 D004   | D        | gync  | MD            | 9
 D005   | E        | skin  | MBBS          | 10
(5 rows)
```

c) Display the details of doctors who have the qualification 'MD'

```
postgres=# select * from hospital where qualification='MD';
```

```
postgres=# select * from hospital where qualification='MD';
 doc_id | doc_name | dept  | qualification | experience
-----+-----+-----+-----+-----
 D001   | A        | skin  | MD            | 1
 D002   | B        | skin  | MD            | 5
 D003   | C        | ortho | MD            | 8
 D004   | D        | gync  | MD            | 9
(4 rows)
```

d) Display all doctors who have more than 5 years experience but do not have the qualification 'MD'

```
postgres=# select * from hospital where experience>5 and qualification!='MD';
```

```
postgres=# select * from hospital where experience>5 and qualification!='MD';
 doc_id | doc_name | dept  | qualification | experience
-----+-----+-----+-----+-----
 D005   | E        | skin  | MBBS          | 10
(1 row)
```

e) Display the doctors in 'Skin' department

```
postgres=# select doc_name,dept from hospital where dept='skin';
```

```
postgres=# select doc_name,dept from hospital where dept='skin';
 doc_name | dept
-----+-----
 A        | skin
 B        | skin
 E        | skin
(3 rows)
```

f) update the experience of doctor with doctored='D003' to 5

```
postgres=# update hospital set experience=5 where doc_id='D003';
```

UPDATE 1

```
postgres=# select * from hospital;
```

doc_id	doc_name	dept	qualification	experience
D001	A	skin	MD	1
D002	B	skin	MD	5
D004	D	gync	MD	5
D005	E	skin	MBBS	10
D003	C	ortho	MD	5

(5 rows)

g)Delete the doctor with DoctorID='D005'

```
postgres=# delete from hospital where doc_id='D005';
```

DELETE 1

```
postgres=# select * from hospital;
```

doc_id	doc_name	dept	qualification	experience
D001	A	skin	MD	1
D002	B	skin	MD	5
D004	D	gync	MD	5
D003	C	ortho	MD	5

(4 rows)

PROGRAM NO:4

BANK INFORMATION**AIM:**

Create the following tables

Bank_customer (accno primary key, cust_name, place)
 Deposit (accno foreign key, deposit_no, damount)
 Loan (accno foreign key loan_no, Lamount)

Write the following queries

- Display the details of the customers
- Display the customers along with deposit amount who have only deposit with the bank
- Display the customers along with loan amount who have only loan with the bank
- Display the customers they have both loan and deposit with the bank
- Display the customer who have neither a loan nor a deposit with the bank

QUERIES:

```
postgres=# create table bank_customer(accno int primary key,cust_name varchar(10),place
varchar(20));
```

```
CREATE TABLE
```

```
postgres=# \d bank_customer;
```

Table "public.bank_customer"				
Column	Type	Collation	Nullable	Default
accno	integer		not null	
cust_name	character varying(10)			
place	character varying(20)			

Indexes:

```
"bank_customer_pkey" PRIMARY KEY, btree (accno)
```

```
postgres=# insert into bank_customer values(1,'Anu','kollam');
```

```
INSERT 0 1
```

```
postgres=# insert into bank_customer values(2,'Abhi','koyilandy');
```

```
INSERT 0 1
```

```
postgres=# insert into bank_customer values(3,'Ammu','calicut');
```

```
INSERT 0 1
```

```

postgres=# insert into bank_customer values(4,'Manu','vadakara');
INSERT 0 1
postgres=# insert into bank_customer values(5,'Adhi','ekm');
INSERT 0 1
postgres=# insert into bank_customer values(6,'Abhi','thirur');
INSERT 0 1
postgres=# insert into bank_customer values(7,'Achu','malappuram');
INSERT 0 1
postgres=# select * from bank_customer;

```

accno	cust_name	place
1	Anu	kollam
2	Abhi	koyilandy
3	Ammu	calicut
4	Manu	vadakara
5	Adhi	ekm
6	Abhi	thirur
7	Achu	malappuram

(7 rows)

```

postgres=# create table deposit(accno int references bank_customer(accno),deposit_no
int,damount int);
CREATE TABLE
postgres=# \d deposit;

```

Table "public.deposit"				
Column	Type	Collation	Nullable	Default
accno	integer			
deposit_no	integer			
damount	integer			

Foreign-key constraints:
 "deposit_accno_fkey" FOREIGN KEY (accno) REFERENCES bank_customer(accno)

```

postgres=# insert into deposit values(1,101,20000);
INSERT 0 1
postgres=# insert into deposit values(2,102,25000);
INSERT 0 1
postgres=# insert into deposit values(3,103,50000);
INSERT 0 1
postgres=# insert into deposit values(4,104,40000);
INSERT 0 1
postgres=# insert into deposit values(6,106,64000);
INSERT 0 1
postgres=# select * from deposit;

```

```
postgres=# select * from deposit;
 accno | deposit_no | damount
-----+-----+-----
      1 |         101 |   20000
      2 |         102 |   25000
      3 |         103 |   50000
      4 |         104 |   40000
      6 |         106 |   64000
(5 rows)
```

```
postgres=# create table loan(accno int references bank_customer(accno),loan_no
int,damount int);
```

```
CREATE TABLE
```

```
postgres=# \d loan;
```

```
Table "public.loan"
Column | Type   | Collation | Nullable | Default
-----+-----+-----+-----+-----
 accno | integer |           |          |
loan_no | integer |           |          |
damount | integer |           |          |
Foreign-key constraints:
 "loan_accno_fkey" FOREIGN KEY (accno) REFERENCES bank_customer(accno)
```

```
postgres=# insert into loan values(1,1,25000);
```

```
INSERT 0 1
```

```
postgres=# insert into loan values(2,2,40000);
```

```
INSERT 0 1
```

```
postgres=# insert into loan values(3,3,50000);
```

```
INSERT 0 1
```

```
postgres=# insert into loan values(4,4,45000);
```

```
INSERT 0 1
```

```
postgres=# insert into loan values(5,5,90000);
```

```
INSERT 0 1
```

```
postgres=# select * from loan;
```

```
 accno | loan_no | damount
-----+-----+-----
      1 |        1 |   25000
      2 |        2 |   40000
      3 |        3 |   50000
      4 |        4 |   45000
      5 |        5 |   90000
(5 rows)
```

a)Display the details of the customers


```
postgres=# select * from bank_customer;
```

accno	cust_name	place
1	Anu	kollam
2	Abhi	koyilandy
3	Ammu	calicut
4	Manu	vadakara
5	Adhi	ekm
6	Abhi	thirur
7	Achu	malappuram

(7 rows)

b) Display the customers along with deposit amount who have only deposit with the bank

```
postgres=# select bank_customer.cust_name,deposit.damount from bank_customer
inner join deposit on bank_customer.accno=deposit.accno and
bank_customer.accno not in(select accno from loan);
```

cust_name	damount
Abhi	64000

(1 row)

c.)Display the customers along with loan amount who have only loan with the bank

```
postgres=# select bank_customer.cust_name,loan.damount from bank_customer inner
join loan on bank_customer.accno=loan.accno and bank_customer.accno not in(select
accno from deposit);
```

cust_name	damount
Adhi	90000

(1 row)

d) Display the customers they have both loan and deposit with the bank

```
postgres=# select accno,cust_name from bank_customer where accno in (select accno
from deposit) and accno in (select accno from loan);
```

accno	cust_name
1	Anu
2	Abhi
3	Ammu
4	Manu

(4 rows)

e) Display the customer who have neither a loan nor a deposit with the bank

postgres=# select accno,cust_name from bank_customer where accno not in (select accno from deposit) and accno not in (select accno from loan);

accno	cust_name
7	Achu

(1 row)

PROGRAM NO:5

EMPLOYEE INFORMATIONAIM:

Create a table employee with fields (EmpID, EName, Salary, Department, and Age). Insert some records. Write SQL queries using aggregate functions and group by clause

- Display the total number of employees.
- Display the name and age of the oldest employee of each department.
- Display the average age of employees of each department
- Display departments and the average salaries
- Display the lowest salary in employee table
- Display the number of employees working in purchase department
- Display the highest salary in sales department;
- Display the difference between highest and lowest salary

QUERIES:

```
postgres=# create table employee(empid varchar(4) primary key,ename varchar(10),salary
int,dept varchar(10),age int);
```

```
CREATE TABLE
```

```
postgres=# \d employee;
```

Table "public.employee"				
Column	Type	Collation	Nullable	Default
empid	character varying(4)		not null	
ename	character varying(10)			
salary	integer			
dept	character varying(10)			
age	integer			

Indexes:

```
"employee_pkey" PRIMARY KEY, btree (empid)
```

```
postgres=# insert into employee values('E001','A',50000,'sales',30);
```

```
INSERT 0 1
```

```
postgres=# insert into employee values('E002','B',50002,'purchase',34);
```

```
INSERT 0 1
```

```
postgres=# insert into employee values('E003','C',50034,'purchase',39);
```

```
INSERT 0 1
```

```
postgres=# insert into employee values('E004','D',54000,'sales',40);
```

INSERT 0 1

postgres=# insert into employee values('E005','E',55000,'sales',35);

INSERT 0 1

postgres=# select * from employee;

```
postgres=# select * from employee;
 empid | ename | salary | dept  | age
-----+-----+-----+-----+----
 E001  | A     | 50000  | sales | 30
 E002  | B     | 50002  | purchase | 34
 E003  | C     | 50034  | purchase | 39
 E004  | D     | 54000  | sales | 40
 E005  | E     | 55000  | sales | 35
(5 rows)
```

A.Display the total number of employees.

postgres=# select count(*) "Number of employee" from employee;

```
Number of employee
-----
                    5
(1 row)
```

B)Display the name and age of the oldest employee of each department.

postgres=# select dept,max(age)"Maximun age" from employee group by dept;

```
 dept | Maximun age
-----+-----
purchase | 39
 sales  | 40
(2 rows)
```

C) Display the average age of employees of each department

select dept,avg(age)"Average age" from employee group by dept;

```
 dept | Average age
-----+-----
purchase | 36.5000000000000000
 sales  | 35.0000000000000000
(2 rows)
```

D)Display departments and the average salaries

postgres=# select dept,avg(salary)"Average salary" from employee group by dept;

dept	Average salary
purchase	50018.000000000000
sales	53000.000000000000
(2 rows)	

E.Display the lowest salary in employee table

postgres=# select min(salary)"Lowest salary" from employee;

Lowest salary
50000
(1 row)

f)Display the number of employees working in purchase department

postgres=# select count(*)"Number of employee" from employee where dept='purchase';

Number of employee
2
(1 row)

g)Display the highest salary in sales department;

postgres=# select max(salary)"Maximun salary" from employee where dept='sales';

Maximun salary
55000
(1 row)

h)Display the difference between highest and lowest salary

postgres=# select max(salary)-min(salary)"Difference in salary" from employee;

Difference in salary
5000
(1 row)

PROGRAM NO:6

PRODUCT INFORMATIONAIM:

Create a table product with the fields (Product_code primary key, Product_Name, Category, Quantity, Price).

Insert some records Write the queries to perform the following.

- Display the records in the descending order of Product_Name
- Display Product_Code, Product_Name with price between 20 and 50
- Display the details of products which belongs to the categories of 'bath soap', 'paste', or 'washing powder'
- Display the products whose Quantity less than 100 or greater than 500
- Display the products whose names starts with 's'
- Display the products which not belongs to the category 'paste'
- Display the products whose second letter is 'u' and belongs to the Category 'washing powder'

QUERIES:

```
postgres=# create table product(P_code varchar(5) primary key,p_name
varchar(10),category varchar(20),qty int,price int);
```

```
CREATE TABLE
```

```
postgres=# \d product;
```

Table "public.product"				
Column	Type	Collation	Nullable	Default
p_code	character varying(5)		not null	
p_name	character varying(10)			
category	character varying(20)			
qty	integer			
price	integer			

Indexes:

```
"product_pkey" PRIMARY KEY, btree (p_code)
```

```
postgres=# insert into product values('P001','sunlight','washing
powder',500,150);
```

```
INSERT 0 1
```

```
postgres=# insert into product values('P002','nirma','washing
      powder',500,120);
```

```
INSERT 0 1
```

```
postgres=# insert into product values('P003','lux','soap',100,30);
```

```
INSERT 0 1
```

```
postgres=# insert into product values('P004','pears','soap',100,50);
```

```
INSERT 0 1
```

```
postgres=# insert into product values('P005','colgate','paste',500,50);
```

```
INSERT 0 1
```

```
postgres=# insert into product values('P006','unibic','cookies',500,150);
```

```
INSERT 0 1
```

```
postgres=# insert into product values('P007','classmate','book',600,150);
```

```
INSERT 0 1
```

```
postgres=# insert into product values('P008','camalin','book',50,150);
```

```
INSERT 0 1
```

```
postgres=# select * from product;
```

p_code	p_name	category	qty	price
P001	sunlight	washing powder	500	150
P002	nirma	washing powder	500	120
P003	lux	soap	100	30
P004	pears	soap	100	50
P005	colgate	paste	500	50
P006	unibic	cookies	500	150
P007	classmate	book	600	150
P008	camalin	book	50	150

(8 rows)

a) Display the records in the descending order of Product_Name

```
postgres=# select * from product order by p_name desc;
```

```
postgres=# select * from product order by p_name desc;
 p_code |  p_name  | category | qty | price
-----+-----+-----+----+-----
 P006   | unibic   | cookies  | 500 | 150
 P001   | sunlight | washing powder | 500 | 150
 P004   | pears    | soap     | 100 | 50
 P002   | nirma    | washing powder | 500 | 120
 P003   | lux       | soap     | 100 | 30
 P005   | colgate  | paste     | 500 | 50
 P007   | classmate | book      | 600 | 150
 P008   | camalin  | book      | 50  | 150
(8 rows)
```

b) Display Product_Code, Product_Name with price between 20 and 50

postgres=# select p_code,p_name from product where price between 20 and 50;

```
 p_code | p_name
-----+-----
 P003   | lux
 P004   | pears
 P005   | colgate
(3 rows)
```

c) Display the details of products which belongs to the categories of 'bath soap', 'paste', or 'washing powder'

postgres=# select * from product where category in('soap','paste','washing powder');

```
 p_code |  p_name  | category | qty | price
-----+-----+-----+----+-----
 P001   | sunlight | washing powder | 500 | 150
 P002   | nirma    | washing powder | 500 | 120
 P003   | lux       | soap     | 100 | 30
 P004   | pears    | soap     | 100 | 50
 P005   | colgate  | paste     | 500 | 50
(5 rows)
```

d) Display the products whose Quantity less than 100 or greater than 500
 postgres=# select * from product where qty<100 or qty>500;

```
 p_code |  p_name  | category | qty | price
-----+-----+-----+----+-----
 P007   | classmate | book      | 600 | 150
 P008   | camalin  | book      | 50  | 150
(2 rows)
```


e) Display the products whose names starts with 's'

```
postgres=# select p_name from product where p_name like 's%';
```

```
p_name
-----
sunlight
(1 row)
```

f) Display the products which not belongs to the category 'paste'

```
postgres=# select * from product where category not in ('paste');
```

p_code	p_name	category	qty	price
P001	sunlight	washing powder	500	150
P002	nirma	washing powder	500	120
P003	lux	soap	100	30
P004	pears	soap	100	50
P006	unibic	cookies	500	150
P007	classmate	book	600	150
P008	camalin	book	50	150

(7 rows)

g) Display the products whose second letter is 'u' and belongs to the Category 'washing powder'

```
postgres=# select p_name from product where p_name like '_u%' and
category='washing powder';
```

```
p_name
-----
sunlight
(1 row)
```

PROGRAM NO:7

EMPLOYEE DATABASE INFORMATION**AIM:**

Consider the employee database given below. Give an expression in SQL for each of the following queries:

EMPLOYEE (Employee-Name, City)

WORKS (Employee-Name, Company-Name,

Salary) COMPANY (Company-Name, City)

MANAGES (Employee-Name, Manager-Name)

a) Find the names of all employees who work in Infosys

b) Find the names and cities of residence of all employees who works in Wipro

c) Find the names, and cities of all employees who work in Infosys and earn more than Rs. 10,000.

d) Find the employees who live in the same cities as the companies for which they work.

e) Find all employees who do not work in Wipro Corporation.

f) Find the company that has the most employees.

QUERIES:-

```
postgres=# create table employ(emp_name varchar(20),city varchar(20));
```

```
CREATE TABLE
```

```
postgres=# \d employ;
```

Table "public.employ"				
Column	Type	Collation	Nullable	Default
emp_name	character varying(20)			
city	character varying(20)			

```
postgres=# insert into employ values('sam','cochin');
```

```
INSERT 0 1
```

```
postgres=# insert into employ values('priya','pune');
```

```
INSERT 0 1
```

```
postgres=# insert into employ values('ram','bengeluru');
```

```
INSERT 0 1
```

```
postgres=# insert into employ values('raju','calicut');
```

INSERT 0 1

postgres=# insert into employ values('appu','tvm');

INSERT 0 1

postgres=# select * from employ;

emp_name	city
sam	cochin
priya	pune
raju	calicut
appu	tvm
ram	bengaluru
(5 rows)	

postgres=# create table work(emp_name varchar(20),company_name varchar(30),salary int);

CREATE TABLE

Table "public.work"				
Column	Type	Collation	Nullable	Default
emp_name	character varying(20)			
company_name	character varying(30)			
salary	integer			

postgres=# insert into work values('sam','wipro',15000);

INSERT 0 1

postgres=# insert into work values('ram','Infosys',25000);

INSERT 0 1

postgres=# insert into work values('priya','wipro',22000);

INSERT 0 1

postgres=# insert into work values('raju','wipro',25000);

INSERT 0 1

postgres=# insert into work values('appu','Infosys',27000);

INSERT 0 1

postgres=# select * from work;

emp_name	company_name	salary
sam	wipro	15000
ram	Infosys	25000
priya	wipro	22000
raju	wipro	25000
appu	Infosys	27000

(5 rows)

```
postgres=# create table company(company_name varchar(30),city
      varchar(20));
CREATE TABLE
postgres=# \d company;
```

Table "public.company"				
Column	Type	Collation	Nullable	Default
company_name	character varying(30)			
city	character varying(20)			

```
postgres=# insert into company values('wipro','bengaluru');
INSERT 0 1
postgres=# insert into company values('infosys','bengaluru');
INSERT 0 1
postgres=# select * from company;
```

company_name	city
wipro	bengaluru
infosys	bengaluru

(2 rows)

```
postgres=# create table managers(emp_name varchar(20),manager_name
      varchar(20));
CREATE TABLE
```

```
postgres=# \d managers;
```

Table "public.managers"				
Column	Type	Collation	Nullable	Default
emp_name	character varying(20)			
manager_name	character varying(20)			

```
postgres=# insert into managers values('sam','diya');
```

```
INSERT 0 1
```

```
postgres=# insert into managers values('ram','arul');
```

```
INSERT 0 1
```

```
postgres=# insert into managers values('priya','das');
```

```
INSERT 0 1
```

```
postgres=# select *from managers;
```

emp_name	manager_name
sam	diya
ram	arul
priya	das
(3 rows)	

a) **Find the names of all employees who work in Infosys**

```
postgres=# select emp_name from work where company_name='Infosys';
```

emp_name
ram
appu
(2 rows)

b) **Find the names and cities of residence of all employees who works in Wipro**

```
select employ.emp_name,employ.city from employ,work where
employ.emp_name=work.emp_name and work.company_name='wipro';
```

emp_name	city
sam	cochin
priya	pune
raju	calicut

(3 rows)

c) Find the names, and cities of all employees who work in Infosys and earn more than Rs. 10,000.

```
postgres=# select employ.emp_name,employ.city from employ,work
whereemploy.emp_name=work.emp_name and
work.company_name='Infosys'and salary>10000;
```

emp_name	city
ram	bengaluru
appu	tvm

(2 rows)

d) Find the employees who live in the same cities as the companies for which they work.

```
select employ.emp_name from employ,work,company where
company.company_name=work.company_name and
company.city=employ.city;
```

emp_name
ram

(1 row)

e) Find all employees who do not work in Wipro Corporation.

```
postgres=# select * from work where company_name not in ('wipro');
```

emp_name	company_name	salary
ram	Infosys	25000
appu	Infosys	27000

(2 rows)

f)Find the company that has the most employees.

postgres=# select company_name from work group by company_name having count(distinct emp_name)>=all (select count(distinct emp_name)from work group by company_name);

```
company_name
-----
wipro
(1 row)
```

PROGRAM NO:8

AREA OF A CIRCLE
.....**AIM:**

Write a program code to calculate the area of a circle for a value of radius varying from 3 to 7. Store the radius and the corresponding value of calculated area in an empty table named areas with field's radius and area.

QUERIES:

```
postgres=# create table areas(radius int,area numeric);
```

```
postgres=# \d areas;
```

```
postgres=# \d areas;
```

Column	Type	Collation	Nullable	Default
radius	integer			
area	numeric			

PL/SQL CODE:

```
postgres=#create or replace function radius() returns void as $$
```

```
    declare
```

```
        r int;
```

```
        a int;
```

```
    begin
```

```
        for r in 3..7
```

```
        loop
```

```
            a:=3.14*r*r;
```

```
            insert into areas values(r,a);
```

```
        end loop;
```

```
    end $$ language plpgsql;
```

```
postgres=# select radius();
```



```
postgres=# select radius();  
radius  
-----  
  
(1 row)
```

postgres=# select * from areas;

radius	area
3	28
4	50
5	79
6	113
7	154

(5 rows)

PROGRAM NO:9

ELECTRICITY BILL CALCULATION**AIM:**

Write a program block to calculate the electricity bill by accepting cust_no and units_consumed

QUERIES:-

```
postgres=# CREATE TABLE ELEC_BILL(CNO INT PRIMARY
KEY,CNAME VARCHAR(10),UC INT,BAMT NUMERIC);
CREATE TABLE
```

```
postgres=# \d ELEC_BILL;
```

Table "public.elec_bill"				
Column	Type	Collation	Nullable	Default
cno	integer		not null	
cname	character varying(10)			
uc	integer			
bamt	numeric			

Indexes:

```
"elec_bill_pkey" PRIMARY KEY, btree (cno)
```

```
postgres=# insert into ELEC_BILL values(1,'SMITHA',250,NULL);
```

```
INSERT 0 1
```

```
postgres=# insert into ELEC_BILL values(2,'SACHIN',200,NULL);
```

```
INSERT 0 1
```

```
postgres=# insert into ELEC_BILL values(3,'NISHAF',100,NULL);
```

```
INSERT 0 1
```

```
postgres=# insert into ELEC_BILL values(4,'SHIHAB',101,NULL);
```

```
INSERT 0 1
```

```
postgres=# insert into ELEC_BILL values(5,'SHAFAF',85,NULL);
```

```
INSERT 0 1
```

```
postgres=# select * from ELEC_BILL;
```

cno	cname	uc	bamt
1	SMITHA	250	
2	SACHIN	200	
3	NISHAF	100	
4	SHIHAB	101	
5	SHAFAF	85	

(5 rows)

```
create or replace function bill_calculation()
returns void as $$
declare
ba numeric;
custno int;
unit int;
cur1 cursor for select cno,uc from ELEC_BILL;
begin
open cur1;
loop
fetch cur1 into custno,unit;
exit when not found;
ba:=unit*3;
update ELEC_BILL set bamt=ba
where cno=custno;
end loop;
close cur1;
end $$ language plpgsql;
```

CREATE FUNCTION

```
postgres=# select bill_calculation();
postgres=# select * from ELEC_BILL;
```

```
postgres=# select * from ELEC_BILL;
```

cno	cname	uc	bamt
1	SMITHA	250	750
2	SACHIN	200	600
3	NISHAF	100	300
4	SHIHAB	101	303
5	SHAFAF	85	255

(5 rows)

PROGRAM NO:10

FIBONACCI SERIES**AIM:**

Create a procedure to print Fibonacci number up to a limit, limit is passed as an argument

QUERIES:-

```
postgres=#create function fibonacii(n int) returns void as $$  
    declare  
    t1 int:=0;  
    t2 int:=1;  
    i int;  
    t3 int:=0;  
    begin  
    raise notice '%',t1;  
    raise notice '%',t2;  
    for i in 2..n loop  
    t3:=t1+t2;  
    t1:=t2;  
    t2:=t3;  
    raise notice '%',t3;  
    end loop;  
    end $$ language plpgsql;  
postgres=# select fibonacii(10);
```

```
NOTICE: 0  
NOTICE: 1  
NOTICE: 1  
NOTICE: 2  
NOTICE: 3  
NOTICE: 5  
NOTICE: 8  
NOTICE: 13  
NOTICE: 21  
NOTICE: 34  
NOTICE: 55
```

```
  fibonacii
```

```
-----
```

```
(1 row)
```

PROGRAM NO:11

PRIME OR NOT**AIM:**

Create a function to check whether a given number is prime or not

QUERIES:-

postgres\$#create or replace function primeno(n int) returns void as \$\$

```
declare
i int;
f int:=0;
begin
if n=2 then
    f:=0;
else
    for i in 2..(n-1)
    loop
        if(mod(n,i)=0)then
            f:=1;
            exit;
        end if;
    end loop;
end if;
if(f=0)then
    raise notice 'prime number';
else
    raise notice 'not prime number';
end if;
end $$ language plpgsql;
```

CREATE FUNCTION

```
postgres=# select primeno(10);
NOTICE: not prime number
 primeno
-----
(1 row)

postgres=# select primeno(5);
NOTICE: prime number
 primeno
-----
(1 row)

postgres=#
```


PROGRAM NO:12

SALARY CALCULATION**AIM:**

create a table emp_salary(empno,ename,dept,salary)

Write a function to return the average salary of a particular department by accepting departmentname as argument.

QUERIES:-

```
postgres=# create table empsal(eno int ,ename varchar(10),dept
varchar(10),sal int);
```

```
postgres=# \d empsal;
```

```
postgres=# \d empsal;
```

Column	Type	Collation	Nullable	Default
eno	integer		not null	
ename	character varying(10)			
dept	character varying(10)			
sal	integer			

```
Indexes:
    "empsal_pkey" PRIMARY KEY, btree (eno)
```

```
postgres=# insert into empsal values(101,'anu','purchase',25000);
INSERT 0 1
```

```
postgres=# insert into empsal values(102,'adharsh','sales',27000);
INSERT 0 1
```

```
postgres=# insert into empsal values(103,'appu','sales',21000);
INSERT 0 1
```

```
postgres=# insert into empsal values(104,'appu','grocery',15000);
INSERT 0 1
```

```
select * from empsal;
```

```
postgres=# select * from empsal;
eno |  ename  | dept   | sal
-----+-----+-----+-----
101 |  anu    | purchase | 25000
102 |  adharsh | sales   | 27000
104 |  appu   | grocery | 15000
103 |  achu   | sales   | 21000
(4 rows)
```

```
postgres=# create or replace function sal_calc (d varchar)returns numeric as
    $$
    declare
    av numeric;
    curl cursor for select avg(sal)
    from empsal where dept =d;
    begin
    open curl;
    fetch curl into av;
    return av;
    close curl;
    end $$ language plpgsql;
```

CREATE FUNCTION

```
postgres=# select sal_calc('sales');
```

```
sal_calc
-----
24000.0000000000000000
(1 row)
```

PROGRAM NO:13

GRADE CALCULATION**AIM:**

Create a table stud_mark (reg_no,name,avgmark)

Insert few records into the table and Write a procedure to display the number of students got distinction,firstclass,second class,third class or failed.

QUERIES:

```
postgres=# create table stud_mark(rno int primary
key,snamevarchar(10),avg_mark numeric);
```

```
CREATE TABLE
```

```
postgres=# \d stud_mark;
```

Table "public.stud_mark"				
Column	Type	Collation	Nullable	Default
rno	integer		not null	
sname	character varying(10)			
avg_mark	numeric			

Indexes:

"stud_mark_pkey" PRIMARY KEY, btree (rno)

```
postgres=# insert into stud_mark values(103,'sikha',92);
```

```
INSERT 0 1
```

```
postgres=# insert into stud_mark values(102,'shyma',79);
```

```
INSERT 0 1
```

```
postgres=# insert into stud_mark values(101,'sarath',59);
```

```
INSERT 0 1
```

```
postgres=# insert into stud_mark values(104,'Akash',89);
```

```
INSERT 0 1
```

```
postgres=# insert into stud_mark values(105,'Athul',69);
```

```
INSERT 0 1
```

```
postgres=# select * from stud_mark;
```

rno	sname	avg_mark
103	sikha	92
102	shyma	79
101	sarath	59
104	Akash	89
105	Athul	69

```
postgres=# create function stud_mark() returns void as $$
declare
d int:=0;
fc int:=0;
sc int:=1;
tc int:=0;
f int:=0;
curl cursor for select avg_mark from stud_mark;
av numeric;
begin
open curl;
loop fetch curl into av;
exit when not found;
if(av>=90 and av<=100)
then d:=d+1;
elsif(av>=75 and av<=89)
then
fc:=fc+1;
elsif(av>=60 and av<=74)
then
sc:=sc+1;
elsif (av>=50 and av<59)
then
tc:=tc+1;
else
f:=f+1;
end if;
end loop;
```

```
close curl;
raise notice 'distinction=%',d;
raise notice 'first class=%',fc;
raise notice 'second class=%',sc;
raise notice 'third class=%',tc;
raise notice 'failed=%',f;
end $$ language plpgsql;
CREATE FUNCTION
```

```
postgres=# select stud_mark();
```

```
NOTICE:  distinction=1
NOTICE:  first class=2
NOTICE:  second class=2
NOTICE:  third class=0
NOTICE:  failed=1
 stud_mark
-----
(1 row)
```

PROGRAM NO:14

MARK CALCULATION**AIM:**

Create a table stud_mark (reg_no,name,avgmark)

Insert few records into the table and Write a procedure to display the number of students got distinction,firstclass,second class,third class or failed.

QUERIES:

```
postgres=# create table student(regno varchar(10),sname varchar(15),sub1
int,sub2 int,sub3 int,sub4 int,sub5 int,mark_total int,avg_mark int);
```

```
CREATE TABLE
```

```
postgres=# \d student;
```

Table "public.student"				
Column	Type	Collation	Nullable	Default
regno	character varying(10)			
sname	character varying(15)			
sub1	integer			
sub2	integer			
sub3	integer			
sub4	integer			
sub5	integer			
mark_total	integer			
avg_mark	integer			

```
create or replace function mcalc() returns trigger as $$
```

```
begin
```

```
NEW.mark_total=NEW.sub1+ NEW.sub2+ NEW.sub3+ NEW.sub4+
NEW.sub5;
```

```
NEW.avg_mark=( NEW.sub1+ NEW.sub2+ NEW.sub3+ NEW.sub4+
NEW.sub5) /5;
```

```
Return NEW;
```

```
end $$ language plpgsql;
```

```
CREATE FUNCTION
```

```

postgres=# create trigger score_calc before insert on student
postgres=# for each row
postgres=# execute procedure mcalc();
CREATE TRIGGER

```

```

postgres=# insert into student values(1,'A',69,78,45,98,81,0,0);
INSERT 0 1
postgres=# insert into student values(2,'B',36,78,65,58,89,0,0);
INSERT 0 1
postgres=# insert into student values(3,'C',50,38,47,95,88,0,0);
INSERT 0 1
postgres=# insert into student values(4,'D',79,88,85,98,67,0,0);
INSERT 0 1
postgres=# insert into student values(5,'E',89,88,75,98,57,0,0);
INSERT 0 1
postgres=# select * from student;    postgres=# select * from student;

```

regno	sname	sub1	sub2	sub3	sub4	sub5	mark_total	avg_mark
1	A	69	78	45	98	81	371	74
2	B	36	78	65	58	89	326	65
3	C	50	38	47	95	88	318	63
4	D	79	88	85	98	67	417	83
5	E	89	88	75	98	57	407	81

(5 rows)

PROGRAM NO:15

TRIGGER IMPLEMENTATION**AIM:**

Create a table phonebook(pname,mobno)
 create a trigger to insert the old from the table phonebook to
 del_phonebook(pname,mobno,modifydate) whenever a record or updated in the
 phone book table.

QUERIES:

```
postgres=# create table phonebook(pname varchar(20),mobno int);
```

```
CREATE TABLE
```

```
postgres=# \d phonebook;
```

Table "public.phonebook"				
Column	Type	Collation	Nullable	Default
pname	character varying(20)			
mobno	integer			

```
postgres=# insert into phonebook values('A',246);
```

```
INSERT 0 1
```

```
postgres=# insert into phonebook values('B',801);
```

```
INSERT 0 1
```

```
postgres=# insert into phonebook values('C',357);
```

```
INSERT 0 1
```

```
postgres=# insert into phonebook values('D',924);
```

```
INSERT 0 1
```

```
postgres=# select * from phonebook;
```

pname	mobno
A	246
B	801
C	357
D	924
(4 rows)	

```
postgres=# create table del_phonebook(dname varchar(20),del_pho
```



```

        int,d_date date);
CREATE TABLE
postgres=# \d del_phonebook;

```

Table "public.del_phonebook"				
Column	Type	Collation	Nullable	Default
dname	character varying(20)			
del_pho	integer			
d_date	date			

```

create or replace function pf() returns trigger as $$
begin
insert into del_phonebook(OLD.pname,OLD.mobno,now()::date);
Return NEW;
end $$ language plpgsql;
CREATE FUNCTION

```

```

create trigger phone_trigger
after delete or update on phonebook
for each row
execute procedure pf()
CREATE TRIGGER

```

```

postgres=# delete from phonebook where pname='A';
DELETE 1

```

```

postgres=# select * from del_phonebook;

```

dname	del_pho	d_date
A	246	2021-03-23
(1 row)		