College Of Engineering Trivandrum

Application Software Development Lab
Final Exam Report

Abhishek Manoharan
S5 CSE Roll No:2

TVE17CS002

Department of Computer Science

November 6, 2019

Contents

List of Figures

ASD LAB EXAM

1 Description

Exam contained table creation ,different queries ,trigger ,cursor ,PLPGSL questions

2 Questions

2.1 Tables with constraints

Create and populate shown tables.
Use appropriate data types and Check the following
constraints:
Define appropriate Primary keys and foreign keys for all tables
Customer_id should be of the format “C00__”
Account_id should be of the format “AC00__”
Tx_id should be of the format “TX00__”
Email should be of the format “abc@xyz.com ”

2.1.1 Table Creation

**Create table Customer(customer_id char(5) primary key,
first_name varchar(20),last_name varchar(20),
state varchar(20),phone varchar(20),email varchar(50),
check (customer_id like 'C00__'),check (email like '%@%.com'));**

**create table Account(Account_id char(6) primary key,
Type varchar(10),Balance int ,
Credit_line int,check(Account_id like 'AC00__'));**

**create table Transactions(Tx_id char(6) primary key,Account_id char(6),
Amount int,Type varchar(6),check(Tx_id like 'TX00__'),
foreign key(Account_id) references Account(Account_id)
on update cascade on delete cascade);**

**create table Customer_Accounts(Customer_id char(5),Account_id char(6),
foreign key(Account_id) references Account(Account_id)
on update cascade on delete cascade,
foreign key(Customer_id) references Customer(Customer_id)
on update cascade on delete cascade);**

```
insert into customer values('C0001','Richard','Collins','New York','9485178542','rcollins@test.com');
insert into customer values('C0002','Peter','Parker','Queens','6254791542','peter@test.com');
insert into customer values('C0003','Stan','Glen','Nevada','741256342','stang@test.com');
insert into customer values('C0004','Steve','Green','New York','6597534242','steven@test.com');
insert into customer values('C0005','John','Adams','Nevada','8632415973','johna@test.com');
```

```
insert into Account values('AC0001','Savings',30000,50000);
insert into Account values('AC0002','Savings',50000,50000);
insert into Account values('AC0003','Current',75000,100000);
insert into Account values('AC0004','Savings',80000,50000);
insert into Account values('AC0005','Current',25000,20000);
insert into Account values('AC0006','Current',22000,50000);
insert into Account values('AC0007','Savings',18000,20000);
```

```
insert into Transactions values('TX0001','AC0003',5000,'Credit');
insert into Transactions values('TX0002','AC0002',7500,'Debit');
insert into Transactions values('TX0003','AC0003',8000,'Debit');
insert into Transactions values('TX0004','AC0001',500,'Credit');
insert into Transactions values('TX0005','AC0005',4000,'Debit');
insert into Transactions values('TX0006','AC0004',1000,'Debit');
insert into Transactions values('TX0007','AC0007',2000,'Credit');
insert into Transactions values('TX0008','AC0006',7000,'Debit');
insert into Transactions values('TX0009','AC0001',3000,'Credit');
insert into Transactions values('TX0010','AC0006',2500,'Credit');
```

```
insert into Customer_Accounts values('C0001','AC0002');
insert into Customer_Accounts values('C0003','AC0004');
insert into Customer_Accounts values('C0005','AC0001');
insert into Customer_Accounts values('C0002','AC0006');
insert into Customer_Accounts values('C0004','AC0005');
insert into Customer_Accounts values('C0005','AC0007');
insert into Customer_Accounts values('C0003','AC0003');
```

```
exam=# select * from customer;
 customer_id | first_name | last_name |  state   |   phone    |       email
-------------+------------+-----------+----------+------------+-------------------
 C0001       | Richard    | Collins   | New York | 9485178542 | rcollins@test.com
 C0002       | Peter      | Parker    | Queens   | 6254791542 | peter@test.com
 C0004       | Steve      | Green     | New York | 6597534242 | steven@test.com
 C0005       | John       | Adams     | Nevada   | 8632415973 | johna@test.com
 C0003       | Stan       | Glen      | Nevada   | 741256342  | stang@test.com
(5 rows)
```

Figure 1: customer Table

```
exam=# \d+ customer
                                        Table "public.customer"
   Column    |         Type          | Collation | Nullable | Default |
-------------+-----------------------+-----------+----------+---------+
 customer_id | character(5)          |           | not null |         |
 first_name  | character varying(20) |           |          |         |
 last_name   | character varying(20) |           |          |         |
 state       | character varying(20) |           |          |         |
 phone       | character varying(20) |           |          |         |
 email       | character varying(50) |           |          |         |
Indexes:
    "customer_pkey" PRIMARY KEY, btree (customer_id)
Check constraints:
    "customer_customer_id_check" CHECK (customer_id ~~ 'C00__'::text)
    "customer_email_check" CHECK (email::text ~~ '%@%.com'::text)
```

Figure 2: customer Table

```
exam=# select * from Account;
 account_id |  type   | balance | credit_line
------------+---------+---------+-------------
 AC0001     | Savings |   30000 |       50000
 AC0002     | Savings |   50000 |       50000
 AC0003     | Current |   75000 |      100000
 AC0004     | Savings |   80000 |       50000
 AC0005     | Current |   25000 |       20000
 AC0006     | Current |   22000 |       50000
 AC0007     | Savings |   18000 |       20000
(7 rows)
```

Figure 3: Account Table

```
exam=# \d+ account;
                                        Table "public.account"
   Column    |         Type          | Collation | Nullable | Default | Storage  |
-------------+-----------------------+-----------+----------+---------+----------+
 account_id  | character(6)          |           | not null |         | extended |
 type        | character varying(10) |           |          |         | extended |
 balance     | integer               |           |          |         | plain    |
 credit_line | integer               |           |          |         | plain    |
Indexes:
    "account_pkey" PRIMARY KEY, btree (account_id)
Check constraints:
    "account_account_id_check" CHECK (account_id ~~ 'AC00__'::text)
```

Figure 4: Account Table

```
exam=# select * from transactions ;
 tx_id   | account_id | amount |  type
---------+------------+--------+--------
 TX0001 | AC0003     |   5000 | Credit
 TX0002 | AC0002     |   7500 | Debit
 TX0003 | AC0003     |   8000 | Debit
 TX0004 | AC0001     |    500 | Credit
 TX0005 | AC0005     |   4000 | Debit
 TX0006 | AC0004     |   1000 | Debit
 TX0007 | AC0007     |   2000 | Credit
 TX0008 | AC0006     |   7000 | Debit
 TX0009 | AC0001     |   3000 | Credit
 TX0010 | AC0006     |   2500 | Credit
(10 rows)
```

Figure 5: Transaction Table

```
exam=# \d+ transactions
                                    Table "public.transactions"
   Column   |         Type         | Collation | Nullable | Default | Storage  | Stats target | Description
------------+----------------------+-----------+----------+---------+----------+--------------+-------------
 tx_id      | character(6)         |           | not null |         | extended |              |
 account_id | character(6)         |           |          |         | extended |              |
 amount     | integer              |           |          |         | plain    |              |
 type       | character varying(6) |           |          |         | extended |              |
Indexes:
    "transactions_pkey" PRIMARY KEY, btree (tx_id)
Check constraints:
    "transactions_tx_id_check" CHECK (tx_id ~~ 'TX00__'::text)
Foreign-key constraints:
    "transactions_account_id_fkey" FOREIGN KEY (account_id) REFERENCES account(account_id) ON UPDATE CASCADE ON DELETE CASCADE
```

Figure 6: Transaction Table

```
exam=# select * from customer_accounts ;
 customer_id | account_id
-------------+------------
 C0001       | AC0002
 C0003       | AC0004
 C0005       | AC0001
 C0002       | AC0006
 C0004       | AC0005
 C0005       | AC0007
 C0003       | AC0003
(7 rows)
```

Figure 7: Customer Account Table

```
exam=# \d+ customer_accounts
                          Table "public.customer_accounts"
   Column    |     Type     | Collation | Nullable | Default | Storage  | Stats target | Description
-------------+--------------+-----------+----------+---------+----------+--------------+-------------
 customer_id | character(5) |           |          |         | extended |              |
 account_id  | character(6) |           |          |         | extended |              |
Foreign-key constraints:
    "customer_accounts_account_id_fkey" FOREIGN KEY (account_id) REFERENCES account(account_id) ON UPDATE CASCADE ON DELETE CASCADE
    "customer_accounts_customer_id_fkey" FOREIGN KEY (customer_id) REFERENCES customer(customer_id) ON UPDATE CASCADE ON DELETE CASCADE
```

Figure 8: Customer Account

2.2 Query 1

Display the details of all customers whose first name begins with S and last name ends with n.

```
select * from Customer where first_name like 'S%'and last_name like '%n';
```

```
exam=# select * from Customer where first_name like 'S%'and last_name like '%n';
 customer_id | first_name | last_name |  state   |   phone    |      email
-------------+------------+-----------+----------+------------+-----------------
 C0003       | Stan       | Glen      | New York | 741256342  | stang@test.com
 C0004       | Steve      | Green     | New York | 6597534242 | steven@test.com
(2 rows)
```

Figure 9: First name start with S last name ends with n

2.3 Query 2

Display all the transactions that have taken place in the account belonging to John Adams in ascending order.

```
    select T.*
from customer as C , Account as A ,Transactions as T,Customer_Accounts as CA
where CA.Customer_id = C.Customer_id AND CA.Account_id = A.Account_id
AND T.Account_id = A.Account_id AND C.first_name like 'John'
AND C.last_name like 'Adams' ORDER BY T.tx_id;
```

```
exam=# select T.*
exam-# from customer as C , Account as A ,Transactions as T,Customer_Accounts as CA
exam-# where CA.Customer_id = C.Customer_id AND CA.Account_id = A.Account_id
exam-# AND T.Account_id = A.Account_id AND C.first_name like 'John' AND C.last_name like 'Adams' ORDER BY T.tx_id;
 tx_id  | account_id | amount |  type
--------+------------+--------+--------
 TX0004 | AC0001     |    500 | Credit
 TX0007 | AC0007     |   2000 | Credit
 TX0009 | AC0001     |   3000 | Credit
(3 rows)
```

Figure 10: John Adams Transactions

2.4 Query 3

Display name of Customers having at least two Savings Accounts

```
    Select concat(first_name,last_name) as name
from Customer
where Customer_id IN
(select C.Customer_id
from Customer as C , Account as A,Customer_Accounts as CA
WHERE C.Customer_id = CA.Customer_id AND A.Account_id = CA.Account_id
AND A.Type like 'Savings'
GROUP BY C.Customer_id
Having Count(*) >=2);
```

```
exam=# Select concat(first_name,last_name) as name
exam-# from Customer
exam-# where Customer_id IN
exam-# (select C.Customer_id
exam(# from Customer as C , Account as A,Customer_Accounts as CA
exam(# WHERE C.Customer_id = CA.Customer_id AND A.Account_id = CA.Account_id
exam(# AND  A.Type like 'Savings'
exam(# GROUP BY C.Customer_id
exam(# Having Count(*) >=2);
   name
-----------
 JohnAdams
(1 row)
```

Figure 11: Customer having 2 Savings Account

2.5 Query 4

Display the total number of customers in the states in which the total balance is greater than 25,000

```
    select state,count(*)
from customer
```

```
where state in
(select C.state
from Account as A ,Customer as C,Customer_Accounts as CA
where CA.Customer_id = C.Customer_id AND CA.Account_id = A.Account_id
group by c.state
having sum(Balance) > 25000)
group by state;
```

```
exam=# select state,count(*)
exam-# from customer
exam-# where state in
exam-# (select C.state
exam(# from Account as A ,Customer as C,Customer_Accounts as CA
exam(# where CA.Customer_id = C.Customer_id AND CA.Account_id = A.Account_id
exam(# group by c.state
exam(# having sum(Balance) > 25000)
exam-# group by state;
  state   | count
----------+-------
 Nevada   |     2
 New York |     2
(2 rows)
```

Figure 12: State and count

2.6 Trigger When new Transaction Occurred

Create a trigger that updates the account balance whenever a new transaction is entered.

2.6.1 Code

```
CREATE OR REPLACE FUNCTION update() RETURNS TRIGGER AS
$$
DECLARE
BEGIN
IF NEW.TYPE = 'Credit'  THEN
  UPDATE ACCOUNT SET BALANCE = (BALANCE +NEW.Amount)
  WHERE Account_id = NEW.Account_id;
 ELSE
  UPDATE ACCOUNT SET BALANCE = (BALANCE -NEW.Amount)
  WHERE Account_id = NEW.Account_id;
 END IF;
RETURN NEW;
END;
$$
language plpgsql;

CREATE TRIGGER new_t
AFTER INSERT ON Transactions
FOR EACH ROW EXECUTE PROCEDURE update();
```

2.6.2 Output

```
exam=#  CREATE OR REPLACE FUNCTION update() RETURNS TRIGGER AS
exam-#  $$
exam$#  DECLARE
exam$#  BEGIN
exam$#
exam$#   IF NEW.TYPE = 'Credit'  THEN
exam$#    UPDATE ACCOUNT SET BALANCE = (BALANCE +NEW.Amount)
exam$#    WHERE Account_id = NEW.Accound_id;
exam$#   ELSE
exam$#    UPDATE ACCOUNT SET BALANCE = (BALANCE -NEW.Amount)
exam$#    WHERE Account_id = NEW.Account_id;
exam$#   END IF;
exam$#  RETURN NEW;
exam$#  END;
exam$#  $$
exam-#  language plpgsql;
CREATE FUNCTION
exam=#
exam=# CREATE TRIGGER new_t
exam-# AFTER INSERT ON Transactions
exam-# FOR EACH STATEMENT EXECUTE PROCEDURE update();
CREATE TRIGGER
```

Figure 13: Update balance when transaction occured

```
exam=# select * from account ;
 account_id |  type   | balance | credit_line
------------+---------+---------+-------------
 AC0001     | Savings |   30000 |       50000
 AC0002     | Savings |   50000 |       50000
 AC0003     | Current |   75000 |      100000
 AC0004     | Savings |   80000 |       50000
 AC0005     | Current |   25000 |       20000
 AC0006     | Current |   22000 |       50000
 AC0007     | Savings |   18000 |       20000
(7 rows)

exam=# insert into Transactions values('TX0011','AC0006',2500,'Credit');
INSERT 0 1
exam=# select * from account ;
 account_id |  type   | balance | credit_line
------------+---------+---------+-------------
 AC0001     | Savings |   30000 |       50000
 AC0002     | Savings |   50000 |       50000
 AC0003     | Current |   75000 |      100000
 AC0004     | Savings |   80000 |       50000
 AC0005     | Current |   25000 |       20000
 AC0007     | Savings |   18000 |       20000
 AC0006     | Current |   24500 |       50000
(7 rows)

exam=# insert into Transactions values('TX0011','AC0006',2500,'Debit');
ERROR:  duplicate key value violates unique constraint "transactions_pkey"
DETAIL:  Key (tx_id)=(TX0011) already exists.
exam=# insert into Transactions values('TX0012','AC0006',2500,'Debit');
INSERT 0 1
exam=# select * from account ;
 account_id |  type   | balance | credit_line
------------+---------+---------+-------------
 AC0001     | Savings |   30000 |       50000
 AC0002     | Savings |   50000 |       50000
 AC0003     | Current |   75000 |      100000
 AC0004     | Savings |   80000 |       50000
 AC0005     | Current |   25000 |       20000
 AC0007     | Savings |   18000 |       20000
 AC0006     | Current |   22000 |       50000
(7 rows)
```

Figure 14: Before and After

2.7 Creating a View

Create a view "Defaulters" containing the name, state and phone number of all customers whose current balance is below credit line.

CREATE OR REPLACE VIEW Defaulters as
(Select C.first_name,C.last_name,C.state,c.phone
from customer as C , Account as A ,Customer_Accounts as CA
where CA.Customer_id = C.Customer_id AND CA.Account_id = A.Account_id
AND A.Balance ¡A.credit_line);

```
select * from Defaulters;
```

```
exam=# CREATE OR REPLACE VIEW  Defaulters as
exam-# (Select C.first_name,C.last_name,C.state,c.phone
exam(# from customer as C , Account as A ,Customer_Accounts as CA
exam(# where CA.Customer_id = C.Customer_id AND CA.Account_id = A.Account_id
exam(# AND  A.Balance <A.credit_line);
CREATE VIEW
exam=# select * from defaulters ;
 first_name | last_name | state  |   phone
------------+-----------+--------+------------
 John       | Adams     | Nevada | 8632415973
 Peter      | Parker    | Queens | 6254791542
 John       | Adams     | Nevada | 8632415973
 Stan       | Glen      | Nevada | 741256342
(4 rows)

exam=#
```

Figure 15: View Defaulters

2.8 Cursor to reduce credit line

Using a cursor, reduce the credit line for all accounts that have balance greater than20000 by 3%.

2.8.1 Code

```
CREATE OR REPLACE FUNCTION reduce() RETURNS VOID AS
$$
DECLARE
row RECORD;
getrow CURSOR
FOR SELECT * FROM Account;
BEGIN
OPEN getrow;
LOOP
FETCH getrow INTO row;
EXIT WHEN NOT FOUND;
IF row.Balance >20000 THEN
UPDATE Account SET Credit_line=Credit_line*0.97 where Account_id =row.Account_id;
END IF;
END LOOP;
CLOSE getrow;
```

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

2.8.2 Output

```
exam=# select * from account ;
 account_id |  type   | balance | credit_line
------------+---------+---------+-------------
 AC0001     | Savings |   30000 |       50000
 AC0002     | Savings |   50000 |       50000
 AC0003     | Current |   75000 |      100000
 AC0004     | Savings |   80000 |       50000
 AC0005     | Current |   25000 |       20000
 AC0007     | Savings |   18000 |       20000
 AC0006     | Current |   22000 |       50000
(7 rows)

exam=# select from reduce();
--
(1 row)

exam=# select * from account ;
 account_id |  type   | balance | credit_line
------------+---------+---------+-------------
 AC0007     | Savings |   18000 |       20000
 AC0001     | Savings |   30000 |       48500
 AC0002     | Savings |   50000 |       48500
 AC0003     | Current |   75000 |       97000
 AC0004     | Savings |   80000 |       48500
 AC0005     | Current |   25000 |       19400
 AC0006     | Current |   22000 |       48500
(7 rows)
```

Figure 16: Reducing credit line

2.9 Query 5

You are given a table, Functions, containing two columns: X and Y
Two pairs (X1, Y1) and (X2, Y2) are said to be symmetric pairs if X1 = Y2 and X2 = Y1.
Write a query to output all such symmetric pairs in ascending order by the value of X.

```
 SELECT DISTINCT( t1.*)


from XY as t1 , XY as t2
where t1.x =t2.y AND t1.y = t2.x ORDER by t1.x;
```

```
exam=# SELECT DISTINCT( t1.*)
  from XY as t1 , XY as t2
where t1.x =t2.y AND t1.y = t2.x ORDER by t1.x;
 x  | y
----+----
 20 | 20
 20 | 21
 21 | 20
 22 | 23
 23 | 22
(5 rows)
```

Figure 17: Symmetric pairs

2.10 Deposit when low balance

For all customers with balance less than 50000, deposit Rs 1000. You must update the balance as well as create a transaction entry.

2.10.1 Code

```
create or replace function help() returns void as $$
declare
    num integer;
    i integer=0;
    txid char(6);
    c1 cursor for select * from account;
    r record;
    len int;
begin
    select count(*) from transactions into num;
    num=num+1;
    select count(*) from account into len;
    open c1;
    loop
        fetch c1 into r;
        i=i+1;
        if i>len then
            exit;
        end if;
        if r.balance<50000 then
            txid=concat('TX00',num);
            num=num+1;
            insert into transactions
            values(txid,r.account_id,1000,'Credit');
        end if;
        raise notice '%',r.balance;
    end loop;
    close c1;
end;
```

```
$$language plpgsql;
```

2.10.2 Output

```
exam=# create or replace function help() returns void as $$
exam$# declare
exam$# num integer;
exam$# i integer=0;
exam$# txid char(6);
exam$# c1 cursor for select * from account;
exam$# r record;
exam$# len int;
exam$# begin
exam$# select count(*) from transactions into num;
exam$# num=num+1;
exam$# select count(*) from account into len;
exam$# open c1;
exam$# loop
exam$# fetch c1 into r;
exam$# i=i+1;
exam$# if i>len then
exam$# exit;
Use control-D to quit.
exam$# end if;
exam$# if r.balance<50000 then
exam$# txid=concat('TX00',num);
exam$# num=num+1;
exam$# insert into transactions
exam$# values(txid,r.account_id,1000,'Credit');
exam$# end if;
exam$# raise notice '%',r.balance;
exam$# end loop;
exam$# close c1;
exam$# end;
exam$# $$language plpgsql;
CREATE FUNCTION
```

Figure 18: Function help

```
exam=# select  * from account ;
 account_id |  type   | balance | credit_line
------------+---------+---------+-------------
 AC0007     | Savings |   18000 |       20000
 AC0001     | Savings |   30000 |       48500
 AC0002     | Savings |   50000 |       48500
 AC0003     | Current |   75000 |       97000
 AC0004     | Savings |   80000 |       48500
 AC0005     | Current |   25000 |       19400
 AC0006     | Current |   22000 |       48500
(7 rows)

exam=# select * from transactions ;
 tx_id  | account_id | amount |  type
--------+------------+--------+--------
 TX0001 | AC0003     |   5000 | Credit
 TX0002 | AC0002     |   7500 | Debit
 TX0003 | AC0003     |   8000 | Debit
 TX0004 | AC0001     |    500 | Credit
 TX0005 | AC0005     |   4000 | Debit
 TX0006 | AC0004     |   1000 | Debit
 TX0007 | AC0007     |   2000 | Credit
 TX0008 | AC0006     |   7000 | Debit
 TX0009 | AC0001     |   3000 | Credit
 TX0010 | AC0006     |   2500 | Credit
 TX0011 | AC0006     |   2500 | Credit
 TX0012 | AC0006     |   2500 | Debit
(12 rows)

exam=# select * from help();
NOTICE:  18000
NOTICE:  30000
NOTICE:  50000
NOTICE:  75000
NOTICE:  80000
NOTICE:  25000
NOTICE:  22000
 help
------

(1 row)
```

Figure 19: Old values

```
exam=# select * from transactions ;
 tx_id   | account_id | amount |  type
---------+------------+--------+--------
 TX0001 | AC0003     |   5000 | Credit
 TX0002 | AC0002     |   7500 | Debit
 TX0003 | AC0003     |   8000 | Debit
 TX0004 | AC0001     |    500 | Credit
 TX0005 | AC0005     |   4000 | Debit
 TX0006 | AC0004     |   1000 | Debit
 TX0007 | AC0007     |   2000 | Credit
 TX0008 | AC0006     |   7000 | Debit
 TX0009 | AC0001     |   3000 | Credit
 TX0010 | AC0006     |   2500 | Credit
 TX0011 | AC0006     |   2500 | Credit
 TX0012 | AC0006     |   2500 | Debit
 TX0013 | AC0007     |   1000 | Credit
 TX0014 | AC0001     |   1000 | Credit
 TX0015 | AC0005     |   1000 | Credit
 TX0016 | AC0006     |   1000 | Credit
(16 rows)

exam=# select  * from account ;
 account_id |  type   | balance | credit_line
------------+---------+---------+-------------
 AC0002     | Savings |   50000 |       48500
 AC0003     | Current |   75000 |       97000
 AC0004     | Savings |   80000 |       48500
 AC0007     | Savings |   19000 |       20000
 AC0001     | Savings |   31000 |       48500
 AC0005     | Current |   26000 |       19400
 AC0006     | Current |   23000 |       48500
(7 rows)
```

Figure 20: New values

2.11 Finding prime

Given 2 distinct prime integers, p q (2 ¡ p,q), find a prime number n such that p ¡ n ¡q and (q-n)*(n-p) is maximum.
If two such n exist print the minimum of the two and
If no prime number exist between them, print -1.

2.11.1 Code

```
CREATE OR REPLACE FUNCTION prime(p INT,q INT) RETURNS INT AS
$$
```

```
DECLARE
i INT ;
j INT;
num INT = -1;
max INT =0;
isprime INT =1;
BEGIN
FOR i IN (p+1)..(q-1) LOOP
isprime = 1;
FOR j in 2 ..(i/2) LOOP
IF i%j = 0 THEN
isprime = 0;
END IF;
END LOOP;
IF isprime = 1 THEN
IF (i-p)*(q-i) > max THEN
num = i;
max = (i-p)*(q-i);
END IF;
END IF;
END LOOP;
RETURN num;
END;
$$
LANGUAGE PLPGSQL;
```

2.11.2 Output

```
exam=# CREATE OR REPLACE FUNCTION prime(p INT,q INT) RETURNS INT AS
exam-# $$
exam$# DECLARE
exam$#  i INT ;
exam$#  j INT;
exam$#  num INT = -1;
exam$#  max INT =0;
exam$#  isprime INT =1;
exam$# BEGIN
exam$#  FOR i IN (p+1)..(q-1) LOOP
exam$#   isprime = 1;
exam$#   FOR j in 2 ..(i/2) LOOP
exam$#    IF i%j = 0 THEN
exam$#     isprime = 0;
exam$#    END IF;
exam$#   END LOOP;
exam$#   IF isprime = 1 THEN
exam$#    IF (i-p)*(q-i) > max THEN
exam$#     num = i;
exam$#     max = (i-p)*(q-i);
exam$#    END IF;
exam$#   END IF;
exam$#  END LOOP;
exam$#  RETURN num;
exam$# END;
exam$# $$
exam-# LANGUAGE PLPGSQL;
CREATE FUNCTION
```

Figure 21: Function Finding prime

```
exam=# select * from prime(3,7);
 prime
-------
     5
(1 row)

exam=# select * from prime(3,5);
 prime
-------
    -1
(1 row)

exam=# select * from prime(3,11);
 prime
-------
     7
(1 row)

exam=# select * from prime(3,13);
 prime
-------
     7
(1 row)

exam=# select * from prime(3,19);
 prime
-------
    11
(1 row)
```

Figure 22: Function finding prime