DBMS ASSIGNMENT- 4
ON
PRODUCT SUPPLY MANAGEMENT SYSTEM

Submitted as a part of course curriculum for

UE19CS301 DATABASE MANAGEMENT SYSTEM

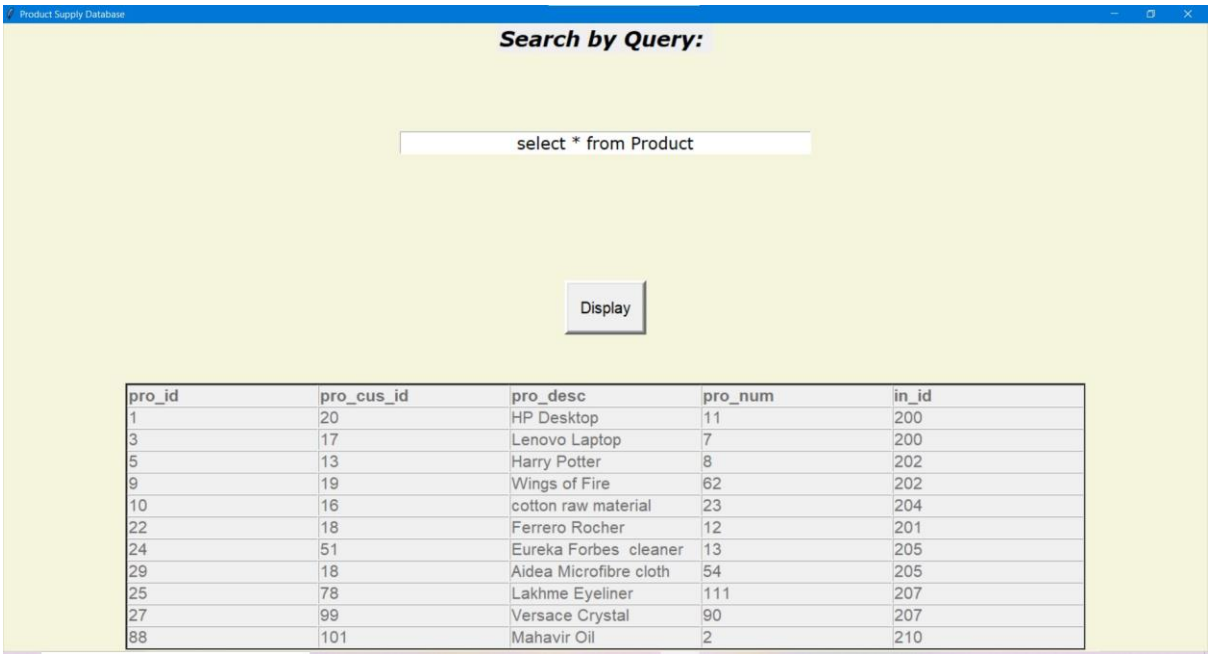
Names	SRN
Anagha Suresh	PES2UG19CS037

Part 1:

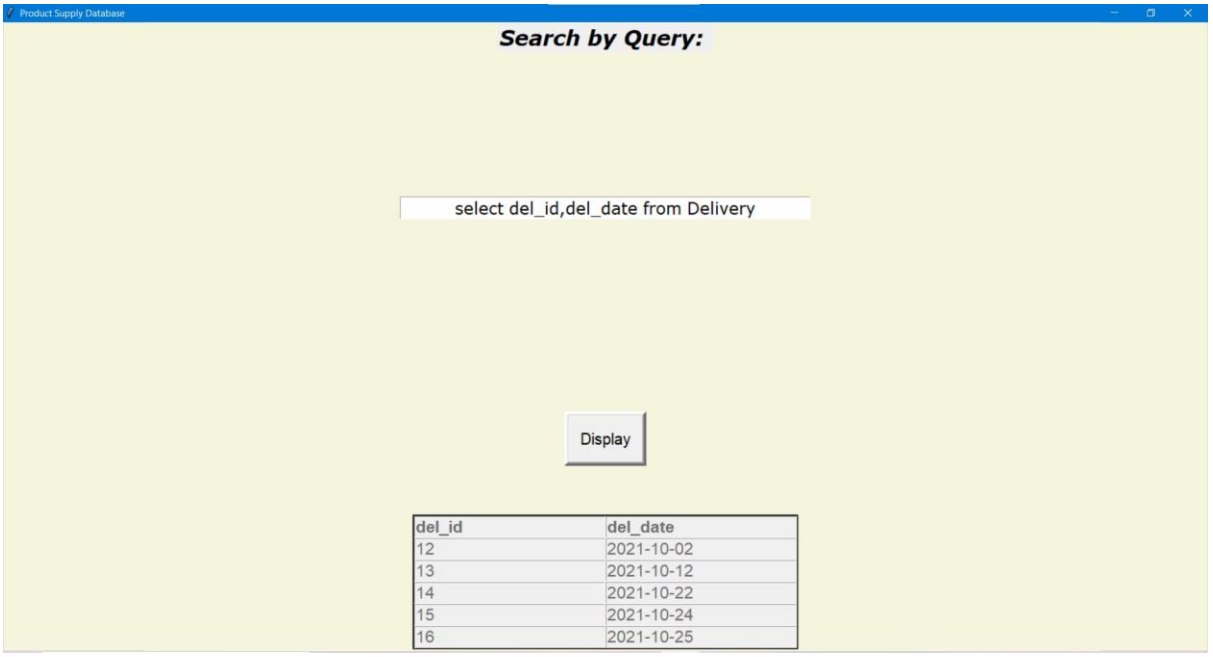
Front End using Python (tkinter):

1. An entry field to enter the query
2. A button that executes the entered query and display the results
3. A display field

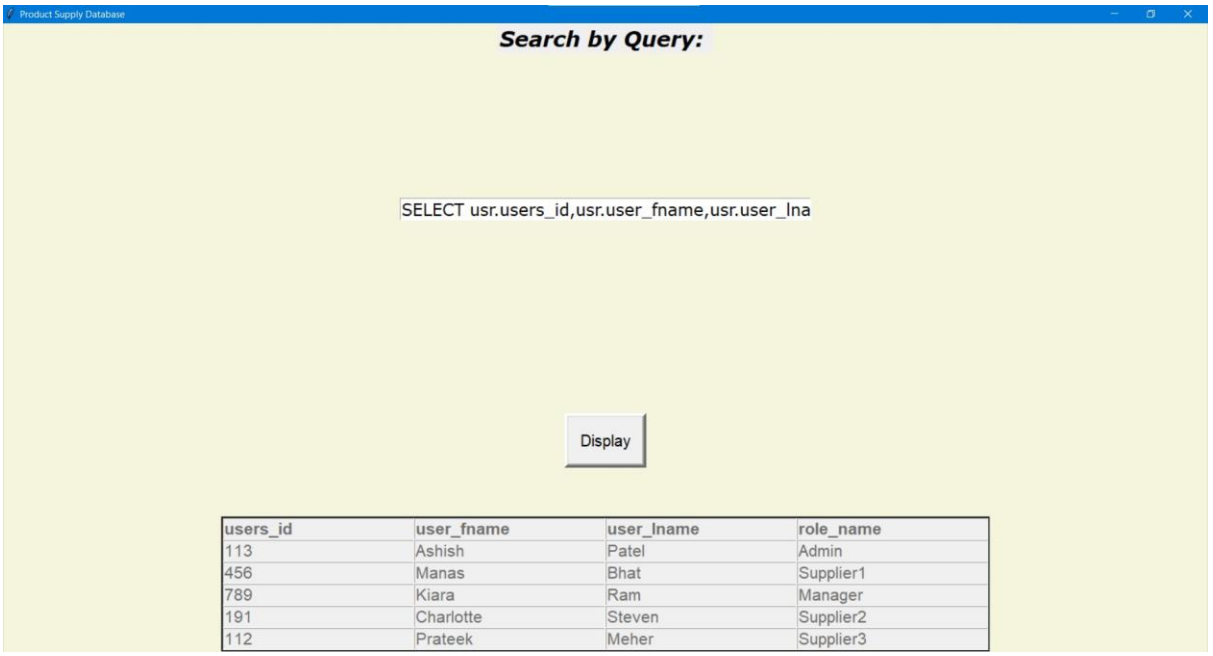
The result generated when a query is executed and the results are displayed as table



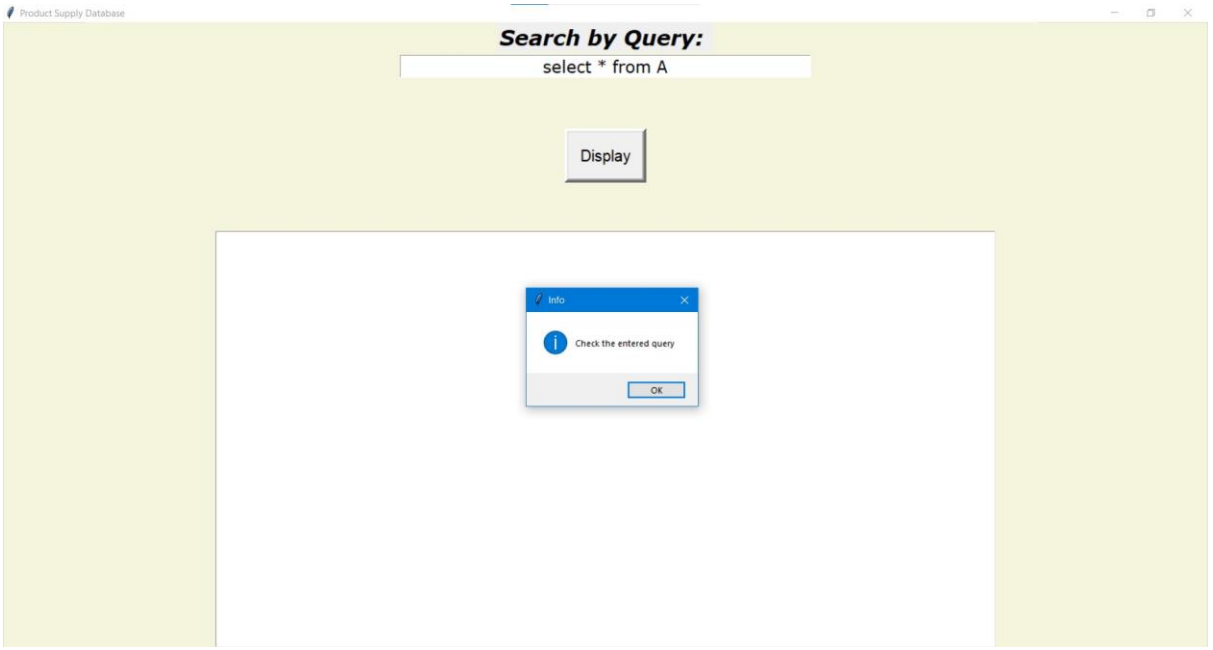
pro_id	pro_cus_id	pro_desc	pro_num	in_id
1	20	HP Desktop	11	200
3	17	Lenovo Laptop	7	200
5	13	Harry Potter	8	202
9	19	Wings of Fire	62	202
10	16	cotton raw material	23	204
22	18	Ferrero Rocher	12	201
24	51	Eureka Forbes cleaner	13	205
29	18	Aidea Microfibre cloth	54	205
25	78	Lakhme Eyeliner	111	207
27	99	Versace Crystal	90	207
88	101	Mahavir Oil	2	210



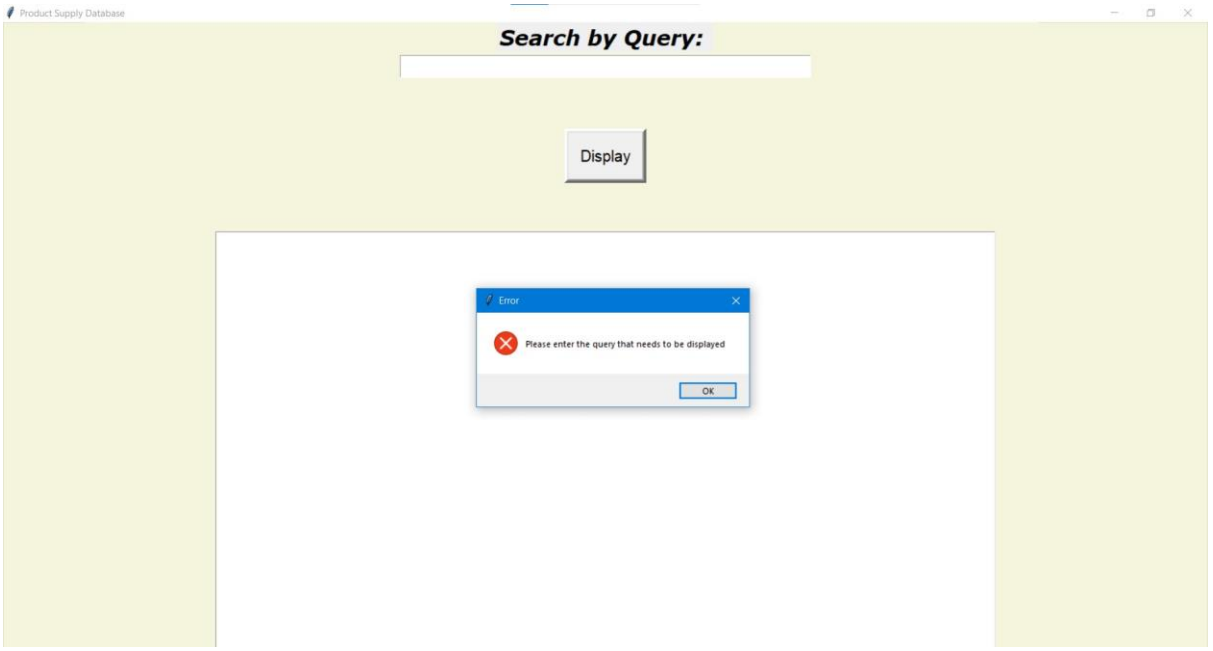
Output on applying Natural join:



On entering a query that doesn't exist (a table that doesn't exist): An information message box is shown



On trying to execute the query without entering it, an error message box is shown



Part 2:
Additional Queries

1.Using constraints ,alter the table Membership to check for the purchase date greater than 19-11-2021.

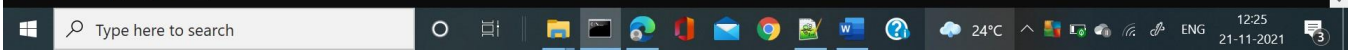
```
INSERT 0 1
INSERT 0 1
INSERT 0 1

D:\Install\POSTGRESQL\bin>psql -U postgres
Password for user postgres:
psql (14.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# \c productsupply
You are now connected to database "productsupply" as user "postgres".
productsupply=# ALTER TABLE Membership
productsupply-# ADD CONSTRAINT check_date
productsupply-# CHECK(Purchase_date>='10-03-2021');
ALTER TABLE
productsupply=#
```


```
productsupply=# \d Membership
                    Table "public.membership"
     Column      |          Type          | Collation | Nullable | Default
-----------------+------------------------+-----------+----------+---------
 purchase_id     | integer                |           | not null |
 expiry_date     | date                   |           | not null |
 purchase_details | character varying(500) |           |          |
 purchase_date   | date                   |           | not null |
Indexes:
    "membership_pkey" PRIMARY KEY, btree (purchase_id)
Check constraints:
    "check_date" CHECK (purchase_date >= '2021-03-10'::date)
Referenced by:
    TABLE "expressdelivery" CONSTRAINT "expressdelivery_pur_id_fkey" FOREIGN KEY (pur_id) REFERENCES membership
(purchase_id)


productsupply=#
```

2.Using constraints set the Login_role_id of table Login to not null.

ALTER TABLE Login
ALTER COLUMN Login_role_id NOT NULL;

```
INSERT 0 1

D:\Install\POSTGRESQL\bin>psql -U postgres
Password for user postgres:
psql (14.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# \c productsupply
You are now connected to database "productsupply" as user "postgres".
productsupply=# ALTER TABLE Login
productsupply-# ALTER COLUMN Login_role_id SET NOT NULL;
ALTER TABLE
productsupply=# _
```



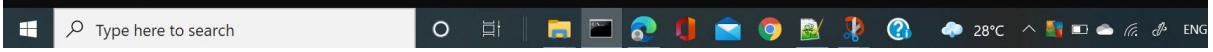
3.Using constraints to alter the table Login to add a unique key.

```
ALTER TABLE
productsupply=# ALTER TABLE Login
productsupply-# ADD UNIQUE(Login_id);
ALTER TABLE
productsupply=#
```

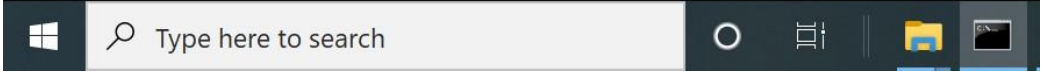

```
productsupply=# ALTER TABLE Login
productsupply-# ALTER COLUMN Login_role_id SET NOT NULL;
ALTER TABLE
productsupply=# ALTER TABLE Login
productsupply-# ADD UNIQUE(Login_id);
ALTER TABLE
productsupply=# \d Login
                    Table "public.login"
     Column     |         Type          | Collation | Nullable | Default
----------------+-----------------------+-----------+----------+---------
 login_id       | integer               |           | not null |
 login_role_id  | integer               |           | not null |
 login_username | character varying(20) |           | not null |
 user_password  | character varying(20) |           | not null |
Indexes:
    "login_pkey" PRIMARY KEY, btree (login_id, login_username)
    "login_login_id_key" UNIQUE CONSTRAINT, btree (login_id)
Referenced by:
    TABLE "permission" CONSTRAINT "permission_login_num_login_name_fkey" FOREIGN KEY (login_num, login_name) RE
FERENCES login(login_id, login_username)


productsupply=#
```

4.Adding a new column Gender into the Users table and also inserting values into the column created.

```
productsupply=# ALTER TABLE Users
productsupply-# ADD COLUMN gender varchar(10)
productsupply-# DEFAULT '';
ALTER TABLE
productsupply=# \d Users
                              Table "public.users"
    Column    |          Type          | Collation | Nullable |        Default
--------------+------------------------+-----------+----------+-----------------------
 users_id     | integer                |           | not null |
 user_fname   | character varying(20)  |           | not null |
 user_lname   | character varying(20)  |           | not null |
 user_email   | character varying(20)  |           |          |
 user_address | character varying(100) |           |          |
 user_mobile  | bigint                 |           |          |
 gender       | character varying(10)  |           |          | ''::character varying
Indexes:
    "users_pkey" PRIMARY KEY, btree (users_id)
Referenced by:
    TABLE "roles" CONSTRAINT "roles_usernum_fkey" FOREIGN KEY (usernum) REFERENCES users(users_id)


productsupply=#
```



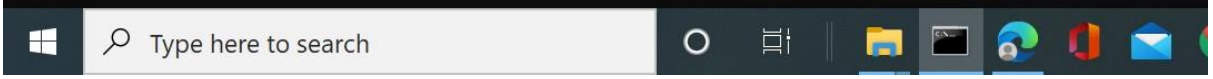

```
productsupply=# UPDATE Users
productsupply-# SET gender='M'
productsupply-# WHERE Users_id=113;
UPDATE 1
productsupply=# UPDATE Users
productsupply-# SET gender='M'
productsupply-# WHERE Users_id=456;
UPDATE 1
productsupply=# UPDATE Users
productsupply-# SET gender='F'
productsupply-# WHERE Users_id=789;
UPDATE 1
productsupply=# UPDATE Users
productsupply-# SET gender='F'
productsupply-# WHERE Users_id=191;
UPDATE 1
productsupply=# UPDATE Users
productsupply-# SET gender='M'
productsupply-# WHERE Users_id=112;
UPDATE 1
productsupply=#
```



```
productsupply=# select * from Users
productsupply-# ;
 users_id | user_fname | user_lname |      user_email      | user_address | user_mobile | gender
----------+------------+------------+----------------------+--------------+-------------+--------
      113 | Ashish     | Patel      | ashishp@gmail.com    | XYZ          |  9845237812 | M
      456 | Manas      | Bhat       | manasb@gmail.com     | CDF          |  9324127722 | M
      789 | Kiara      | Ram        | kiarar01@gmail.com   | ABC          |  9513236789 | F
      191 | Charlotte  | Steven     | charlottes@gmail.com | KLM          |  9867231299 | F
      112 | Prateek    | Meher      | prateekm@gmail.com   | GHI          |  9966234511 | M
(5 rows)


productsupply=#
```

5.Write a query to rename the existing schema name.

```
productsupply=# ALTER SCHEMA PUBLIC RENAME TO Product_Supply
productsupply-# ;
ALTER SCHEMA
productsupply=#
```



Part 3:
Choice of NO-SQL Database Variety incase of migration from the present choice

The database implementation in this project has been carried out using postgresql a relational database which is an advanced version of SQL that provides support to different functions of SQL.It is noticeable that in case of a relational database it leads to having too many tables, complicated queries and slower performance in many instances.Due to performance issues or other requirements, if the current database should be migrated to an alternative one,our choice is a No-sql database of Mongodb which is a document oriented database where it stores data in collections instead of tables.They allow storing objects without introducing additional entities, thus allowing for faster and simpler development process.It also allows us to have objects in one collection that can have different sets of fields that is information specific to only certain records if needed.

Simple Queries:

1.Query to view products whose customer id is greater than 5.

```
productsupply=# select * from Product  where pro_cus_id>5;
 pro_id | pro_cus_id |        pro_desc        | pro_num | in_id
--------+------------+------------------------+---------+-------
      1 |         20 | HP Desktop             |      11 |   200
      3 |         17 | Lenovo Laptop          |       7 |   200
      5 |         13 | Harry Potter           |       8 |   202
      9 |         19 | Wings of Fire          |      62 |   202
     10 |         16 | cotton raw material    |      23 |   204
     22 |         18 | Ferrero Rocher         |      12 |   201
     24 |         51 | Eureka Forbes  cleaner |      13 |   205
     29 |         18 | Aidea Microfibre cloth |      54 |   205
     25 |         78 | Lakhme Eyeliner        |     111 |   207
     27 |         99 | Versace Crystal        |      90 |   207
     88 |        101 | Mahavir Oil            |       2 |   210
(11 rows)
```

2.Name all users whose name starts with letter ‘A’.

```
productsupply=# select user_fname from users where user_fname like 'A%';
 user_fname
------------
 Ashish
(1 row)
```

3.Query to view stock available which has stock id less than 1235 and inventory id less than 205.

```
productsupply=# select * from stock where stk_id<1235 and iid<205;
 stk_id |      stk_desc       | iid
--------+---------------------+-----
   1230 | Lenovo Laptop       | 200
   1234 | Hp desktop          | 200
   1232 | Cotton Raw material | 204
(3 rows)
```

4.Query to view number of products whose product number is greater than 10.

```
productsupply=# Select COUNT(*) from product where pro_num>10;
 count
-------
     8
(1 row)
```

5.Query to view product id numbers above 8 but not in (22,29).

```
productsupply=# select * from product where pro_id>8
productsupply-# EXCEPT
productsupply-# select * from product where pro_id in (22,29);
 pro_id | pro_cus_id |        pro_desc         | pro_num | in_id
--------+------------+-------------------------+---------+-------
     88 |        101 | Mahavir Oil             |       2 |   210
     24 |         51 | Eureka Forbes  cleaner  |      13 |   205
     10 |         16 | cotton raw material     |      23 |   204
     27 |         99 | Versace Crystal         |      90 |   207
     25 |         78 | Lakhme Eyeliner         |     111 |   207
      9 |         19 | Wings of Fire           |      62 |   202
(6 rows)
```

Complex queries:

1.Retrieve details of all products which are present in stock.

```
productsupply=# select * from product p where EXISTS (select * from stock s where s.iid=p.in_id);
 pro_id | pro_cus_id |        pro_desc         | pro_num | in_id
--------+------------+-------------------------+---------+-------
      1 |         20 | HP Desktop              |      11 |   200
      3 |         17 | Lenovo Laptop           |       7 |   200
      5 |         13 | Harry Potter            |       8 |   202
      9 |         19 | Wings of Fire           |      62 |   202
     10 |         16 | cotton raw material     |      23 |   204
     24 |         51 | Eureka Forbes  cleaner  |      13 |   205
     29 |         18 | Aidea Microfibre cloth  |      54 |   205
     25 |         78 | Lakhme Eyeliner         |     111 |   207
     27 |         99 | Versace Crystal         |      90 |   207
     88 |        101 | Mahavir Oil             |       2 |   210
(10 rows)
```

2.Retrieve product description ,id and inventory id whose inventory id is less than maximum.

```
productsupply=# select pro_desc,pro_id,in_id from product where in_id in(select inv_id from invitems where inv_id<(select MAX(inv_id) from invitems));
       pro_desc        | pro_id | in_id
-----------------------+--------+-------
 HP Desktop            |      1 |   200
 Lenovo Laptop         |      3 |   200
 Harry Potter          |      5 |   202
 Wings of Fire         |      9 |   202
 cotton raw material   |     10 |   204
 Ferrero Rocher        |     22 |   201
 Eureka Forbes  cleaner|     24 |   205
 Aidea Microfibre cloth|     29 |   205
 Lakhme Eyeliner       |     25 |   207
 Versace Crystal       |     27 |   207
(10 rows)
```

3.Query to retrieve distinct product description,inventory id from product which has id greater than 204.

```
productsupply=# select DISTINCT pro_desc,in_id from product where  in_id in (select DISTINCT  inv_id from  invtype  where inv_id>204) ;
        PRODUCTSUPPLY
       pro_desc        | in_id
-----------------------+-------
 Mahavir Oil           |   210
 Aidea Microfibre cloth|   205
 Versace Crystal       |   207
 Lakhme Eyeliner       |   207
 Eureka Forbes  cleaner|   205
(5 rows)
```

4.Query to retrieve express delivery date and description of delivery from express delivery whose description is in the delivery table whose id is greater than 13.

```
productsupply=# select del_desc,express_del_date from expressdelivery where del_desc in(select del_desc from delivery wh
ere del_id>13);
      del_desc      | express_del_date
--------------------+------------------
 electronic devices | 2021-10-14
 electronic devices | 2021-10-12
 books              | 2021-10-15
 electronic devices | 2021-10-19
(4 rows)
```

5.Query to retrieve permission names,login number which are present in both permission and roles.

```
productsupply=# select per_name,login_num from permission p where EXISTS(select * from roles r where r.role_name=p.per_name);
 per_name  | login_num
-----------+-----------
 Supplier1 |       345
 Manager   |       678
 Supplier2 |       910
 Supplier3 |       101
(4 rows)
```

Triggers:

1. Create a trigger that adds the details of the information inserted to the delivery table to a new table notify

```
productsupply=# CREATE TABLE Notify(
productsupply(# Delivery_date DATE NOT NULL,
productsupply(# Delivery_item VARCHAR(500) NOT NULL);
CREATE TABLE
productsupply=# CREATE OR REPLACE FUNCTION add_del() RETURNS trigger AS $add$
productsupply$# BEGIN
productsupply$# INSERT INTO Notify
productsupply$# (Delivery_date,Delivery_item) values(new.Del_date,new.Del_desc);
productsupply$# RETURN NEW;
productsupply$# END;
productsupply$# $add$
productsupply-# language plpgsql;
CREATE FUNCTION
```

```
productsupply=# CREATE TRIGGER add
productsupply-# AFTER INSERT
productsupply-# ON Delivery
productsupply-# FOR EACH ROW
productsupply-# EXECUTE PROCEDURE add_del();
CREATE TRIGGER
```

```
productsupply=# INSERT INTO
productsupply-# Delivery
productsupply-# VALUES(17,'2021-11-28',509,'books');
INSERT 0 1
productsupply=# INSERT INTO
productsupply-# Delivery
productsupply-# VALUES(18,'2021-11-30',120,'oil-supplies');
INSERT 0 1
productsupply=# select * from Notify;
 delivery_date | delivery_item
---------------+---------------
 2021-11-28    | books
 2021-11-30    | oil-supplies
(2 rows)
```

2. Create a trigger that notifies when the inventory has been updated and sends a message via a table created to display message

```
productsupply=# CREATE TABLE message(
productsupply(# text VARCHAR(500));
CREATE TABLE
productsupply=# CREATE OR REPLACE FUNCTION update_inv() RETURNS trigger as $inv_up$
productsupply$# BEGIN
productsupply$# INSERT INTO
productsupply$# message
productsupply$# (text) values('Inventory has been updated');
productsupply$# RETURN NEW;
productsupply$# END;
productsupply$# $inv_up$
productsupply-# language plpgsql;
CREATE FUNCTION
```

```
productsupply=# CREATE TRIGGER inv_up
productsupply-# AFTER UPDATE
productsupply-# ON Inventory
productsupply-# FOR EACH ROW
productsupply-# EXECUTE PROCEDURE update_inv();
CREATE TRIGGER
```

```
productsupply=# UPDATE Inventory
productsupply-# SET Inv_num=22
productsupply-# WHERE Inv_id = 210;
UPDATE 1
productsupply=# UPDATE Inventory
productsupply-# SET Inv_num=80
productsupply-# WHERE Inv_id = 202;
UPDATE 1
productsupply=# select * from message;
            text
----------------------------
 Inventory has been updated
 Inventory has been updated
(2 rows)
```

3. Create a trigger that adds the user name when a new user is been added to the database

```
productsupply=# CREATE TABLE User_details(
productsupply(# fname VARCHAR(500),
productsupply(# lname VARCHAR(500));
CREATE TABLE
productsupply=# CREATE OR REPLACE FUNCTION details() RETURNS trigger AS $det$
productsupply$# BEGIN
productsupply$# INSERT INTO User_details
productsupply$# (fname,lname) values(new.User_fname,new.User_lname);
productsupply$# RETURN NEW;
productsupply$# END;
productsupply$# $det$
productsupply-# language plpgsql;
CREATE FUNCTION
```

```
productsupply=# CREATE TRIGGER det
productsupply-# AFTER INSERT
productsupply-# ON Users
productsupply-# FOR EACH ROW
productsupply-# EXECUTE PROCEDURE details();
CREATE TRIGGER
```

```
productsupply=# insert into Users values(221,'Sanya','Shetty','sanya@yahoo.com','DFG',9125235108);
INSERT 0 1
productsupply=# insert into Users values(705,'Dev','Kumar','dev_kumar@yahoo.com','KLD',8933335108);
INSERT 0 1
productsupply=# select * from User_details;
 fname | lname
-------+--------
 Sanya | Shetty
 Dev   | Kumar
(2 rows)
```

4. Create a trigger to raise an information when a new product is been added to the Product relation ensure that the product is added by checking whether pro_num > 0

```
productsupply=# CREATE OR REPLACE FUNCTION ProductAdd() RETURNS TRIGGER AS $$
productsupply$# BEGIN
productsupply$# if NEW.Pro_num > 0 THEN
productsupply$# RAISE INFO 'A new product has been added';
productsupply$# END IF;
productsupply$# RETURN NULL;
productsupply$# END;
productsupply$# $$
productsupply-# LANGUAGE PLPGSQL;
CREATE FUNCTION
```

```
productsupply=# CREATE TRIGGER Pro_add
productsupply-# AFTER INSERT
productsupply-# ON Product
productsupply-# FOR EACH ROW
productsupply-# EXECUTE PROCEDURE ProductAdd();
CREATE TRIGGER
```

```
productsupply=# INSERT INTO Product VALUES(32,88,'Revlon Lipstick',20,205);
INFO:  A new product has been added
INSERT 0 1
productsupply=# INSERT INTO Product VALUES(35,43,'Kajal',0,205);
INSERT 0 1
```

5.Create trigger that adds product when product has been updated and displays message through table created to display message.

```
productsupply=# CREATE table msgs(
productsupply(# text varchar(500));
CREATE TABLE
productsupply=# CREATE or REPLACE function update_pro() returns trigger as $pro_up$
productsupply$# BEGIN
productsupply$# INSERT INTO
productsupply$# msgs
productsupply$# (text) values('product has been updated');
productsupply$# RETURN NEW;
productsupply$# END;
productsupply$# $pro_up$
productsupply-# language plpgsql;
CREATE FUNCTION
```

```
productsupply=# CREATE TRIGGER prod_up
productsupply-# AFTER INSERT
productsupply-# ON PRODUCT
productsupply-# FOR EACH ROW
productsupply-# EXECUTE PROCEDURE update_pro();
CREATE TRIGGER
```

```
productsupply=# insert into product values(35,129,'dark chocolate',21,201);
INSERT 0 1
productsupply=# select * from msgs;
          text
--------------------------
 product has been updated
(1 row)


productsupply=# insert into product values(40,145,'white chocolate',40,201);
INSERT 0 1
productsupply=# select * from msgs;
          text
--------------------------
 product has been updated
 product has been updated
(2 rows)
```

Functions

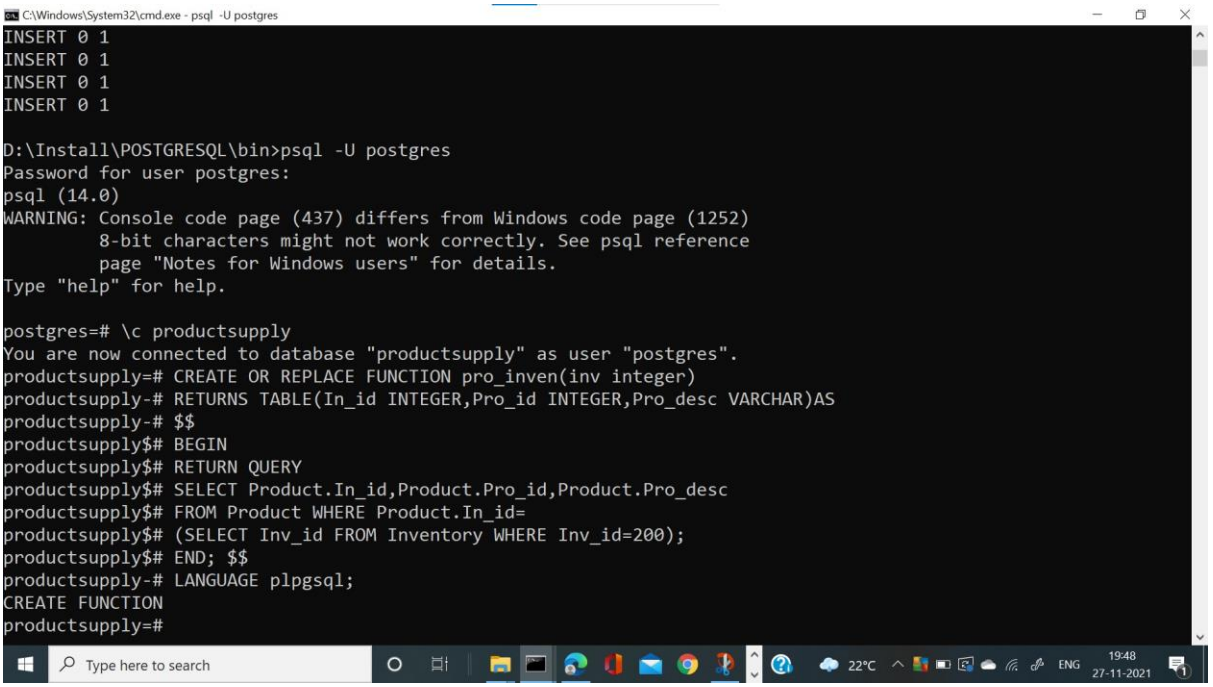
1. Write a function to fetch the user details who have made changes to the table roles using their role_id

```
productsupply=# CREATE OR REPLACE FUNCTION get_user_details(Id integer)
productsupply-# RETURNS TABLE(UserId INTEGER, UserName VARCHAR, Mail VARCHAR) AS
productsupply-# $$
productsupply$# BEGIN
productsupply$# RETURN QUERY
productsupply$# SELECT Users.Users_id, Users.User_fname, Users.User_email
productsupply$# FROM Users WHERE Users.Users_id=
productsupply$# (SELECT Usernum FROM Roles WHERE Roles.Role_id=Id);
productsupply$# END; $$
productsupply-# LANGUAGE plpgsql;
CREATE FUNCTION
```

```python
import psycopg2

try:
    ps_connection = psycopg2.connect(user="postgres",
                                     password="1234",
                                     host="::1",
                                     port="5432",
                                     database="productsupply")

    cursor = ps_connection.cursor()

    # call procedure
    cursor.callproc('get_user_details', [131, ])

    print("\nFetching User details who pushed changes\n")
    result = cursor.fetchall()
    for row in result:
        print("Id = ", row[0], )
        print("Name = ", row[1])
        print("Email Address  = ", row[2])

except (Exception, psycopg2.DatabaseError) as error:
    print("Error while connecting to PostgreSQL", error)

finally:
    # closing database connection.
    if ps_connection:
        cursor.close()
        ps_connection.close()
        print("\nPostgreSQL connection is closed")
```

```
C:\Users\Anusha\Desktop\DBMS project>python3 stored1.py

Fetching User details who pushed changes

Id =  789
Name =  Kiara
Email Address  =  kiarar01@gmail.com

PostgreSQL connection is closed
```

2.Write a function to display all the products that belong to inventory id 200.



```python
import psycopg2
try:
    ps_connection = psycopg2.connect(user="postgres",
                                     password="anu123",
                                     host="::1",
                                     port="5432",
                                     database="productsupply")
    cursor=ps_connection.cursor()

    #Call stored procedure
    cursor.callproc('pro_inven',[200,])

    print("\nFetching all the products of inventory 200 \n")
    result=cursor.fetchall()
    for row in result:
        print("Inventory number:",row[0], )
        print("Product Id:",row[1], )
        print("Product Description:",row[2], )


except (Exception, psycopg2.DatabaseError)as error:
    print("Error while connecting to postgresql",error)

finally:
```

Python file length : 1,013 lines : 31 Ln : 14 Col : 29 Sel : 0 | 0 Windows (CR LF) UTF-8 INS

```python
    cursor=ps_connection.cursor()

    #Call stored procedure
    cursor.callproc('pro_inven',[200,])

    print("\nFetching all the products of inventory 200 \n")
    result=cursor.fetchall()
    for row in result:
        print("Inventory number:",row[0], )
        print("Product Id:",row[1], )
        print("Product Description:",row[2], )


except (Exception, psycopg2.DatabaseError)as error:
    print("Error while connecting to postgresql",error)

finally:
    #closing the database connection.
    if ps_connection:
        cursor.close()
        ps_connection.close()
        print("\nPostgresql connection is closed")
```

ython file length : 1,013 lines : 31 Ln : 14 Col : 29 Sel : 0 | 0 Windows (CR LF) UTF-8 INS

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. All rights reserved.

D:\Install\Python3.7.4>python func2.py

Fetching all the products of inventory 200

Inventory number: 200
Product Id: 1
Product Description: HP Desktop
Inventory number: 200
Product Id: 3
Product Description: Lenovo Laptop

Postgresql connection is closed

D:\Install\Python3.7.4>
```

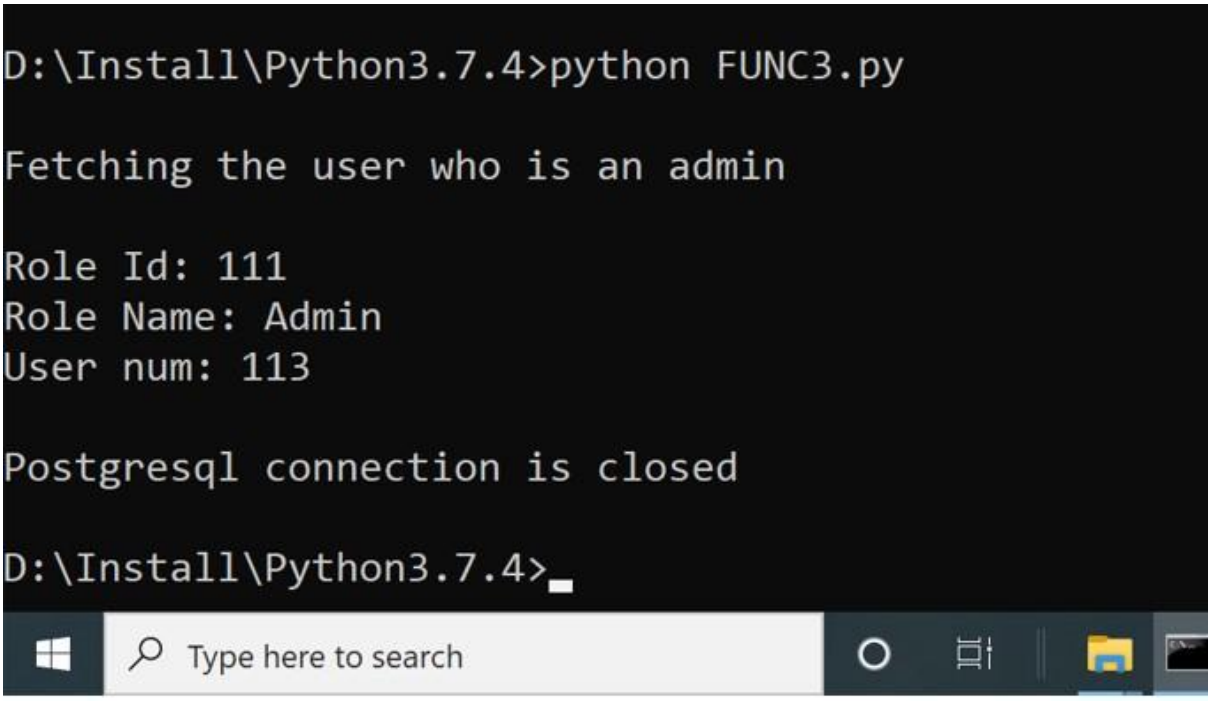
Type here to search 22°C ENG 19:51 27-11-2021

3.Write a function to fetch the id of the user who is the Administrator

```
postgres=# \c productsupply
You are now connected to database "productsupply" as user "postgres".
productsupply=# CREATE OR REPLACE FUNCTION role_admin(userrole integer)
productsupply-# RETURNS TABLE(Role_id INTEGER,Role_name VARCHAR,Usernum INTEGER)AS
productsupply-# $$
productsupply$# BEGIN
productsupply$# RETURN QUERY
productsupply$# SELECT Roles.Role_id,Roles.Role_name,Roles.Usernum
productsupply$# FROM Roles WHERE Roles.Usernum=
productsupply$# (SELECT Roles.Usernum FROM Roles WHERE Roles.Role_id=111);
productsupply$# END; $$
productsupply-# LANGUAGE plpgsql;
CREATE FUNCTION
productsupply=#
```


```python
import psycopg2
try:
    ps_connection = psycopg2.connect(user="postgres",
                                     password="anu123",
                                     host="::1",
                                     port="5432",
                                     database="productsupply")
    cursor=ps_connection.cursor()

    #Call stored procedure
    cursor.callproc('role_admin',[111,])

    print("\nFetching the user who is an admin \n")
    result=cursor.fetchall()
    for row in result:
        print("Role Id:",row[0], )
        print("Role Name:",row[1], )
        print("User num:",row[2], )


except (Exception, psycopg2.DatabaseError)as error:
    print("Error while connecting to postgresql",error)

finally:
```

```
    cursor=ps_connection.cursor()

    cursor.callproc('role_admin',[111,])

    print("\nFetching the user who is an admin \n")
    result=cursor.fetchall()
    for row in result:
        print("Role Id:",row[0], )
        print("Role Name:",row[1], )
        print("User num:",row[2], )


except (Exception, psycopg2.DatabaseError)as error:
    print("Error while connecting to postgresql",error)

finally:
    #closing the database connection.
    if ps_connection:
        cursor.close()
        ps_connection.close()
        print("\nPostgresql connection is closed")
```



Stored Procedures:

1. Create a stored procedure using transactions to modify the changes made to the Delivery table

```
productsupply=# CREATE OR REPLACE PROCEDURE del_date_changes()
productsupply-# LANGUAGE plpgsql
productsupply-# AS
productsupply-# $$
productsupply$# begin
productsupply$# UPDATE Delivery
productsupply$# SET Del_date='2021-11-12'
productsupply$# WHERE Del_id=14;
productsupply$# COMMIT;
productsupply$# END;
productsupply$# $$
productsupply-# ;
CREATE PROCEDURE
productsupply=# call del_date_changes();
CALL
```

```
productsupply=# SELECT * FROM   Delivery;
 del_id |  del_date   | del_cus_id |      del_desc
--------+-------------+------------+--------------------
     15 | 2021-10-24 |        241 | books
     16 | 2021-10-25 |        134 | cleaning supplies
     18 | 2021-10-30 |        120 | oil-supplies
     13 | 2021-10-29 |        236 | cosmetics
     12 | 2021-10-27 |        234 | electronic devices
     14 | 2021-11-12 |        221 | electronic devices
(6 rows)
```

2. Create a stored procedure on Membership table to insert values and use rollback to avoid the changes to reflect on the table

```
productsupply=# CREATE OR REPLACE PROCEDURE membership_insert()
productsupply-# LANGUAGE plpgsql
productsupply-# AS
productsupply-# $$
productsupply$# begin
productsupply$# INSERT INTO Membership VALUES(55,'2021-10-31','3 months membership','2021-08-01');
productsupply$# INSERT INTO Membership VALUES(83,'2021-11-24','12 months membership','2020-11-24');
productsupply$# ROLLBACK;
productsupply$# END;
productsupply$# $$
productsupply-# ;
CREATE PROCEDURE
productsupply=# call membership_insert();
CALL
productsupply=# SELECT * FROM Membership;
 purchase_id | expiry_date |   purchase_details   | purchase_date
-------------+-------------+----------------------+---------------
          78 | 2021-09-10  | 6 months membership  | 2021-03-10
          79 | 2021-10-11  | 3 months membership  | 2021-07-11
          87 | 2021-11-17  | 12 months membership | 2022-11-17
          89 | 2021-12-19  | 1 month membership   | 2021-11-19
          91 | 2021-12-24  | 6 months membership  | 2021-06-24
(5 rows)
```

USER PRIVILEGES:

1.Grant select on product and stock to user anu.

```
productsupply=# create user anu_gupta with password 'anu';
CREATE ROLE
productsupply=# GRANT select on product,stock TO anu_gupta;
GRANT
```

```
postgres=> \c productsupply;
You are now connected to database "productsupply" as user "anu_gupta".
productsupply=> select * from product where pro_id<8;
 pro_id | pro_cus_id |   pro_desc    | pro_num | in_id
--------+------------+---------------+---------+-------
      1 |         20 | HP Desktop    |      11 |   200
      3 |         17 | Lenovo Laptop |       7 |   200
      5 |         13 | Harry Potter  |       8 |   202
(3 rows)
```

```
productsupply=# REVOKE select on product,stock from anu_gupta;
REVOKE
productsupply=#
```

```
productsupply=> select * from product;
ERROR:  permission denied for relation product
productsupply=>
```

2.Grant select on product and stock where id is greater than 200 from user ann.

```
productsupply=# create user ann with password 'anu';
CREATE ROLE
productsupply=# GRANT select on product,stock TO anu;
ERROR:  role "anu" does not exist
productsupply=# GRANT select on product,stock TO ann;
GRANT
productsupply=#
```

```
postgres-> \c productsupply
You are now connected to database "productsupply" as user "ann".
productsupply-> Select * from product where in_id in (select iid from stock where iid>200);
ERROR:  syntax error at or near "/"
LINE 1: /c productsupplt
        ^
productsupply=> Select * from product where in_id in (select iid from stock where iid>200);
 pro_id | pro_cus_id |        pro_desc        | pro_num | in_id
--------+------------+------------------------+---------+-------
      5 |         13 | Harry Potter           |       8 |   202
      9 |         19 | Wings of Fire          |      62 |   202
     10 |         16 | cotton raw material    |      23 |   204
     24 |         51 | Eureka Forbes  cleaner |      13 |   205
     29 |         18 | Aidea Microfibre cloth |      54 |   205
     27 |         99 | Versace Crystal        |      90 |   207
     88 |        101 | Mahavir Oil            |       2 |   210
     25 |         78 | Nykaa perfume          |     111 |   207
(8 rows)
```

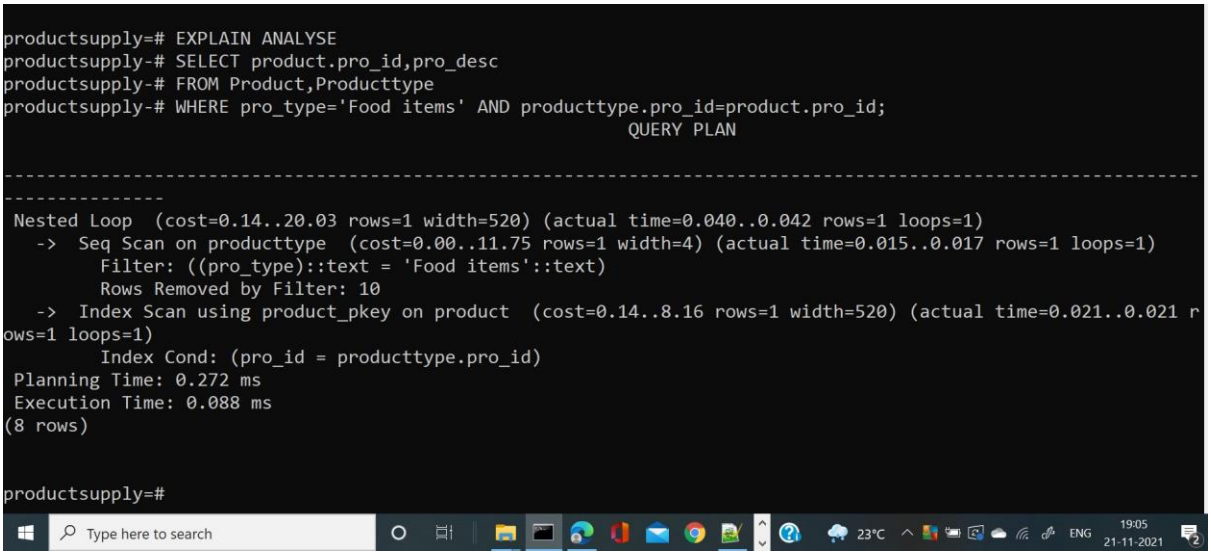
```
productsupply=# REVOKE select on product,stock from ann;
REVOKE
productsupply=#
```

Query Performance Analysis:

Simple Queries Performance Analysis:

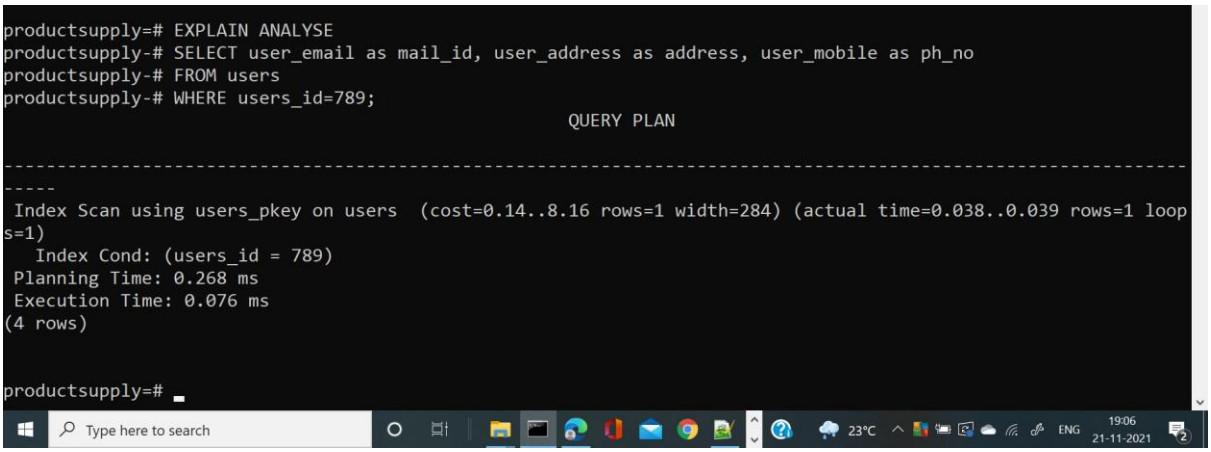
1. A query to retrieve product id and product description of type = ‘Food items’

SELECT product.pro_id,pro_desc
FROM Product,Producttype
WHERE pro_type='Food items' AND producttype.pro_id=product.pro_id;



2. Retrieve user email as mail_id, user’s address as address and mobilenumber as ph_no of users_id = 789

SELECT user_email as mail_id, user_address as address, user_mobile as ph_no FROM users
WHERE users_id=789;



3.Retrieve role name of an user named ‘Charlotte’

SELECT Role_name
FROM Roles,Users
WHERE User_fname='Charlotte' AND Usernum=Users_id;

```
postgres=# \c productsupply
You are now connected to database "productsupply" as user "postgres".
productsupply=# EXPLAIN ANALYSE
productsupply-# SELECT Role_name
productsupply-# FROM Roles,Users
productsupply-# WHERE User_fname='Charlotte' AND Usernum=Users_id;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=12.26..23.15 rows=1 width=516) (actual time=0.122..0.125 rows=1 loops=1)
   Hash Cond: (roles.usernum = users.users_id)
   ->  Seq Scan on roles  (cost=0.00..10.70 rows=70 width=520) (actual time=0.043..0.045 rows=5 loops=1)
   ->  Hash  (cost=12.25..12.25 rows=1 width=4) (actual time=0.052..0.053 rows=1 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Seq Scan on users  (cost=0.00..12.25 rows=1 width=4) (actual time=0.036..0.037 rows=1 loops=1)
               Filter: ((user_fname)::text = 'Charlotte'::text)
               Rows Removed by Filter: 4
 Planning Time: 1.893 ms
 Execution Time: 0.315 ms
(10 rows)
```

4.Display product description as Product and product number as availability from product table ordered by product number

SELECT Pro_desc as Product,Pro_num as Availability
FROM Product
ORDER BY Pro_num;

```
productsupply=# EXPLAIN ANALYSE
productsupply-# SELECT Pro_desc as Product,Pro_num as Availability
productsupply-# FROM Product
productsupply-# ORDER BY Pro_num;
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Sort  (cost=16.39..16.74 rows=140 width=520) (actual time=0.202..0.204 rows=11 loops=1)
   Sort Key: pro_num
   Sort Method: quicksort  Memory: 25kB
   ->  Seq Scan on product  (cost=0.00..11.40 rows=140 width=520) (actual time=0.070..0.073 rows=11 loops=1)
 Planning Time: 0.835 ms
 Execution Time: 0.229 ms
(6 rows)


productsupply=# _
```


5.Retrieve delivery description as product_to_deliver in between the delivery date 2-10-2021 and 12-10-2021

SELECT Del_desc as Product_to_deliver
FROM Delivery
WHERE Del_date BETWEEN '2021-10-2' AND '2021-10-12';

```
productsupply=# EXPLAIN ANALYSE
productsupply-# SELECT Del_desc as Product_to_deliver
productsupply-# FROM Delivery
productsupply-# WHERE Del_date BETWEEN '2021-10-2' AND '2021-10-12';
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Seq Scan on delivery  (cost=0.00..12.10 rows=1 width=516) (actual time=0.030..0.033 rows=2 loops=1)
   Filter: ((del_date >= '2021-10-02'::date) AND (del_date <= '2021-10-12'::date))
   Rows Removed by Filter: 3
 Planning Time: 0.528 ms
 Execution Time: 0.093 ms
(5 rows)


productsupply=#
```

6.Query to view products whose customer id is greater than 5.

select * from Product where pro_cus_id>5;

```
postgres=# EXPLAIN ANALYSE
postgres-# select * from Product where pro_cus_id>5;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Seq Scan on product  (cost=0.00..13.88 rows=103 width=234) (actual time=3.596..3.609 rows=11 loops=1)
   Filter: (pro_cus_id > 5)
 Planning Time: 23.665 ms
 Execution Time: 7.799 ms
(4 rows)


postgres=#
```

7.Name all users whose name starts with letter ‘A’. select

user_fname from users where user_fname liks 'A%';

```
productsupply=#
productsupply=# EXPLAIN ANALYSE
productsupply-# select user_fname from users where user_fname like 'A%';
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Seq Scan on users  (cost=0.00..12.25 rows=1 width=58) (actual time=0.621..0.624 rows=1 loops=1)
   Filter: ((user_fname)::text ~~ 'A%'::text)
   Rows Removed by Filter: 4
 Planning Time: 10.101 ms
 Execution Time: 0.651 ms
(5 rows)


productsupply=#
```

8.Query to view stock available which has stock id less than 1235 and inventory id less than 205.

select * from stock where stk_id<1235 and iid<205;

```
postgres=# EXPLAIN ANALYSE
postgres-# select * from stock where stk_id<1235 and iid<205;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Seq Scan on stock  (cost=0.00..17.95 rows=59 width=126) (actual time=0.020..0.020 rows=0 loops=1)
   Filter: ((stk_id < 1235) AND (iid < 205))
 Planning Time: 5.594 ms
 Execution Time: 0.045 ms
(4 rows)


postgres=#
```

9.Query to view number of products whose product number is greater than 10.

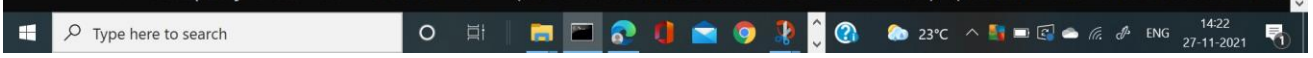
select COUNT(*) from product where pro_num>10;

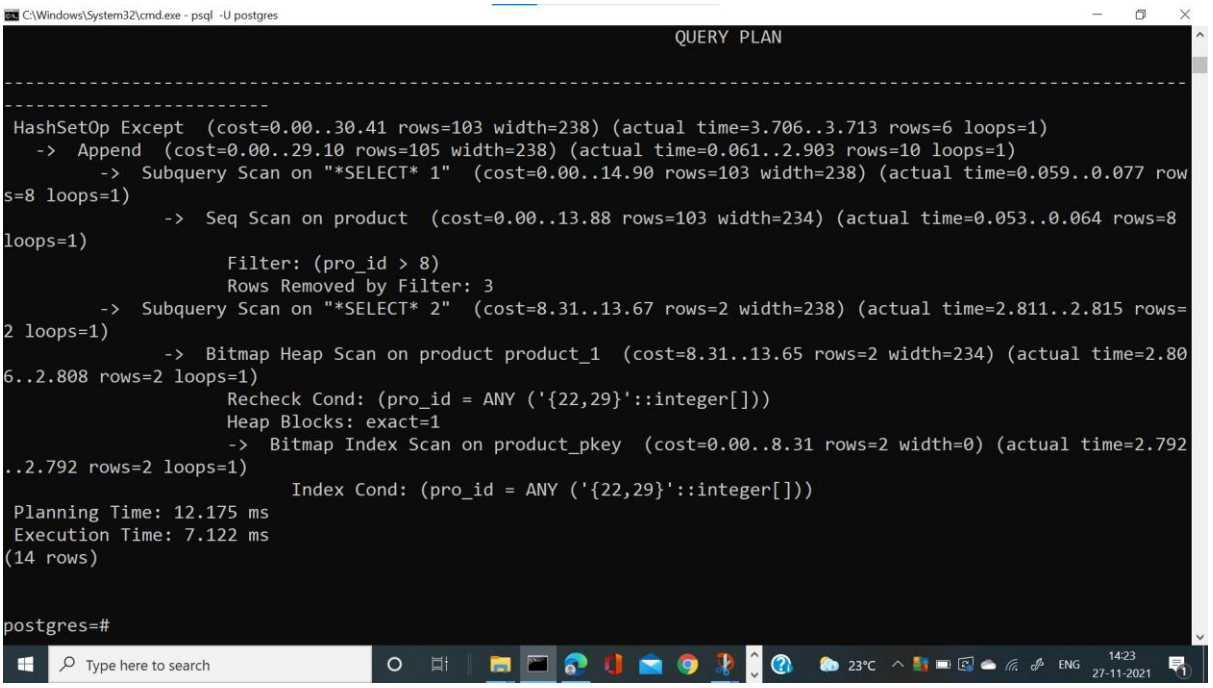
```
postgres=# EXPLAIN ANALYSE
postgres-# select COUNT(*) from product where pro_num>10;
                                              QUERY PLAN
------------------------------------------------------------------------------------------------------
 Aggregate  (cost=14.13..14.14 rows=1 width=8) (actual time=0.092..0.094 rows=1 loops=1)
   ->  Seq Scan on product  (cost=0.00..13.88 rows=103 width=0) (actual time=0.061..0.067 rows=8 loops=1)
         Filter: (pro_num > 10)
         Rows Removed by Filter: 3
 Planning Time: 3.816 ms
 Execution Time: 24.127 ms
(6 rows)


postgres=#
```

10.Query to view product id numbers above 8 but not in (22,29).

select * from product where pro_id>8 except
select * from product where pro_id in (22,29)

```
postgres=# EXPLAIN ANALYSE
postgres-# select * from product where pro_id>8 except select * from product where pro_id in (22,29)
postgres-# ;
                                                    QUERY PLAN


-------------------------------------------------------------------------------------------------------------------
-------------------------
 HashSetOp Except  (cost=0.00..30.41 rows=103 width=238) (actual time=3.706..3.713 rows=6 loops=1)
   ->  Append  (cost=0.00..29.10 rows=105 width=238) (actual time=0.061..2.903 rows=10 loops=1)
         ->  Subquery Scan on "*SELECT* 1"  (cost=0.00..14.90 rows=103 width=238) (actual time=0.059..0.077 row
```

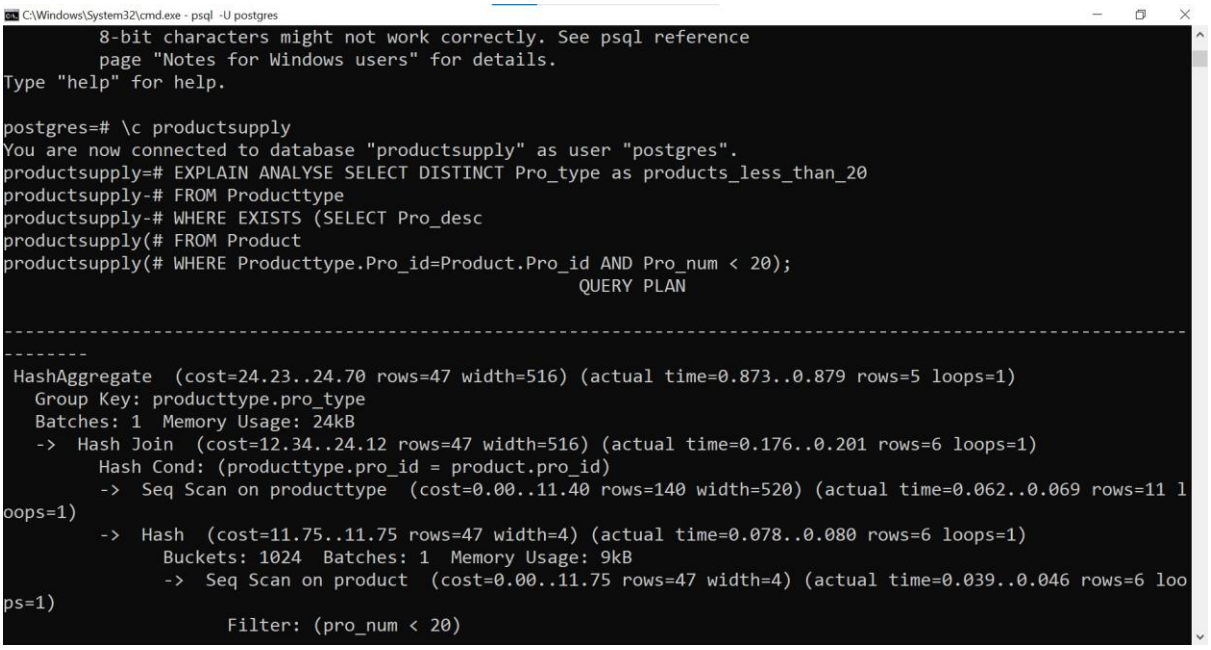


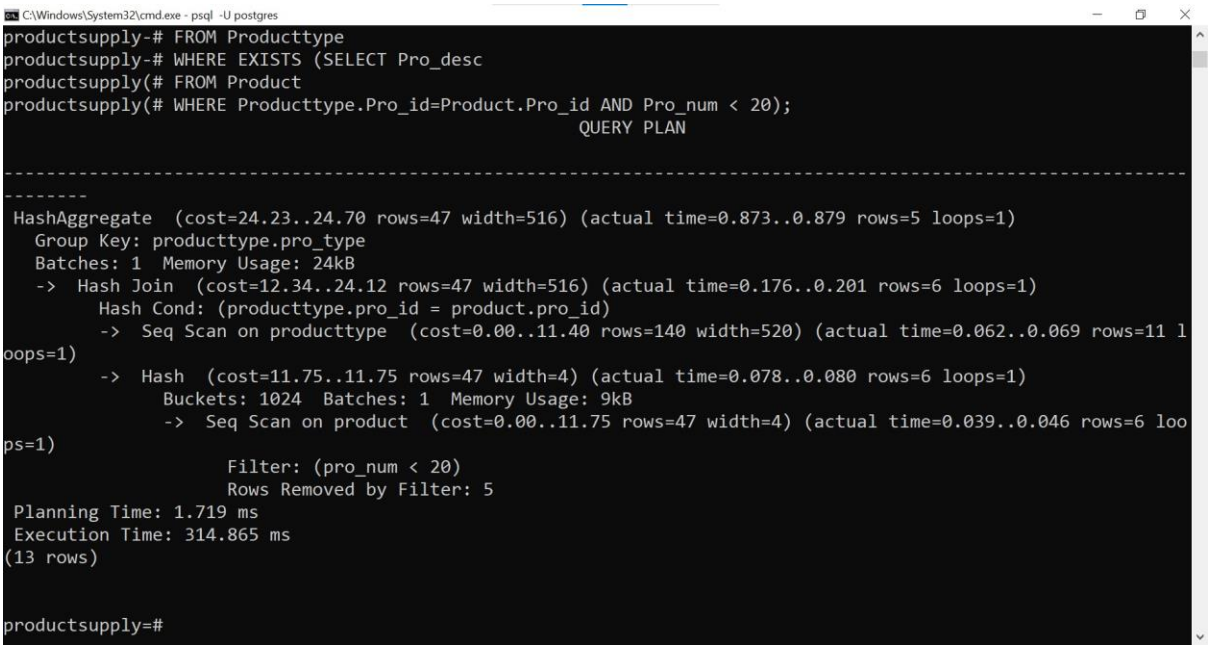


Complex Queries Analysis:

1. Display distinct pro_type of those products that are less than 20

```
SELECT DISTINCT Pro_type as products_less_than_20
FROM Producttype
WHERE EXISTS (SELECT Pro_desc
FROM Product
WHERE Producttype.Pro_id=Product.Pro_id AND Pro_num < 20);
```





2.2. Display the permission names given to the user with per_id 10,14, and 15

```
SELECT Per_name
FROM Permission
WHERE Per_id IN (10,14,15);
```

```
productsupply=# EXPLAIN ANALYSE
productsupply-# SELECT Per_name
productsupply-# FROM Permission
productsupply-# WHERE Per_id IN (10,14,15);
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on permission  (cost=0.00..10.69 rows=3 width=516) (actual time=0.018..0.020 rows=3 loops=1)
   Filter: (per_id = ANY ('{10,14,15}'::integer[]))
   Rows Removed by Filter: 2
 Planning Time: 0.122 ms
 Execution Time: 0.037 ms
(5 rows)


productsupply=#
```

3.Display user_id, user_fname, user_lname and role_name from table users joined with roles

```
SELECT usr.users_id,usr.user_fname,usr.user_lname,role.role_name
FROM Users as usr
NATURAL JOIN Roles as role
WHERE Usernum=Users_id;
```

```
C:\Windows\System32\cmd.exe - psql -U postgres
 Planning Time: 0.122 ms
 Execution Time: 0.037 ms
(5 rows)


productsupply=# EXPLAIN ANALYSE
productsupply-# SELECT usr.users_id,usr.user_fname,usr.user_lname,role.role_name
productsupply-# FROM Users as usr
productsupply-# NATURAL JOIN Roles as role
productsupply-# WHERE Usernum=Users_id;
                                                  QUERY PLAN

-------------------------------------------------------------------------------------------------------------
-----
 Hash Join  (cost=11.57..24.75 rows=70 width=636) (actual time=0.076..0.079 rows=5 loops=1)
   Hash Cond: (usr.users_id = role.usernum)
   ->  Seq Scan on users usr  (cost=0.00..11.80 rows=180 width=120) (actual time=0.023..0.024 rows=5 loops=1)
   ->  Hash  (cost=10.70..10.70 rows=70 width=520) (actual time=0.045..0.046 rows=5 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Seq Scan on roles role  (cost=0.00..10.70 rows=70 width=520) (actual time=0.023..0.023 rows=5 loop
s=1)
 Planning Time: 0.775 ms
 Execution Time: 0.098 ms
(8 rows)


productsupply=#
```

4.Retrieve the count of stk_type grouped by stk_type

```
SELECT COUNT(Stk_type), Stk_type
FROM Stocktype
GROUP BY Stk_type;
```

```
productsupply=# EXPLAIN ANALYSE
productsupply-# SELECT COUNT(Stk_type), Stk_type
productsupply-# FROM Stocktype
productsupply-# GROUP BY Stk_type;
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=12.10..13.50 rows=140 width=524) (actual time=0.020..0.022 rows=6 loops=1)
   Group Key: stk_type
   Batches: 1  Memory Usage: 40kB
   ->  Seq Scan on stocktype  (cost=0.00..11.40 rows=140 width=516) (actual time=0.010..0.010 rows=9 loops=1)
 Planning Time: 0.228 ms
 Execution Time: 0.048 ms
(6 rows)


productsupply=#
productsupply=#
```

5. Create a column named quantity conclusion which displays ‘The quantity is less than 50’ if pro_num less than 50 and ‘The quantity is more than 50’ if pro_num is greater than and ‘The quantity is 50’ otherwise

```
SELECT Pro_id,Pro_num,
CASE
WHEN Pro_num > 50 THEN 'The quantity is greater than 50'
WHEN Pro_num < 50 THEN 'The quantity is less than 50'
ELSE 'The quantity is 50'
END AS QuantityConclusion
FROM Product;
```

```
productsupply=#
productsupply=# EXPLAIN ANALYSE
productsupply-# SELECT Pro_id,Pro_num,
productsupply-# CASE
productsupply-# WHEN Pro_num > 50 THEN 'The quantity is greater than 50'
productsupply-# WHEN Pro_num < 50 THEN 'The quantity is less than 50'
productsupply-# ELSE 'The quantity is 50'
productsupply-# END AS QuantityConclusion
productsupply-# FROM Product;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------
 Seq Scan on product  (cost=0.00..12.10 rows=140 width=40) (actual time=0.056..0.064 rows=11 loops=1)
 Planning Time: 0.176 ms
 Execution Time: 0.112 ms
(3 rows)


productsupply=#
```

6.Retrieving the details of all products which are present in stock.

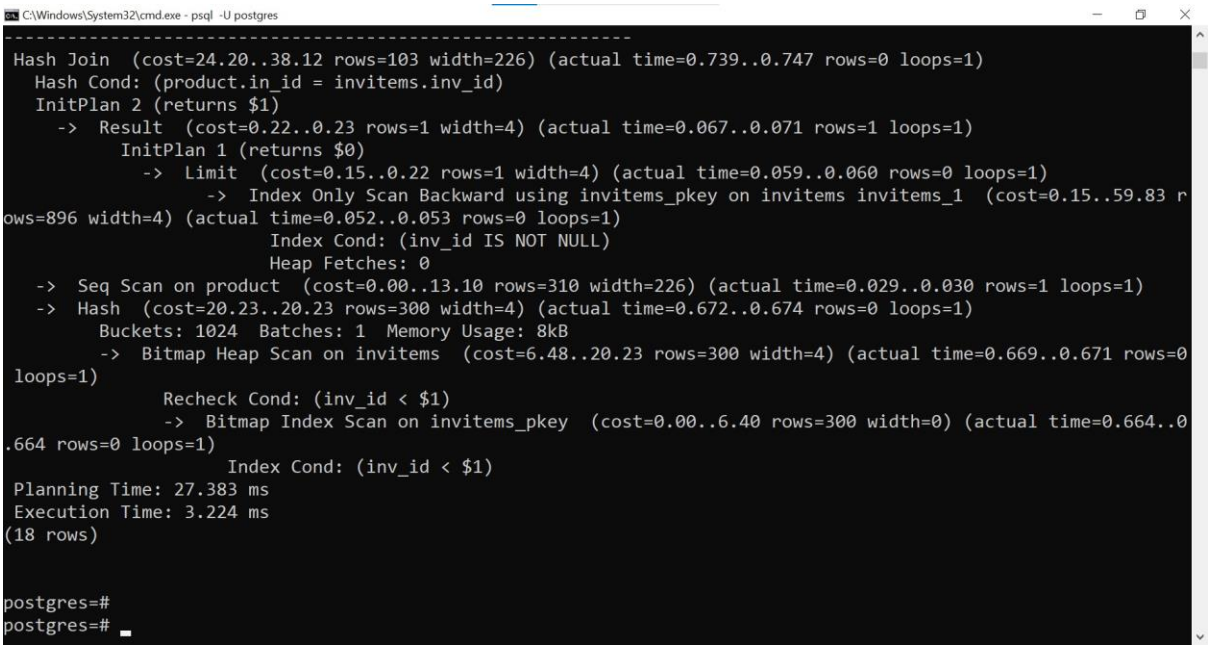
```
select * from product p where EXISTS (select * from stock s where s.iid=p.in_id);
```

```
postgres=# EXPLAIN ANALYSE
postgres-# select * from product p where EXISTS (select * from stock s where s.iid=p.in_id);
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
-------
 Hash Join  (cost=21.13..36.76 rows=155 width=234) (actual time=0.253..0.255 rows=0 loops=1)
   Hash Cond: (p.in_id = s.iid)
   ->  Seq Scan on product p  (cost=0.00..13.10 rows=310 width=234) (actual time=0.200..0.200 rows=1 loops=1)
   ->  Hash  (cost=18.63..18.63 rows=200 width=4) (actual time=0.024..0.025 rows=0 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 8kB
         ->  HashAggregate  (cost=16.63..18.63 rows=200 width=4) (actual time=0.023..0.023 rows=0 loops=1)
               Group Key: s.iid
               Batches: 1  Memory Usage: 40kB
               ->  Seq Scan on stock s  (cost=0.00..15.30 rows=530 width=4) (actual time=0.017..0.018 rows=0 l
ops=1)
 Planning Time: 215.073 ms
 Execution Time: 3.651 ms
(11 rows)
```

7.Retrieve product description ,id and inventory id whose inventory id is less than maximum.

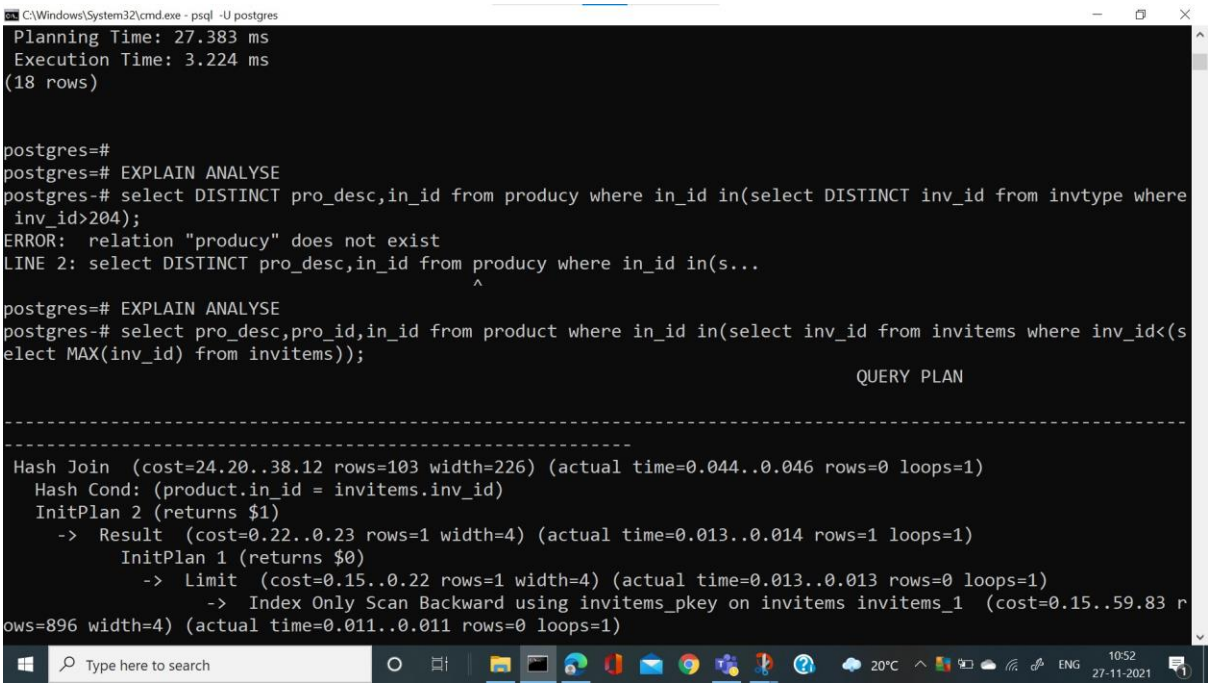
select pro_desc,pro_id,in_id from product where in_id in(select inv_id from invitems where inv_id<(select MAX(inv_id) from invitems));

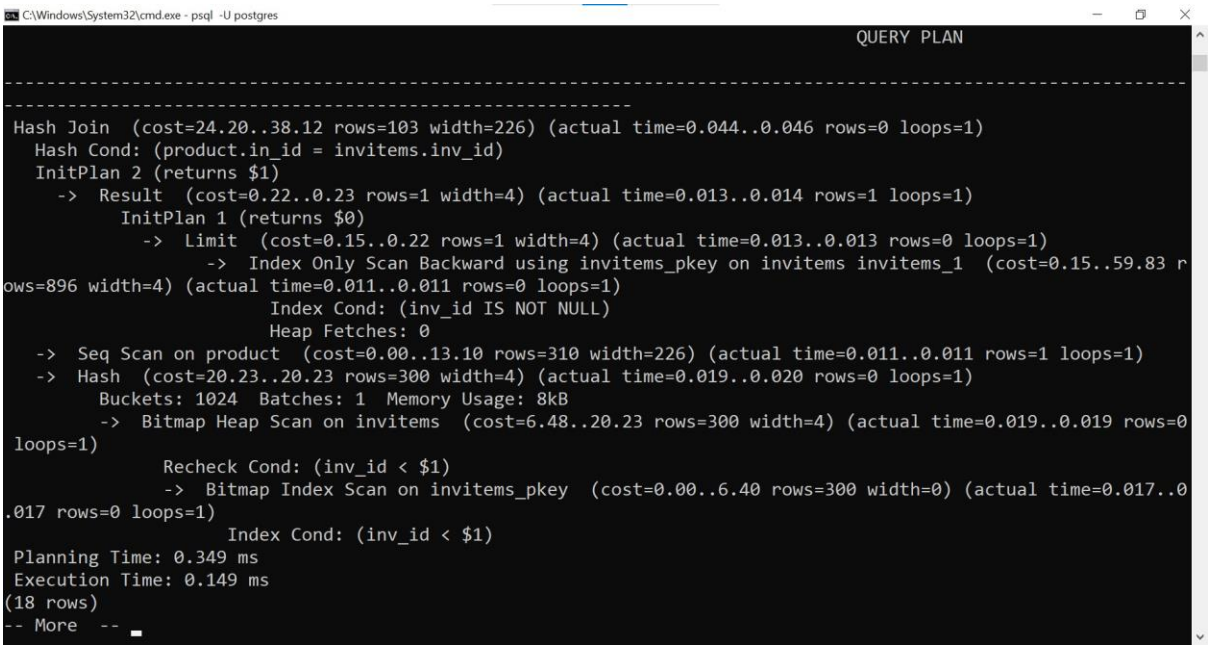
```
postgres=# EXPLAIN ANALYSE
postgres-# select pro_desc,pro_id,in_id from product where in_id in(select inv_id from invitems where inv_id<(s
elect MAX(inv_id) from invitems));
                                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------
 Hash Join  (cost=24.20..38.12 rows=103 width=226) (actual time=0.739..0.747 rows=0 loops=1)
   Hash Cond: (product.in_id = invitems.inv_id)
   InitPlan 2 (returns $1)
     ->  Result  (cost=0.22..0.23 rows=1 width=4) (actual time=0.067..0.071 rows=1 loops=1)
           InitPlan 1 (returns $0)
             ->  Limit  (cost=0.15..0.22 rows=1 width=4) (actual time=0.059..0.060 rows=0 loops=1)
                   ->  Index Only Scan Backward using invitems_pkey on invitems invitems_1  (cost=0.15..59.83 r
ows=896 width=4) (actual time=0.052..0.053 rows=0 loops=1)
                         Index Cond: (inv_id IS NOT NULL)
                         Heap Fetches: 0
   ->  Seq Scan on product  (cost=0.00..13.10 rows=310 width=226) (actual time=0.029..0.030 rows=1 loops=1)
   ->  Hash  (cost=20.23..20.23 rows=300 width=4) (actual time=0.672..0.674 rows=0 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 8kB
         ->  Bitmap Heap Scan on invitems  (cost=6.48..20.23 rows=300 width=4) (actual time=0.669..0.671 rows=0
 loops=1)
               Recheck Cond: (inv_id < $1)
```



8.Query to retrieve distinct product description,inventory id from product which has id greater than 204.

select DISTINCT pro_desc,in_id from product where in_id in(select DISTINCT inv_id from invtype where inv_id>204);





9.Query to retrieve express delivery date and description of delivery from express delivery whose description is in delivery table whose id is greater than 13.

select del_desc,express_del_date from expressdelivery where del_desc in(select del_desc from delivery where del_id>13);

```
postgres=# \c productsupply
You are now connected to database "productsupply" as user "postgres".
productsupply=# EXPLAIN ANALYSE
productsupply-# select del_desc,express_del_date from expressdelivery where del_desc in(select del_desc from delivery where del_id>13);
                                                    QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------------
 Hash Semi Join  (cost=12.34..24.63 rows=47 width=520) (actual time=0.237..0.248 rows=4 loops=1)
   Hash Cond: ((expressdelivery.del_desc)::text = (delivery.del_desc)::text)
   ->  Seq Scan on expressdelivery  (cost=0.00..11.40 rows=140 width=520) (actual time=0.067..0.070 rows=5 loops=1)
   ->  Hash  (cost=11.75..11.75 rows=47 width=516) (actual time=0.116..0.117 rows=3 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Seq Scan on delivery  (cost=0.00..11.75 rows=47 width=516) (actual time=0.043..0.047 rows=3 loops=1)
               Filter: (del_id > 13)
               Rows Removed by Filter: 2
 Planning Time: 39.244 ms
 Execution Time: 0.596 ms
(10 rows)


productsupply=#
```

10.Query to retrieve permission names,login number which are present in both permission and roles.

select per_name,login_num from permission p where EXISTS(select * from roles r where r.role_name=p.per_name);

```
productsupply=# EXPLAIN ANALYSE
productsupply-# select per_name,login_num from permission p where EXISTS(select * from roles r where r.role_nam
e=p.per_name);
                                                QUERY PLAN

-----------------------------------------------------------------------------------------------------------------
--
 Hash Semi Join  (cost=11.57..22.76 rows=50 width=520) (actual time=0.029..0.032 rows=5 loops=1)
   Hash Cond: ((p.per_name)::text = (r.role_name)::text)
   ->  Seq Scan on permission p  (cost=0.00..10.50 rows=50 width=520) (actual time=0.010..0.011 rows=5 loops=1)
   ->  Hash  (cost=10.70..10.70 rows=70 width=516) (actual time=0.012..0.012 rows=5 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Seq Scan on roles r  (cost=0.00..10.70 rows=70 width=516) (actual time=0.007..0.008 rows=5 loops=1
)
 Planning Time: 17.512 ms
 Execution Time: 0.054 ms
(8 rows)


productsupply=#
productsupply=#
```