DBMS MINI PROJECT ASSIGNMENT 3 REPORT
STORE MANAGEMENT SYSTEM

TEAM NUMBER : 10

PRAMATHA GAJANAN BHAT	**PES1UG19CS339**	F
PRATIKSHA D NAYAK	**PES1UG19CS349**	
PRERANA HADADI	**PES1UG19CS352**	

In this assignment, the actual working of the database is tested using queries. We have appropriately designed queries, which are segregated into:

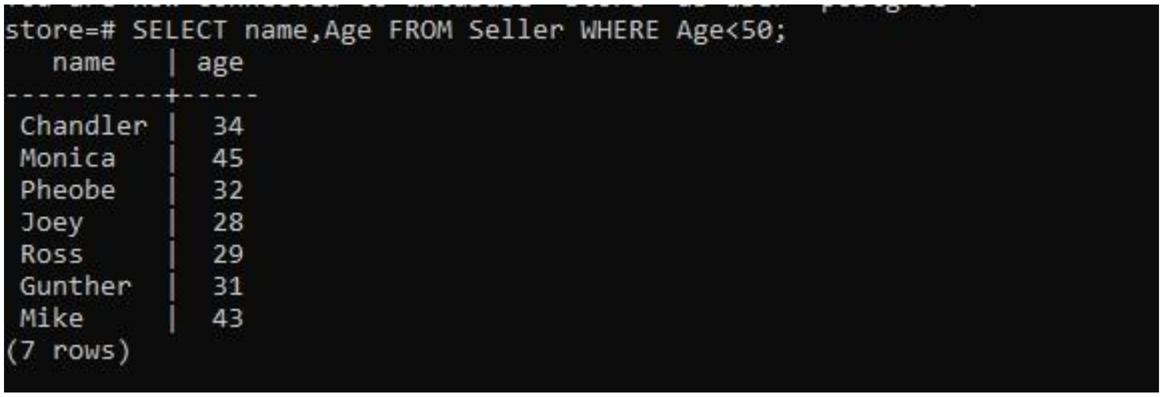
1) Simple Queries
2) Complex Queries (includes nested queries)
3) Roles and Access Privileges
4) Transactions
5) Concurrency Control

In addition to this, query performance is analyzed and the execution plan for queries are generated. In PostgreSQL, this is executed using the 'explain' keyword.

Simple Queries:

1) Retrieve all the sellers whose age is below 50.

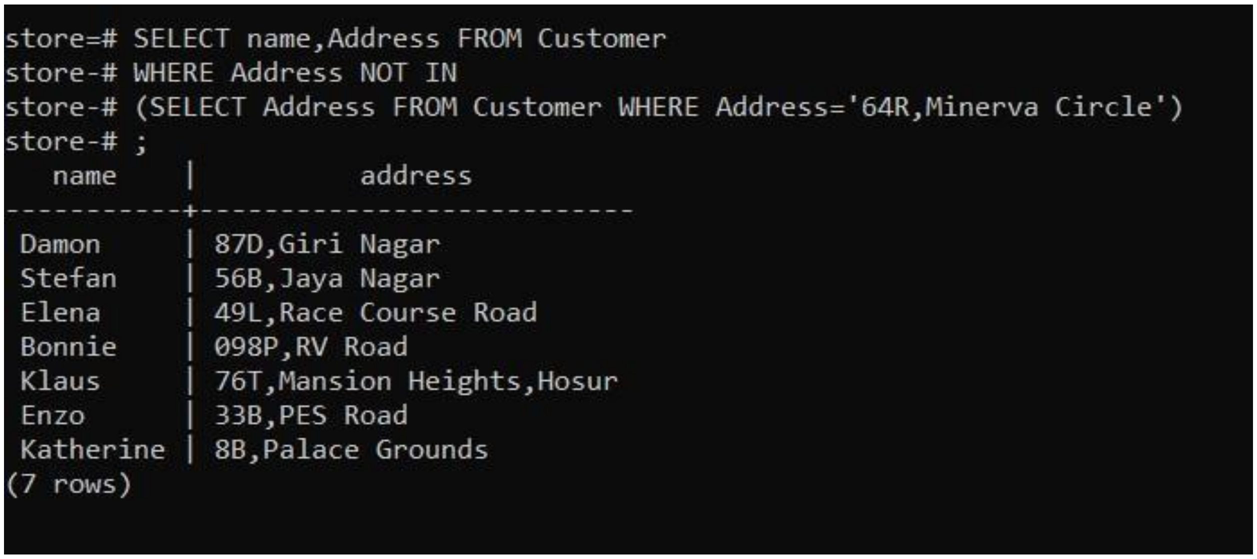
```
SELECT name,Age FROM Seller WHERE Age<50;
```



```
store=# SELECT name,Age FROM Seller WHERE Age<50;
   name   | age
----------+-----
 Chandler |  34
 Monica   |  45
 Pheobe   |  32
 Joey     |  28
 Ross     |  29
 Gunther  |  31
 Mike     |  43
(7 rows)
```

2) Retrieve all the customers who do not live in '64R,Minerva Circle'.

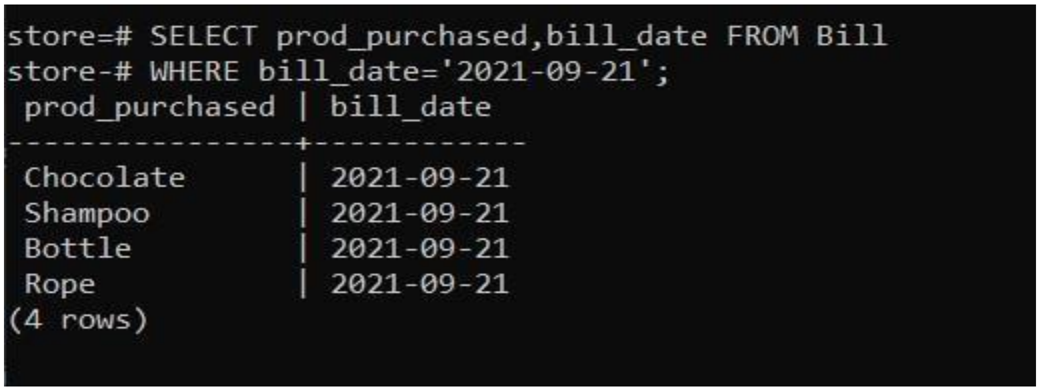
SELECT name,Address   FROM Customer
 WHERE Address NOT IN (SELECT Address FROM Customer
WHERE Address='64R,Minerva Circle')



```
store=# SELECT name,Address FROM Customer
store-# WHERE Address NOT IN
store-# (SELECT Address FROM Customer WHERE Address='64R,Minerva Circle')
store-# ;
   name    |          address
-----------+---------------------------
 Damon     | 87D,Giri Nagar
 Stefan    | 56B,Jaya Nagar
 Elena     | 49L,Race Course Road
 Bonnie    | 098P,RV Road
 Klaus     | 76T,Mansion Heights,Hosur
 Enzo      | 33B,PES Road
 Katherine | 8B,Palace Grounds
(7 rows)
```

3) Retrive the products purchased on 21-09-2021

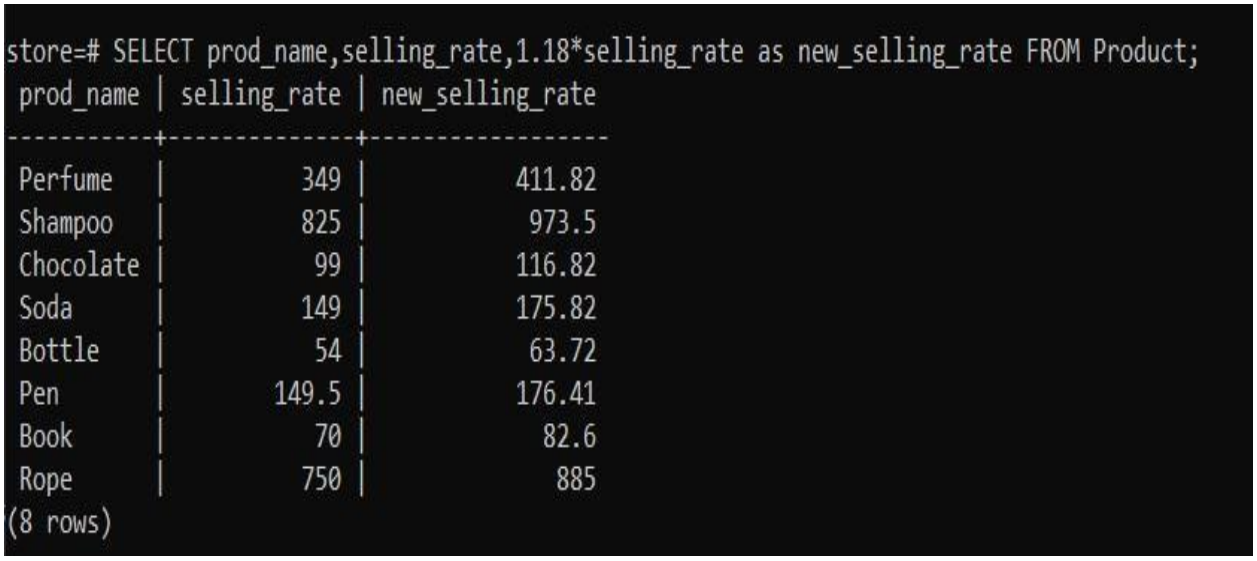
SELECT prod_purchased,bill_date   FROM Bill WHERE bill_date='2021-09-21' ;



```
store=# SELECT prod_purchased,bill_date FROM Bill
store-# WHERE bill_date='2021-09-21';
 prod_purchased | bill_date
----------------+------------
 Chocolate      | 2021-09-21
 Shampoo        | 2021-09-21
 Bottle         | 2021-09-21
 Rope           | 2021-09-21
(4 rows)
```

4)Find the new rate of the products if GST of 18% is added.

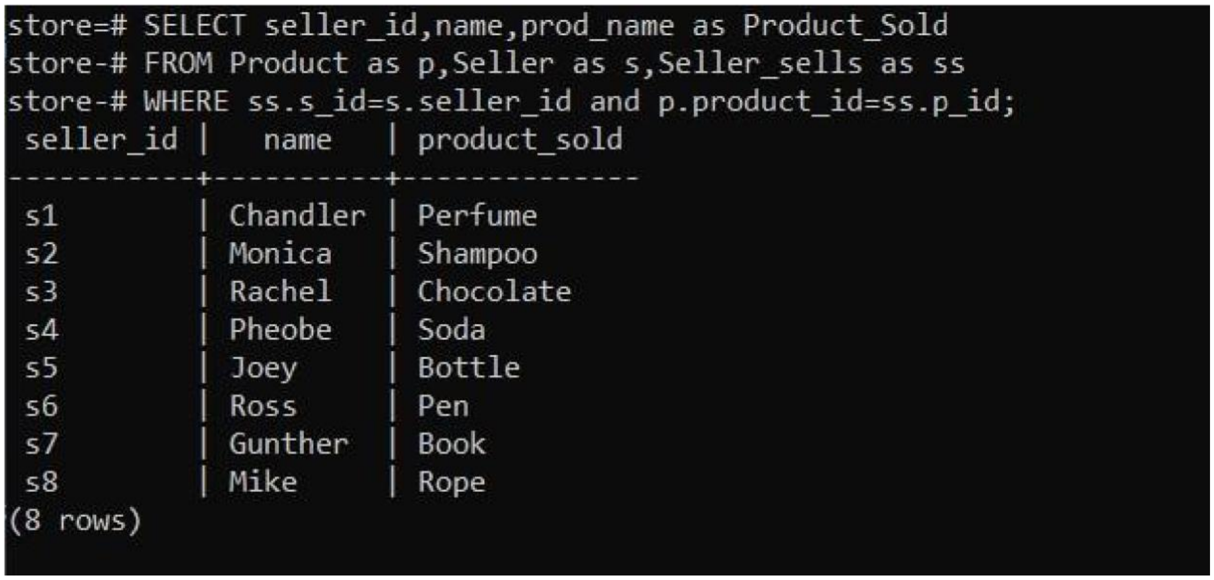
SELECT prod_name,selling_rate,1.18*selling_rate as new_selling_rate FROM Product;



```
store=# SELECT prod_name,selling_rate,1.18*selling_rate as new_selling_rate FROM Product;
 prod_name | selling_rate | new_selling_rate
-----------+--------------+------------------
 Perfume   |          349 |           411.82
 Shampoo   |          825 |            973.5
 Chocolate |           99 |           116.82
 Soda      |          149 |           175.82
 Bottle    |           54 |            63.72
 Pen       |        149.5 |           176.41
 Book      |           70 |             82.6
 Rope      |          750 |              885
(8 rows)
```

5) Retrieve all the products sold by each seller.

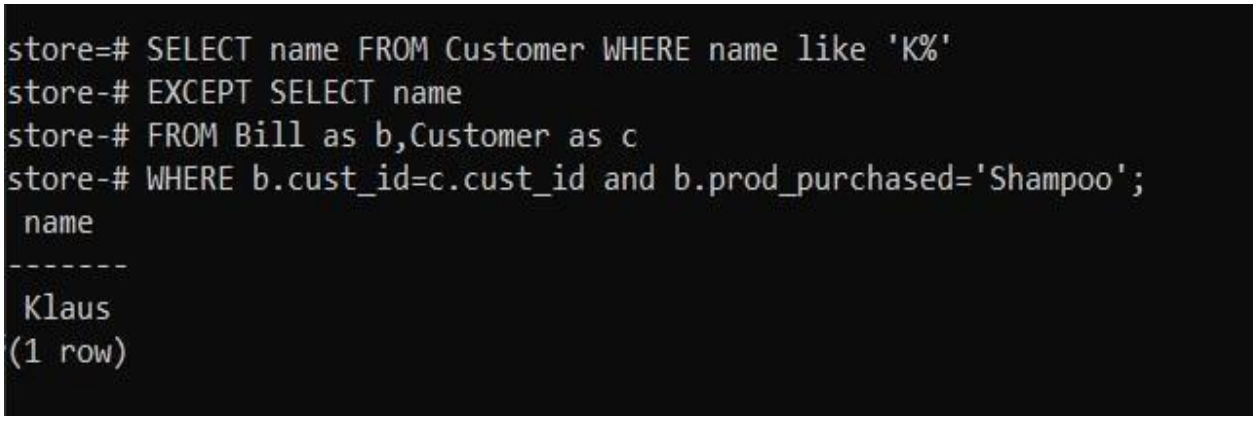
```
SELECT seller_id,name,prod_name as Product_Sold
FROM Product as p,Seller as s,Seller_sells as ss
WHERE ss.s_id=s.seller_id and p.product_id=ss.p_id;
```



```
store=# SELECT seller_id,name,prod_name as Product_Sold
store-# FROM Product as p,Seller as s,Seller_sells as ss
store-# WHERE ss.s_id=s.seller_id and p.product_id=ss.p_id;
 seller_id |   name   | product_sold
-----------+----------+--------------
 s1        | Chandler | Perfume
 s2        | Monica   | Shampoo
 s3        | Rachel   | Chocolate
 s4        | Pheobe   | Soda
 s5        | Joey     | Bottle
 s6        | Ross     | Pen
 s7        | Gunther  | Book
 s8        | Mike     | Rope
(8 rows)
```

6) Find the customers whose name starts with 'K' and purchased 'shampoo'.

```
SELECT name FROM Customer WHERE name like 'K%'
EXCEPT
SELECT name
FROM Bill as b,Customer as c
WHERE b.cust_id=c.cust_id and b.prod_purchased='Shampoo';
```

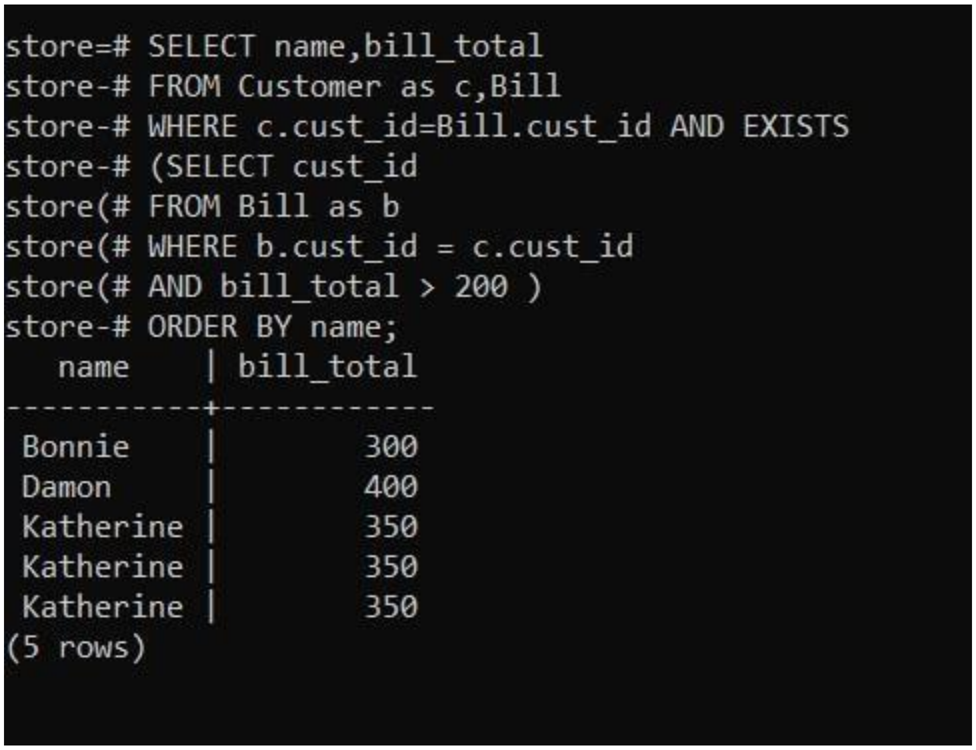


```
store=# SELECT name FROM Customer WHERE name like 'K%'
store-# EXCEPT SELECT name
store-# FROM Bill as b,Customer as c
store-# WHERE b.cust_id=c.cust_id and b.prod_purchased='Shampoo';
 name
-------
 Klaus
(1 row)
```

Complex Queries (includes nested query)

7) Find customers who have at least one payment whose amount is greater than 200.

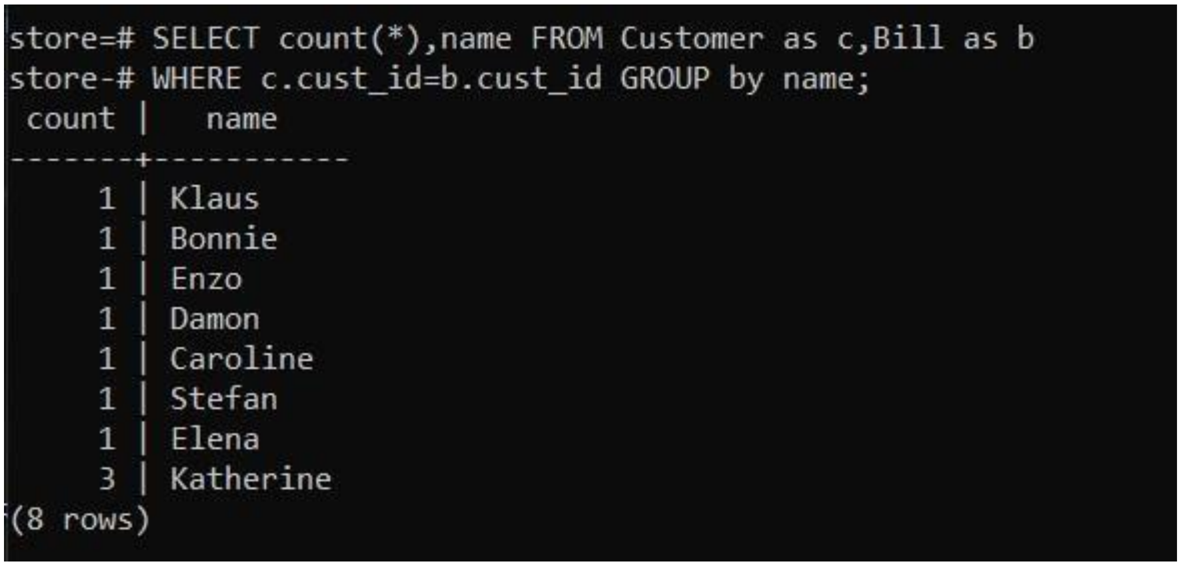
```
SELECT name,bill_total
FROM Customer as c,Bill
WHERE c.cust_id=Bill.cust_id AND EXISTS
(SELECT cust_id
FROM Bill as b
WHERE b.cust_id = c.cust_id AND bill_total > 200 )
ORDER BY name;
```



```
store=# SELECT name,bill_total
store-# FROM Customer as c,Bill
store-# WHERE c.cust_id=Bill.cust_id AND EXISTS
store-# (SELECT cust_id
store(# FROM Bill as b
store(# WHERE b.cust_id = c.cust_id
store(# AND bill_total > 200 )
store-# ORDER BY name;
   name    | bill_total
-----------+------------
 Bonnie    |        300
 Damon     |        400
 Katherine |        350
 Katherine |        350
 Katherine |        350
(5 rows)
```

8) Find all customers who purchased more than one product.

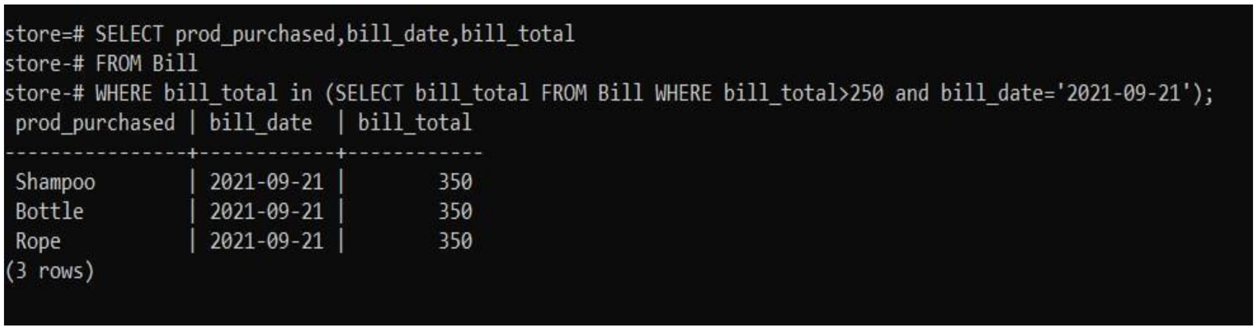
```
SELECT count(*),name
FROM Customer as c,Bill as b
WHERE c.cust_id=b.cust_id GROUP by name;
```



```
store=# SELECT count(*),name FROM Customer as c,Bill as b
store-# WHERE c.cust_id=b.cust_id GROUP by name;
 count |   name
-------+-----------
     1 | Klaus
     1 | Bonnie
     1 | Enzo
     1 | Damon
     1 | Caroline
     1 | Stefan
     1 | Elena
     3 | Katherine
(8 rows)
```

9) Retrive the products purchased on 21-09-2021 SELECT prod_purchased,bill_date,bill_total

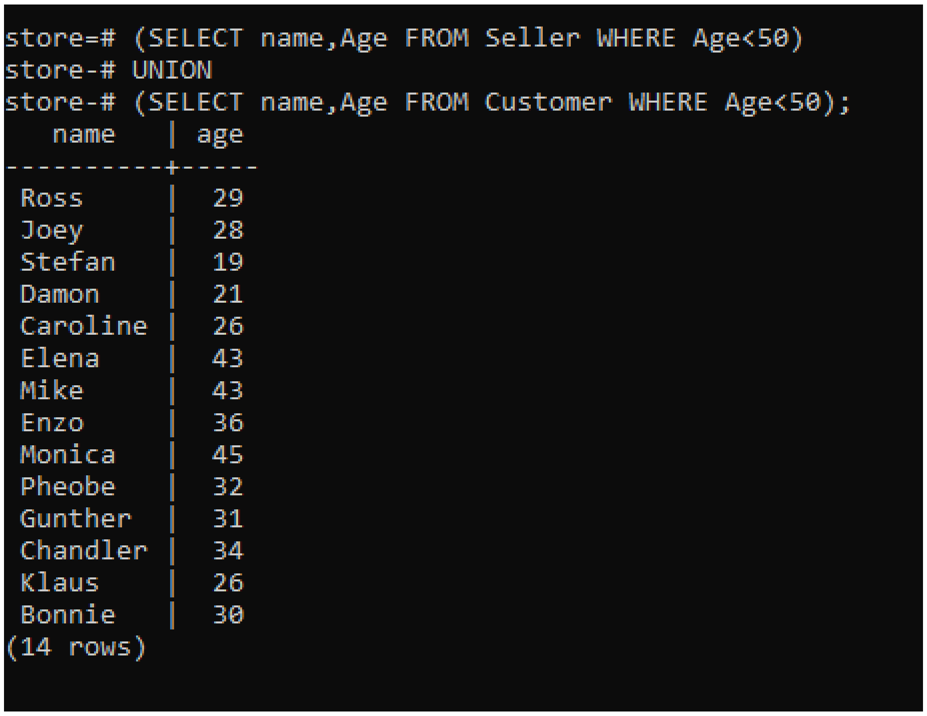
```
FROM Bill
WHERE bill_total in (SELECT bill_total FROM Bill WHERE bill_total>250 and
bill_date='2021-09-21');
```



```
store=# SELECT prod_purchased,bill_date,bill_total
store-# FROM Bill
store-# WHERE bill_total in (SELECT bill_total FROM Bill WHERE bill_total>250 and bill_date='2021-09-21');
 prod_purchased | bill_date  | bill_total
----------------+------------+-----------
 Shampoo        | 2021-09-21 |       350
 Bottle         | 2021-09-21 |       350
 Rope           | 2021-09-21 |       350
(3 rows)
```

10) Retrieve all the sellers and customers whose age is below 50.

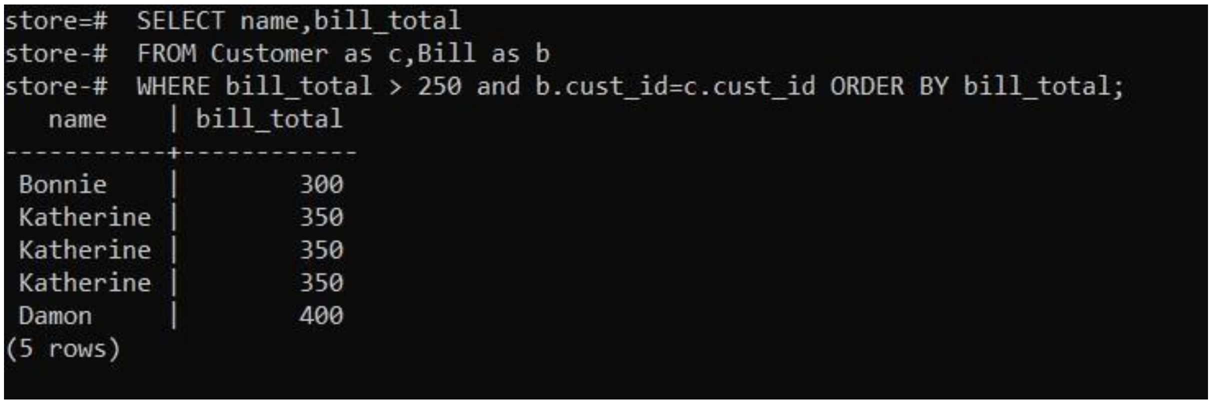
```
    (SELECT name,Age FROM Seller WHERE Age<50)
  UNION
    (SELECT name,Age FROM Customer WHERE Age<50);
```



```
store=# (SELECT name,Age FROM Seller WHERE Age<50)
store-# UNION
store-# (SELECT name,Age FROM Customer WHERE Age<50);
   name    | age
-----------+-----
 Ross      |  29
 Joey      |  28
 Stefan    |  19
 Damon     |  21
 Caroline  |  26
 Elena     |  43
 Mike      |  43
 Enzo      |  36
 Monica    |  45
 Pheobe    |  32
 Gunther   |  31
 Chandler  |  34
 Klaus     |  26
 Bonnie    |  30
(14 rows)
```

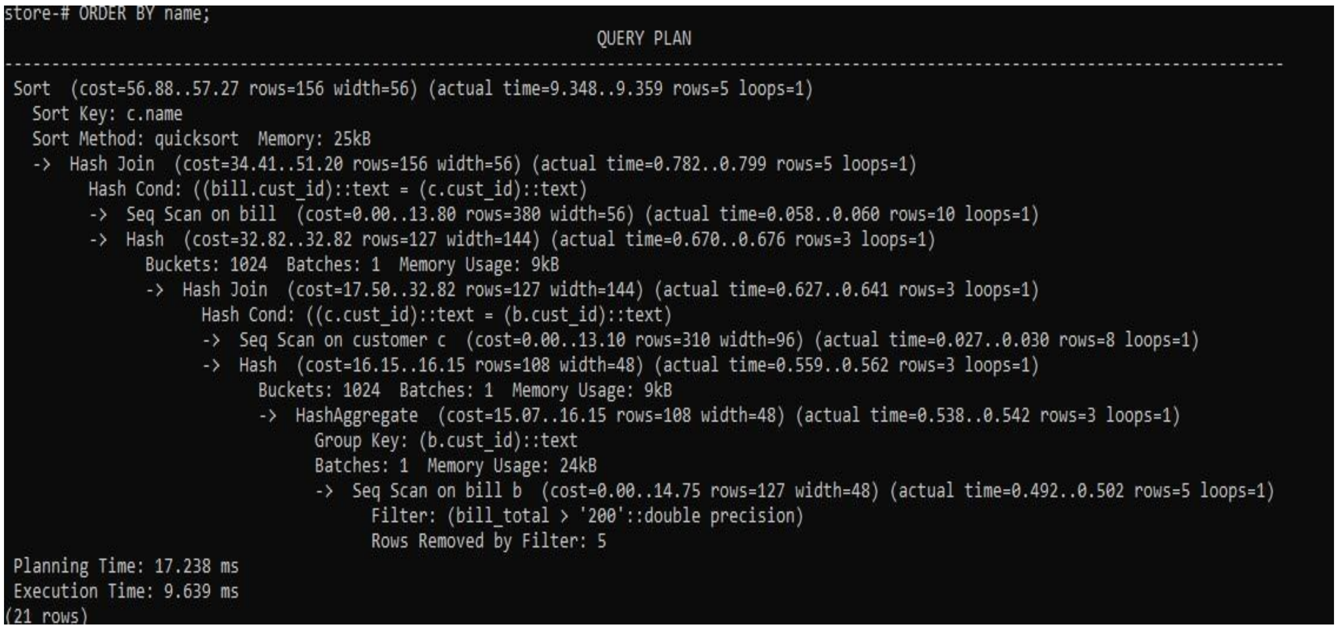
11)Retrieve all the customers who's bill total is greater than 250 rupees.

```
SELECT name,bill_total
FROM Customer as c,Bill as b
WHERE bill_total > 250 and b.cust_id=c.cust_id ORDER BY bill_total;
```

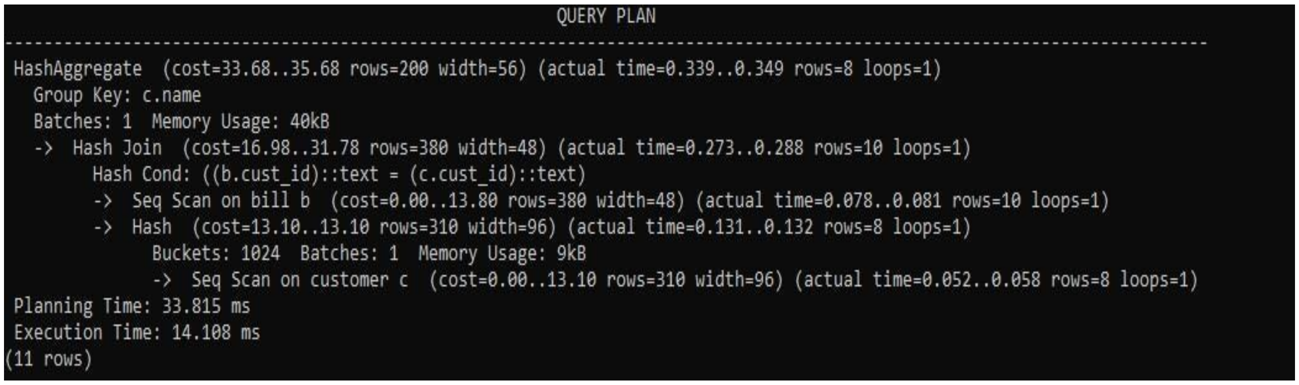


```
store=#  SELECT name,bill_total
store-#  FROM Customer as c,Bill as b
store-#  WHERE bill_total > 250 and b.cust_id=c.cust_id ORDER BY bill_total;
   name    | bill_total
-----------+------------
 Bonnie    |        300
 Katherine |        350
 Katherine |        350
 Katherine |        350
 Damon     |        400
(5 rows)
```

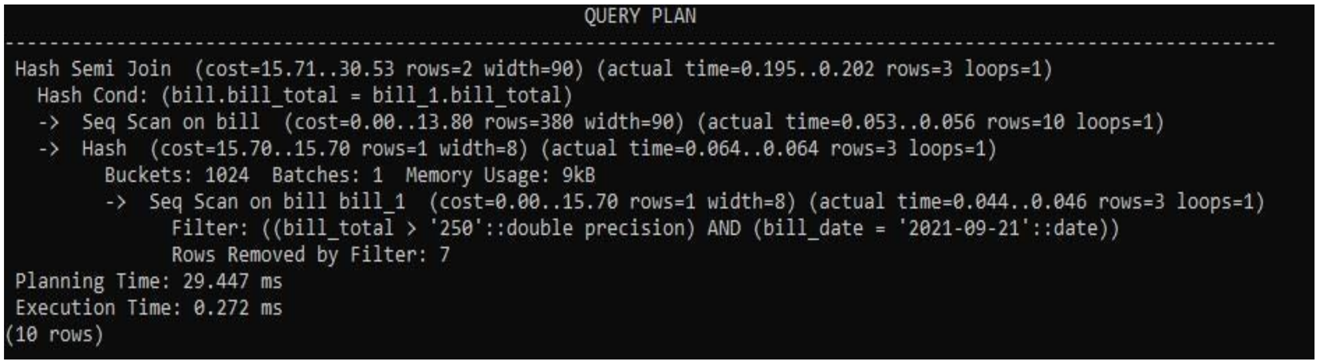
For query performance, the explain analyze command is executed for each of the complex queries, the results are as shown below for the 5 complex queries. Note that the order of the below screenshots is the same as that of the complex queries listed above



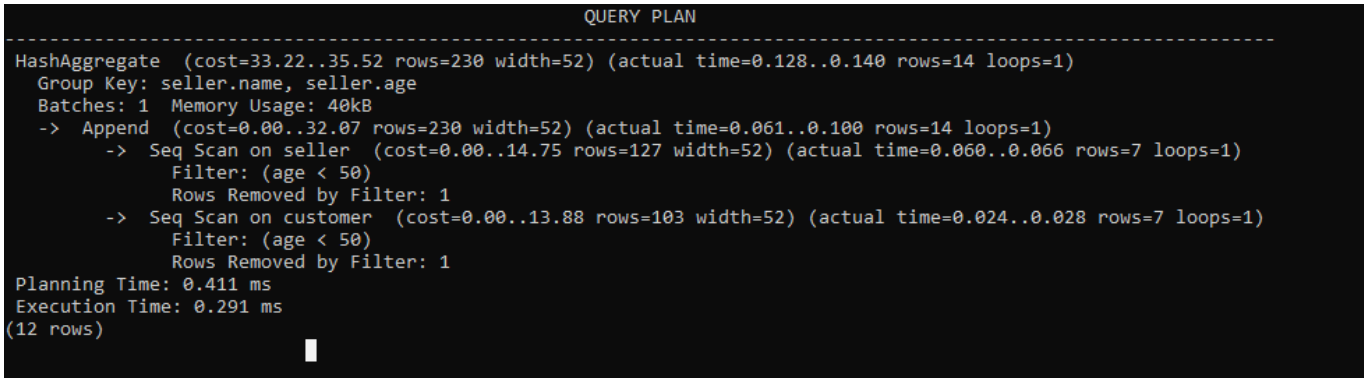
```
store-# ORDER BY name;
                                                         QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=56.88..57.27 rows=156 width=56) (actual time=9.348..9.359 rows=5 loops=1)
   Sort Key: c.name
   Sort Method: quicksort  Memory: 25kB
   ->  Hash Join  (cost=34.41..51.20 rows=156 width=56) (actual time=0.782..0.799 rows=5 loops=1)
         Hash Cond: ((bill.cust_id)::text = (c.cust_id)::text)
         ->  Seq Scan on bill  (cost=0.00..13.80 rows=380 width=56) (actual time=0.058..0.060 rows=10 loops=1)
         ->  Hash  (cost=32.82..32.82 rows=127 width=144) (actual time=0.670..0.676 rows=3 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Hash Join  (cost=17.50..32.82 rows=127 width=144) (actual time=0.627..0.641 rows=3 loops=1)
                     Hash Cond: ((c.cust_id)::text = (b.cust_id)::text)
                     ->  Seq Scan on customer c  (cost=0.00..13.10 rows=310 width=96) (actual time=0.027..0.030 rows=8 loops=1)
                     ->  Hash  (cost=16.15..16.15 rows=108 width=48) (actual time=0.559..0.562 rows=3 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 9kB
                           ->  HashAggregate  (cost=15.07..16.15 rows=108 width=48) (actual time=0.538..0.542 rows=3 loops=1)
                                 Group Key: (b.cust_id)::text
                                 Batches: 1  Memory Usage: 24kB
                                 ->  Seq Scan on bill b  (cost=0.00..14.75 rows=127 width=48) (actual time=0.492..0.502 rows=5 loops=1)
                                       Filter: (bill_total > '200'::double precision)
                                       Rows Removed by Filter: 5
 Planning Time: 17.238 ms
 Execution Time: 9.639 ms
(21 rows)
```



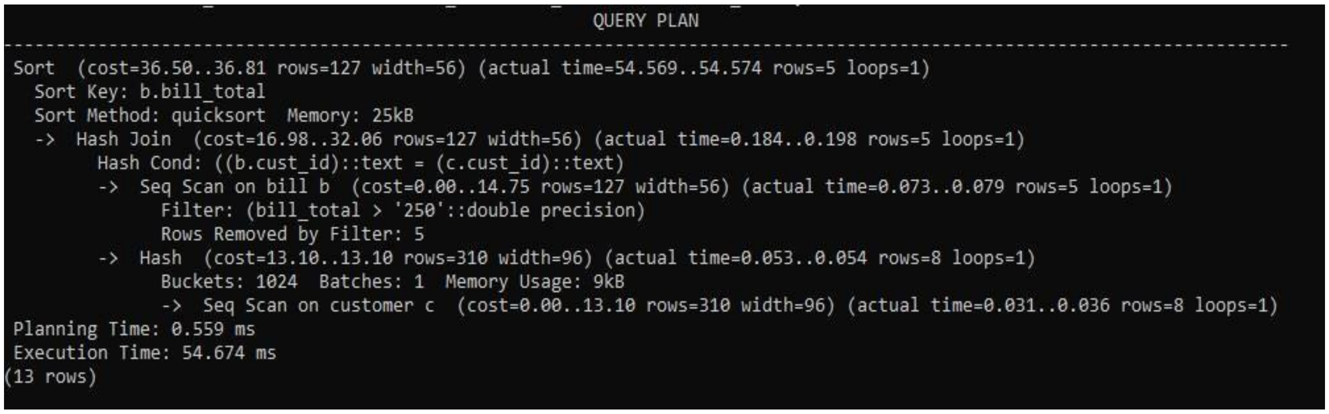
```
                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=33.68..35.68 rows=200 width=56) (actual time=0.339..0.349 rows=8 loops=1)
   Group Key: c.name
   Batches: 1  Memory Usage: 40kB
   ->  Hash Join  (cost=16.98..31.78 rows=380 width=48) (actual time=0.273..0.288 rows=10 loops=1)
         Hash Cond: ((b.cust_id)::text = (c.cust_id)::text)
         ->  Seq Scan on bill b  (cost=0.00..13.80 rows=380 width=48) (actual time=0.078..0.081 rows=10 loops=1)
         ->  Hash  (cost=13.10..13.10 rows=310 width=96) (actual time=0.131..0.132 rows=8 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on customer c  (cost=0.00..13.10 rows=310 width=96) (actual time=0.052..0.058 rows=8 loops=1)
 Planning Time: 33.815 ms
 Execution Time: 14.108 ms
(11 rows)
```



```
                                                  QUERY PLAN
--------------------------------------------------------------------------------------------------------------------
 Hash Semi Join  (cost=15.71..30.53 rows=2 width=90) (actual time=0.195..0.202 rows=3 loops=1)
   Hash Cond: (bill.bill_total = bill_1.bill_total)
   ->  Seq Scan on bill  (cost=0.00..13.80 rows=380 width=90) (actual time=0.053..0.056 rows=10 loops=1)
   ->  Hash  (cost=15.70..15.70 rows=1 width=8) (actual time=0.064..0.064 rows=3 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Seq Scan on bill bill_1  (cost=0.00..15.70 rows=1 width=8) (actual time=0.044..0.046 rows=3 loops=1)
               Filter: ((bill_total > '250'::double precision) AND (bill_date = '2021-09-21'::date))
               Rows Removed by Filter: 7
 Planning Time: 29.447 ms
 Execution Time: 0.272 ms
(10 rows)
```



```
                                                  QUERY PLAN
--------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=33.22..35.52 rows=230 width=52) (actual time=0.128..0.140 rows=14 loops=1)
   Group Key: seller.name, seller.age
   Batches: 1  Memory Usage: 40kB
   ->  Append  (cost=0.00..32.07 rows=230 width=52) (actual time=0.061..0.100 rows=14 loops=1)
         ->  Seq Scan on seller  (cost=0.00..14.75 rows=127 width=52) (actual time=0.060..0.066 rows=7 loops=1)
               Filter: (age < 50)
               Rows Removed by Filter: 1
         ->  Seq Scan on customer  (cost=0.00..13.88 rows=103 width=52) (actual time=0.024..0.028 rows=7 loops=1)
               Filter: (age < 50)
               Rows Removed by Filter: 1
 Planning Time: 0.411 ms
 Execution Time: 0.291 ms
(12 rows)
```



```
                                                  QUERY PLAN
--------------------------------------------------------------------------------------------------------------------
 Sort  (cost=36.50..36.81 rows=127 width=56) (actual time=54.569..54.574 rows=5 loops=1)
   Sort Key: b.bill_total
   Sort Method: quicksort  Memory: 25kB
   ->  Hash Join  (cost=16.98..32.06 rows=127 width=56) (actual time=0.184..0.198 rows=5 loops=1)
         Hash Cond: ((b.cust_id)::text = (c.cust_id)::text)
         ->  Seq Scan on bill b  (cost=0.00..14.75 rows=127 width=56) (actual time=0.073..0.079 rows=5 loops=1)
               Filter: (bill_total > '250'::double precision)
               Rows Removed by Filter: 5
         ->  Hash  (cost=13.10..13.10 rows=310 width=96) (actual time=0.053..0.054 rows=8 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on customer c  (cost=0.00..13.10 rows=310 width=96) (actual time=0.031..0.036 rows=8 loops=1)
 Planning Time: 0.559 ms
 Execution Time: 54.674 ms
(13 rows)
```

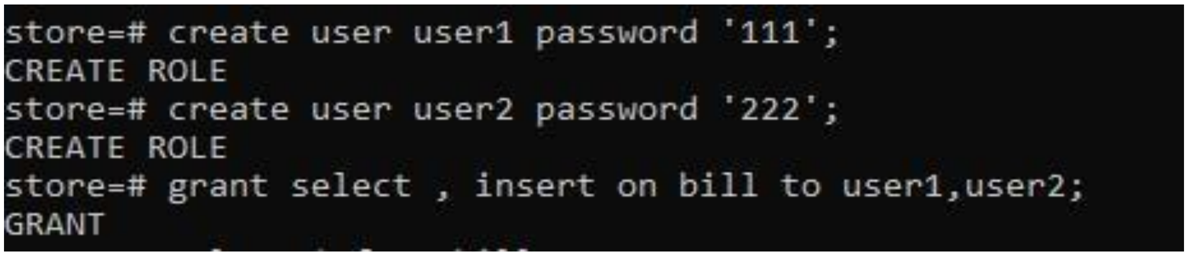
Access Privileges:

This task included adding access privileges that were relevant to the database. This included adding relevant roles, and providing them with only necessary privileges. For accessing the database with the role provided to the user, the following command is used:

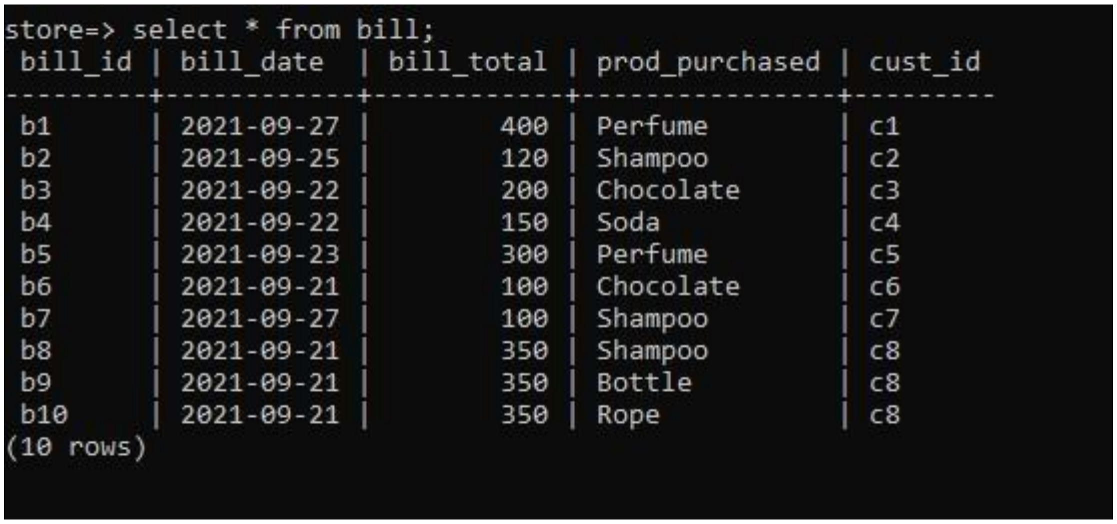
```
psql --host=localhost --dbname=store  --username=user
```

1)USER (Customer)

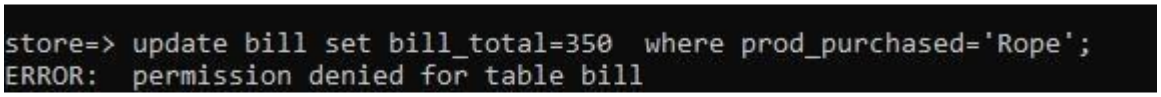
a) They can do the selection and insertion on the bill



```
store=# create user user1 password '111';
CREATE ROLE
store=# create user user2 password '222';
CREATE ROLE
store=# grant select , insert on bill to user1,user2;
GRANT
```



```
store=> select * from bill;
 bill_id | bill_date  | bill_total | prod_purchased | cust_id
---------+------------+------------+----------------+---------
 b1      | 2021-09-27 |        400 | Perfume        | c1
 b2      | 2021-09-25 |        120 | Shampoo        | c2
 b3      | 2021-09-22 |        200 | Chocolate      | c3
 b4      | 2021-09-22 |        150 | Soda           | c4
 b5      | 2021-09-23 |        300 | Perfume        | c5
 b6      | 2021-09-21 |        100 | Chocolate      | c6
 b7      | 2021-09-27 |        100 | Shampoo        | c7
 b8      | 2021-09-21 |        350 | Shampoo        | c8
 b9      | 2021-09-21 |        350 | Bottle         | c8
 b10     | 2021-09-21 |        350 | Rope           | c8
(10 rows)
```



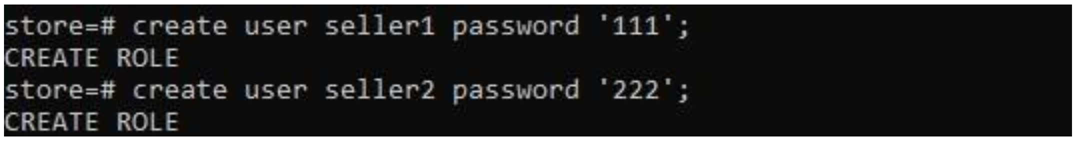
```
store=> update bill set bill_total=350  where prod_purchased='Rope';
ERROR:  permission denied for table bill
```

It can be seen that access to any other table other than the bill, is denied t

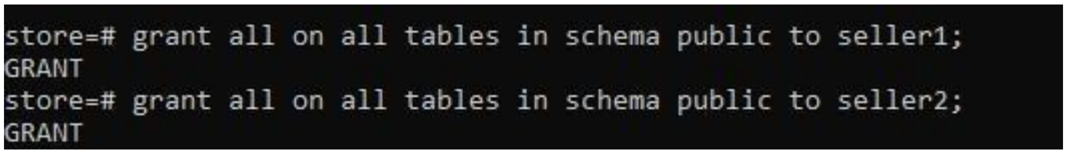
2)Seller

a) They are provided with the entire database's access.

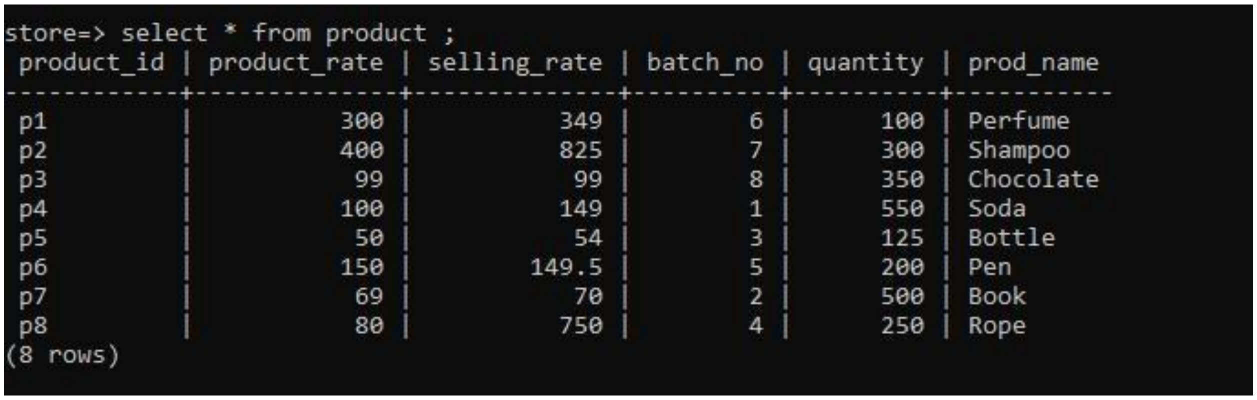
b) They can do all necessary operations on all the schemas in the database



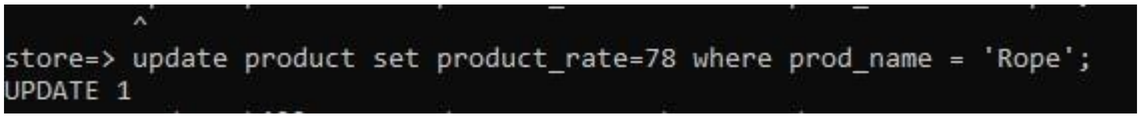
```
store=# create user seller1 password '111';
CREATE ROLE
store=# create user seller2 password '222';
CREATE ROLE
```



```
store=# grant all on all tables in schema public to seller1;
GRANT
store=# grant all on all tables in schema public to seller2;
GRANT
```



```
store=> select * from product ;
 product_id | product_rate | selling_rate | batch_no | quantity | prod_name
------------+--------------+--------------+----------+----------+-----------
 p1         |          300 |          349 |        6 |      100 | Perfume
 p2         |          400 |          825 |        7 |      300 | Shampoo
 p3         |           99 |           99 |        8 |      350 | Chocolate
 p4         |          100 |          149 |        1 |      550 | Soda
 p5         |           50 |           54 |        3 |      125 | Bottle
 p6         |          150 |        149.5 |        5 |      200 | Pen
 p7         |           69 |           70 |        2 |      500 | Book
 p8         |           80 |          750 |        4 |      250 | Rope
(8 rows)
```



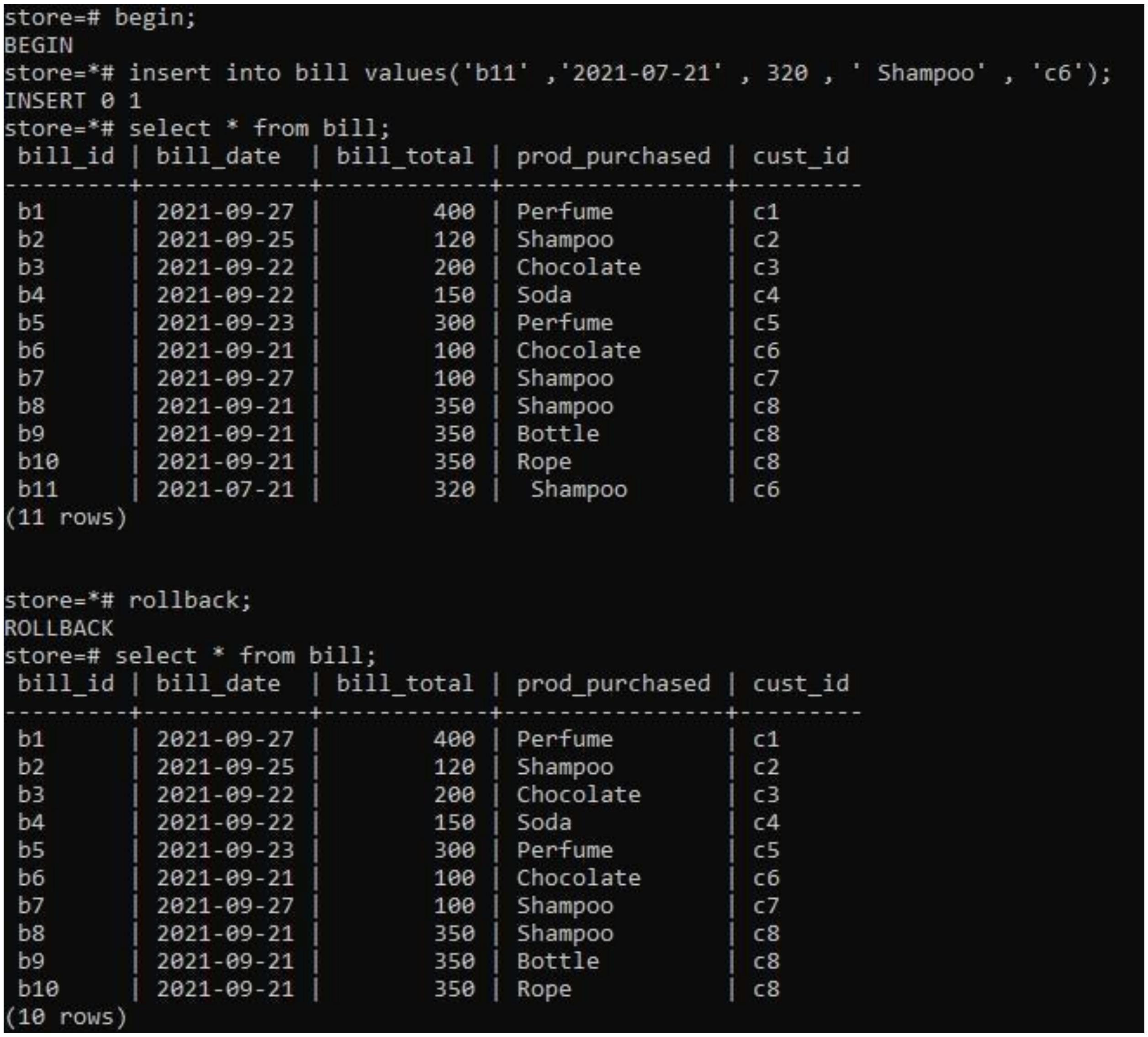
```
store=> update product set product_rate=78 where prod_name = 'Rope';
UPDATE 1
```

Here it can be seen that the seller is able to update the changes

Transactions:

In this section, we address how to execute transactions, and address the issue of concurrent access to the database, which might lead to data inconsistency. First, we exhibit three variations of transactions:

1) Transaction without commit.

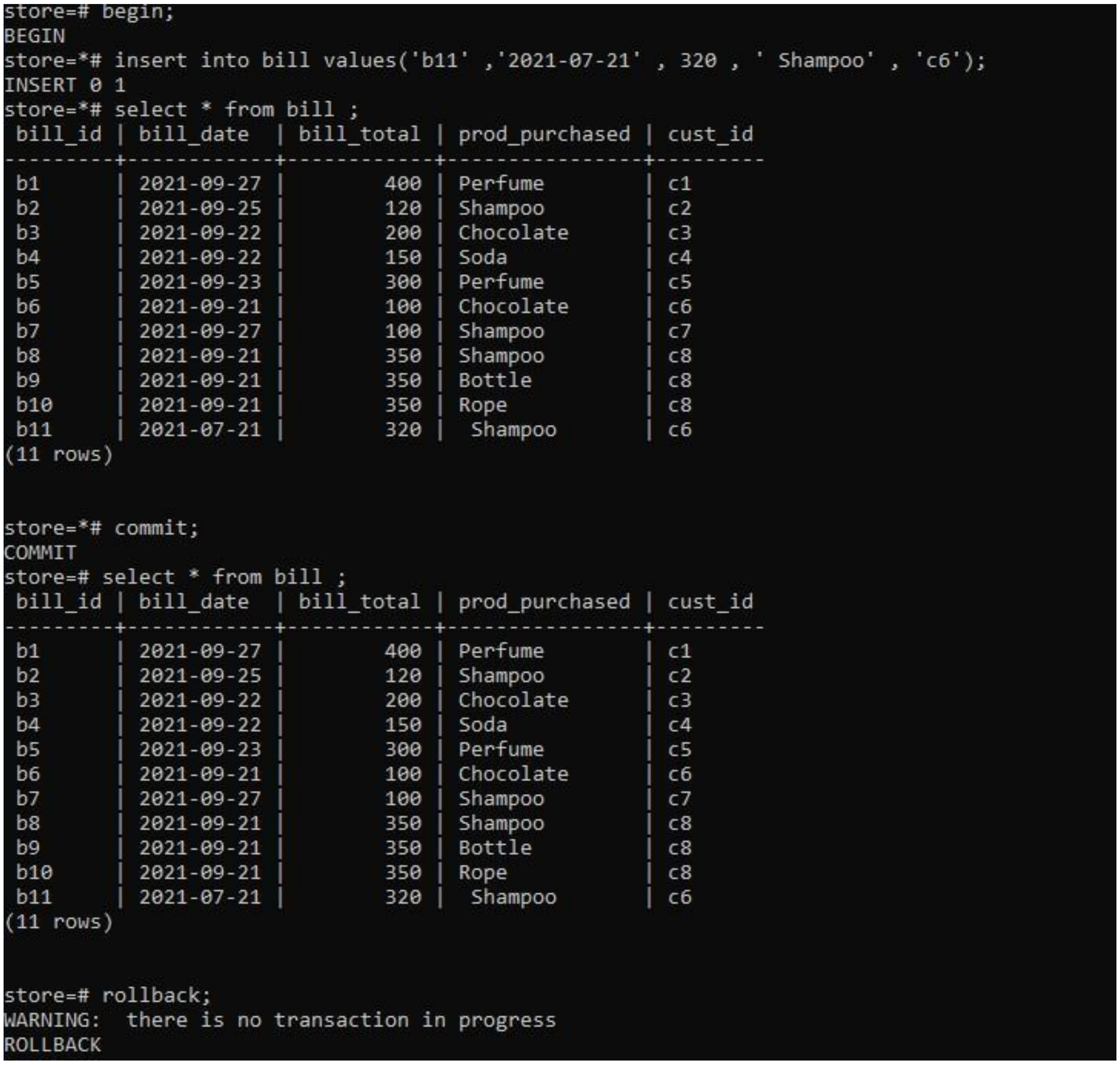


```
store=# begin;
BEGIN
store=*# insert into bill values('b11' ,'2021-07-21' , 320 , ' Shampoo' , 'c6');
INSERT 0 1
store=*# select * from bill;
 bill_id | bill_date  | bill_total | prod_purchased | cust_id
---------+------------+------------+----------------+---------
 b1      | 2021-09-27 |        400 | Perfume        | c1
 b2      | 2021-09-25 |        120 | Shampoo        | c2
 b3      | 2021-09-22 |        200 | Chocolate      | c3
 b4      | 2021-09-22 |        150 | Soda           | c4
 b5      | 2021-09-23 |        300 | Perfume        | c5
 b6      | 2021-09-21 |        100 | Chocolate      | c6
 b7      | 2021-09-27 |        100 | Shampoo        | c7
 b8      | 2021-09-21 |        350 | Shampoo        | c8
 b9      | 2021-09-21 |        350 | Bottle         | c8
 b10     | 2021-09-21 |        350 | Rope           | c8
 b11     | 2021-07-21 |        320 |  Shampoo       | c6
(11 rows)


store=*# rollback;
ROLLBACK
store=# select * from bill;
 bill_id | bill_date  | bill_total | prod_purchased | cust_id
---------+------------+------------+----------------+---------
 b1      | 2021-09-27 |        400 | Perfume        | c1
 b2      | 2021-09-25 |        120 | Shampoo        | c2
 b3      | 2021-09-22 |        200 | Chocolate      | c3
 b4      | 2021-09-22 |        150 | Soda           | c4
 b5      | 2021-09-23 |        300 | Perfume        | c5
 b6      | 2021-09-21 |        100 | Chocolate      | c6
 b7      | 2021-09-27 |        100 | Shampoo        | c7
 b8      | 2021-09-21 |        350 | Shampoo        | c8
 b9      | 2021-09-21 |        350 | Bottle         | c8
 b10     | 2021-09-21 |        350 | Rope           | c8
(10 rows)
```

In this transaction, we update the value of a table, and this change is not committed to the database. Hence, when we rollback, the old table’s values are shown instead of the new values. (inserting the products here)

2) Transaction with commit



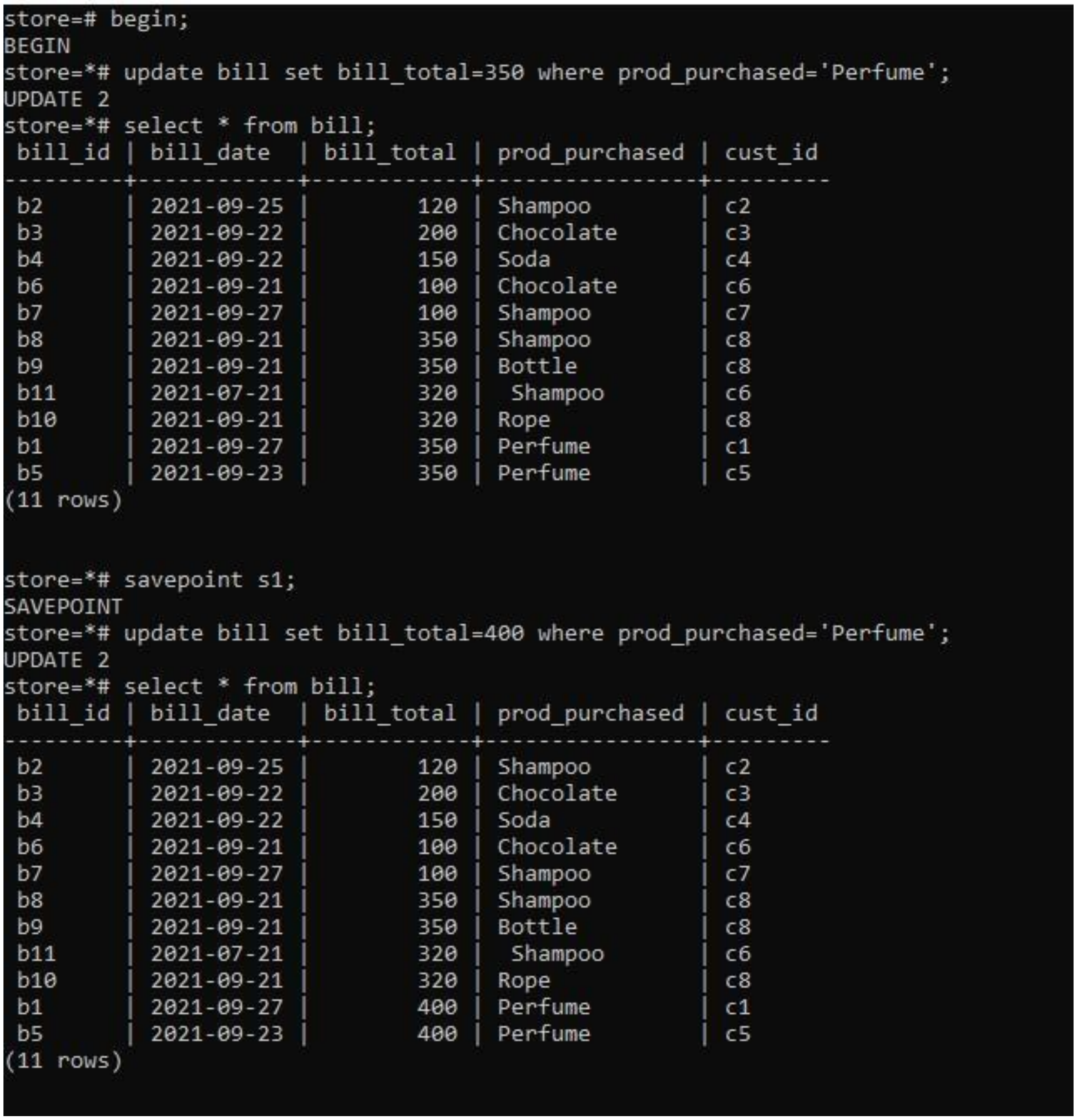
```
store=# begin;
BEGIN
store=*# insert into bill values('b11' ,'2021-07-21' , 320 , ' Shampoo' , 'c6');
INSERT 0 1
store=*# select * from bill ;
 bill_id | bill_date  | bill_total | prod_purchased | cust_id
---------+------------+------------+----------------+---------
 b1      | 2021-09-27 |        400 | Perfume        | c1
 b2      | 2021-09-25 |        120 | Shampoo        | c2
 b3      | 2021-09-22 |        200 | Chocolate      | c3
 b4      | 2021-09-22 |        150 | Soda           | c4
 b5      | 2021-09-23 |        300 | Perfume        | c5
 b6      | 2021-09-21 |        100 | Chocolate      | c6
 b7      | 2021-09-27 |        100 | Shampoo        | c7
 b8      | 2021-09-21 |        350 | Shampoo        | c8
 b9      | 2021-09-21 |        350 | Bottle         | c8
 b10     | 2021-09-21 |        350 | Rope           | c8
 b11     | 2021-07-21 |        320 |  Shampoo       | c6
(11 rows)


store=*# commit;
COMMIT
store=# select * from bill ;
 bill_id | bill_date  | bill_total | prod_purchased | cust_id
---------+------------+------------+----------------+---------
 b1      | 2021-09-27 |        400 | Perfume        | c1
 b2      | 2021-09-25 |        120 | Shampoo        | c2
 b3      | 2021-09-22 |        200 | Chocolate      | c3
 b4      | 2021-09-22 |        150 | Soda           | c4
 b5      | 2021-09-23 |        300 | Perfume        | c5
 b6      | 2021-09-21 |        100 | Chocolate      | c6
 b7      | 2021-09-27 |        100 | Shampoo        | c7
 b8      | 2021-09-21 |        350 | Shampoo        | c8
 b9      | 2021-09-21 |        350 | Bottle         | c8
 b10     | 2021-09-21 |        350 | Rope           | c8
 b11     | 2021-07-21 |        320 |  Shampoo       | c6
(11 rows)


store=# rollback;
WARNING:  there is no transaction in progress
ROLLBACK
```

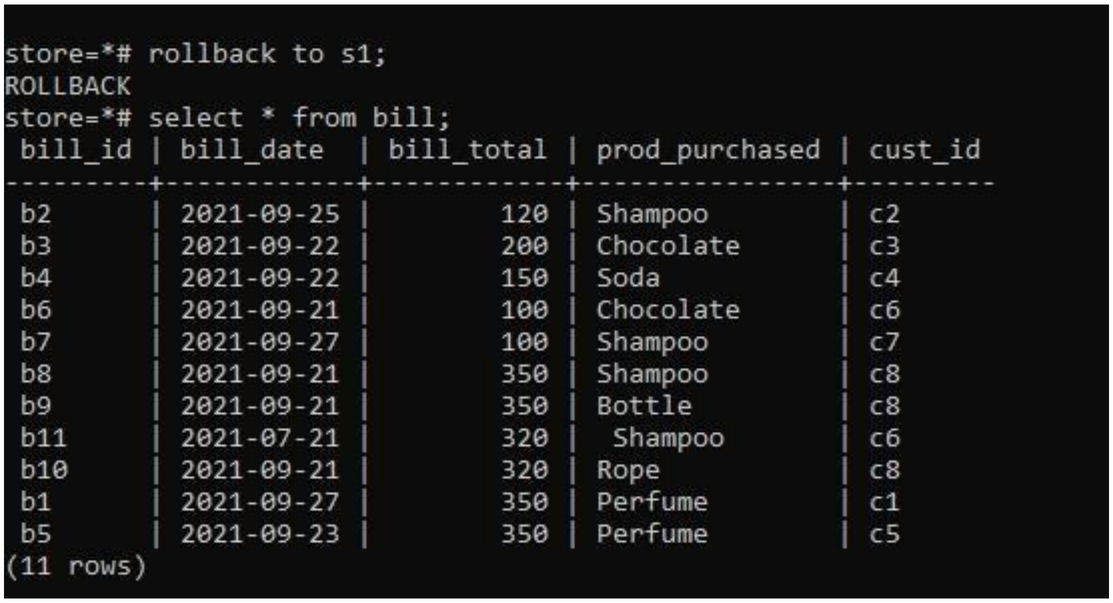
It can be seen that rollback is not allowed after the commit is executed, since all the changes are already reflected in the database and rollback does not have any savepoint to roll back to.

3) Transaction with savepoint



```
store=# begin;
BEGIN
store=*# update bill set bill_total=350 where prod_purchased='Perfume';
UPDATE 2
store=*# select * from bill;
 bill_id | bill_date  | bill_total | prod_purchased | cust_id
---------+------------+------------+----------------+---------
 b2      | 2021-09-25 |        120 | Shampoo        | c2
 b3      | 2021-09-22 |        200 | Chocolate      | c3
 b4      | 2021-09-22 |        150 | Soda           | c4
 b6      | 2021-09-21 |        100 | Chocolate      | c6
 b7      | 2021-09-27 |        100 | Shampoo        | c7
 b8      | 2021-09-21 |        350 | Shampoo        | c8
 b9      | 2021-09-21 |        350 | Bottle         | c8
 b11     | 2021-07-21 |        320 |  Shampoo       | c6
 b10     | 2021-09-21 |        320 | Rope           | c8
 b1      | 2021-09-27 |        350 | Perfume        | c1
 b5      | 2021-09-23 |        350 | Perfume        | c5
(11 rows)


store=*# savepoint s1;
SAVEPOINT
store=*# update bill set bill_total=400 where prod_purchased='Perfume';
UPDATE 2
store=*# select * from bill;
 bill_id | bill_date  | bill_total | prod_purchased | cust_id
---------+------------+------------+----------------+---------
 b2      | 2021-09-25 |        120 | Shampoo        | c2
 b3      | 2021-09-22 |        200 | Chocolate      | c3
 b4      | 2021-09-22 |        150 | Soda           | c4
 b6      | 2021-09-21 |        100 | Chocolate      | c6
 b7      | 2021-09-27 |        100 | Shampoo        | c7
 b8      | 2021-09-21 |        350 | Shampoo        | c8
 b9      | 2021-09-21 |        350 | Bottle         | c8
 b11     | 2021-07-21 |        320 |  Shampoo       | c6
 b10     | 2021-09-21 |        320 | Rope           | c8
 b1      | 2021-09-27 |        400 | Perfume        | c1
 b5      | 2021-09-23 |        400 | Perfume        | c5
(11 rows)
```



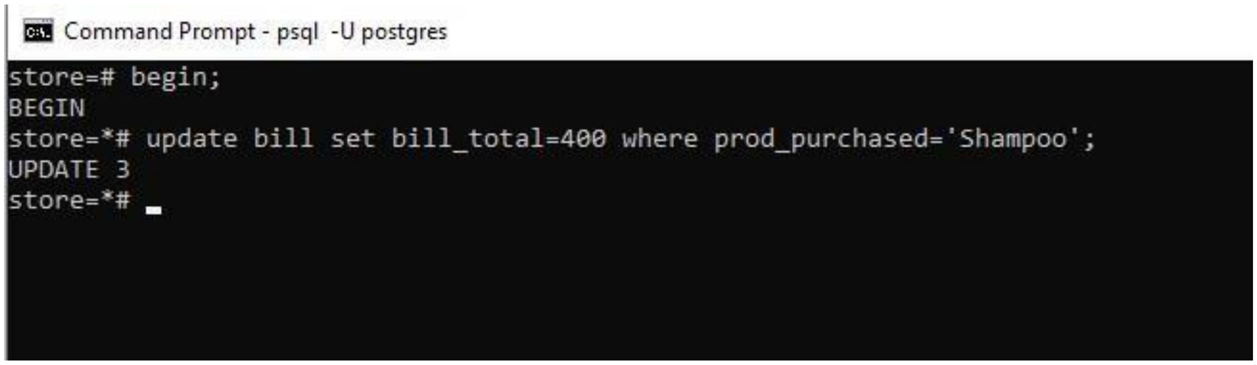
```
store=*# rollback to s1;
ROLLBACK
store=*# select * from bill;
 bill_id | bill_date  | bill_total | prod_purchased | cust_id
---------+------------+------------+----------------+---------
 b2      | 2021-09-25 |        120 | Shampoo        | c2
 b3      | 2021-09-22 |        200 | Chocolate      | c3
 b4      | 2021-09-22 |        150 | Soda           | c4
 b6      | 2021-09-21 |        100 | Chocolate      | c6
 b7      | 2021-09-27 |        100 | Shampoo        | c7
 b8      | 2021-09-21 |        350 | Shampoo        | c8
 b9      | 2021-09-21 |        350 | Bottle         | c8
 b11     | 2021-07-21 |        320 |  Shampoo       | c6
 b10     | 2021-09-21 |        320 | Rope           | c8
 b1      | 2021-09-27 |        350 | Perfume        | c1
 b5      | 2021-09-23 |        350 | Perfume        | c5
(11 rows)
```

It can be seen that rollback to a savepoint helps in preserving the data that was existing at a particular instance of time, and also was saved to a savepoint.

Concurrency Control:

In terms of concurrency control, PostgreSQL handles this using the concept of locks. These locks can be seen in most computer science concepts where concurrency exists, like Deadlocks in process synchronization (in Operating Systems).

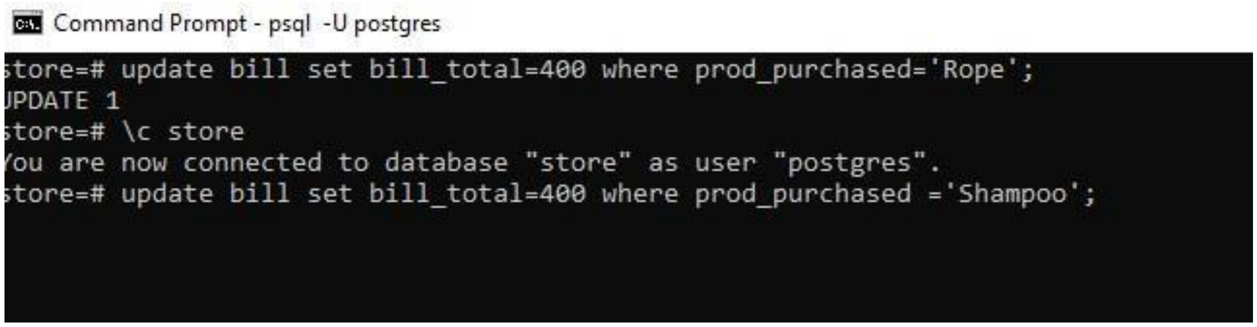
PostgreSQl has implicit locks that handle concurrent access. In the example shown below, PostgreSQL uses the concept of locks. This is emulated using two terminals accessing the same database. Terminal 1 tries to update a row from a table, and hence acquires a 'Row-Level Lock'. Now, since the lock is acquired by this user, no other user can update this specific row. Hence, it can be inferred that other rows in the table are still unlocked



Command Prompt - psql -U postgres

```
store=# begin;
BEGIN
store=*# update bill set bill_total=400 where prod_purchased='Shampoo';
UPDATE 3
store=*#
```

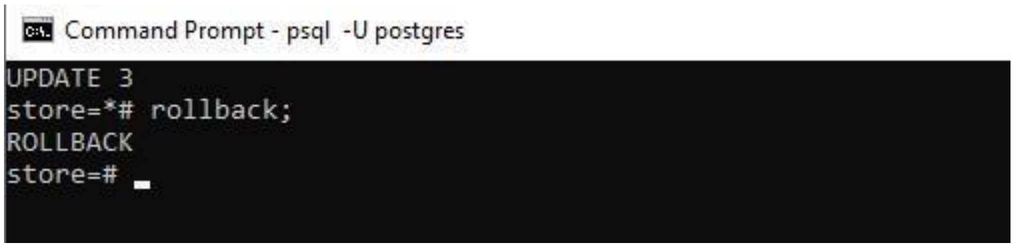
In this example, the user on the second terminal is not able to access that specific row, as can be seen below:



Command Prompt - psql -U postgres

```
tore=# update bill set bill_total=400 where prod_purchased='Rope';
PDATE 1
tore=# \c store
ou are now connected to database "store" as user "postgres".
tore=# update bill set bill_total=400 where prod_purchased ='Shampoo';
```

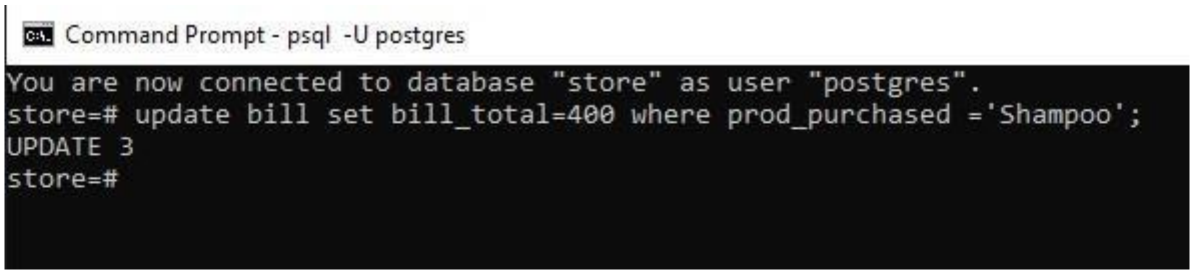
Finally, the user on the first terminal finishes the update and executes a rollback, and the lock is released .



Command Prompt - psql -U postgres

```
UPDATE 3
store=*# rollback;
ROLLBACK
store=#
```

It can now be seen that the user on the second terminal is able to access the previously locked row



Command Prompt - psql -U postgres

```
You are now connected to database "store" as user "postgres".
store=# update bill set bill_total=400 where prod_purchased ='Shampoo';
UPDATE 3
store=#
```

Time Contribution:

PRERANA HADADI
2H- Report Writing
2H- Complex Queries, Access Privileges

PRAMATHA GAJANAN BHAT
2H- Complex Queries, Simple Queries
2H- Access Privileges

PRATIKSHA D NAYAK
2H- Concurrency Control
2H- Transaction