# SUPPLY CHAIN DATA ANALYSIS - B2B CONSUMERS

## Team composition and responsibilities:

- Anika Raisa Chowdhury (Responsible for Query Performance Evaluation and Data Cleaning)
- Anubhuti Hiwase (Responsible for ER Diagrams and Data Cleaning)
- Jahnavi Danda (Responsible for SQL Queries - Data Definition Language, Data Manipulation Language )
- L K Mounika Desu (Responsible for SQL Queries - Data Definition Language, Data Manipulation Language)
- Rutuja Magdum (Responsible for Exploratory Data Analysis and Visualization)

## Project Goal

Our goal is to establish a database management system for a B2B organization to manage its supply chains. They can use SQL to ask the database queries to draw daily reports and meaningful insights for strategic planning. The dataset consists of records of a B2B organization. It holds information on all sales and vendor and client information for the company.

This project will give us the opportunity to test the following concepts learned in class:

- ➢ Being able to examine data and functional needs for a business in a real-world dataset.
- ➢ The capacity to create an ER diagram as a conceptual model for real-world data
- ➢ Understanding of how to convert a conceptual model into a SQL schema using DDL
- ➢ Ability to use DML commands to alter data
- ➢ Writing effective SQL queries for practical aspects. Showcasing concepts like logical equivalence, aggregate functions, and Window functions.
- ➢ The capacity to implement database indexes
- ➢ Being able to assess how well a SQL query or index is performed
- ➢ Being able to present the visualizations.

## Attached Files:

CSV Files:

| FILE NAME | FILE DESCRIPTION |
|---|---|
| OrderList.csv | This file has a list of all orders for the company |
| PlantPorts.csv | This file describes links between the warehouses and shipping ports in the real world |
| ProductsPerPlant.csv | This file lists all supported warehouse-product combinations. |

| | |
|---|---|
| VmiCustomers.csv | This file lists all special cases, where the warehouse is only allowed to support specific customers |
| Warehouse.csv | This file Specifies the cost associated with storing the products in a given warehouse  measured in dollars per unit and number of orders per day |

Python Files:

| FILE NAME | FILE DESCRIPTION |
|---|---|
| CIS556_clean.py | Python file for data health check and cleaning. Here we checked the data to observe if it required any kind of preprocessing prior to being loaded into the database systems. |

SQL Files:

| FILE NAME | FILE DESCRIPTION |
|---|---|
| DDL SupplyChain.sql | DDL statements |
| DML SupplyChain.sql | DML statements |
| SQL Queries | SQL queries executed as detailed in Section |

Text File:

| FILE NAME | FILE DESCRIPTION |
|---|---|
| reproduce.docx | Steps to reproduce the project in another environment |

## Dataset

Supply Chain Dataset - The dataset consists of records of a B2B organization. It holds information on all sales and vendor and client information for the company.

**Dataset Link:** https://www.kaggle.com/datasets/laurinbrechter/supply-chain-data

**Description:** The dataset consists of records of a B2B organization. It holds information on all sales and vendor and client information for the company.

Below is an overview of the data in each table in the database system created:

**OrderList:** This table includes a list of all orders that need to be assigned to a route.

The fields in the table are:

| Field Name | Description |
|---|---|
| order_id | The unique number assigned to each purchase order |
| order_date | The date associated with each purchase order |
| origin_port | The source port number of the port from where the product has been shipped |
| carrier | ID assigned to the carrier |
| tpt | Describes the transaction time |
| service_level | The type of shipping service |
| ship_ahead_day_count | The number of days the product has been shipped earlier |
| ship_late_day_count | The number of days delay in the shipping of the product |
| customer | Customer ID assigned to the customer |
| product_id | The ID associated with each product being shipped |
| plant_code | Code assigned to each warehouse |
| destination_port | The destination port number of the port to where the product has been delivered. |

| | |
|---|---|
| unit_quantity | Quantity of each product sold to the customer |
| weight | Weight of the product |

**Warehouse**: Specifies the cost associated with storing the products in a given warehouse measured in dollars per unit and number of orders per day.

The fields in the table are:

| Field Name | Field Description |
|---|---|
| plant_code | ID assigned to the warehouse |
| cost_per_unit | The manufacturing and operational costs |
| daily_capacity | Daily capacity of the warehouse |

**ProductsPerPlant**: It lists all supported warehouse-product combinations.

The fields in the table are:

| Field Name | Field Description |
|---|---|
| plant_code | ID assigned to the warehouse |
| product_ID | ID assigned to the warehouse |

**VmiCustomers**: It lists all special cases, where the warehouse is only allowed to support specific customers.

The fields in the table are:

| Field Name | Field Description |
|---|---|
| plant_code | ID assigned to the warehouse |

| | |
|---|---|
| customers | ID assigned to each customer |

**PlantPorts**: Describes the allowed links between the warehouses and shipping ports in the real world.

The fields in the table are:

| Field Name | FILE DESCRIPTION |
|---|---|
| plant_code | ID assigned to the warehouse |
| Port | Port number assigned to the warehouse |

### Dataset Transformations

In order to make the data set better for comprehension, we merged the tables ‘warehouse costs’ and ‘warehouse capacities’ into one table ‘warehouse’ keeping all the columns from merging tables intact. Initially the dataset contained a table named ‘Frieghtrates’. We did not wish to use this table in connection with other tables for the queries, and thus we dropped the table from the dataset in the DDL stage. Later on , when we wanted to use this table individually for queries involving only the one table, we created this ‘Freightrates’ table again using CREATE TABLE, INSERTION and COPY statements.

All the code for cleaning the dataset is included in “clean.py”
All the PostgreSQL queries for the table transformations is included in this file.

## Conceptual Design

The ER diagram below provides a faithful, complete, and unambiguous specification of the data stored in the supply chain logistics problem table.

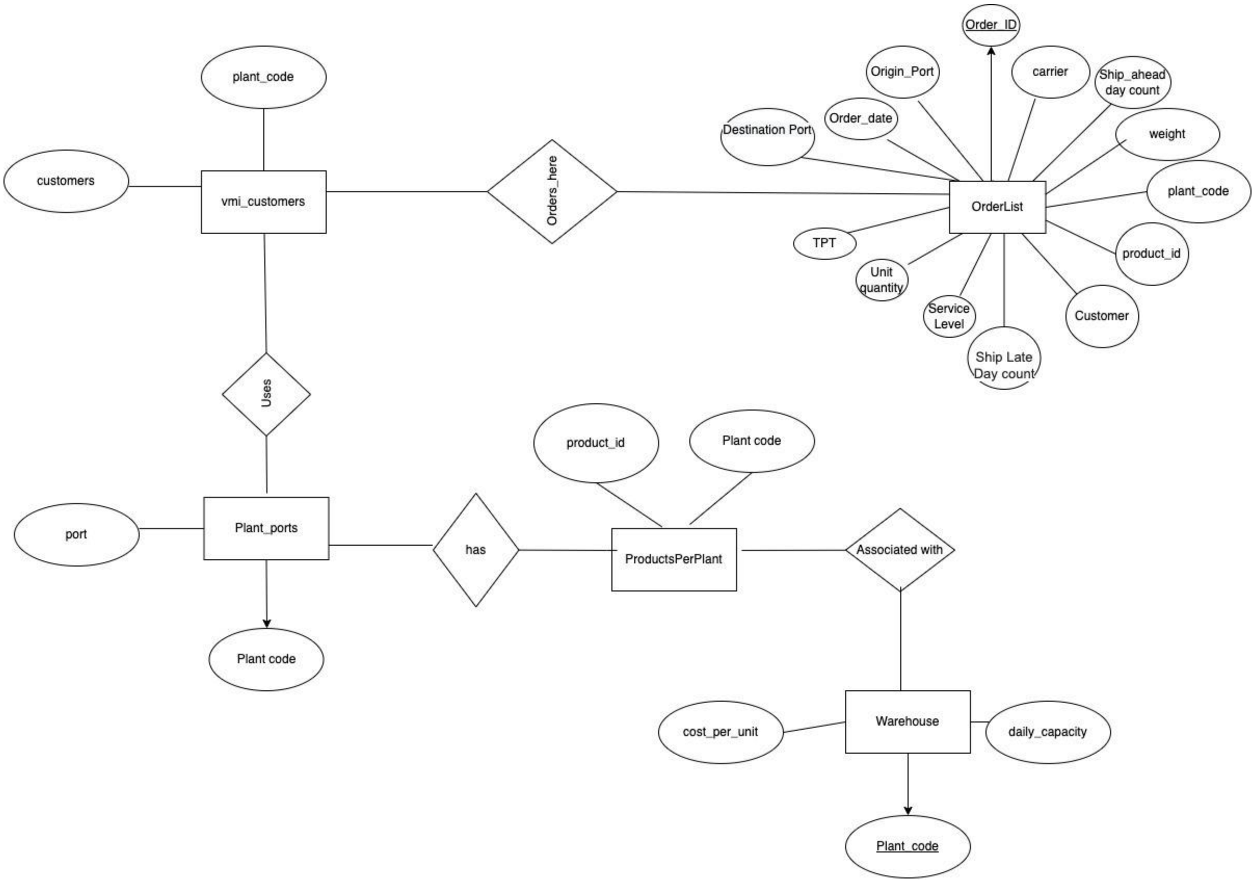


**Figure:** Entity Relationship Diagram for the implemented DBMS. All the tables have many-to- many relationships

## Database Schema

We converted the above conceptual design into the following SQL schema:

| Table Name | Schema |
|---|---|
| vmi_customers | {plantcode, customers} |
| Plant_ports | {plantcode, port} |
| Productsperplant | {plantcode,productid} |
| OrderList | {order_id,order_date,origin_port ,carrier, tpt, |

| | |
|---|---|
| | service_level, ship_ahead_day_count,<br>ship_late_day_count, customer,<br>product_id , plant_code),<br>destination_port,<br>unit_quantity,<br>weight} |
| Warehouse | {plant_code, cost_per_unit,daily_capacity } |

### DML Statements

```
DROP TABLE IF EXISTS plantports;
DROP TABLE IF EXISTS vmicustomers;
DROP TABLE IF EXISTS productsperplant ;
DROP TABLE IF EXISTS warehouse;
DROP TABLE IF EXISTS freightrates;
DROP TABLE IF EXISTS orderlist ;

CREATE TABLE plantports
(
plant_code varchar(50) REFERENCES warehouse(plant_code),

port varchar(50) NOT NULL
);

CREATE TABLE vmicustomers
(
plant_code varchar(50) REFERENCES warehouse(plant_code),
customers varchar(50) NOT NULL
);

CREATE TABLE productsperplant
(
plant_code varchar(50) REFERENCES warehouse(plant_code),
product_id int NOT NULL
);

CREATE TABLE warehouse
(
plant_code varchar(50) PRIMARY KEY,
cost_per_unit decimal NOT NULL,
daily_capacity int NOT NULL
);
```

```
CREATE TABLE freightrates
(
carrier varchar(50) NOT NULL,
orig_port_cd varchar(50),
dest_port_cd varchar(50),
minm_wgh_qty decimal,
max_wgh_qty decimal,
svc_cd varchar(50),
minimum_cost decimal,
rate decimal,
mode_dsc varchar(50),
tpt_day_cnt int,

carrier_type varchar(50) NOT NULL
);

CREATE TABLE orderlist
(
order_id decimal PRIMARY KEY,
order_date DATE ,
origin_port varchar(50),
carrier varchar(50),
tpt int ,
service_level varchar(50),
ship_ahead_day_count int,
ship_late_day_count int,
customer varchar(50),
product_id INT,
plant_code varchar(50) references warehouse(plant_code),
destination_port varchar(50),
unit_quantity int,
weight decimal
);
```

### DML Statements

We populated our schema with the following DML statements:

```
COPY plantports FROM 'C:\Program Files\PostgreSQL\15\project\project556\PlantPorts.csv' delimiter ',' CSV Header;
COPY vmicustomers FROM 'C:\Program Files\PostgreSQL\15\project\project556\VmiCustomers.csv' delimiter ',' CSV Header;
COPY productsperplant FROM 'C:\Program
Files\PostgreSQL\15\project\project556\ProductsPerPlant.csv' delimiter ',' CSV Header;
COPY warehouse FROM 'C:\Program Files\PostgreSQL\15\project\project556\Warehouse.csv'
delimiter ',' CSV Header;
COPY freightrates FROM 'C:\Program Files\PostgreSQL\15\project\project556\FreightRates.csv'
DELIMITER ',' CSV Header;
COPY orderlist FROM 'C:\Program Files\PostgreSQL\15\project\project556\OrderList.csv' delimiter ','
CSV Header;

ALTER TABLE warehouse
ADD CONSTRAINT wh_pkey PRIMARY KEY (wh);

ALTER TABLE warehouse
RENAME COLUMN wh TO plant_code;

ALTER TABLE orderlist
ADD CONSTRAINT order_id_pkey PRIMARY KEY (order_id);
```

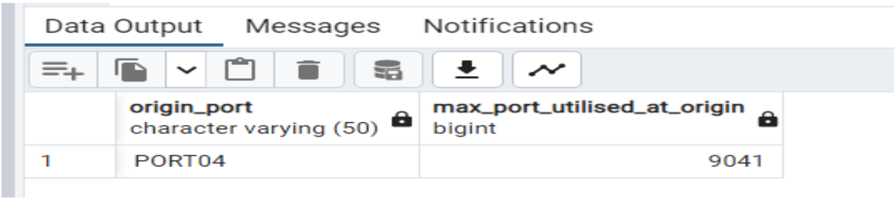
## Queries

**1)Most used port of origin?**

Approach 1 –

```
SELECT Origin_Port, COUNT(Origin_Port) AS max_port_utilised_at_origin

FROM orderlist

GROUP BY Origin_Port

ORDER BY max_port_utilised_at_origin DESC LIMIT 1;
```

**Output:**

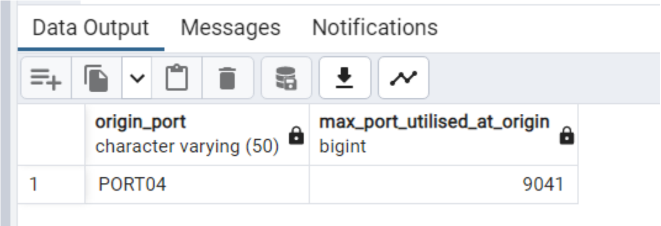


| | origin_port<br>character varying (50) | max_port_utilised_at_origin<br>bigint |
|---|---|---|
| 1 | PORT04 | 9041 |

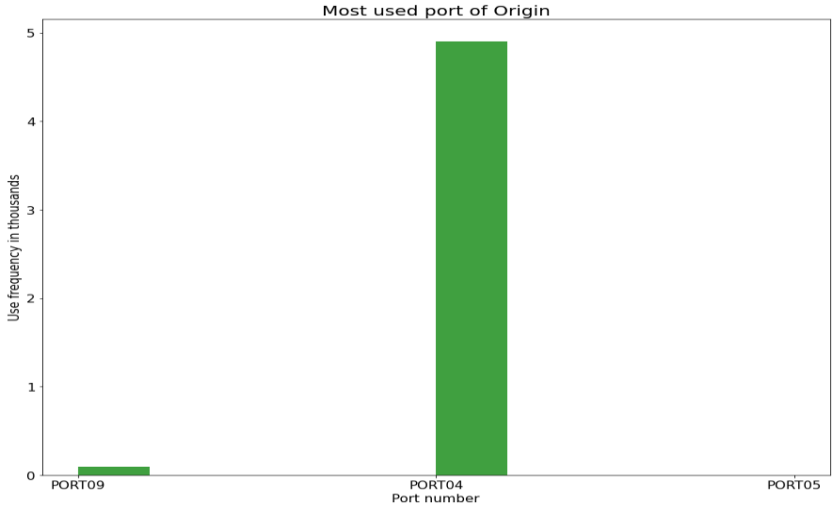
Approach 2 -

```
SELECT Origin_Port, COUNT(Origin_Port)

over (partition by Origin_Port) AS max_port_utilised_at_origin

FROM orderlist limit 1 ;
```

**Output:**



| | origin_port<br>character varying (50) | max_port_utilised_at_origin<br>bigint |
|---|---|---|
| 1 | PORT04 | 9041 |



**2) Most used port of destination?**

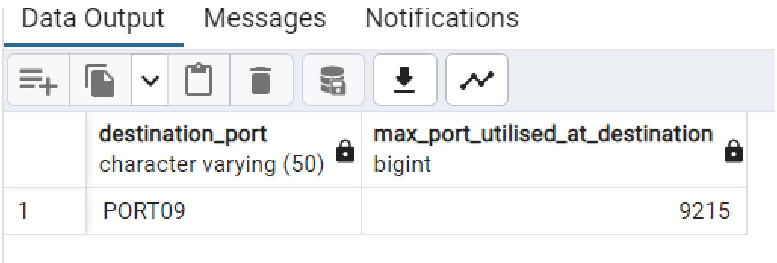
SELECT Destination_Port, COUNT( Destination_Port) AS max_port_utilised_at_destination

FROM orderlist

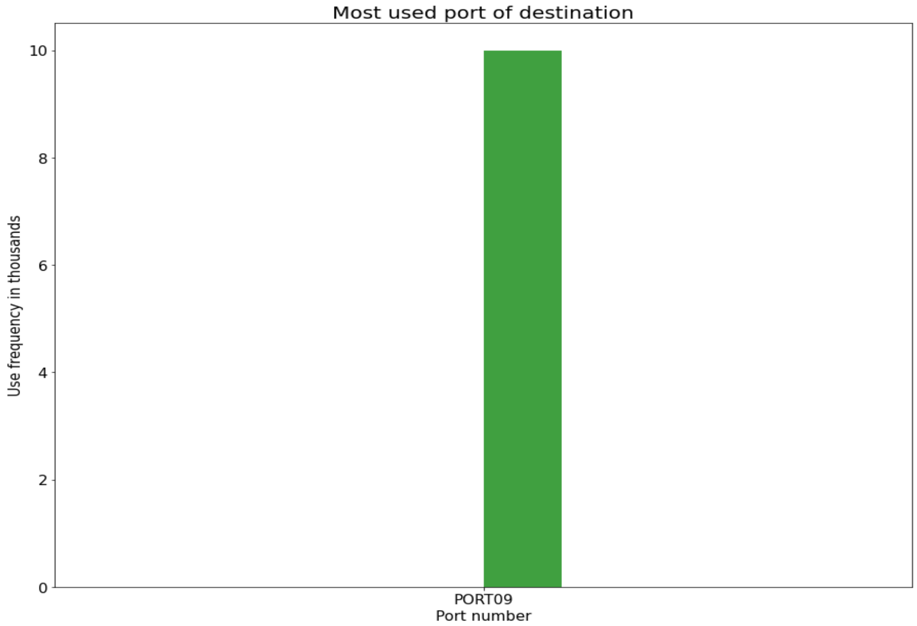
GROUP BY  Destination_Port

ORDER BY max_port_utilised_at_destination DESC LIMIT 1;

**Output:**



| | destination_port<br>character varying (50) | max_port_utilised_at_destination<br>bigint |
|---|---|---|
| 1 | PORT09 | 9215 |



**3)Quantity sold for each product ()**

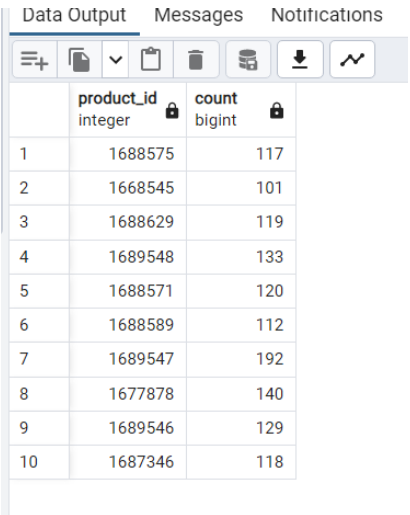
SELECT product_id, count(product_id)

from orderlist

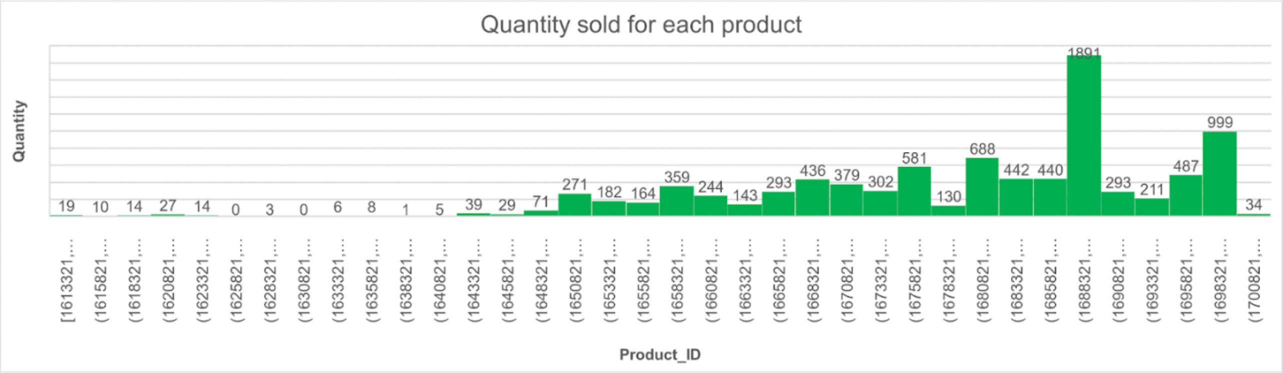
group by product_id

having count(product_id) >100;

**Output:**



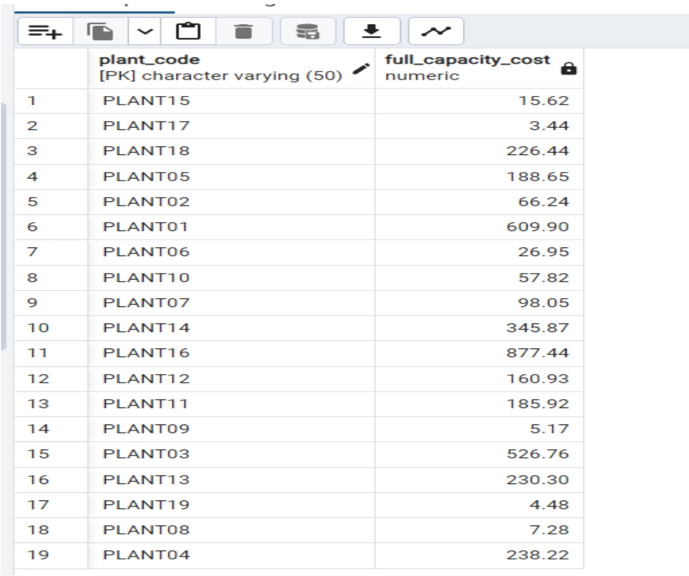
| | product_id<br>integer | count<br>bigint |
|---|---|---|
| 1 | 1688575 | 117 |
| 2 | 1668545 | 101 |
| 3 | 1688629 | 119 |
| 4 | 1689548 | 133 |
| 5 | 1688571 | 120 |
| 6 | 1688589 | 112 |
| 7 | 1689547 | 192 |
| 8 | 1677878 | 140 |
| 9 | 1689546 | 129 |
| 10 | 1687346 | 118 |



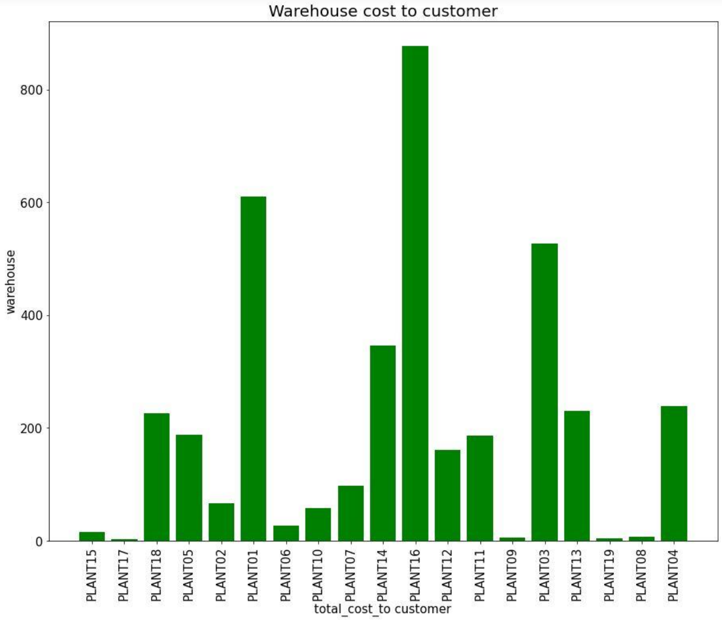
**4) Operating at full capacity, how much does using each warehouse cost to the customer?**

```
Select plant_code , (Cost_Per_Unit * daily_capacity) as full_capacity_cost
from warehouse
```

**Output:**



| | plant_code<br>[PK] character varying (50) | full_capacity_cost<br>numeric |
|---|---|---|
| 1 | PLANT15 | 15.62 |
| 2 | PLANT17 | 3.44 |
| 3 | PLANT18 | 226.44 |
| 4 | PLANT05 | 188.65 |
| 5 | PLANT02 | 66.24 |
| 6 | PLANT01 | 609.90 |
| 7 | PLANT06 | 26.95 |
| 8 | PLANT10 | 57.82 |
| 9 | PLANT07 | 98.05 |
| 10 | PLANT14 | 345.87 |
| 11 | PLANT16 | 877.44 |
| 12 | PLANT12 | 160.93 |
| 13 | PLANT11 | 185.92 |
| 14 | PLANT09 | 5.17 |
| 15 | PLANT03 | 526.76 |
| 16 | PLANT13 | 230.30 |
| 17 | PLANT19 | 4.48 |
| 18 | PLANT08 | 7.28 |
| 19 | PLANT04 | 238.22 |



**5)Most bought products?**

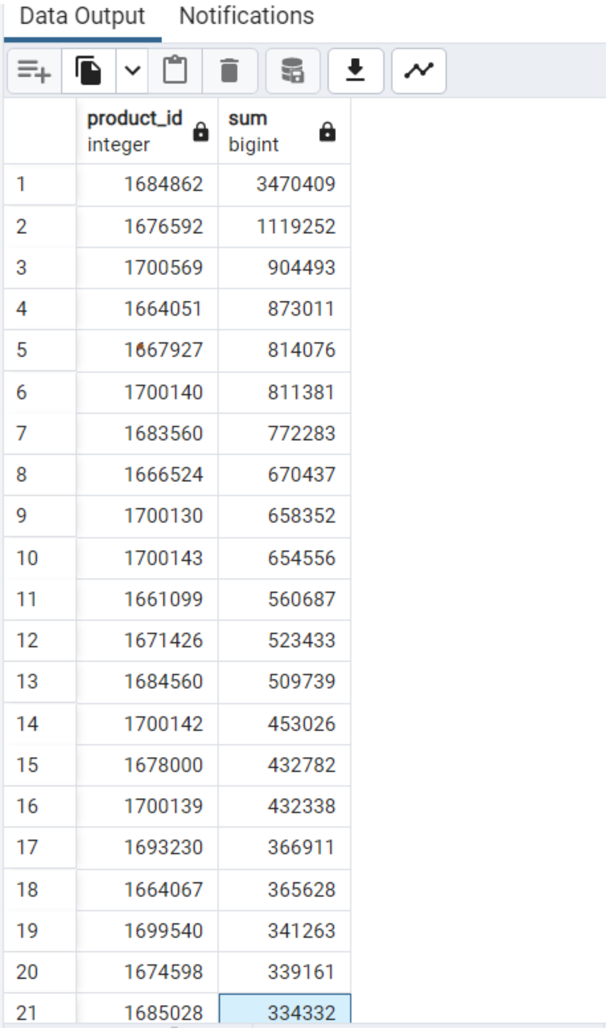
select product_id, sum(Unit_quantity)

from orderlist

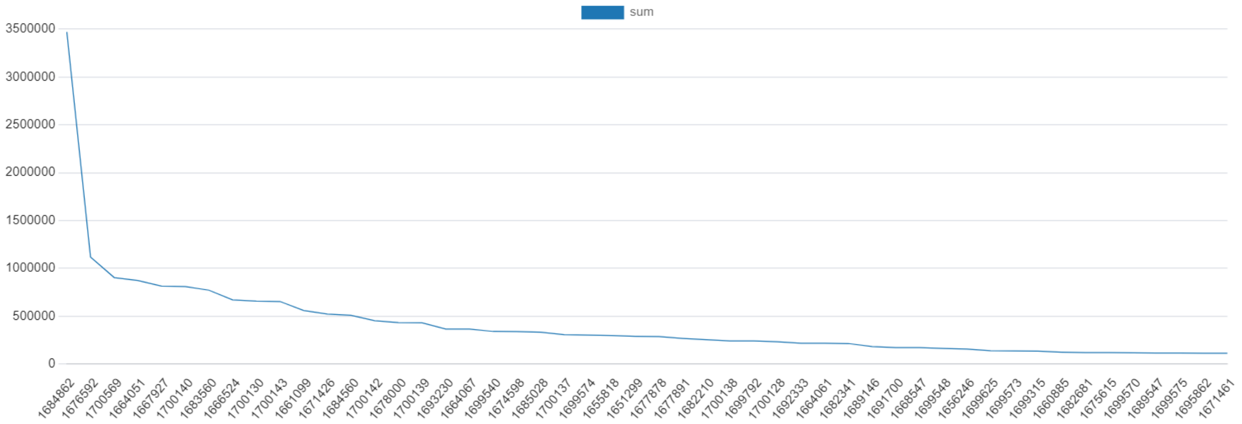
group by product_id

Order by sum(Unit_quantity) desc limit 50;

**Output:**



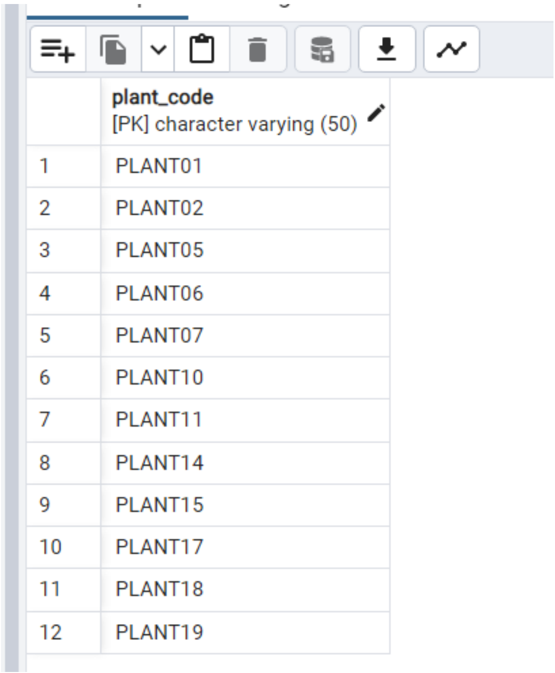
| | product_id<br>integer | sum<br>bigint |
|---|---|---|
| 1 | 1684862 | 3470409 |
| 2 | 1676592 | 1119252 |
| 3 | 1700569 | 904493 |
| 4 | 1664051 | 873011 |
| 5 | 1667927 | 814076 |
| 6 | 1700140 | 811381 |
| 7 | 1683560 | 772283 |
| 8 | 1666524 | 670437 |
| 9 | 1700130 | 658352 |
| 10 | 1700143 | 654556 |
| 11 | 1661099 | 560687 |
| 12 | 1671426 | 523433 |
| 13 | 1684560 | 509739 |
| 14 | 1700142 | 453026 |
| 15 | 1678000 | 432782 |
| 16 | 1700139 | 432338 |
| 17 | 1693230 | 366911 |
| 18 | 1664067 | 365628 |
| 19 | 1699540 | 341263 |
| 20 | 1674598 | 339161 |
| 21 | 1685028 | 334332 |



### 6) What plants are underutilized?

```
select w.Plant_Code
from Warehouse w
where not exists(
select o.Plant_Code
from orderlist o
where o.Plant_Code=w.Plant_Code )
order by w.Plant_Code ;
```

**Output:**

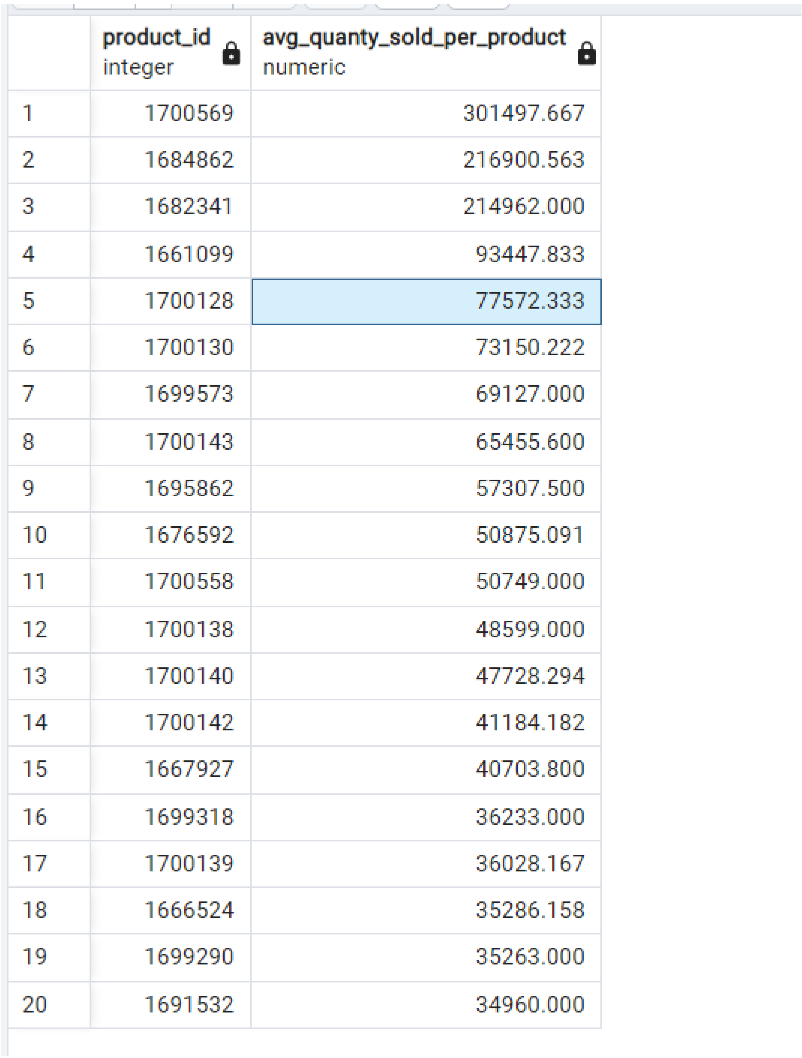


| | plant_code<br>[PK] character varying (50) |
|---|---|
| 1 | PLANT01 |
| 2 | PLANT02 |
| 3 | PLANT05 |
| 4 | PLANT06 |
| 5 | PLANT07 |
| 6 | PLANT10 |
| 7 | PLANT11 |
| 8 | PLANT14 |
| 9 | PLANT15 |
| 10 | PLANT17 |
| 11 | PLANT18 |
| 12 | PLANT19 |

### 7) Top 20 records of Average quantity sold for each product?

```
select product_id, round(avg(unit_quantity),3) as Avg_quanty_sold_per_product
from orderlist
group by product_id
order by Avg_quanty_sold_per_product desc limit 20;
```

**Output:**



| | product_id<br>integer | avg_quanty_sold_per_product<br>numeric |
|---|---|---|
| 1 | 1700569 | 301497.667 |
| 2 | 1684862 | 216900.563 |
| 3 | 1682341 | 214962.000 |
| 4 | 1661099 | 93447.833 |
| 5 | 1700128 | 77572.333 |
| 6 | 1700130 | 73150.222 |
| 7 | 1699573 | 69127.000 |
| 8 | 1700143 | 65455.600 |
| 9 | 1695862 | 57307.500 |
| 10 | 1676592 | 50875.091 |
| 11 | 1700558 | 50749.000 |
| 12 | 1700138 | 48599.000 |
| 13 | 1700140 | 47728.294 |
| 14 | 1700142 | 41184.182 |
| 15 | 1667927 | 40703.800 |
| 16 | 1699318 | 36233.000 |
| 17 | 1700139 | 36028.167 |
| 18 | 1666524 | 35286.158 |
| 19 | 1699290 | 35263.000 |
| 20 | 1691532 | 34960.000 |

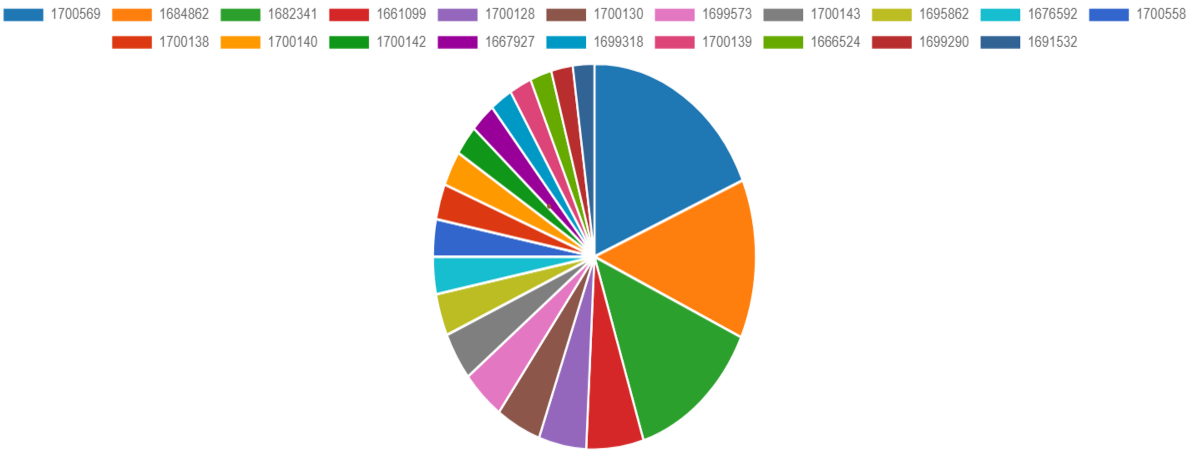
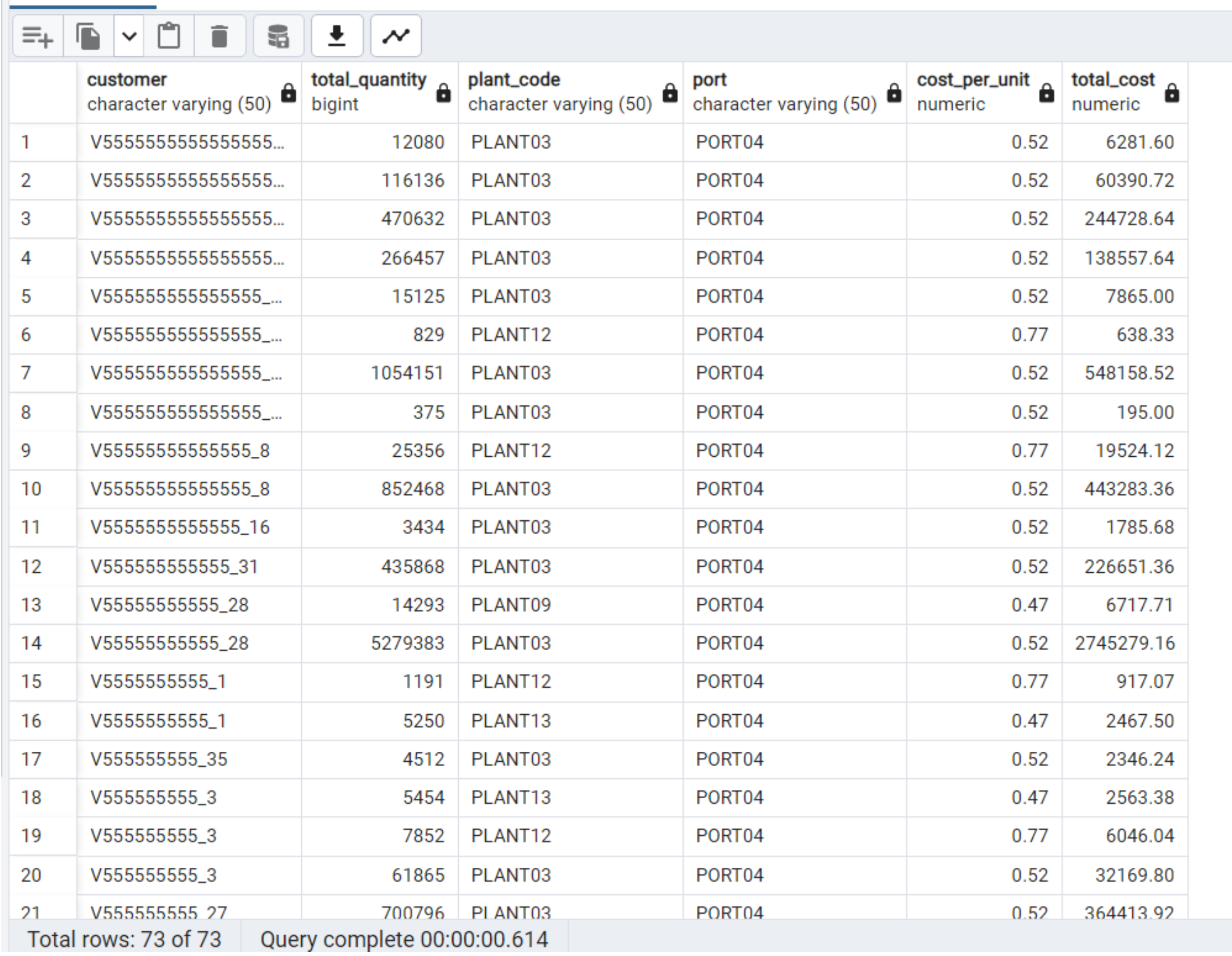


Fig. Distribution of Average quantity sold for 20 different products

### 8) Overall price spent by each costumer for every plant

```
SELECT      distinct(a.customer),
            sum(a.unit_quantity) as total_quantity,
            b.plant_code, b.port,
            c.cost_per_unit,
            sum((a.unit_quantity*c.cost_per_unit)) as total_cost
FROM                orderlist as a
Inner Join  plantports as b
ON          a.plant_code = b.plant_code
Inner Join  warehouse as c
ON          a.plant_code = c.plant_code
GROUP BY    a.customer,
            b.plant_code,
            b.port,
            c.cost_per_unit
ORDER BY    a.customer desc;
```

**Output:**



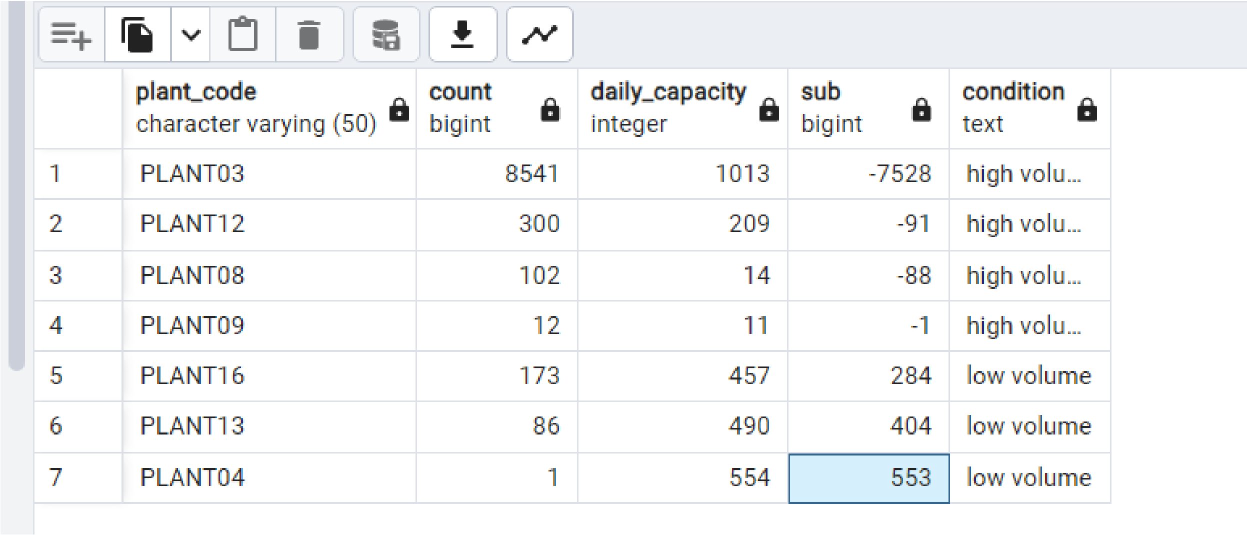
| | customer<br>character varying (50) | total_quantity<br>bigint | plant_code<br>character varying (50) | port<br>character varying (50) | cost_per_unit<br>numeric | total_cost<br>numeric |
|---|---|---|---|---|---|---|
| 1 | V5555555555555555... | 12080 | PLANT03 | PORT04 | 0.52 | 6281.60 |
| 2 | V5555555555555555... | 116136 | PLANT03 | PORT04 | 0.52 | 60390.72 |
| 3 | V5555555555555555... | 470632 | PLANT03 | PORT04 | 0.52 | 244728.64 |
| 4 | V5555555555555555... | 266457 | PLANT03 | PORT04 | 0.52 | 138557.64 |
| 5 | V555555555555555_... | 15125 | PLANT03 | PORT04 | 0.52 | 7865.00 |
| 6 | V555555555555555_... | 829 | PLANT12 | PORT04 | 0.77 | 638.33 |
| 7 | V555555555555555_... | 1054151 | PLANT03 | PORT04 | 0.52 | 548158.52 |
| 8 | V555555555555555_... | 375 | PLANT03 | PORT04 | 0.52 | 195.00 |
| 9 | V55555555555555_8 | 25356 | PLANT12 | PORT04 | 0.77 | 19524.12 |
| 10 | V55555555555555_8 | 852468 | PLANT03 | PORT04 | 0.52 | 443283.36 |
| 11 | V5555555555555_16 | 3434 | PLANT03 | PORT04 | 0.52 | 1785.68 |
| 12 | V555555555555_31 | 435868 | PLANT03 | PORT04 | 0.52 | 226651.36 |
| 13 | V55555555555_28 | 14293 | PLANT09 | PORT04 | 0.47 | 6717.71 |
| 14 | V55555555555_28 | 5279383 | PLANT03 | PORT04 | 0.52 | 2745279.16 |
| 15 | V5555555555_1 | 1191 | PLANT12 | PORT04 | 0.77 | 917.07 |
| 16 | V5555555555_1 | 5250 | PLANT13 | PORT04 | 0.47 | 2467.50 |
| 17 | V555555555_35 | 4512 | PLANT03 | PORT04 | 0.52 | 2346.24 |
| 18 | V555555555_3 | 5454 | PLANT13 | PORT04 | 0.47 | 2563.38 |
| 19 | V555555555_3 | 7852 | PLANT12 | PORT04 | 0.77 | 6046.04 |
| 20 | V555555555_3 | 61865 | PLANT03 | PORT04 | 0.52 | 32169.80 |
| 21 | V555555555_27 | 700796 | PLANT03 | PORT04 | 0.52 | 364413.92 |

Total rows: 73 of 73    Query complete 00:00:00.614

### 9) High and Low Volumes of Warehouses

```
CREATE VIEW volume AS
SELECT          o.plant_code,
                  count(o.order_id),
                 w.daily_capacity,
                  (w.daily_capacity - count(o.order_id)) as sub
 FROM         orderlist as o
 INNER JOIN  warehouse as w
 ON             o.plant_code = w.plant_code
 GROUP BY  o.plant_code,w.daily_capacity;


 SELECT    *,
        Case
              When sub > 0 Then 'low volume'
              Else  'high volume'
              END condition
 FROM  volume
 ORDER BY sub;
```

**Output:**

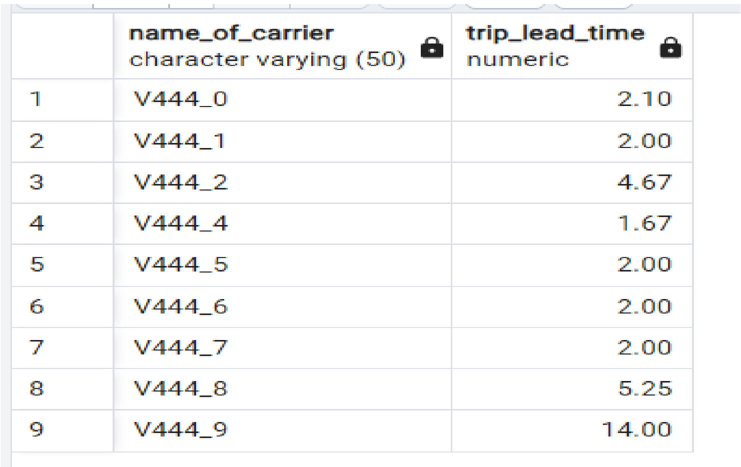


| | plant_code<br>character varying (50) | count<br>bigint | daily_capacity<br>integer | sub<br>bigint | condition<br>text |
|---|---|---|---|---|---|
| 1 | PLANT03 | 8541 | 1013 | -7528 | high volu… |
| 2 | PLANT12 | 300 | 209 | -91 | high volu… |
| 3 | PLANT08 | 102 | 14 | -88 | high volu… |
| 4 | PLANT09 | 12 | 11 | -1 | high volu… |
| 5 | PLANT16 | 173 | 457 | 284 | low volume |
| 6 | PLANT13 | 86 | 490 | 404 | low volume |
| 7 | PLANT04 | 1 | 554 | 553 | low volume |

**10) Discover the average number of transport trips per carrier.**

```
SELECT      f.carrier as Name_of_carrier,
            round(avg(tpt_day_cnt),2) as trip_lead_time
FROM                   freightrates f
GROUP BY    f.carrier
ORDER BY    f.carrier;
```

**Output:**



| | name_of_carrier<br>character varying (50) 🔒 | trip_lead_time<br>numeric 🔒 |
|---|---|---|
| 1 | V444_0 | 2.10 |
| 2 | V444_1 | 2.00 |
| 3 | V444_2 | 4.67 |
| 4 | V444_4 | 1.67 |
| 5 | V444_5 | 2.00 |
| 6 | V444_6 | 2.00 |
| 7 | V444_7 | 2.00 |
| 8 | V444_8 | 5.25 |
| 9 | V444_9 | 14.00 |

**INDEXING**

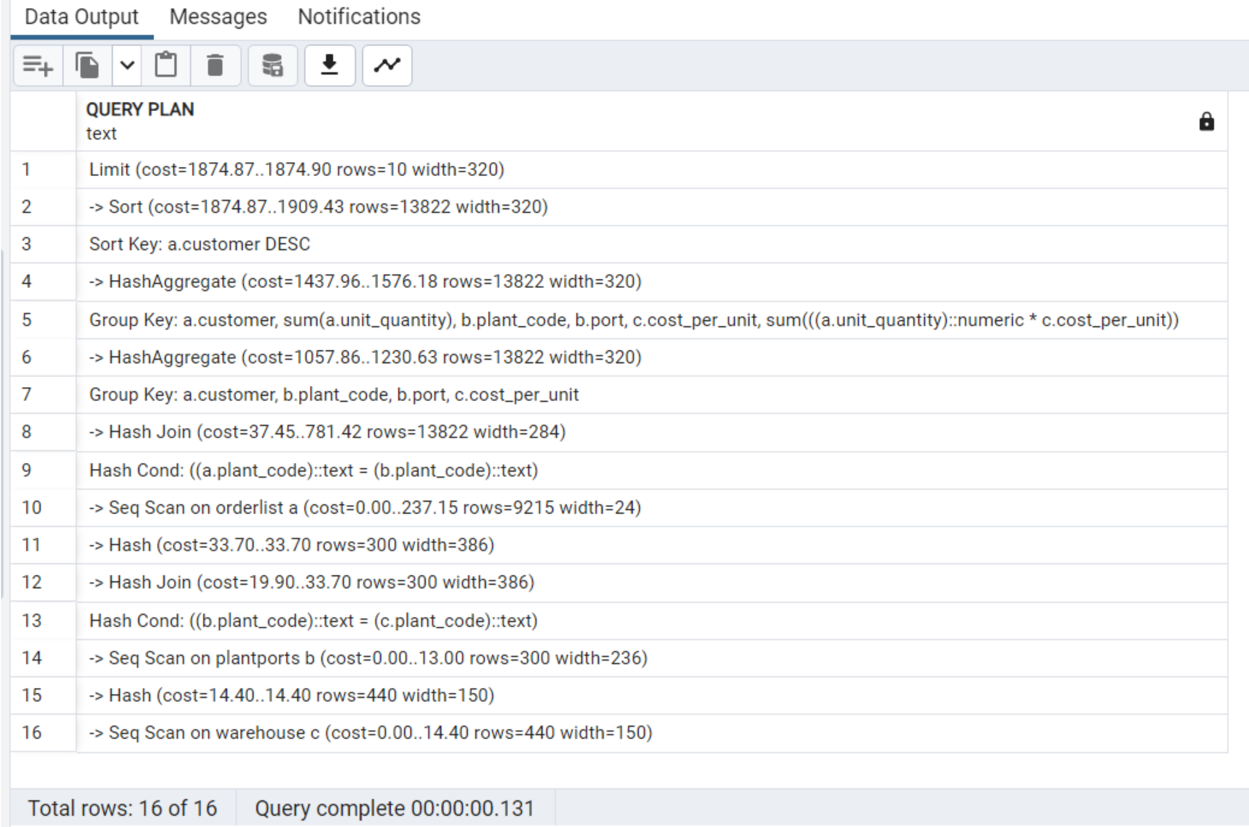
```
create Index plant_index on orderlist(plant_code);
```

## QUERY OPTIMIZATION

**Used on query 8)**

Before Creating Index for Orderlist table on plant_index, we get query analysis by using explain

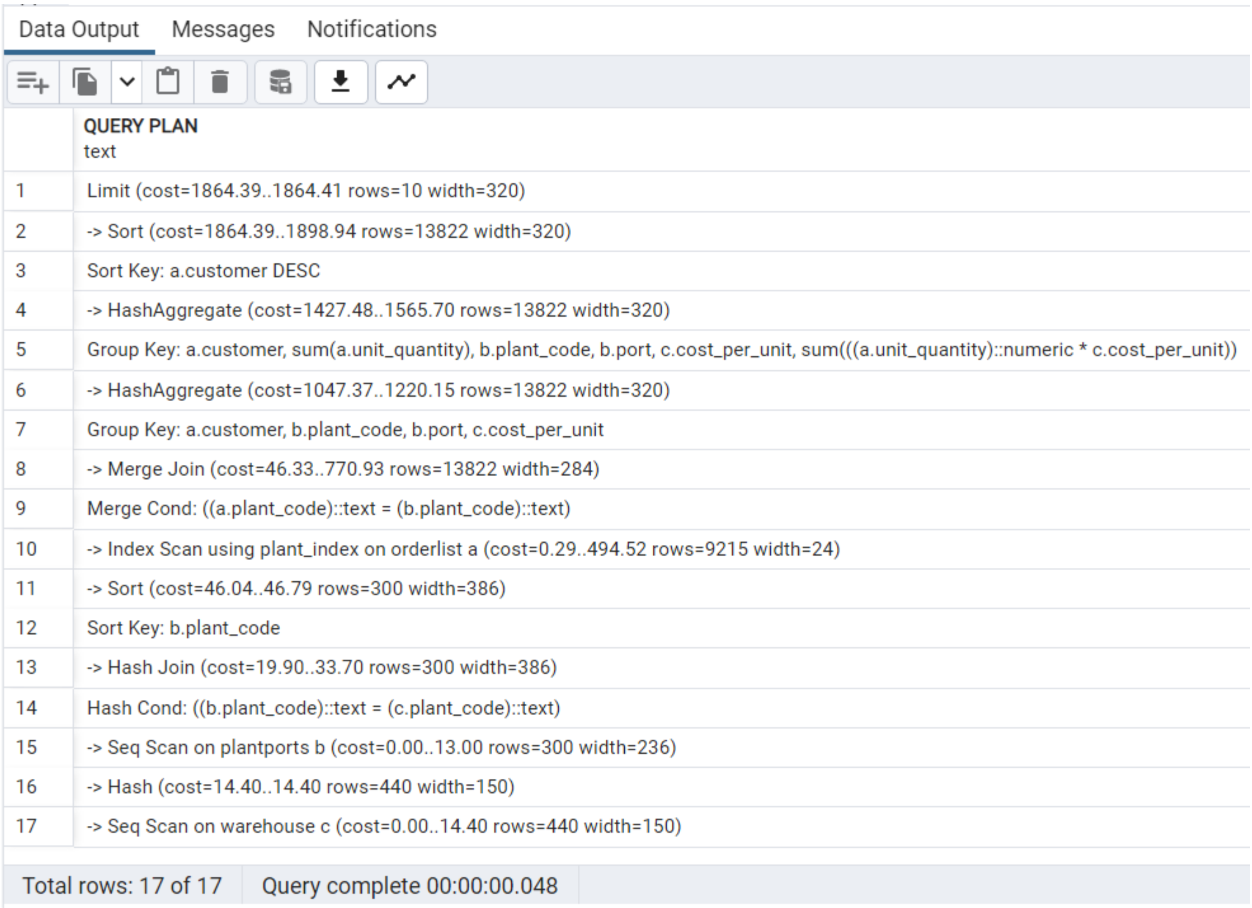
```
EXPLAIN
SELECT   distinct(a.customer),
                   sum(a.unit_quantity) as total_quantity,
                   b.plant_code, b.port,
                   c.cost_per_unit,
                   sum((a.unit_quantity*c.cost_per_unit)) as total_cost
FROM                        orderlist as a
Inner Join      plantports as b
ON              a.plant_code = b.plant_code
Inner Join      warehouse as c
ON              a.plant_code = c.plant_code
GROUP BY        a.customer,
                b.plant_code,
                b.port,
                c.cost_per_unit
        ORDER BY                a.customer desc;
```



```
QUERY PLAN
text
1	Limit (cost=1874.87..1874.90 rows=10 width=320)
2	-> Sort (cost=1874.87..1909.43 rows=13822 width=320)
3	Sort Key: a.customer DESC
4	-> HashAggregate (cost=1437.96..1576.18 rows=13822 width=320)
5	Group Key: a.customer, sum(a.unit_quantity), b.plant_code, b.port, c.cost_per_unit, sum(((a.unit_quantity)::numeric * c.cost_per_unit))
6	-> HashAggregate (cost=1057.86..1230.63 rows=13822 width=320)
7	Group Key: a.customer, b.plant_code, b.port, c.cost_per_unit
8	-> Hash Join (cost=37.45..781.42 rows=13822 width=284)
9	Hash Cond: ((a.plant_code)::text = (b.plant_code)::text)
10	-> Seq Scan on orderlist a (cost=0.00..237.15 rows=9215 width=24)
11	-> Hash (cost=33.70..33.70 rows=300 width=386)
12	-> Hash Join (cost=19.90..33.70 rows=300 width=386)
13	Hash Cond: ((b.plant_code)::text = (c.plant_code)::text)
14	-> Seq Scan on plantports b (cost=0.00..13.00 rows=300 width=236)
15	-> Hash (cost=14.40..14.40 rows=440 width=150)
16	-> Seq Scan on warehouse c (cost=0.00..14.40 rows=440 width=150)

Total rows: 16 of 16	Query complete 00:00:00.131
```

**Fig:** With statistics but without indexes

After Creating Index for Orderlist table on plant_index, we get query analysis by using explain



| | QUERY PLAN<br>text |
|---|---|
| 1 | Limit (cost=1864.39..1864.41 rows=10 width=320) |
| 2 | -> Sort (cost=1864.39..1898.94 rows=13822 width=320) |
| 3 | Sort Key: a.customer DESC |
| 4 | -> HashAggregate (cost=1427.48..1565.70 rows=13822 width=320) |
| 5 | Group Key: a.customer, sum(a.unit_quantity), b.plant_code, b.port, c.cost_per_unit, sum(((a.unit_quantity)::numeric * c.cost_per_unit)) |
| 6 | -> HashAggregate (cost=1047.37..1220.15 rows=13822 width=320) |
| 7 | Group Key: a.customer, b.plant_code, b.port, c.cost_per_unit |
| 8 | -> Merge Join (cost=46.33..770.93 rows=13822 width=284) |
| 9 | Merge Cond: ((a.plant_code)::text = (b.plant_code)::text) |
| 10 | -> Index Scan using plant_index on orderlist a (cost=0.29..494.52 rows=9215 width=24) |
| 11 | -> Sort (cost=46.04..46.79 rows=300 width=386) |
| 12 | Sort Key: b.plant_code |
| 13 | -> Hash Join (cost=19.90..33.70 rows=300 width=386) |
| 14 | Hash Cond: ((b.plant_code)::text = (c.plant_code)::text) |
| 15 | -> Seq Scan on plantports b (cost=0.00..13.00 rows=300 width=236) |
| 16 | -> Hash (cost=14.40..14.40 rows=440 width=150) |
| 17 | -> Seq Scan on warehouse c (cost=0.00..14.40 rows=440 width=150) |

Total rows: 17 of 17 Query complete 00:00:00.048

**Fig:** With both statistic and indexes

## Methodology

To test its execution time, we used the command EXPLAIN <query8>.
We run Q8 in three different situations:
1. Before any statistics collection or indexing
2. With statistics but without indexes
3. With both statistic and indexes

We used the following indexing scheme:
CREATE index plant_index on orderlist(plant_code);

## Benchmarks

In this section we report the observed performance for each query execution , together with the query plans generated by the optimizer.

## Instructions for reproducing the experiments

**DDL Statements:**

```
DROP TABLE IF EXISTS plantports;
DROP TABLE IF EXISTS vmicustomers;
DROP TABLE IF EXISTS productsperplant ;
DROP TABLE IF EXISTS warehouse;
DROP TABLE IF EXISTS freightrates;
DROP TABLE IF EXISTS orderlist ;

CREATE TABLE plantports
(
plant_code varchar(50) REFERENCES warehouse(plant_code),

port varchar(50) NOT NULL
);


CREATE TABLE vmicustomers
(
plant_code varchar(50) REFERENCES warehouse(plant_code),
customers varchar(50) NOT NULL
);

CREATE TABLE productsperplant
(
plant_code varchar(50) REFERENCES warehouse(plant_code),
product_id int NOT NULL
);

CREATE TABLE warehouse
(
plant_code varchar(50) PRIMARY KEY,
cost_per_unit decimal NOT NULL,
daily_capacity int NOT NULL
);

CREATE TABLE freightrates
(
carrier varchar(50) NOT NULL,
orig_port_cd varchar(50),
dest_port_cd varchar(50),
minm_wgh_qty decimal,
max_wgh_qty decimal,
svc_cd varchar(50),
minimum_cost decimal,
rate decimal,
mode_dsc varchar(50),
tpt_day_cnt int,

carrier_type varchar(50) NOT NULL
);
```

```
CREATE TABLE orderlist
(
order_id decimal PRIMARY KEY,
order_date DATE ,
origin_port varchar(50),
carrier varchar(50),
tpt int ,
service_level varchar(50),
ship_ahead_day_count int,
ship_late_day_count int,
customer varchar(50),
product_id INT,
plant_code varchar(50) references warehouse(plant_code),
destination_port varchar(50),
unit_quantity int,
weight decimal
);
```

**DML Statements:**

```
COPY plantports FROM 'C:\Program Files\PostgreSQL\15\project\project556\PlantPorts.csv' delimiter ',' CSV Header;
COPY vmicustomers FROM 'C:\Program Files\PostgreSQL\15\project\project556\VmiCustomers.csv' delimiter ',' CSV Header;
COPY productsperplant FROM 'C:\Program Files\PostgreSQL\15\project\project556\ProductsPerPlant.csv' delimiter ',' CSV Header;
COPY warehouse FROM 'C:\Program Files\PostgreSQL\15\project\project556\Warehouse.csv' delimiter ',' CSV Header;
COPY freightrates FROM 'C:\Program Files\PostgreSQL\15\project\project556\FreightRates.csv' DELIMITER ',' CSV Header;
COPY orderlist FROM 'C:\Program Files\PostgreSQL\15\project\project556\OrderList.csv' delimiter ',' CSV Header;

ALTER TABLE warehouse
ADD CONSTRAINT wh_pkey PRIMARY KEY (wh);

ALTER TABLE warehouse
RENAME COLUMN wh TO plant_code;

ALTER TABLE orderlist
ADD CONSTRAINT order_id_pkey PRIMARY KEY (order_id);
```

**Indexes:**

```
CREATE index plant_index on orderlist(plant_code);
CREATE index product_idindex on orderlist(product_id);
```

## Final Conclusions

For the given dataset, analysis on supply chain management data is performed, and some meaningful insights are gained which are mentioned above, as well as ER diagram for effective database construction.