Department of Computer Science and Engineering

Database Management System

UE19CS304

Assignment 3 and 4

SEM V

TEAM 18

SECTION B

Name: Gourav.Aravinda PES2UG19CS130
Bhargav Narayanan P PES2UG19CS088
Ashish Upadhya PES2UG19CS069

Simple Queries

1) Selecting number of Orders A Customer has ordered

Command Prompt - psql -U postgres

```
sales_catalog_management=# Select C.id, O.id, O.Product_ID, C.name
sales_catalog_management-# From Customers as C, Customer_Orders as O
sales_catalog_management-# Where C.id=O.Customer_ID;
  id   |  id   | product_id | name
-------+-------+------------+------
 cus_1 | co_1  | pc_1       | A
 cus_1 | co_2  | pc_2       | A
 cus_2 | co_3  | pc_1       | B
 cus_3 | co_4  | pc_4       | C
 cus_3 | co_5  | pc_2       | C
 cus_3 | co_6  | pc_6       | C
 cus_4 | co_7  | pc_7       | D
 cus_5 | co_8  | pc_4       | E
 cus_5 | co_9  | pc_9       | E
 cus_5 | co_10 | pc_10      | E
 cus_1 | co_11 | pc_1       | A
 cus_1 | co_12 | pc_1       | A
 cus_3 | co_13 | pc_1       | C
 cus_5 | co_14 | pc_8       | E
(14 rows)


sales_catalog_management=#
sales_catalog_management=# Explain Analyze Select C.id,(Count (*))
sales_catalog_management-#                 From Customers as C, Customer_Orders as O
sales_catalog_management-#                 Where C.id=O.Customer_ID
sales_catalog_management-#                 Group By C.id;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=25.05..26.25 rows=120 width=40) (actual time=0.030..0.032 rows=5 loops=1)
   Group Key: c.id
   Batches: 1  Memory Usage: 40kB
   ->  Hash Join  (cost=12.93..24.45 rows=120 width=32) (actual time=0.022..0.025 rows=14 loops=1)
         Hash Cond: ((o.customer_id)::text = (c.id)::text)
         ->  Seq Scan on customer_orders o  (cost=0.00..11.20 rows=120 width=32) (actual time=0.010..0.010 rows=14 loops=1)
         ->  Hash  (cost=11.30..11.30 rows=130 width=32) (actual time=0.007..0.007 rows=5 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on customers c  (cost=0.00..11.30 rows=130 width=32) (actual time=0.004..0.005 rows=5 loops=1)
 Planning Time: 0.109 ms
 Execution Time: 0.058 ms
(11 rows)
```

```
sales_catalog_management=#
sales_catalog_management=# Select C.id,(Count (*))
sales_catalog_management-# From Customers as C, Customer_Orders as O
sales_catalog_management-# Where C.id=O.Customer_ID
sales_catalog_management-# Group By C.id;
  id   | count
-------+-------
 cus_5 |     4
 cus_3 |     4
 cus_1 |     4
 cus_4 |     1
 cus_2 |     1
(5 rows)


sales_catalog_management=#
```

2) Selecting number of Orders A Driver is driving

Command Prompt - psql -U postgres

```
sales_catalog_management=# Select D.id, O.transportation_driver_id, D.name
sales_catalog_management-# From Transportation_Drivers as D, Order_Delivered_By as O
sales_catalog_management-# Where D.id=O.transportation_driver_id;
  id  | transportation_driver_id |    name
------+--------------------------+------------
 td_1 | td_1                     | Kumar
 td_1 | td_1                     | Kumar
 td_3 | td_3                     | Shreya
 td_3 | td_3                     | Shreya
 td_5 | td_5                     | Anushka
 td_6 | td_6                     | Ash
 td_7 | td_7                     | Amit Kumar
 td_8 | td_8                     | Aman
 td_9 | td_9                     | Arnab
 td_9 | td_9                     | Arnab
 td_1 | td_1                     | Kumar
 td_1 | td_1                     | Kumar
 td_4 | td_4                     | Ravi
(13 rows)


sales_catalog_management=#
sales_catalog_management=# Explain Analyze Select D.id,(Count (*))
sales_catalog_management-#                 From Transportation_Drivers as D, Order_Delivered_By as O
sales_catalog_management-#                 Where D.id=O.transportation_driver_id
sales_catalog_management-#                 Group By D.id;
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=38.25..39.45 rows=120 width=46) (actual time=0.030..0.032 rows=8 loops=1)
   Group Key: d.id
   Batches: 1  Memory Usage: 40kB
   ->  Hash Join  (cost=12.70..33.85 rows=880 width=38) (actual time=0.022..0.025 rows=13 loops=1)
         Hash Cond: ((o.transportation_driver_id)::text = (d.id)::text)
         ->  Seq Scan on order_delivered_by o  (cost=0.00..18.80 rows=880 width=32) (actual time=0.009..0.009 rows=13 loops=1)
         ->  Hash  (cost=11.20..11.20 rows=120 width=38) (actual time=0.008..0.008 rows=9 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on transportation_drivers d  (cost=0.00..11.20 rows=120 width=38) (actual time=0.004..0.004 rows=9 loops=1)
 Planning Time: 0.116 ms
 Execution Time: 0.057 ms
(11 rows)
```

```
sales_catalog_management=#
sales_catalog_management=# Select D.id,(Count (*))
sales_catalog_management-# From Transportation_Drivers as D, Order_Delivered_By as O
sales_catalog_management-# Where D.id=O.transportation_driver_id
sales_catalog_management-# Group By D.id;
  id   | count
------+-------
 td_8 |     1
 td_5 |     1
 td_7 |     1
 td_4 |     1
 td_6 |     1
 td_3 |     2
 td_9 |     2
 td_1 |     4
(8 rows)


sales_catalog_management=#
```

3) Selecting number of Products Handled by a SalesPerson

Command Prompt - psql -U postgres

```
sales_catalog_management-# From Sales_Person as S,Product_Catalog as P
sales_catalog_management-# Where S.id=P.Sales_Person_ID;
   id   |       name        |  id   |        name
--------+-------------------+-------+--------------------
 sp_101 | Amit Kumar        | pc_2  | Hamam Soap
 sp_103 | Augustine Ronaldo | pc_3  | Shampoo
 sp_201 | Ravi Kumar        | pc_4  | Air pods
 sp_101 | Amit Kumar        | pc_5  | Nutrilite
 sp_203 | Ravi Ash          | pc_6  | Duracell
 sp_301 | Amit Kumar        | pc_7  | Rice
 sp_202 | Anushka Viswas    | pc_9  | Laptop PC
 sp_303 | Arnab Ladwani     | pc_10 | Apples
 sp_103 | Augustine Ronaldo | pc_1  | Glister Toothpaste
 sp_303 | Arnab Ladwani     | pc_8  | Cereal
(10 rows)


sales_catalog_management=#
sales_catalog_management=# Explain Analyze Select S.id,(Count (*))
sales_catalog_management-#                 From Sales_Person as S,Product_Catalog as P
sales_catalog_management-#                 Where S.id=P.Sales_Person_ID
sales_catalog_management-#                 Group By S.id;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=24.47..25.47 rows=100 width=46) (actual time=0.029..0.030 rows=7 loops=1)
   Group Key: s.id
   Batches: 1  Memory Usage: 24kB
   ->  Hash Join  (cost=12.70..23.97 rows=100 width=38) (actual time=0.022..0.025 rows=10 loops=1)
         Hash Cond: ((p.sales_person_id)::text = (s.id)::text)
         ->  Seq Scan on product_catalog p  (cost=0.00..11.00 rows=100 width=38) (actual time=0.009..0.010 rows=10 loops=1)
         ->  Hash  (cost=11.20..11.20 rows=120 width=38) (actual time=0.008..0.008 rows=9 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on sales_person s  (cost=0.00..11.20 rows=120 width=38) (actual time=0.004..0.004 rows=9 loops=1)
 Planning Time: 0.104 ms
 Execution Time: 0.056 ms
(11 rows)
```

```
sales_catalog_management=#
sales_catalog_management=# Select S.id,(Count (*))
sales_catalog_management-# From Sales_Person as S,Product_Catalog as P
sales_catalog_management-# Where S.id=P.Sales_Person_ID
sales_catalog_management-# Group By S.id;
   id    | count
--------+-------
 sp_101 |     2
 sp_303 |     2
 sp_301 |     1
 sp_203 |     1
 sp_103 |     2
 sp_202 |     1
 sp_201 |     1
(7 rows)


sales_catalog_management=#
```

4) Selecting Number of Employees that work for each transportation company

Command Prompt - psql -U postgres

```
sales_catalog_management=# Select E.id,E.name,C.id,C.name
sales_catalog_management-# From Transportation_Drivers as E, Transportation_Company as C
sales_catalog_management-# Where E.Transportation_Company_ID=C.id;
 id  |    name     |  id  |   name
-----+-------------+------+-----------
 td_1 | Kumar      | tc_1 | Company A
 td_2 | Shukhla    | tc_2 | Company B
 td_3 | Shreya     | tc_1 | Company A
 td_4 | Ravi       | tc_3 | Company C
 td_5 | Anushka    | tc_2 | Company B
 td_6 | Ash        | tc_1 | Company A
 td_7 | Amit Kumar | tc_1 | Company A
 td_8 | Aman       | tc_2 | Company B
 td_9 | Arnab      | tc_3 | Company C
(9 rows)


sales_catalog_management=#
sales_catalog_management=# Explain  Analyze Select C.id,(Count (*))
sales_catalog_management-# From Transportation_Drivers as E, Transportation_Company as C
sales_catalog_management-# Where E.Transportation_Company_ID=C.id
sales_catalog_management-# Group By C.id;
                                                              QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=25.28..26.48 rows=120 width=46) (actual time=0.024..0.025 rows=3 loops=1)
   Group Key: c.id
   Batches: 1  Memory Usage: 40kB
   ->  Hash Join  (cost=13.15..24.68 rows=120 width=38) (actual time=0.018..0.020 rows=9 loops=1)
         Hash Cond: ((e.transportation_company_id)::text = (c.id)::text)
         ->  Seq Scan on transportation_drivers e  (cost=0.00..11.20 rows=120 width=38) (actual time=0.008..0.008 rows=9 loops=1)
         ->  Hash  (cost=11.40..11.40 rows=140 width=38) (actual time=0.006..0.006 rows=3 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on transportation_company c  (cost=0.00..11.40 rows=140 width=38) (actual time=0.003..0.003 rows=3 loops=1)
 Planning Time: 0.103 ms
 Execution Time: 0.048 ms
(11 rows)
```

```
sales_catalog_management=#
sales_catalog_management=# Select C.id,(Count (*))
sales_catalog_management-# From Transportation_Drivers as E, Transportation_Company as C
sales_catalog_management-# Where E.Transportation_Company_ID=C.id
sales_catalog_management-# Group By C.id;
  id   | count
-------+-------
 tc_2 |     3
 tc_3 |     2
 tc_1 |     4
(3 rows)
```

5) Selecting Number of Employees that work for each sales company

Command Prompt - psql -U postgres

```
sales_catalog_management=# Select E.id,E.name,C.id,C.name
sales_catalog_management-# From Sales_Person as E, Sales_Company as C
sales_catalog_management-# Where E.Sales_Company_ID=C.id;
   id   |       name        |  id  |       name
--------+-------------------+------+-----------------
 sp_103 | Augustine Ronaldo | sc_1 | Amway
 sp_101 | Amit Kumar        | sc_1 | Amway
 sp_102 | Rajeev Shukhla    | sc_1 | Amway
 sp_203 | Ravi Ash          | sc_2 | Reliance Digital
 sp_202 | Anushka Viswas    | sc_2 | Reliance Digital
 sp_201 | Ravi Kumar        | sc_2 | Reliance Digital
 sp_302 | Amanni Usha       | sc_3 | D-Mart
 sp_301 | Amit Kumar        | sc_3 | D-Mart
 sp_303 | Arnab Ladwani     | sc_3 | D-Mart
(9 rows)


sales_catalog_management=#
sales_catalog_management=# Explain Analyze Select C.id,(Count (*))
sales_catalog_management-#         From Sales_Person as E, Sales_Company as C
sales_catalog_management-#         Where E.Sales_Company_ID=C.id
sales_catalog_management-#         Group By C.id;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=25.28..26.48 rows=120 width=46) (actual time=0.024..0.025 rows=3 loops=1)
   Group Key: c.id
   Batches: 1  Memory Usage: 40kB
   ->  Hash Join  (cost=13.15..24.68 rows=120 width=38) (actual time=0.019..0.021 rows=9 loops=1)
         Hash Cond: ((e.sales_company_id)::text = (c.id)::text)
         ->  Seq Scan on sales_person e  (cost=0.00..11.20 rows=120 width=38) (actual time=0.008..0.009 rows=9 loops=1)
         ->  Hash  (cost=11.40..11.40 rows=140 width=38) (actual time=0.006..0.006 rows=3 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on sales_company c  (cost=0.00..11.40 rows=140 width=38) (actual time=0.003..0.003 rows=3 loops=1)
 Planning Time: 0.105 ms
 Execution Time: 0.049 ms
(11 rows)
```

```
sales_catalog_management=#
sales_catalog_management=# Select C.id,(Count (*))
sales_catalog_management-# From Sales_Person as E, Sales_Company as C
sales_catalog_management-# Where E.Sales_Company_ID=C.id
sales_catalog_management-# Group By C.id;
  id   | count
-------+-------
 sc_2 |     3
 sc_1 |     3
 sc_3 |     3
(3 rows)
```

Complex Queries

1) Selecting Transportation Company that have more then n employees

Command Prompt - psql -U postgres

```
sales_catalog_management=# Explain Analyze Select C.id,(Count (*))
sales_catalog_management-#         From Transportation_Drivers as E, Transportation_Company as C
sales_catalog_management-#         Where E.Transportation_Company_ID=C.id
sales_catalog_management-#         Group By C.id
sales_catalog_management-#         Having Count(*)>2
sales_catalog_management-#         Order BY C.id;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=27.84..27.94 rows=40 width=46) (actual time=0.712..0.713 rows=2 loops=1)
   Sort Key: c.id
   Sort Method: quicksort  Memory: 25kB
   ->  HashAggregate  (cost=25.28..26.78 rows=40 width=46) (actual time=0.030..0.032 rows=2 loops=1)
         Group Key: c.id
         Filter: (count(*) > 2)
         Batches: 1  Memory Usage: 40kB
         Rows Removed by Filter: 1
         ->  Hash Join  (cost=13.15..24.68 rows=120 width=38) (actual time=0.024..0.027 rows=9 loops=1)
               Hash Cond: ((e.transportation_company_id)::text = (c.id)::text)
               ->  Seq Scan on transportation_drivers e  (cost=0.00..11.20 rows=120 width=38) (actual time=0.006..0.006 rows=9 loops=1)
               ->  Hash  (cost=11.40..11.40 rows=140 width=38) (actual time=0.012..0.013 rows=3 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on transportation_company c  (cost=0.00..11.40 rows=140 width=38) (actual time=0.004..0.005 rows=3 loops=1)
 Planning Time: 0.120 ms
 Execution Time: 0.738 ms
(16 rows)


sales_catalog_management=#
sales_catalog_management=# Select C.id,(Count (*))
sales_catalog_management-# From Transportation_Drivers as E, Transportation_Company as C
sales_catalog_management-# Where E.Transportation_Company_ID=C.id
sales_catalog_management-# Group By C.id
sales_catalog_management-# Having Count(*)>2
sales_catalog_management-# Order BY C.id;
  id   | count
-------+-------
 tc_1 |     4
 tc_2 |     3
(2 rows)


sales_catalog_management=#
```

2) Selecting Sales Company that have more then n employees

Command Prompt - psql -U postgres

```
sales_catalog_management=# Explain Analyze Select C.id,(Count (*))
sales_catalog_management-#         From Sales_Person as E, Sales_Company as C
sales_catalog_management-#         Where E.Sales_Company_ID=C.id
sales_catalog_management-#         Group By C.id
sales_catalog_management-#         Having Count(*)>2
sales_catalog_management-#         Order By C.id;
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=27.84..27.94 rows=40 width=46) (actual time=0.038..0.039 rows=3 loops=1)
   Sort Key: c.id
   Sort Method: quicksort  Memory: 25kB
   ->  HashAggregate  (cost=25.28..26.78 rows=40 width=46) (actual time=0.027..0.028 rows=3 loops=1)
         Group Key: c.id
         Filter: (count(*) > 2)
         Batches: 1  Memory Usage: 40kB
         ->  Hash Join  (cost=13.15..24.68 rows=120 width=38) (actual time=0.021..0.023 rows=9 loops=1)
               Hash Cond: ((e.sales_company_id)::text = (c.id)::text)
               ->  Seq Scan on sales_person e  (cost=0.00..11.20 rows=120 width=38) (actual time=0.008..0.008 rows=9 loops=1)
               ->  Hash  (cost=11.40..11.40 rows=140 width=38) (actual time=0.006..0.007 rows=3 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on sales_company c  (cost=0.00..11.40 rows=140 width=38) (actual time=0.003..0.003 rows=3 loops=1)
 Planning Time: 0.113 ms
 Execution Time: 0.069 ms
(15 rows)


sales_catalog_management=#
sales_catalog_management=#
sales_catalog_management=# Select C.id,(Count (*))
sales_catalog_management-# From Sales_Person as E, Sales_Company as C
sales_catalog_management-# Where E.Sales_Company_ID=C.id
sales_catalog_management-# Group By C.id
sales_catalog_management-# Having Count(*)>2
sales_catalog_management-# Order By C.id;
  id  | count
------+-------
 sc_1 |     3
 sc_2 |     3
 sc_3 |     3
(3 rows)


sales_catalog_management=#
```

3) Selecting Customers With More then any order unpaid for

Command Prompt - psql -U postgres

```
sales_catalog_management=# Explain Analyze Select C.id,(Count (*))
sales_catalog_management-#         From Customers as C, Customer_Orders as O
sales_catalog_management-#         Where C.id=O.Customer_ID AND O.Payment_Status='pending'
sales_catalog_management-#         Group By C.id
sales_catalog_management-#         Order By C.id;
                                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=19.79..19.81 rows=1 width=40) (actual time=1.484..1.485 rows=1 loops=1)
   Group Key: c.id
   ->  Sort  (cost=19.79..19.80 rows=1 width=32) (actual time=1.465..1.466 rows=1 loops=1)
         Sort Key: c.id
         Sort Method: quicksort  Memory: 25kB
         ->  Nested Loop  (cost=0.14..19.78 rows=1 width=32) (actual time=1.458..1.460 rows=1 loops=1)
               ->  Seq Scan on customer_orders o  (cost=0.00..11.50 rows=1 width=32) (actual time=0.011..0.012 rows=1 loops=1)
                     Filter: (payment_status = 'pending'::text)
                     Rows Removed by Filter: 13
               ->  Index Only Scan using customers_pkey on customers c  (cost=0.14..8.16 rows=1 width=32) (actual time=1.446..1.446 rows=1 loops=1)
                     Index Cond: (id = (o.customer_id)::text)
                     Heap Fetches: 1
 Planning Time: 0.110 ms
 Execution Time: 1.509 ms
(14 rows)


sales_catalog_management=#
sales_catalog_management=# Select C.id,(Count (*))
sales_catalog_management-# From Customers as C, Customer_Orders as O
sales_catalog_management-# Where C.id=O.Customer_ID AND O.Payment_Status='pending'
sales_catalog_management-# Group By C.id
sales_catalog_management-# Order By C.id;
  id   | count
-------+-------
 cus_3 |     1
(1 row)


sales_catalog_management=#
```

4) Selecting Products Sold and Sorting them as per quantity

Command Prompt - psql -U postgres

```
sales_catalog_management=# Explain Analyze Select P.name,(Count (*))
sales_catalog_management-#         From Product_Catalog as P,Customer_Orders as O
sales_catalog_management-#         Where P.id=O.Product_ID
sales_catalog_management-#         Group By P.name
sales_catalog_management-#         Order By Count(*);
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=28.70..28.95 rows=100 width=524) (actual time=0.040..0.041 rows=8 loops=1)
   Sort Key: (count(*))
   Sort Method: quicksort  Memory: 25kB
   ->  HashAggregate  (cost=24.38..25.38 rows=100 width=524) (actual time=0.026..0.028 rows=8 loops=1)
         Group Key: p.name
         Batches: 1  Memory Usage: 24kB
         ->  Hash Join  (cost=12.25..23.78 rows=120 width=516) (actual time=0.019..0.022 rows=14 loops=1)
               Hash Cond: ((o.product_id)::text = (p.id)::text)
               ->  Seq Scan on customer_orders o  (cost=0.00..11.20 rows=120 width=32) (actual time=0.005..0.006 rows=14 loops=1)
               ->  Hash  (cost=11.00..11.00 rows=100 width=554) (actual time=0.008..0.009 rows=10 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on product_catalog p  (cost=0.00..11.00 rows=100 width=554) (actual time=0.004..0.005 rows=10 loops=1)
 Planning Time: 0.125 ms
 Execution Time: 0.062 ms
(14 rows)


sales_catalog_management=#
sales_catalog_management=# Select P.name,(Count (*))
sales_catalog_management-# From Product_Catalog as P,Customer_Orders as O
sales_catalog_management-# Where P.id=O.Product_ID
sales_catalog_management-# Group By P.name
sales_catalog_management-# Order By Count(*);
        name        | count
--------------------+-------
 Cereal             |     1
 Duracell           |     1
 Apples             |     1
 Rice               |     1
 Laptop PC          |     1
 Hamam Soap         |     2
 Air pods           |     2
 Glister Toothpaste |     5
(8 rows)


sales_catalog_management=#
```

5) Get The Order Delivery Location and the Warehouse_Location to pick it up

Command Prompt - psql -U postgres

```
sales_catalog_management=# Explain Analyze Select P.id,C.Address,W.Location
sales_catalog_management-#         From Product_Catalog as P, Customers as C, Warehouse_Location as W,Customer_Orders as O
sales_catalog_management-#         Where P.Warehouse_ID=W.id AND O.Product_ID=P.id AND O.Customer_ID=C.id
sales_catalog_management-#         Order By P.id;
                                                                    QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=54.42..54.72 rows=120 width=102) (actual time=0.069..0.071 rows=14 loops=1)
   Sort Key: p.id
   Sort Method: quicksort  Memory: 26kB
   ->  Hash Join  (cost=38.10..50.28 rows=120 width=102) (actual time=0.046..0.054 rows=14 loops=1)
         Hash Cond: ((o.customer_id)::text = (c.id)::text)
         ->  Hash Join  (cost=25.18..37.03 rows=120 width=102) (actual time=0.032..0.037 rows=14 loops=1)
               Hash Cond: ((p.warehouse_id)::text = (w.id)::text)
               ->  Hash Join  (cost=12.25..23.78 rows=120 width=108) (actual time=0.017..0.021 rows=14 loops=1)
                     Hash Cond: ((o.product_id)::text = (p.id)::text)
                     ->  Seq Scan on customer_orders o  (cost=0.00..11.20 rows=120 width=64) (actual time=0.004..0.004 rows=14 loops=1)
                     ->  Hash  (cost=11.00..11.00 rows=100 width=76) (actual time=0.007..0.007 rows=10 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 9kB
                           ->  Seq Scan on product_catalog p  (cost=0.00..11.00 rows=100 width=76) (actual time=0.003..0.004 rows=10 loops=1)
               ->  Hash  (cost=11.30..11.30 rows=130 width=70) (actual time=0.010..0.010 rows=5 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on warehouse_location w  (cost=0.00..11.30 rows=130 width=70) (actual time=0.004..0.004 rows=5 loops=1)
         ->  Hash  (cost=11.30..11.30 rows=130 width=64) (actual time=0.010..0.010 rows=5 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on customers c  (cost=0.00..11.30 rows=130 width=64) (actual time=0.006..0.007 rows=5 loops=1)
 Planning Time: 1.294 ms
 Execution Time: 0.114 ms
(21 rows)
```

```
sales_catalog_management=#
sales_catalog_management=# Select P.id,C.Address,W.Location
sales_catalog_management-# From Product_Catalog as P, Customers as C, Warehouse_Location as W,Customer_Orders as O
sales_catalog_management-# Where P.Warehouse_ID=W.id AND O.Product_ID=P.id AND O.Customer_ID=C.id
sales_catalog_management-# Order By P.id;
  id   | address |                location
-------+---------+----------------------------------------
 pc_1  | ban_1   | Kudlu Gate
 pc_1  | ban_3   | Kudlu Gate
 pc_1  | ban_1   | Kudlu Gate
 pc_1  | ban_1   | Kudlu Gate
 pc_1  | ban_2   | Kudlu Gate
 pc_10 | ban_5   | Mangamma Palya Rd
 pc_2  | ban_3   | Mangamma Palya Rd
 pc_2  | ban_1   | Mangamma Palya Rd
 pc_4  | ban_5   | Mangamma Palya Rd
 pc_4  | ban_3   | Mangamma Palya Rd
 pc_6  | ban_3   | Kudlu Gate
 pc_7  | ban_4   | shop no.34, 3rd floor, Mangamma Palya Rd
 pc_8  | ban_5   | Kudlu Gate
 pc_9  | ban_5   | shop no.34, 3rd floor, Mangamma Palya Rd
(14 rows)


sales_catalog_management=#
```

Nested Queries

1) Get Customers Who have an order that is worth more then 1000

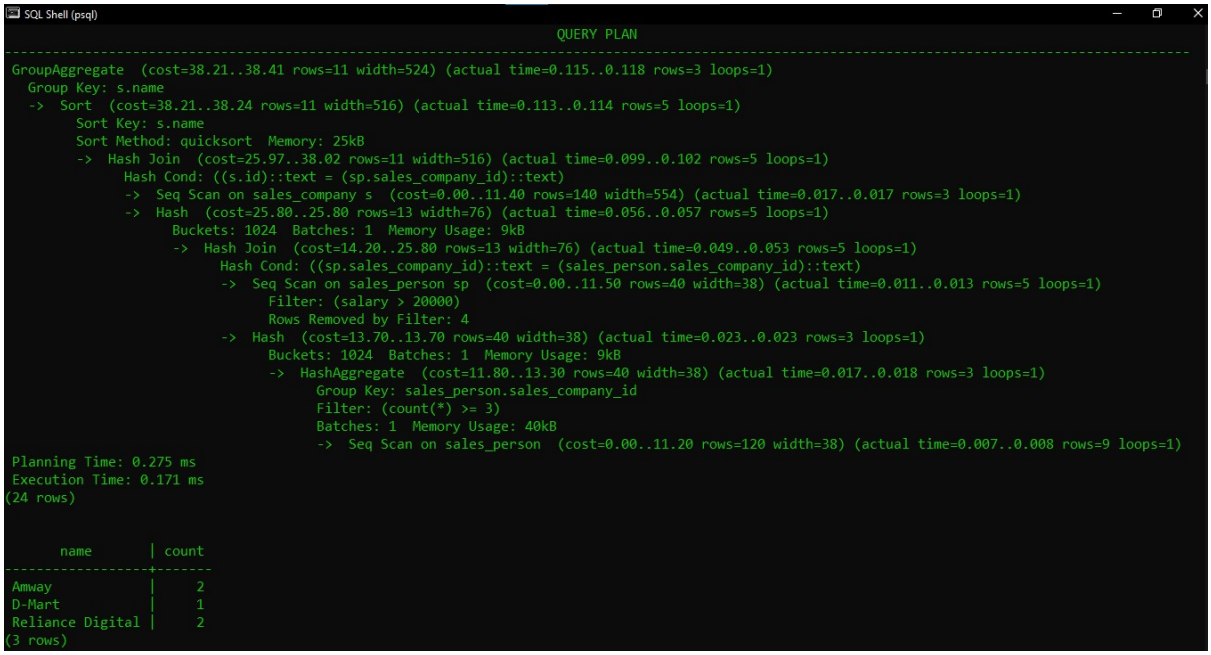
Command Prompt - psql -U postgres

```
sales_catalog_management=# Explain Analyze Select C.name,C.id
sales_catalog_management-#                From Customers as C, Customer_Orders as E
sales_catalog_management-#                Where E.id in (  Select O.id
sales_catalog_management(#                                 From Customer_Orders as O,Product_Catalog as P
sales_catalog_management(#                                 Where O.Product_ID=P.Id and O.Quantity*P.Price>1000)
sales_catalog_management-#                       AND
sales_catalog_management-#                     E.Customer_Id=C.id;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=36.73..48.91 rows=40 width=548) (actual time=0.043..0.044 rows=2 loops=1)
   Hash Cond: ((c.id)::text = (e.customer_id)::text)
   ->  Seq Scan on customers c  (cost=0.00..11.30 rows=130 width=548) (actual time=0.004..0.005 rows=5 loops=1)
   ->  Hash  (cost=36.23..36.23 rows=40 width=32) (actual time=0.032..0.033 rows=2 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Hash Semi Join  (cost=24.27..36.23 rows=40 width=32) (actual time=0.030..0.032 rows=2 loops=1)
               Hash Cond: ((e.id)::text = (o.id)::text)
               ->  Seq Scan on customer_orders e  (cost=0.00..11.20 rows=120 width=64) (actual time=0.003..0.004 rows=14 loops=1)
               ->  Hash  (cost=23.77..23.77 rows=40 width=32) (actual time=0.024..0.024 rows=2 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Hash Join  (cost=12.25..23.77 rows=40 width=32) (actual time=0.019..0.021 rows=2 loops=1)
                           Hash Cond: ((o.product_id)::text = (p.id)::text)
                           Join Filter: (((o.quantity)::double precision * p.price) > '1000'::double precision)
                           Rows Removed by Join Filter: 12
                           ->  Seq Scan on customer_orders o  (cost=0.00..11.20 rows=120 width=68) (actual time=0.002..0.003 rows=14 loops=1)
                           ->  Hash  (cost=11.00..11.00 rows=100 width=46) (actual time=0.008..0.008 rows=10 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                 ->  Seq Scan on product_catalog p  (cost=0.00..11.00 rows=100 width=46) (actual time=0.003..0.004 rows=10 loops=1)
 Planning Time: 0.516 ms
 Execution Time: 0.080 ms
(20 rows)
```

```
sales_catalog_management=#
sales_catalog_management=# Select C.name,C.id
sales_catalog_management-# From Customers as C, Customer_Orders as E
sales_catalog_management-# Where E.id in (  Select O.id
sales_catalog_management(#                 From Customer_Orders as O,Product_Catalog as P
sales_catalog_management(#                 Where O.Product_ID=P.Id and O.Quantity*P.Price>1000)
sales_catalog_management-#         AND
sales_catalog_management-#         E.Customer_Id=C.id;
 name |  id
------+-------
 C    | cus_3
 E    | cus_5
(2 rows)
```

2) To retreive Company name and number of employees who earn more than 20,000 given that atleast 3 employees are working in that company.

```
Explain Analyze Select S.name, count(*)
                from Sales_Company as S, Sales_Person as SP
                where S.id = SP.Sales_Company_ID and SP.salary > 20000 and
S.id in
                        (select Sales_Company_ID
                        from Sales_Person
                        group by Sales_Company_ID
                        having count(*) >= 3)
                group by S.name;
```


SQL Shell (psql)
QUERY PLAN
GroupAggregate (cost=38.21..38.41 rows=11 width=524) (actual time=0.115..0.118 rows=3 loops=1)
Group Key: s.name
-> Sort (cost=38.21..38.24 rows=11 width=516) (actual time=0.113..0.114 rows=5 loops=1)
Sort Key: s.name
Sort Method: quicksort Memory: 25kB
-> Hash Join (cost=25.97..38.02 rows=11 width=516) (actual time=0.099..0.102 rows=5 loops=1)
Hash Cond: ((s.id)::text = (sp.sales_company_id)::text)
-> Seq Scan on sales_company s (cost=0.00..11.40 rows=140 width=554) (actual time=0.017..0.017 rows=3 loops=1)
-> Hash (cost=25.80..25.80 rows=13 width=76) (actual time=0.056..0.057 rows=5 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Hash Join (cost=14.20..25.80 rows=13 width=76) (actual time=0.049..0.053 rows=5 loops=1)
Hash Cond: ((sp.sales_company_id)::text = (sales_person.sales_company_id)::text)
-> Seq Scan on sales_person sp (cost=0.00..11.50 rows=40 width=38) (actual time=0.011..0.013 rows=5 loops=1)
Filter: (salary > 20000)
Rows Removed by Filter: 4
-> Hash (cost=13.70..13.70 rows=40 width=38) (actual time=0.023..0.023 rows=3 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> HashAggregate (cost=11.80..13.30 rows=40 width=38) (actual time=0.017..0.018 rows=3 loops=1)
Group Key: sales_person.sales_company_id
Filter: (count(*) >= 3)
Batches: 1 Memory Usage: 40kB
-> Seq Scan on sales_person (cost=0.00..11.20 rows=120 width=38) (actual time=0.007..0.008 rows=9 loops=1)
Planning Time: 0.275 ms
Execution Time: 0.171 ms
(24 rows)
name | count
Amway | 2
D-Mart | 1
Reliance Digital | 2
(3 rows)

Functions and Triggers

Running all the functions and the triggers

Command Prompt - psql -U postgres

```
sales_catalog_management=# \i 'C:\\Coding\\Database Management Systems\\Assignment 3\\Functions.sql';
DROP FUNCTION
CREATE FUNCTION
DROP FUNCTION
CREATE FUNCTION
psql:C:/Coding/Database Management Systems/Assignment 3/Functions.sql:32: NOTICE:  drop cascades to trigger decrease_quantity_in_product_quantity on table product_catalog
DROP FUNCTION
CREATE FUNCTION
CREATE TRIGGER
DROP FUNCTION
CREATE FUNCTION
psql:C:/Coding/Database Management Systems/Assignment 3/Functions.sql:95: NOTICE:  drop cascades to trigger assign_driver_check on table order_delivered_by
DROP FUNCTION
CREATE FUNCTION
CREATE TRIGGER
sales_catalog_management=#
```

1) Ordering a Product

i) Code

File Edit Selection View Go Run Terminal Help — Functions.SQL - Visual Studio Code

Functions.SQL | BasicTables.sql | Queries.sql | Stored_Procedures.sql

C: > Coding > Database Management Systems > Assignment 3 > Functions.SQL

```
--Allows a customer to place an order by giving their payment method,product id, quantity and their customer_id. Will update the customer_orders.
Drop Function Ordering_Product(varchar,varchar,varchar,varchar,int);
Create or replace function Ordering_Product(Customer_ID varchar, Order_Id varchar, Payment_Method varchar(50),Product_ID varchar,Quantity int)
   returns  void
   language plpgsql
  as
$$
begin
    perform Decrease_Quantity(Product_ID,Quantity) ;
    Insert into Customer_Orders Values(Customer_ID,Order_Id,Payment_Method,'Unpaid',Product_ID,Quantity);
    perform Assign_Driver(Product_ID,Order_ID);
end;
$$;
```

ii)Working

We have to allow a customer to place an order. We have decrease the quantity of the product and assign a driver to deliver their product to them. We can see the working of both the Assign Driver and Decrease Quantity Function that will be called here as well.

Select Command Prompt - psql -U postgres

```
sales_catalog_management=#
sales_catalog_management=#
sales_catalog_management=# Select Ordering_Product('cus_2','co_15','cash','pc_8',2);
 ordering_product
------------------

(1 row)


sales_catalog_management=# Select * from Customer_Orders;
 customer_id |  id   | payment_method | payment_status | product_id | quantity
-------------+-------+----------------+----------------+------------+----------
 cus_1       | co_1  | cash           | success        | pc_1       |        1
 cus_1       | co_2  | cash           | success        | pc_2       |        1
 cus_2       | co_3  | cc             | success        | pc_1       |        2
 cus_3       | co_4  | cc             | success        | pc_4       |        2
 cus_3       | co_5  | cc             | success        | pc_2       |        1
 cus_3       | co_6  | cash           | pending        | pc_6       |        3
 cus_4       | co_7  | dc             | success        | pc_7       |        2
 cus_5       | co_8  | dc             | success        | pc_4       |        1
 cus_5       | co_9  | cash           | success        | pc_9       |        1
 cus_5       | co_10 | cash           | success        | pc_10      |        2
 cus_1       | co_11 | cash           | Unpaid         | pc_1       |        1
 cus_1       | co_12 | cash           | Unpaid         | pc_1       |        1
 cus_3       | co_13 | cash           | Unpaid         | pc_1       |        3
 cus_5       | co_14 | cash           | Unpaid         | pc_8       |        3
 cus_2       | co_15 | cash           | Unpaid         | pc_8       |        2
(15 rows)


sales_catalog_management=#
```

2) Decreasing Quantity Of Product

i) Code

```
--Decreasing Quantity()--Will decrease the quantity. Updates the record. There is a trigger check for that. Will update the Product_Catalog
Drop Function If Exists Decrease_Quantity(varchar,int);
Create or Replace function Decrease_Quantity(Product_ID_Decrease varchar(50),Quantity_Decrease int)
   returns void
   language plpgsql
  as
$$
begin
    Update Product_Catalog
    Set Available_QTY=Available_QTY-Quantity_Decrease
    Where ID=Product_ID_Decrease;
end;
$$;

```

ii) Working

The code here will decrease the quantity of the item when an order is placed by a customer or when a sales person wants to decrease the quantity as it got damaged.

Command Prompt - psql -U postgres

```
sales_catalog_management=#
sales_catalog_management=#
sales_catalog_management=# Select *
sales_catalog_management-# From Product_Catalog
sales_catalog_management-# Where Id='pc_4';
   name   |  id  | price | available_qty | description | sales_company_id | sales_person_id | product_warehouse_id | transportation_company_id | warehouse_id
----------+------+-------+---------------+-------------+------------------+-----------------+----------------------+---------------------------+--------------
 Air pods | pc_4 |   799 |             5 | Wired       | sc_2             | sp_201          | Pware_2              | tc_1                      | ware_5
(1 row)


sales_catalog_management=# Select Decrease_Quantity('pc_4',1);
 decrease_quantity
-------------------

(1 row)


sales_catalog_management=# Select *
sales_catalog_management-# From Product_Catalog
sales_catalog_management-# Where Id='pc_4';
   name   |  id  | price | available_qty | description | sales_company_id | sales_person_id | product_warehouse_id | transportation_company_id | warehouse_id
----------+------+-------+---------------+-------------+------------------+-----------------+----------------------+---------------------------+--------------
 Air pods | pc_4 |   799 |             4 | Wired       | sc_2             | sp_201          | Pware_2              | tc_1                      | ware_5
(1 row)


sales_catalog_management=#
```

3) Decreasing Quantity Trigger

i) Code

```
--This trigger will ensure that the quantity of the product as per product_catalog is not taken to a negative value
Drop Function If Exists Decrease_Quantity_Trigger_Check() Cascade;
Create or Replace function Decrease_Quantity_Trigger_Check()
   returns TRIGGER
   language plpgsql
  as
$$
begin
    if NEW.Available_QTY<0 then
        Raise Exception 'Quantity is decreasing below zero. Please Order a Quantity that is lower then the one you have chosen';
    end if;
    return new;
end;
$$;
CREATE TRIGGER Decrease_Quantity_In_Product_Quantity
  BEFORE UPDATE
  ON Product_Catalog
  FOR EACH ROW
  EXECUTE PROCEDURE Decrease_Quantity_Trigger_Check();
```

ii) Working

When the quantity that was to be updated has crossed a its available quantity, it will not be allowed to update and an exception is raised asking us to decrease the quantity we want to order.

```
sales_catalog_management=# Select *
sales_catalog_management-# From Product_Catalog
sales_catalog_management-# Where Id='pc_4';
   name   |  id  | price | available_qty | description | sales_company_id | sales_person_id | product_warehouse_id | transportation_company_id | warehouse_id
----------+------+-------+---------------+-------------+------------------+-----------------+----------------------+---------------------------+--------------
 Air pods | pc_4 |   799 |             4 | Wired       | sc_2             | sp_201          | Pware_2              | tc_1                      | ware_5
(1 row)


sales_catalog_management=# Select Decrease_Quantity('pc_4',2000);
ERROR:  Quantity is decreasing below zero. Please Order a Quantity that is lower then the one you have chosen
CONTEXT:  PL/pgSQL function decrease_quantity_trigger_check() line 4 at RAISE
SQL statement "Update Product_Catalog
    Set Available_QTY=Available_QTY-Quantity_Decrease
    Where ID=Product_ID_Decrease"
PL/pgSQL function decrease_quantity(character varying,integer) line 3 at SQL statement
sales_catalog_management=#
```

4) Assigning a Driver

i) Code

```
56  --Assign_Driver()--Will select from one of the available drivers and update the Order_Delivered_By table
57  Drop Function if exists Assign_Driver(varchar) Cascade;
58  Create or replace function Assign_Driver(Customer_ID varchar)
59     returns void
60     language plpgsql
61    as
62  $$
63  declare
64      check_insert integer :=0;
65      cur cursor for select Name, ID from Transportation_Drivers;
66      pointer record;
67  begin
68      open cur;
69      loop
70          fetch cur into pointer;
71          exit when not found;
72          Insert into Order_Delivered_By Values(pointer.ID,Customer_ID);
73          get diagnostics check_insert = ROW_COUNT;
74          if check_insert = 1 then
75              raise notice 'Order has been alloted to driver % with Driver_ID: %', pointer.Name, pointer.ID ;
76              exit;
77          end if;
78      end loop;
79      close cur;
80      if check_insert = 0 then
81          raise exception 'All drivers already have 3 orders to deliver!!';
82      end if;
83  end;
84  $$;
85
```

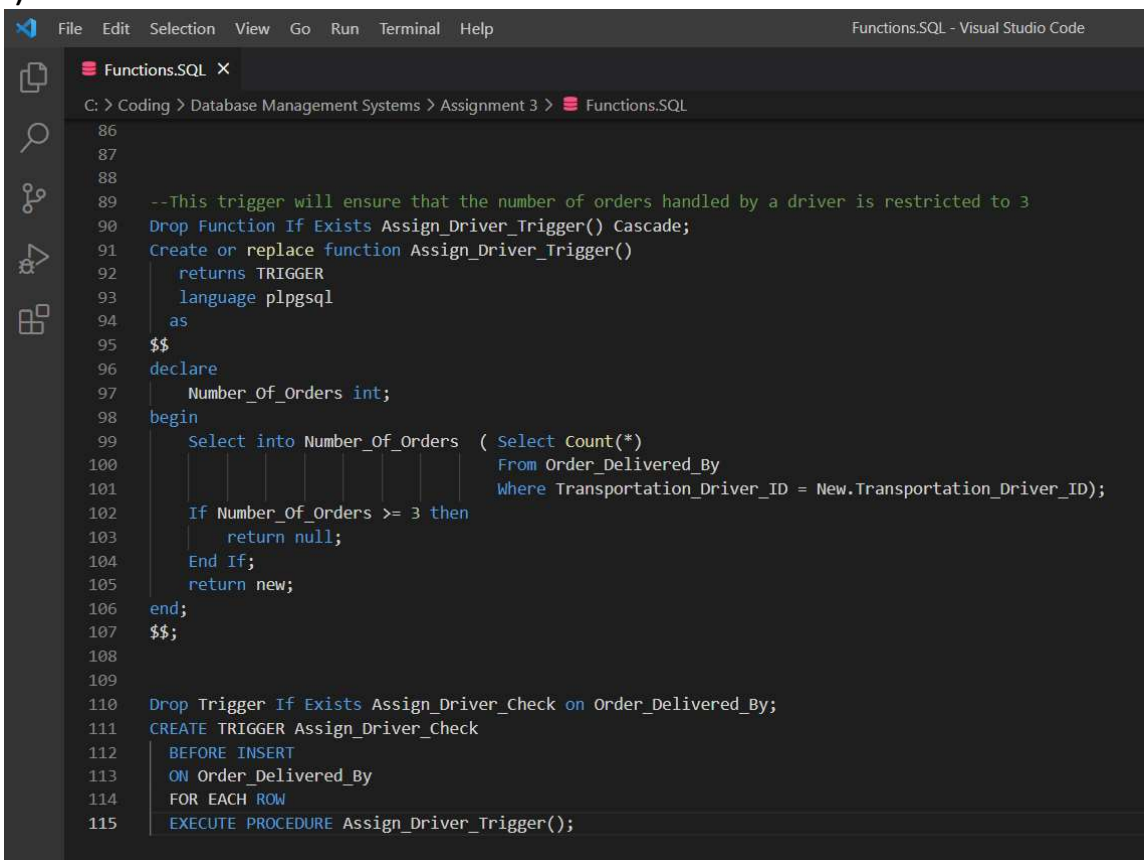
ii) Working

We need to assign a driver to take care of a customer order. This cannot be shown directly as we have to get an order. It will be shown using an example of an order.

```
sales_catalog_management=# select Ordering_Product('cus_5','co_15','cash', 'pc_1',1);
NOTICE:  Order alloted to driver
 ordering_product
------------------

(1 row)


sales_catalog_management=# select * from order_delivered_by;
 transportation_driver_id | customer_order_id
--------------------------+-------------------
 td_1                     | co_1
 td_1                     | co_2
 td_3                     | co_3
 td_3                     | co_4
 td_5                     | co_5
 td_6                     | co_6
 td_7                     | co_7
 td_8                     | co_8
 td_9                     | co_9
 td_9                     | co_10
 td_1                     | co_13
 td_1                     | co_14
 td_2                     | co_15
(13 rows)
```

5) Assigning Driver trigger

i) Code




```
86
87
88
89  --This trigger will ensure that the number of orders handled by a driver is restricted to 3
90  Drop Function If Exists Assign_Driver_Trigger() Cascade;
91  Create or replace function Assign_Driver_Trigger()
92     returns TRIGGER
93     language plpgsql
94    as
95  $$
96  declare
97      Number_Of_Orders int;
98  begin
99      Select into Number_Of_Orders  ( Select Count(*)
100                                      From Order_Delivered_By
101                                      Where Transportation_Driver_ID = New.Transportation_Driver_ID);
102     If Number_Of_Orders >= 3 then
103         return null;
104     End If;
105     return new;
106 end;
107 $$;
108
109
110 Drop Trigger If Exists Assign_Driver_Check on Order_Delivered_By;
111 CREATE TRIGGER Assign_Driver_Check
112   BEFORE INSERT
113   ON Order_Delivered_By
114   FOR EACH ROW
115   EXECUTE PROCEDURE Assign_Driver_Trigger();
```

ii) Working

We will only allow a driver to be able to deliver at the max 3 orders at a time. We wont allow more then that.

```
CREATE TRIGGER
sales_catalog_management=# select Ordering_Product('cus_5','co_16','cash', 'pc_1',1);
ERROR:  All drivers have 3 orders to take care of
CONTEXT:  PL/pgSQL function assign_driver(character varying,character varying) line 22 at RAISE
SQL statement "SELECT Assign_Driver(Product_ID,Order_ID)"
PL/pgSQL function ordering_product(character varying,character varying,character varying,charact
sales_catalog_management=#
```

6) Add Product to catalog- <u>Function with cursors</u>

I) Code

```sql
/*Function to insert new entries into product catalog */
create or replace function CatalogInsert(Fname varchar, Product_ID varchar, price float, qty int, descp text, scompid varchar, sperid varchar, prodwarehid varchar, transpcompid varchar, warehouse varchar)
returns void
language plpgsql
as
$$
    declare
        check_insert integer;
        curs cursor for select Name, ID from Sales_Person;
        pointer record;

    begin

        insert into Product_Catalog values(Fname, Product_ID, price, qty, descp, scompid, sperid, prodwarehid, transpcompid, warehouse);
        get diagnostics check_insert = ROW_COUNT;

        if check_insert = 0 then
            raise notice 'Sales Person with ID:% already handles 3 products ', sperid;
            open curs;
            loop
                fetch curs into pointer;
                exit when not found;

                insert into Product_Catalog values(Fname, Product_ID, price, qty, descp, scompid, pointer.ID, prodwarehid, transpcompid, warehouse);
                get diagnostics check_insert = ROW_COUNT;

                if check_insert = 1 then
                    raise notice 'Order has been alloted to Sales Person with ID: %', pointer.ID ;
                    exit;
                end if;
            end loop;

            if check_insert = 0 then
                raise exception 'All Sales Persons have been alloted 3 products!!';
            end if;


        else
            raise notice 'Order has been alloted to Sales Person with ID: %', sperid;
        end if;
        close curs;
    end;
$$;

```

ii)Working

Adds a product catalog and assigns a sales person.

```
sales_catalog_management=# select CatalogInsert('Ponds', 'pc_11', 200.0, 15, 'Multi-Action Toothpaste With Herbals', 'sc_1', 'sp_103', 'Pware_1', 'tc
_1', 'ware_3');
NOTICE:  Order has been alloted to Sales Person with ID: sp_103
 cataloginsert
---------------

(1 row)
```

7) Preventing redundant catalog inserts - <u>trigger with cursors</u>

I) Code

```
/* Trigger to check for redundant entries and handle them accordingly in product catalog*/
create or replace function remove_redundant_product()
returns trigger As
$$
    declare
    cur cursor for select * from Product_Catalog as C;
    pointer record;
    flag integer :=0;

    begin
        open cur;
    loop
        fetch cur into pointer;
        exit when not found;
        if pointer.Name = NEW.Name then
            flag := 1;
            Raise Exception 'Product Named % already exists with ID %! Instead perform an UPDATE.', pointer.Name, pointer.ID;
            exit;
        end if;
    end loop;
    close cur;

    if flag = 1 then
        return null;
    else
        return NEW;
    end if;
    end;
$$ language plpgsql;

create or replace trigger redundantprod
    before insert on Product_Catalog
    for each row
    execute procedure remove_redundant_product();
```

II) Working - Exception is raised if product with the same name already exists. Otherwise insert to product catalog.

```
sales_catalog_management=# select CatalogInsert('Glister Toothpaste', 'pc_11', 200.0, 15, 'Multi-Action Toothpaste With Herbals', 'sc_1', 'sp_103', '
Pware_1', 'tc_1', 'ware_3');
ERROR:  Product Named Glister Toothpaste already exists with ID pc_1! Instead perform an UPDATE.
CONTEXT:  PL/pgSQL function remove_redundant_product() line 14 at RAISE
SQL statement "insert into Product_Catalog values(Fname, Product_ID, price, qty, descp, scompid, sperid, prodwarehid, transpcompid, warehouse)"
PL/pgSQL function cataloginsert(character varying,character varying,double precision,integer,text,character varying,character varying,character varyi
ng,character varying,character varying) line 9 at SQL statement
sales_catalog_management=#
```

8) Assign Sales Person - Triggers

I) Code

```
Create or replace function Assign_SalesPerson_Trigger()
   returns TRIGGER
   language plpgsql
  as
$$
declare
    Number_Of_Products int;
begin
    Select into Number_Of_Products  ( Select Count(*)
                                      From Product_Catalog
                                      Where Sales_Person_ID = New.Sales_Person_ID);
    If Number_Of_Products >= 3 then
        return null;
    End If;
    return new;
end;
$$;


CREATE OR REPLACE TRIGGER Assign_SalesPerson_Check
  BEFORE INSERT
  ON Product_Catalog
  FOR EACH ROW
  EXECUTE PROCEDURE Assign_SalesPerson_Trigger();
```

II) Working

Newly inserted Product is assigned to a Sales Person who handles less than 3 Products.

```
sales_catalog_management=# select CatalogInsert('Pears', 'pc_12', 200.0, 15, 'Multi-Action Toothpaste With Herbals', 'sc_1', 'sp_103', 'Pware_1', 'tc
_1', 'ware_3');
NOTICE:  Sales Person with ID:sp_103 already handles 3 products
NOTICE:  Order has been alloted to Sales Person with ID: sp_101
 cataloginsert
---------------

(1 row)
```

9) Update quantity in product Catalog - Function

I) Code

```
create or replace function CatalogUpdate(Product_ID varchar, Quantity_Increase int)
returns void
language plpgsql
as
$$
    declare
        rowsUpserted integer;
    begin
        Update Product_Catalog
        Set Available_QTY=Available_QTY + Quantity_Increase
        Where ID=Product_ID;

        get diagnostics rowsUpserted = ROW_COUNT;
        if rowsUpserted = 0 then
            Raise Exception 'Product ID not found';
        end if;
    end;
$$;

```

II) Working

To update the quantity of items.

```
sales_catalog_management=# select CatalogUpdate('pc_11', 15);
 catalogupdate
---------------


(1 row)
```

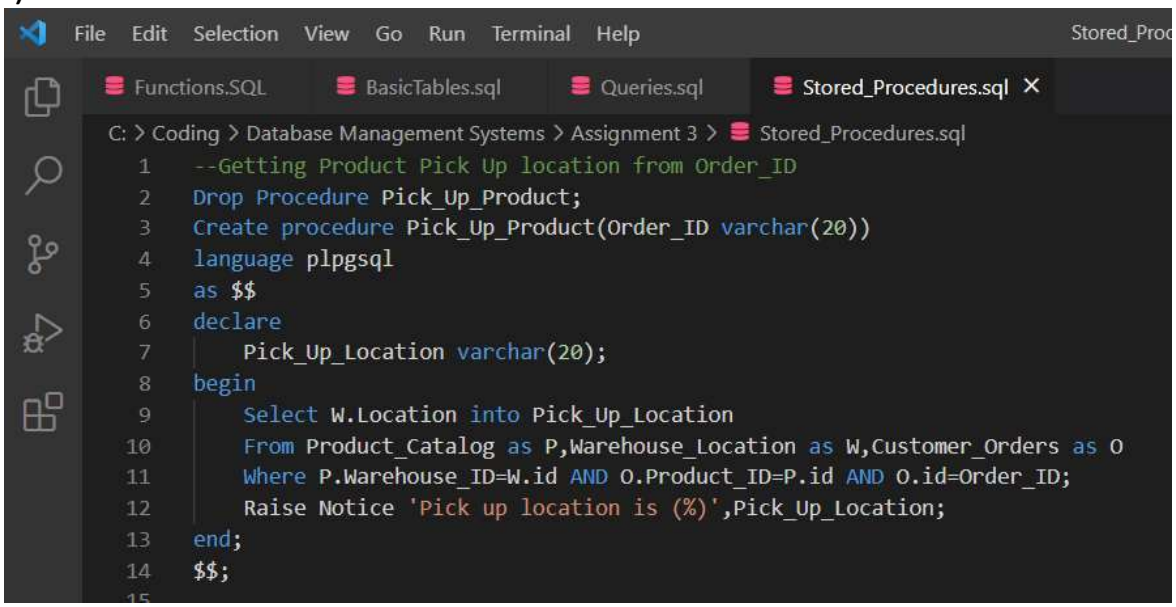
SQL Shell (psql)

```
sales_catalog_management=# select * from product_catalog;
        name        |  id   | price | available_qty |                 description                  | sales_company_id | sales_person_id | product_w
arehouse_id | transportation_company_id | warehouse_id
--------------------+-------+-------+---------------+----------------------------------------------+------------------+-----------------+----------
------------+---------------------------+-------------
 Glister Toothpaste | pc_1  |   200 |            15 | Multi-Action Toothpaste With Herbals         | sc_1             | sp_103          | Pware_1
           | tc_1                      | ware_3
 Hamam Soap         | pc_2  |   100 |            12 | Herbal Soap                                  | sc_1             | sp_101          | Pware_1
           | tc_1                      | ware_1
 Shampoo            | pc_3  |   150 |            25 | Hairfall Control With Conditioner            | sc_1             | sp_103          | Pware_1
           | tc_2                      | ware_1
 Air pods           | pc_4  |   799 |             5 | Wired                                        | sc_2             | sp_201          | Pware_2
           | tc_1                      | ware_5
 Nutrilite          | pc_5  |  1735 |            25 | Multivitamin and Multimineral Tablets        | sc_1             | sp_101          | Pware_1
           | tc_1                      | ware_4
 Duracell           | pc_6  |    75 |           500 | AAA batteries                                | sc_2             | sp_203          | Pware_2
           | tc_3                      | ware_3
 Rice               | pc_7  |   102 |            12 | Red rice (1 qty = 2Kg Sac)                   | sc_3             | sp_301          | Pware_3
           | tc_2                      | ware_4
 Cereal             | pc_8  |   200 |             7 | Breakfast Cereal Protein & Fiber             | sc_3             | sp_303          | Pware_3
           | tc_3                      | ware_3
 Laptop PC          | pc_9  | 74999 |             8 | HP Envy Ryzen 5                              | sc_2             | sp_202          | Pware_2
           | tc_2                      | ware_4
 Apples             | pc_10 |   359 |             9 | Kashmiri Fresh Fruit Apples (1 qty = 2 apples) | sc_3             | sp_303          | Pware_3
           | tc_3                      | ware_5
 Pears              | pc_12 |   200 |            15 | Multi-Action Toothpaste With Herbals         | sc_1             | sp_101          | Pware_1
           | tc_1                      | ware_3
 Ponds              | pc_11 |   200 |            30 | Multi-Action Toothpaste With Herbals         | sc_1             | sp_103          | Pware_1
           | tc_1                      | ware_3
(12 rows)
```

Stored Procedures

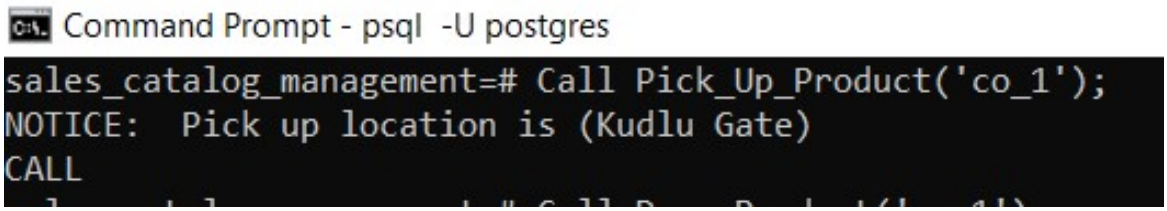
1) Fetching Pick Up location for a customer order

i) Code



ii) Working

We will fetch the location that a driver has to go and pick up the customer order so that he can pick the order up.



2) Fetching Drop location for a customer order

i) Code

```
15
16  --Getting Product Drop Location from Order_ID
17  Drop Procedure Drop_Product;
18  Create procedure Drop_Product(Order_ID varchar(20))
19  language plpgsql
20  as $$
21  declare
22      Drop_Location varchar(20);
23  begin
24      Select C.Address into Drop_Location
25      From Product_Catalog as P, Customers as C,Customer_Orders as O
26      Where O.Product_ID=P.id AND O.Customer_ID=C.id And O.id=Order_ID;
27      Raise Notice 'Drop location is (%)',Drop_Location;
28  end;
29  $$;
```

ii) Working

We will fetch the location of the customer so that we can tell the driver to deliver the order to that specific location.

```
CALL
sales_catalog_management=# Call Drop_Product('co_1');
NOTICE:  Drop location is (ban_1)
CALL
sales_catalog_management=#
```

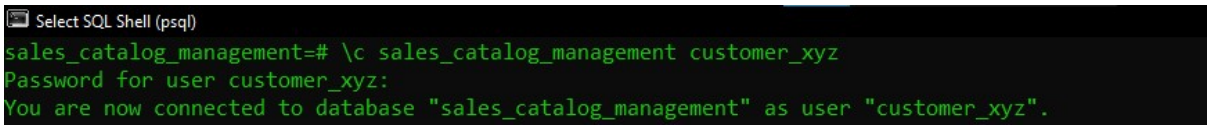
Materialized Views with Access Privileges

Codes:

```
create materialized view prodCatalog_view
AS
select * from Product_Catalog;

create materialized view custOrders_view
AS
select * from Customer_Orders;

create materialized view deliv_view
AS
select * from Order_Delivered_By;


create user customer_xyz
with
password 'passwd101';
grant select on prodCatalog_view to customer_xyz;

create user SalesPerson_xyz
with
password 'passwd102';
grant select on custOrders_view to SalesPerson_xyz;

create user driver_xyz
with
password 'passwd103';
grant select on deliv_view to driver_xyz;

```

Displaying Various Views of each User:

Customer:



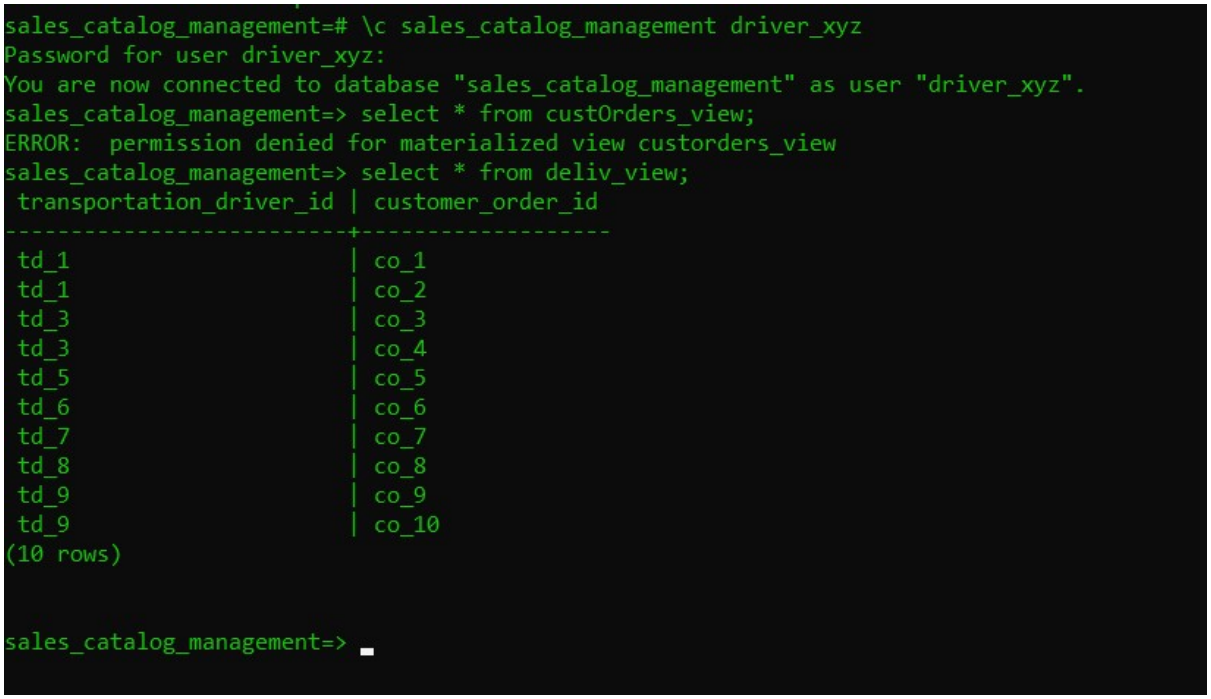
```
sales_catalog_management=> select * from prodCatalog_view;
        name        |  id   | price | available_qty |                description                 | sales_company_id | sales_person_id | product_w
arehouse_id | transportation_company_id | warehouse_id
--------------------+-------+-------+---------------+--------------------------------------------+------------------+-----------------+----------
------------+---------------------------+-------------
 Glister Toothpaste | pc_1  |   200 |            15 | Multi-Action Toothpaste With Herbals       | sc_1             | sp_103          | Pware_1
           | tc_1                      | ware_3
 Hamam Soap         | pc_2  |   100 |            12 | Herbal Soap                                | sc_1             | sp_101          | Pware_1
           | tc_1                      | ware_1
 Shampoo            | pc_3  |   150 |            25 | Hairfall Control With Conditioner          | sc_1             | sp_103          | Pware_1
           | tc_2                      | ware_1
 Air pods           | pc_4  |   799 |             5 | Wired                                      | sc_2             | sp_201          | Pware_2
           | tc_1                      | ware_5
 Nutrilite          | pc_5  |  1735 |            25 | Multivitamin and Multimineral Tablets      | sc_1             | sp_101          | Pware_1
           | tc_1                      | ware_4
 Duracell           | pc_6  |    75 |           500 | AAA batteries                              | sc_2             | sp_203          | Pware_2
           | tc_3                      | ware_3
 Rice               | pc_7  |   102 |            12 | Red rice (1 qty = 2Kg Sac)                 | sc_3             | sp_301          | Pware_3
           | tc_2                      | ware_4
 Cereal             | pc_8  |   200 |             7 | Breakfast Cereal Protein & Fiber           | sc_3             | sp_303          | Pware_3
           | tc_3                      | ware_3
 Laptop PC          | pc_9  | 74999 |             8 | HP Envy Ryzen 5                            | sc_2             | sp_202          | Pware_2
           | tc_2                      | ware_4
 Apples             | pc_10 |   359 |             9 | Kashmiri Fresh Fruit Apples (1 qty = 2 apples) | sc_3         | sp_303          | Pware_3
           | tc_3                      | ware_5
 Pears              | pc_12 |   200 |            15 | Multi-Action Toothpaste With Herbals       | sc_1             | sp_101          | Pware_1
           | tc_1                      | ware_3
 Ponds              | pc_11 |   200 |            30 | Multi-Action Toothpaste With Herbals       | sc_1             | sp_103          | Pware_1
           | tc_1                      | ware_3
(12 rows)
```

```
sales_catalog_management=> select * from custOrders_view;
ERROR:  permission denied for materialized view custorders_view
sales_catalog_management=>
```

Sales Person:

```
SQL Shell (psql)
sales_catalog_management=# select * from custOrders_view;
 customer_id |  id   | payment_method | payment_status | product_id | quantity
-------------+-------+----------------+----------------+------------+----------
 cus_1       | co_1  | cash           | success        | pc_1       |        1
 cus_1       | co_2  | cash           | success        | pc_2       |        1
 cus_2       | co_3  | cc             | success        | pc_1       |        2
 cus_3       | co_4  | cc             | success        | pc_4       |        2
 cus_3       | co_5  | cc             | success        | pc_2       |        1
 cus_3       | co_6  | cash           | pending        | pc_6       |        3
 cus_4       | co_7  | dc             | success        | pc_7       |        2
 cus_5       | co_8  | dc             | success        | pc_4       |        1
 cus_5       | co_9  | cash           | success        | pc_9       |        1
 cus_5       | co_10 | cash           | success        | pc_10      |        2
(10 rows)


sales_catalog_management=#
```

Driver:



Moving From PostgreSQL to No-SQL(MongoDB)

The process for transferring data from PostgreSQL to MongoDB is clear-cut. Ultimately, the ease of our task depends on the complexity of the PostgreSQL database and the structure of document collections needed in the new MongoDB database. To migrate data, we can extract it from PostgreSQL and then import it to MongoDB using the mongoimport tool. Let's look at the two different ways to extract the data: returning queries as tab-separated values (TSV) or as JSON.

Using TSV to transfer data

The TSV format is the quick and easy option. However, it only works if the original PostgreSQL schema is relatively simple and you don't need to embed documents in other documents with a one-to-many relationship.

For example, if we need to create a collection of documents that represent customers and plan on embedding each customer's order in these documents, TSV won't work because it doesn't support hierarchical data structures. Each row in the TSV will become a document. You can still map values in each row to fields deeper in your documents; you just can't embed documents.

We could create an address field and create nested state and city fields as in the example below. However, you could not store multiple address entities. Let's look at an example query to see how this works.

Consider the PostgreSQL created table “users.”

```
postgres=# select * from users;
 userid | first_name | last_name | logins | upgraded | city | state
--------+------------+-----------+--------+----------+------+-------
      1 | Bob        | Smith     |     10 | t        | NYC  | NY
```

```
COPY (SELECT
      userid AS "userId.int32()",
      logins AS "loginCount.int32()",
    first_name AS "firstName.auto()",
    last_name AS "lastName.auto()",
      CASE WHEN upgraded=TRUE THEN 'TRUE' ELSE 'FALSE' END AS "upgraded
Account.boolean()",
      city AS "address.city.auto()",
      state AS "address.state.auto()"
  FROM
      users
) TO '/tmp/users.tsv' WITH (FORMAT CSV, HEADER TRUE, DELIMITER E'\t');
```

Notice that we renamed each column we are exporting with the AS command. The format of the alias being created is mongoFieldName.type(). For example, we have userId.int32(). When we execute the import, mongoimport will parse this header and create the fields with the correct types. On most of the

columns, we use the auto() type and let mongoimport determine the type based on context.

For the upgraded column, which is a boolean, PostgreSQL will return a t for true and f for false. This default value won't be recognized by MongoDB, so we use a CASE statement to set values that will work.

We can also do some limited data nesting using the TSV migration process. The city and state fields are an example.

The final line in the query formats the export as CSV with a header using a tab delimiter, which is what makes the file TSV format. We use this command to import the file into MongoDB Atlas:

```
mongoimport --uri mongodb+srv://<mongodb_user>:<mongodb_password>@<altas-cluster>.mongodb.net/<DATABASE>
--collection users --type tsv --file users.tsv --headerline –columnsHaveTypes
```

This will result in the following document in the “users” collection:

```
{
  userId: 1,
  first_name: "Bob",
  last_name: "Smith",
    LoginCount : 10,
    upgraded : true,
  "Address" : {
        "city" : "NYC",
        "state" : "NY" }
}
```

Using JSON to transfer data

Using JSON for data migration is preferable if your PostgreSQL schema is complex and you want to nest arrays of related records inside of a MongoDB document.

To return the results of a PostgreSQL query as JSON, you will need three functions:

1. row_to_json: Returns a row as a JSON object with column names as keys and the values from the row
2. array_to_json: Returns an array of data as a JSON array
3. array_agg: Accepts a set of values and returns an array where each value in the set becomes an element in the array

Let's look at an orders table which in our relational schema keeps a record for every product ordered by the user:

```
id | userid | product | quantity | price
----+--------+---------+----------+-------
 1 |      1 | shoes   |        4 | 50.75
 2 |      1 | razer   |       20 |  1.75
```

Here is an example query using all three functions:

```
COPY (SELECT row_to_json(results)
FROM (
 SELECT userid,first_name, last_name,
  (
    SELECT array_to_json(array_agg(o))
    FROM (
      SELECT id, product, quantity, price
      FROM orders
      WHERE products.userid = users.userid
    ) o
  ) AS orders
 FROM users
) results) TO '/tmp/orders.json' WITH (FORMAT text, HEADER FALSE);
```

The query above will create a file orders.json with JSON documents for each user from the users table:

```
{
  id: 1,
  first_name: "Bob",
 last_name: "Smith",
  "orders" : [
     {
        "id" : 1,
        "product" : "shoes",
        "quantity" : 4,
        "price" : 50.75
     },
     {
        "id" : 2,
        "product" : "razer",
        "quantity" : 20,
        "price" : 1.75
     }
  ]
}
```

Once you have written the query and saved it, you can use mongoimport to import the file:

```
mongoimport --uri mongodb+srv://<mongodb_user>:<mongodb_password>@<atlas-cluster>.mognodb.net/<DATABASE>
--collection orders --jsonArray orders.json
```

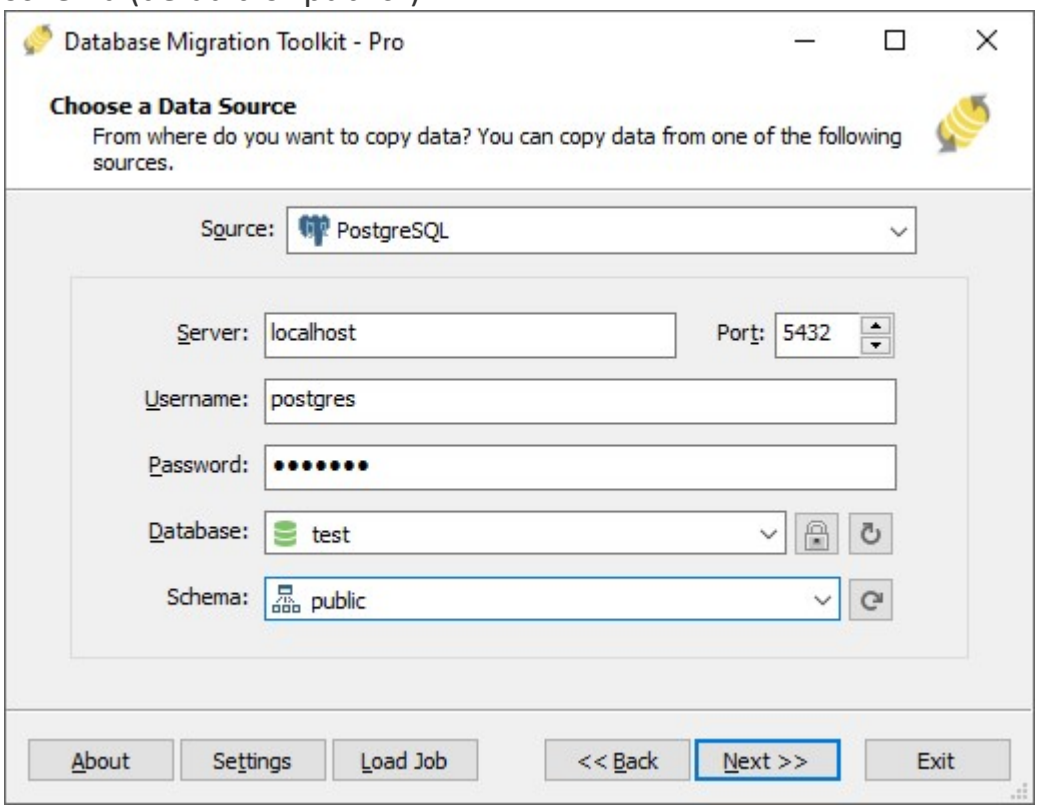
Moving From PostgreSQL to SQL

This is much easier then the above transition to a No-SQL database. We can use the ESF Database Migration Toolkit. It is a powerful and high performance toolkit that can migrate data across various database formats, such as PostgreSQL, SQL Server, etc.
We can directly follow the 4 steps mentioned below to perform the switch.

1. In "Choose a Data Source" dialog, choose "PostgreSQL";

- Input the server name (default: localhost) and port (default: 5432).
- Input the username (default is "postgres") and its password.
- Press "Refresh Database" button to list all databases, then choose an existing database.

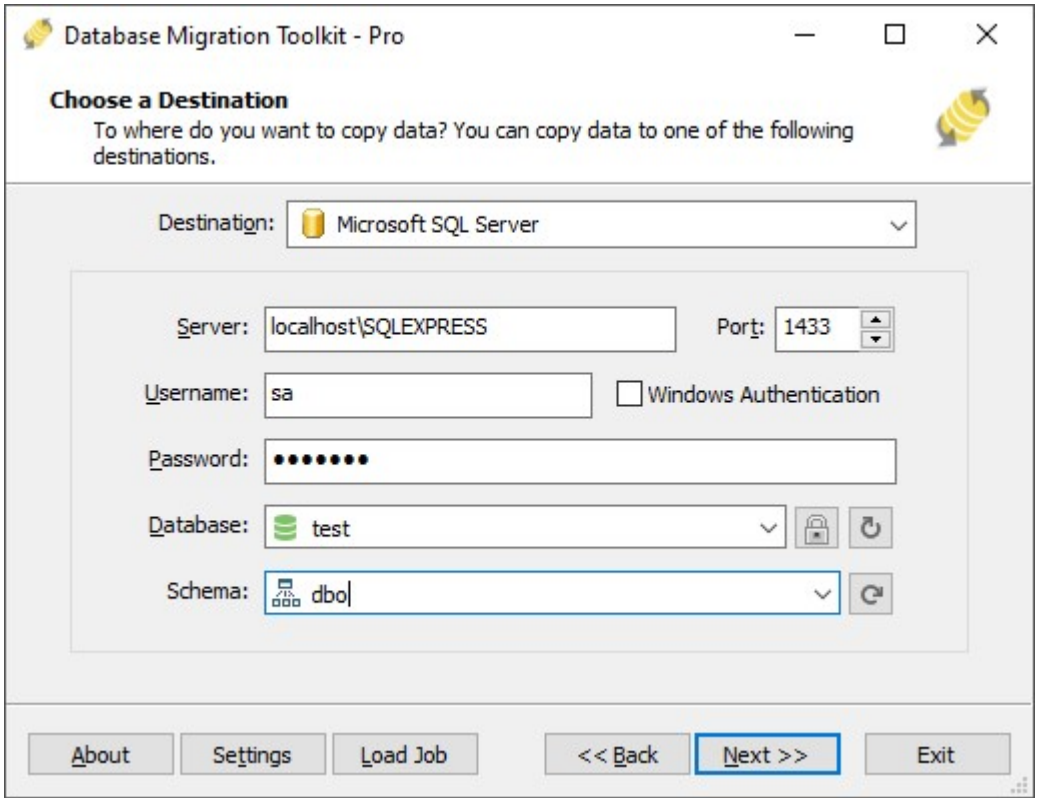
- Press "Refresh" button to list all schemas, then choose an existing schema (default is "public").



2. In "Choose a Destination" dialog, choose "Microsoft SQL Server";

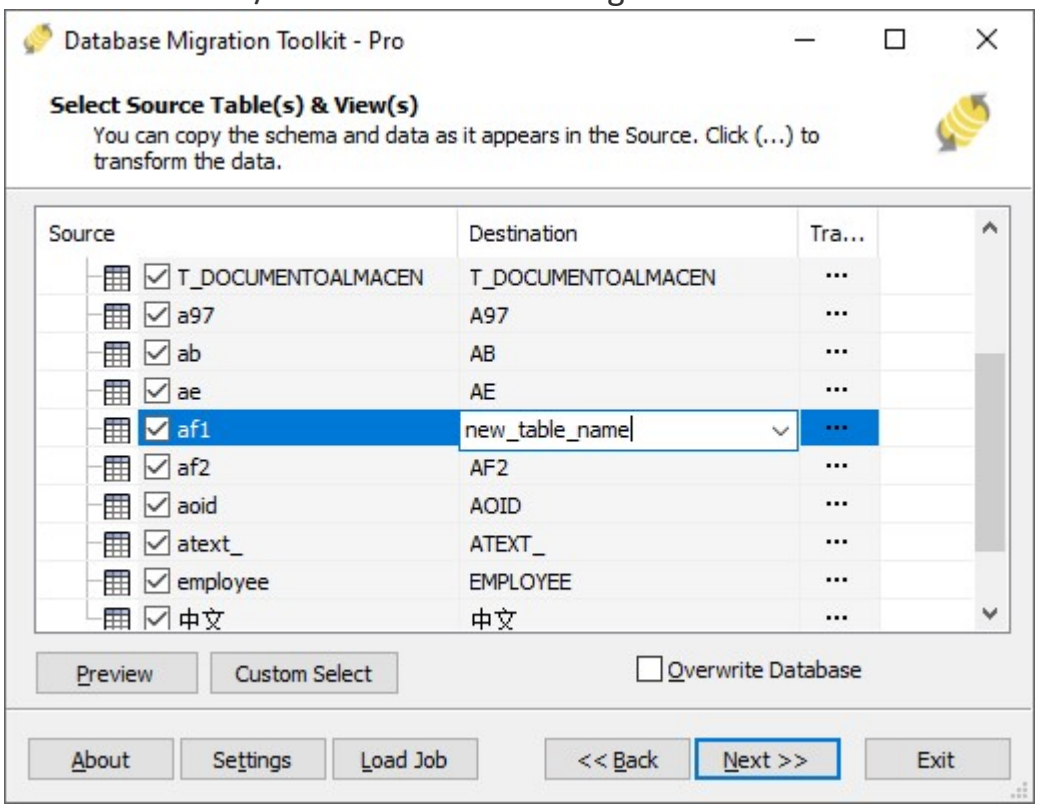
- Input the SQL Server host name (you can also add an instance name). e.g: localhost\sqlexpress.
- If using TCP/IP method then input the server port (default is 0 and using pipe method), in addition, you need to provide a username(e.g: sa) and password.
- Checking "Windows Authentication" checkbox if using Windows authentication.
- Press "Refresh Database" button to list all databases, yeither select an existing database or enter a new database name, this toolkit will automatically create the new database during the migration process.

- Press "Refresh Schema" button to list all schemas, you can select an existing schema or enter a new schema name, this toolkit will automatically create the new schema during the migration process, if you let it empty, the default is "dbo".

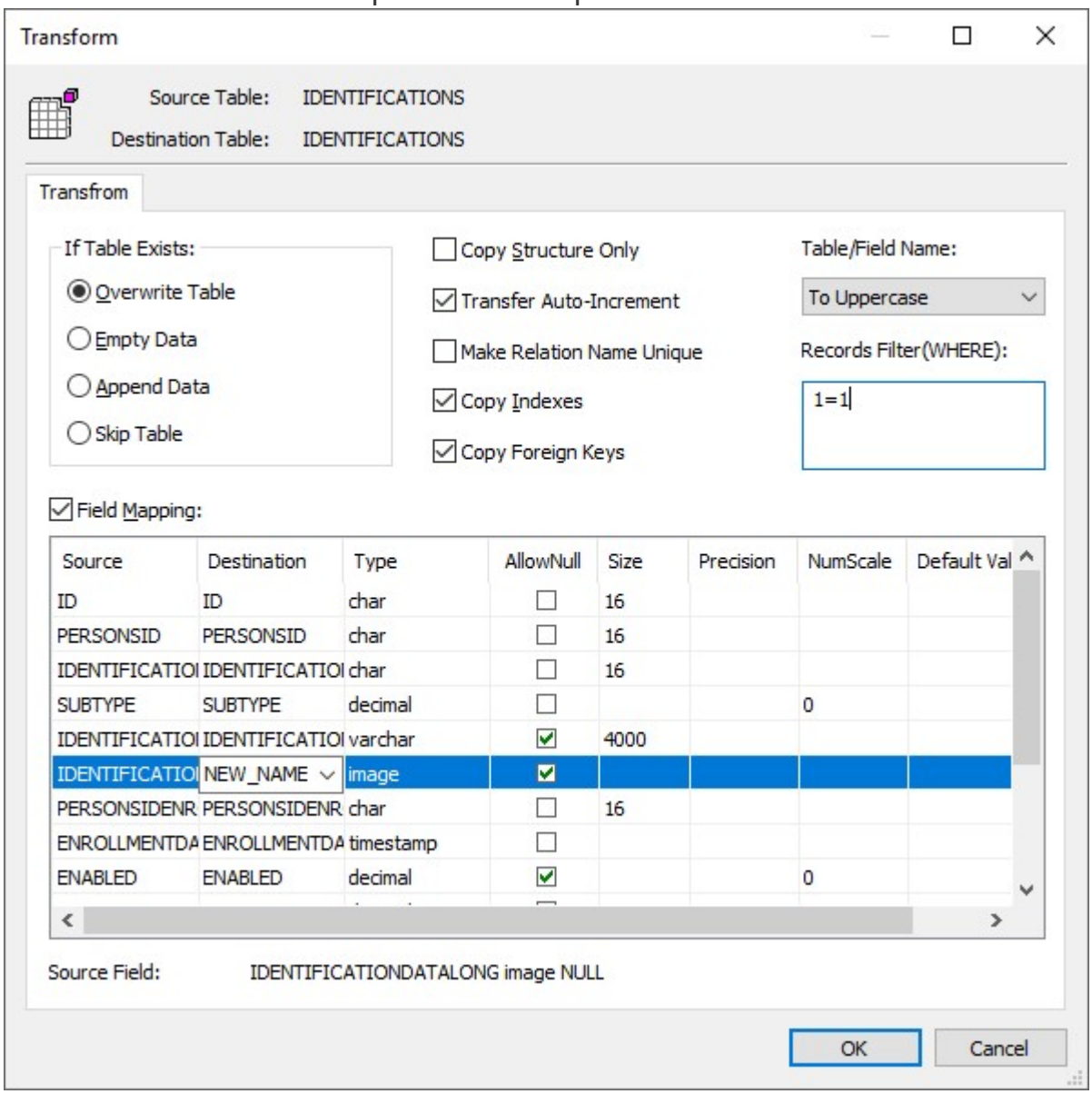


3. In "Select source Tables(s) & View(s)" dialog;

- Select the tables/views which will be migrated.



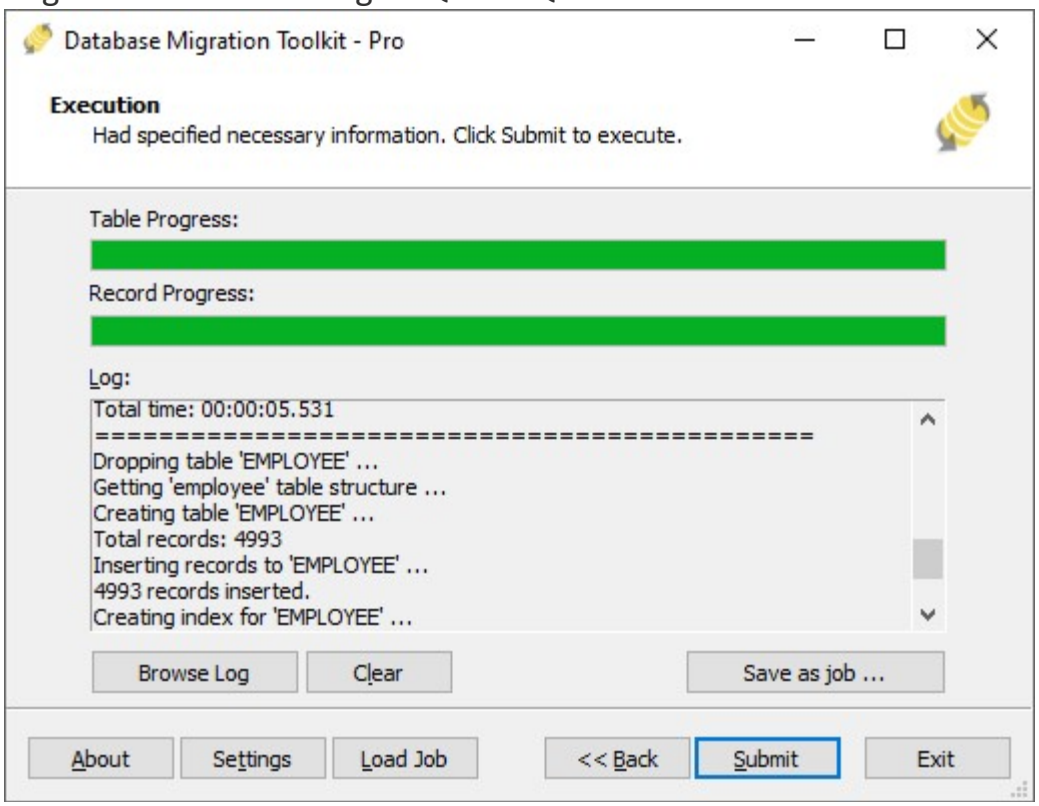
- Click "..." to set the table options or remap the table structure.



- You can set the data transfer method (Overwrite Table/Empty Data/Append Data/Skip Table) or filter the data before transferring.
- Choose "Field Mapping" option, you can redefine the fields in the destination table, e.g.: field name, data type, default value, comment and also.

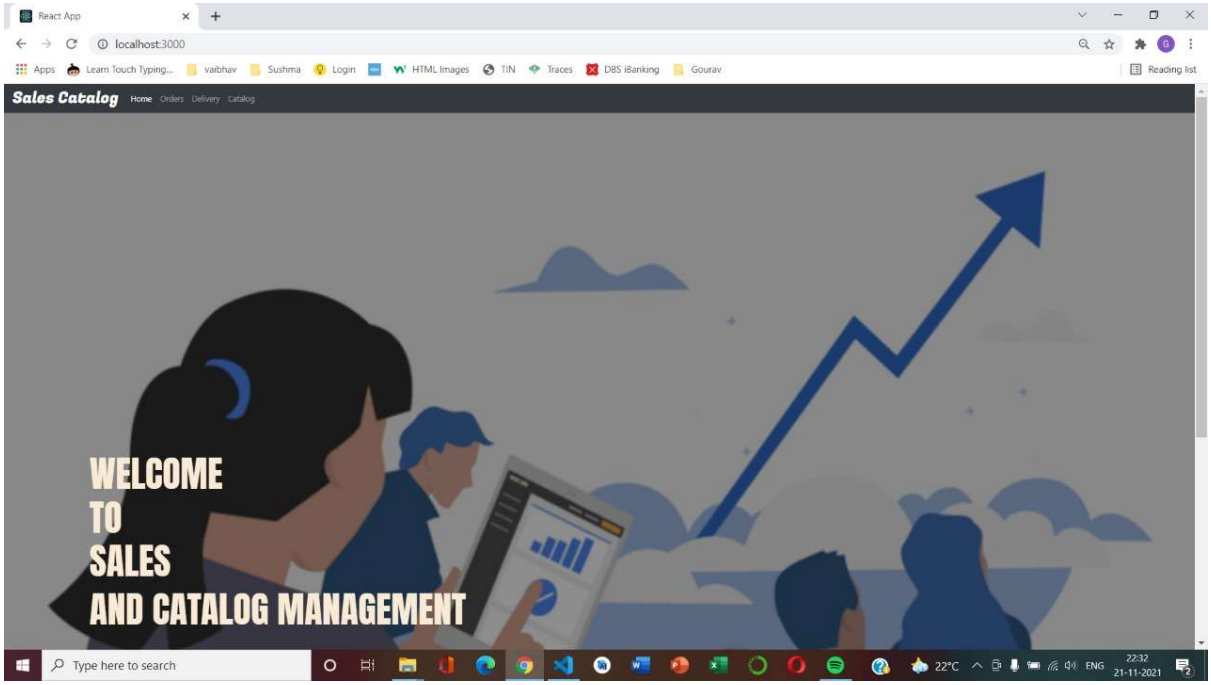
4. In "Execution" Dialog;

- Click "Submit" to begin the migration, this toolkit will help you quickly migrate data from PostgreSQL to SQL Server without intervention.

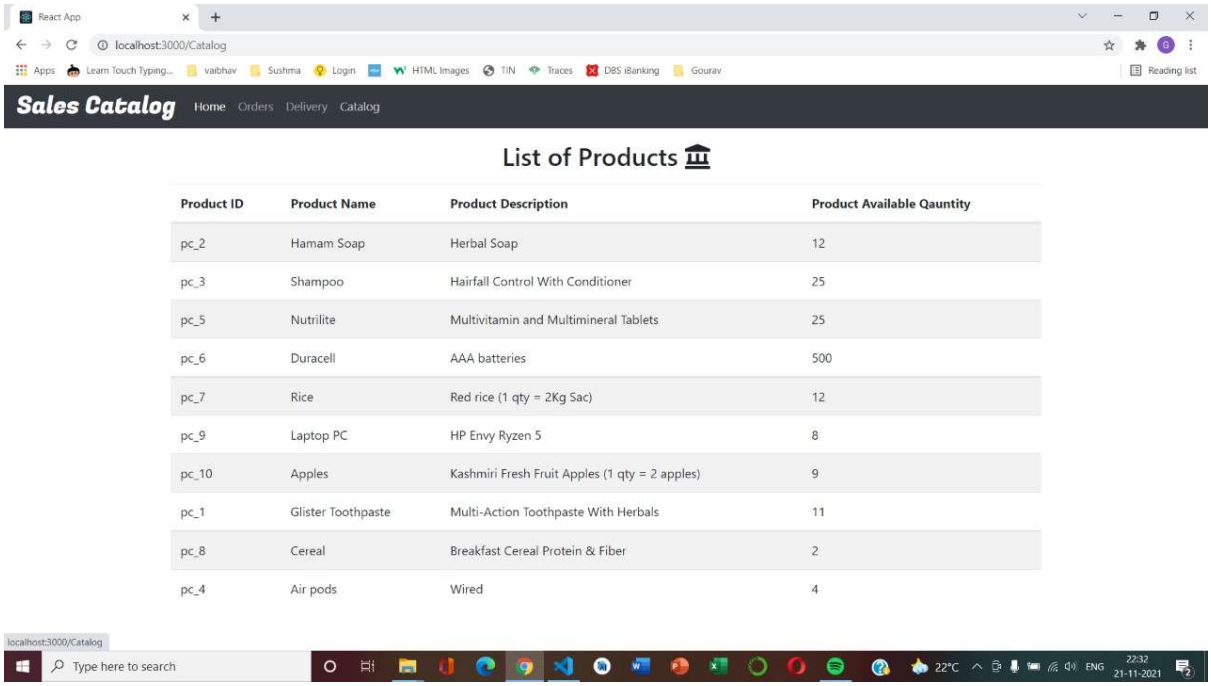


- Click "Browse Log" to visit the full migration log.
- Click "Save as job" to save the migration settings to a job file, so you can "Load Job" to quickly reload the migration job in the future or run the migration job via command-prompt. Run "esf-cmd --help" in command-prompt to get the full command parameters.

Front-End

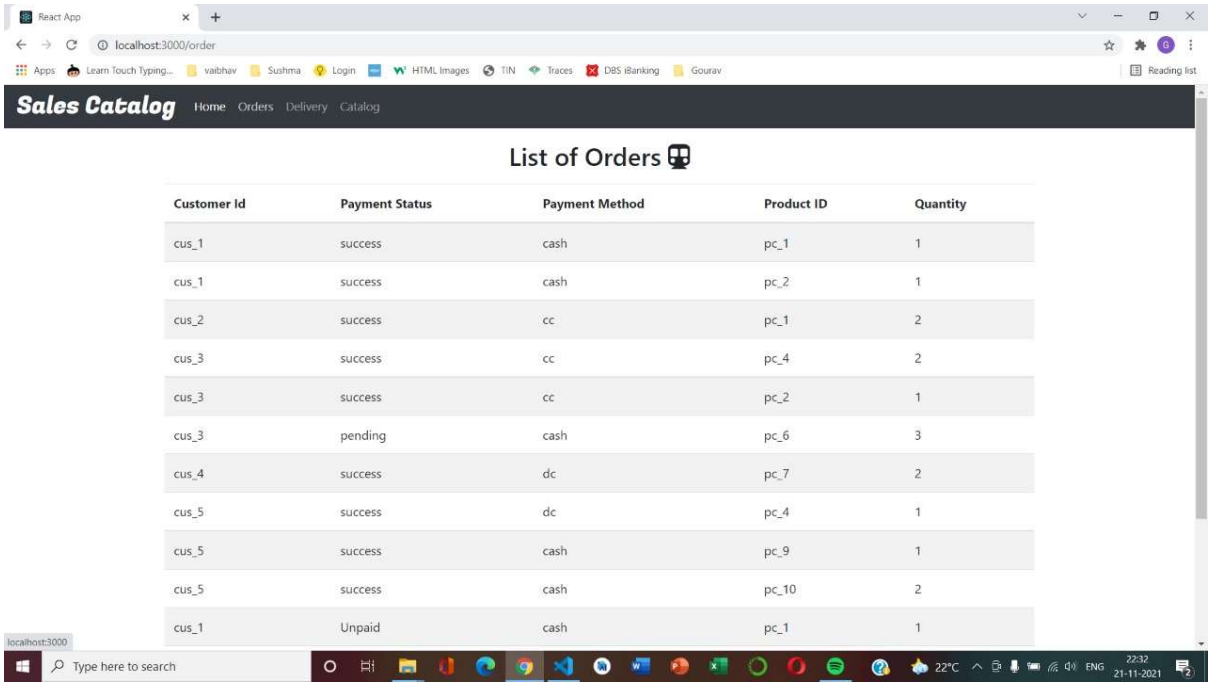


Retrieved Products:

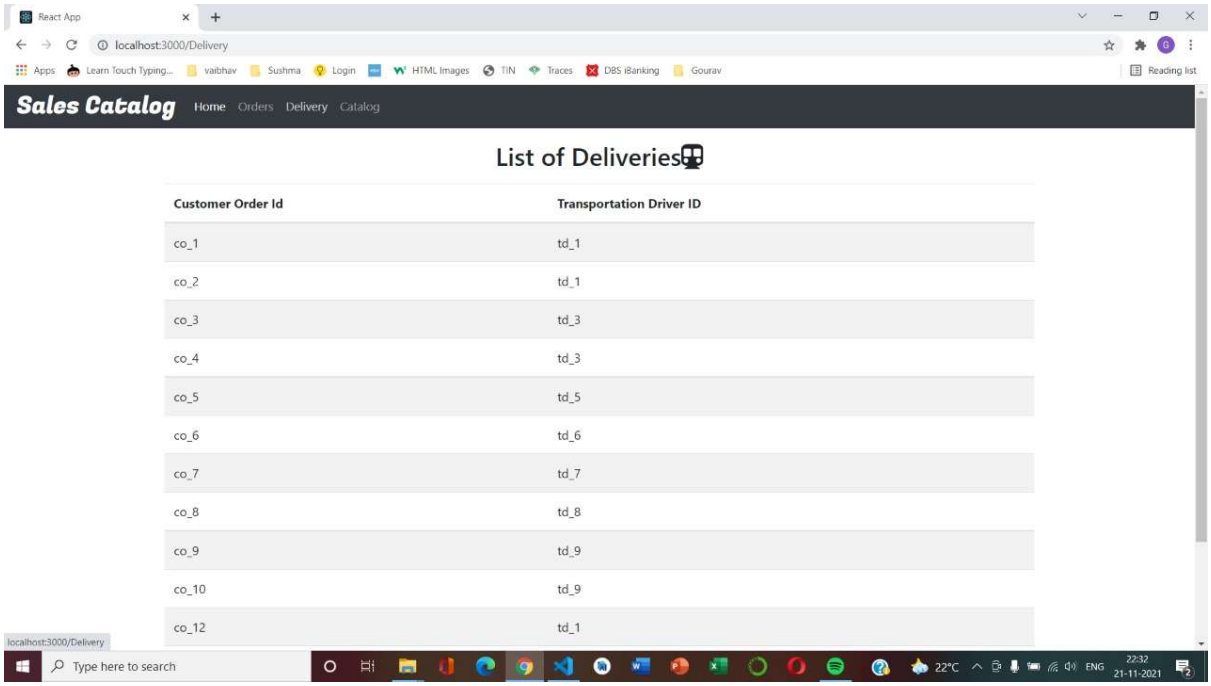


Product ID	Product Name	Product Description	Product Available Qauntity
pc_2	Hamam Soap	Herbal Soap	12
pc_3	Shampoo	Hairfall Control With Conditioner	25
pc_5	Nutrilite	Multivitamin and Multimineral Tablets	25
pc_6	Duracell	AAA batteries	500
pc_7	Rice	Red rice (1 qty = 2Kg Sac)	12
pc_9	Laptop PC	HP Envy Ryzen 5	8
pc_10	Apples	Kashmiri Fresh Fruit Apples (1 qty = 2 apples)	9
pc_1	Glister Toothpaste	Multi-Action Toothpaste With Herbals	11
pc_8	Cereal	Breakfast Cereal Protein & Fiber	2
pc_4	Air pods	Wired	4

Retrieved orders:



Retrieved Deliveries:



Language: *Javascript*

Contributions:

Bhargav Narayanan P : nested queries, 2 triggers, 2 functions with cursors, Materialized Views, Access Privileges, Front End (11 hours)

Gourav Aravinda : Simple queries, complex queries, nested queries, 2 triggers, 3 functions, Stored Procedures, Front End, Documentation (12 hours)

Ashish Uphadya: Front End (2 hours)