Product Performance Analysis for E-commerce Success

Nitesh Padidam
vpadidam@buffalo.edu
50495795
vpadidam

Manoj Paladi
manojpal@buffalo.edu
50496792
manojpal

Thrivikrama Rao Kavuri
thrivikr@buffalo.edu
50496382
thrivikr

I. Problem Statement

The aim of the project is to identify our top-performing and under-performing products by analyzing sales, returns i.e Product Performance Analysis and Seasonal trends in sales i.e Seasonal Demand and customer segmentation analysis from the data.

II. Phase 1 Overview

This project is essential for data-driven decisions on product quality enhancement, product discontinuation, supply chain and inventory planning, targeted marketing campaign creation, etc.. Better business strategies and cost optimization result from this.
Database is used compared to Excel, as it enables scalability, data integrity, multi-user access, improved security, and more powerful data retrieval capabilities. Database can be leveraged for various functions like reporting, analytics, queries, maintaining the systems etc and expected target users are Developers, data/business analysts, customer support, business owners, database administrators etc.
Data is loaded into Database from the source file which are raw csv data files. These are imported into PostgreSQL database and the steps include inspecting data, creating tables, directly importing data, correcting data types and formats etc. Source data have 8 tables and these tables are verified whether they are in BCNF form or not. After verification , database is normalised to BCNF form resulting in 12 tables and ER diagram of same is designed.

III. Indexing

In our database, dataset is relatively small, primary key indexing is implemented as a fundamental practice to enhance data integrity and optimize data retrieval.
Primary key serves as a unique identifier for each record in a table, ensuring that no two rows share identical values in the designated primary key columns. This uniqueness constraint, combined with the absence of NULL values in the primary key columns, facilitates accurate identification and retrieval of specific records.
In PostgreSQL database, there is an automatic creation of a unique index on the primary key column(s). This index is known as a clustered index which organizes the physical order of rows in the table to correspond with the order of the primary key values. This design choice results in more efficient querying capabilities, particularly for operations such as searching, joining, or sorting based on the primary key.
Apart from uniqueness, primary key plays a pivotal role in establishing relationships between tables. One such example is where foreign key relationships are employed, another table's foreign key references the primary key of our table, creating a logical connection between the two.
By implementing primary key indexing, data integrity is maintained and helps in fast data retrieval and enhances query performances.Primary key indexing concepts are particularly helpful when dealing with smaller datasets, but they also provide a strong basis for efficiency and scalability when our database gets larger and more complicated.

IV. Data Counts

After normalization , following are the table and data counts in them.

TABLE I
Product_Name

Table_Name	Table_Schema	Data_Count
products	public	293
product_description	public	293
product_category	public	4
Category_Subcategory	public	37
calendar	public	912
sales	public	56046
product_subcategory	public	37
customers	public	18148
territory	public	10
Continent	public	6
Product_Name	public	293
Returns	public	1809

V. Testing Database

Comprehensive examination of database is performed by employing a variety of SQL queries to test the functionality of essential operations such as inserting, deleting, updating, and selecting data.
Series of SQL queries are created that are organized according to the functions they perform. Every inquiry was meticulously designed to encompass a variety of possible use cases and to reflect real-world situations.
Primary objective of this examination is to evaluate the database's robustness and its ability to handle diverse query types, including join operations, ordering, grouping, and sub-queries.

A. SQL 1

```sql
SELECT "ProductName","ProductColor","ProductSize","
    ProductStyle" from product_description;
```

Listing 1. SQL Query for Table-Product Description

ProductColor character varying	ProductSize character varying	ProductStyle character varying
NA	0	0
Multi	0	U
NA	0	0
NA	0	0
Silver	0	0
Blue	L	U
Blue	M	U
Blue	S	U
NA	0	0
Silver	0	0
Silver	0	0
Black	L	U
Black	M	U
Black	S	U
Black	L	U

Fig. 1. SQL-1: Table without updated productstyle values(0)

```sql
UPDATE product_description SET "ProductStyle" = 'NA'
    WHERE "ProductStyle" = '0';
```

Listing 2. SQL Query for updating table

	ProductName [PK] character varying	ProductColor character varying	ProductSize character varying	ProductStyle character varying
249	Mountain Bottle Cage	NA	0	NA
250	Road Bottle Cage	NA	0	NA
251	Patch Kit/8 Patches	NA	0	NA
252	Hitch Rack - 4-Bike	NA	0	NA
253	Bike Wash - Dissolver	NA	0	NA
254	Fender Set - Mountain	NA	0	NA
255	All-Purpose Bike Stand	NA	0	NA
256	Hydration Pack - 70 oz.	Silver	70	NA
257	Rear Derailleur	Silver	0	NA
258	Rear Brakes	Silver	0	NA
259	LL Mountain Seat/Saddle	NA	0	NA
260	ML Mountain Seat/Saddle	NA	0	NA
261	HL Mountain Seat/Saddle	NA	0	NA

Fig. 2. SQL-1:Table with updated productstyle values

B. SQL 2

```sql
ALTER TABLE calendar
ADD COLUMN month VARCHAR,
ADD COLUMN year INTEGER;

UPDATE calendar
SET
  month = TO_CHAR(e_date, 'Month'),
  year = EXTRACT(YEAR FROM e_date);
```

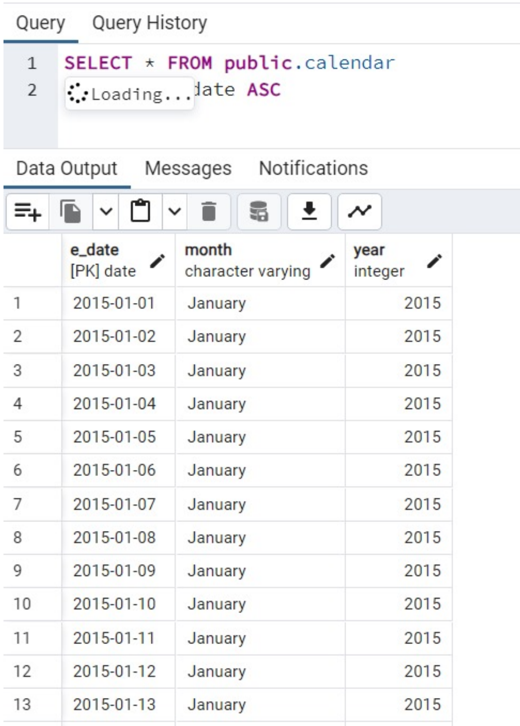
Listing 3. SQL Query- Altering table

```sql
ALTER TABLE calendar
ADD COLUMN month VARCHAR,
ADD COLUMN year INTEGER;

UPDATE calendar
SET
  month = TO_CHAR(e_date, 'Month'),
  year = EXTRACT(YEAR FROM e_date);
```

Data Output | Messages | Notifications

```
UPDATE 912

Query returned successfully in 91 msec.
```

Fig. 3. SQL-2: execution Status



	e_date [PK] date	month character varying	year integer
1	2015-01-01	January	2015
2	2015-01-02	January	2015
3	2015-01-03	January	2015
4	2015-01-04	January	2015
5	2015-01-05	January	2015
6	2015-01-06	January	2015
7	2015-01-07	January	2015
8	2015-01-08	January	2015
9	2015-01-09	January	2015
10	2015-01-10	January	2015
11	2015-01-11	January	2015
12	2015-01-12	January	2015
13	2015-01-13	January	2015

Fig. 4. SQL-2: Table with additional month and year column

C. SQL 3

```
INSERT INTO products(productkey,
    productsubcategorykey,productcost,productprice)
VALUES(607,14,897.64,1285.87);
```

Listing 4. SQL Query for Inserting Values

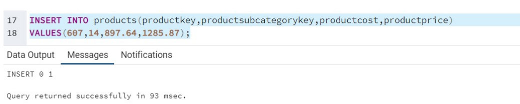


Fig. 5. SQL-3: Successful execution of INSERT statement

D. SQL 4

```
DELETE from products
where productkey=607;
```

Listing 5. SQL Query for Delete statement

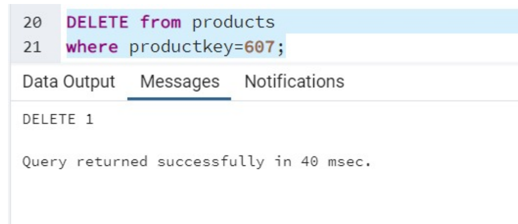
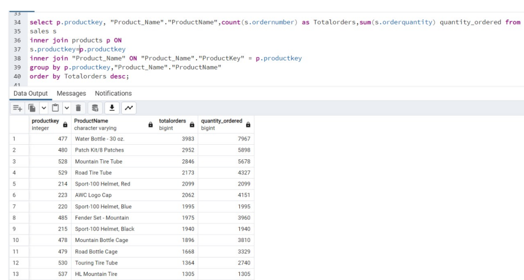


Fig. 6. SQL-4: Successful Execution of DELETE Statement

E. SQL 5

```
select p.productkey, "Product_Name"."ProductName",
    count(s.ordernumber) as Totalorders,sum(s.
    orderquantity) quantity_ordered from
sales s
inner join products p ON
s.productkey=p.productkey
inner join "Product_Name" ON "Product_Name"."
    ProductKey" = p.productkey
group by p.productkey,"Product_Name"."ProductName"
order by Totalorders desc;
```

Listing 6. SQL Query for Total orders, Total quantity ordered of each product



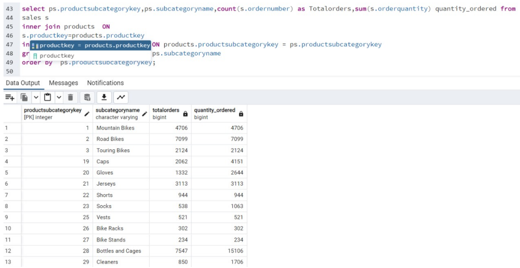
	productkey integer	ProductName character varying	totalorders bigint	quantity_ordered bigint
1	477	Water Bottle - 30 oz.	3983	7967
2	480	Patch Kit/8 Patches	2952	5898
3	528	Mountain Tire Tube	2846	5678
4	529	Road Tire Tube	2173	4327
5	214	Sport-100 Helmet, Red	2099	2099
6	223	AWC Logo Cap	2062	4151
7	220	Sport-100 Helmet, Blue	1995	1995
8	485	Fender Set - Mountain	1975	3960
9	215	Sport-100 Helmet, Black	1940	1940
10	478	Mountain Bottle Cage	1896	3810
11	479	Road Bottle Cage	1668	3329
12	530	Touring Tire Tube	1364	2740
13	537	HL Mountain Tire	1305	1305

Fig. 7. Results of SQL-5

F. SQL 6

```
select ps.productsubcategorykey,ps.subcategoryname,
    count(s.ordernumber) as Totalorders,sum(s.
    orderquantity) quantity_ordered from
sales s
inner join products  ON
s.productkey=products.productkey
inner join product_subcategory ps ON products.
    productsubcategorykey = ps.productsubcategorykey
group by ps.productsubcategorykey,ps.subcategoryname
order by  ps.productsubcategorykey;
```

Listing 7. SQL Query for Total orders, Total quantity ordered w.r.t Product Subcategory



```
43 select ps.productsubcategorykey,ps.subcategoryname,count(s.ordernumber) as Totalorders,sum(s.orderquantity) quantity_ordered from
44 sales s
45 inner join products  ON
46 s.productkey=products.productkey
47 in productkey = products.productkey ON products.productsubcategorykey = ps.productsubcategorykey
48 gr  productkey                       ps.subcategoryname
49 order by  ps.productsubcategorykey;
50
```

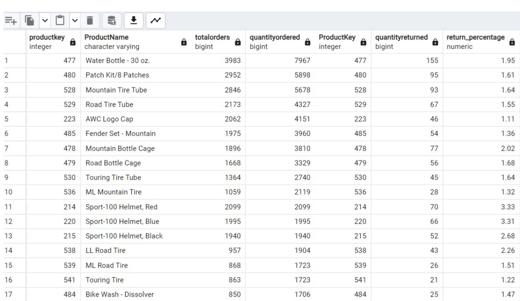
	productsubcategorykey [PK] integer	subcategoryname character varying	totalorders bigint	quantity_ordered bigint
1	1	Mountain Bikes	4706	4706
2	2	Road Bikes	7099	7099
3	3	Touring Bikes	2124	2124
4	19	Caps	2062	4151
5	20	Gloves	1332	2644
6	21	Jerseys	3113	3113
7	22	Shorts	944	944
8	23	Socks	538	1063
9	25	Vests	521	521
10	26	Bike Racks	302	302
11	27	Bike Stands	234	234
12	28	Bottles and Cages	7547	15106
13	29	Cleaners	850	1706

Fig. 8. Results of SQL-6

G. SQL 7

```
with tot as (SELECT *
FROM (
    SELECT
        p.productkey,
        "Product_Name"."ProductName",
        COUNT(s.ordernumber) AS TotalOrders,
        SUM(s.orderquantity) AS QuantityOrdered
    FROM
        sales s
    INNER JOIN products p ON s.productkey = p.
        productkey
    INNER JOIN "Product_Name" ON "Product_Name"."
        ProductKey" = p.productkey
    GROUP BY
        p.productkey, "Product_Name"."ProductName"
) AS orders
INNER JOIN (
    SELECT
        r."ProductKey",
        SUM(r."ReturnQuantity") AS QuantityReturned
    FROM
        "Returns" r
    GROUP BY
        r."ProductKey"
) AS qr ON orders.productkey = qr."ProductKey")
SELECT *,round((quantityreturned::decimal /
    quantityordered)*100,2) as return_percentage
    from tot
order by quantityordered DESC;
```

Listing 8. SQL Query: Total orders and Return Percentage of each product



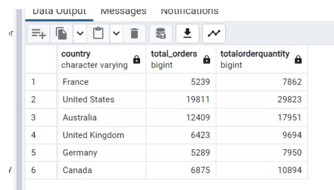
	productkey integer	ProductName character varying	totalorders bigint	quantityordered bigint	ProductKey integer	quantityreturned bigint	return_percentage numeric
1	477	Water Bottle - 30 oz.	3983	7967	477	155	1.95
2	480	Patch Kit/8 Patches	2952	5898	480	95	1.61
3	528	Mountain Tire Tube	2846	5678	528	93	1.64
4	529	Road Tire Tube	2173	4327	529	67	1.55
5	223	AWC Logo Cap	2062	4151	223	46	1.11
6	485	Fender Set - Mountain	1975	3960	485	54	1.36
7	478	Mountain Bottle Cage	1896	3810	478	77	2.02
8	479	Road Bottle Cage	1668	3329	479	56	1.68
9	530	Touring Tire Tube	1364	2740	530	45	1.64
10	536	ML Mountain Tire	1059	2119	536	28	1.32
11	214	Sport-100 Helmet, Red	2099	2099	214	70	3.33
12	220	Sport-100 Helmet, Blue	1995	1995	220	66	3.31
13	215	Sport-100 Helmet, Black	1940	1940	215	52	2.68
14	538	LL Road Tire	957	1904	538	43	2.26
15	539	ML Road Tire	868	1723	539	26	1.51
16	541	Touring Tire	863	1723	541	21	1.22
17	484	Bike Wash - Dissolver	850	1706	484	25	1.47

Fig. 9. SQL-7 : results

H. SQL 8

```
SELECT
territory.country, count(s.ordernumber) as
    Total_orders, sum(orderquantity) as
    Totalorderquantity
from sales s
right join territory ON territory.salesterritorykey
    = s.territorykey
group By territory.country
```

Listing 9. SQL Query: Total orders w.r.t Country



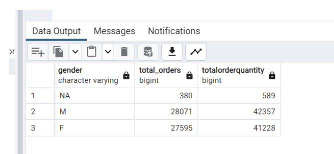
	country character varying	total_orders bigint	totalorderquantity bigint
1	France	5239	7862
2	United States	19811	29823
3	Australia	12409	17951
4	United Kingdom	6423	9694
5	Germany	5289	7950
6	Canada	6875	10894

Fig. 10. SQL-8 : results

I. SQL 9

```
SELECT
customers.gender, count(s.ordernumber) as
    Total_orders, sum(orderquantity) as
    Totalorderquantity
from sales s
inner join customers ON customers.customerkey = s.
    customerkey
group By customers.gender
```

Listing 10. SQL Query: Total orders w.r.t Gender



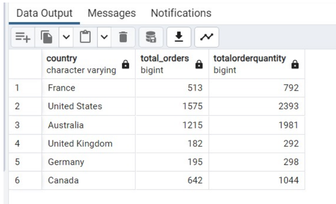
	gender character varying	total_orders bigint	totalorderquantity bigint
1	NA	380	589
2	M	28071	42357
3	F	27595	41228

Fig. 11. SQL-9 : results

J. SQL 10

```
SELECT
territory.country, count(s.ordernumber) as
Total_orders, sum(orderquantity) as
Totalorderquantity
from sales s
inner join territory ON territory.salesterritorykey=
     s.territorykey
inner join customers ON customers.customerkey = s.
    customerkey
where customers.totalchildren >4
group By territory.country
```

Listing 11. SQL Query: Purchasing capacity of large family



Data Output Messages Notifications

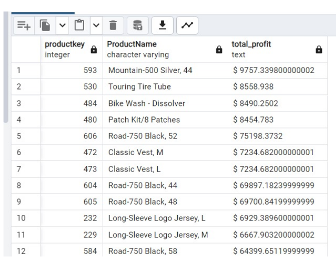
	country character varying	total_orders bigint	totalorderquantity bigint
1	France	513	792
2	United States	1575	2393
3	Australia	1215	1981
4	United Kingdom	182	292
5	Germany	195	298
6	Canada	642	1044

Fig. 12. SQL-10 : results

K. SQL 11

```sql
select
Ord.productkey, pn."ProductName",
'$'||' '||(products.productprice - products.
    productcost)*Ord.Totalorderquantity as
    Total_Profit
FROM
(SELECT sales.productkey,sum(orderquantity) as
Totalorderquantity from sales
group by sales.productkey) as Ord
inner join products on products.productkey = Ord.
    productkey
inner join public."Product_Name" pn ON pn."
    ProductKey" = products.productkey
ORDER by Total_Profit DESC;
```

Listing 12. SQL Query: Profitable products



	productkey integer	ProductName character varying	total_profit text
1	593	Mountain-500 Silver, 44	$ 9757.339800000002
2	530	Touring Tire Tube	$ 8558.938
3	484	Bike Wash - Dissolver	$ 8490.2502
4	480	Patch Kit/8 Patches	$ 8454.783
5	606	Road-750 Black, 52	$ 75198.3732
6	472	Classic Vest, M	$ 7234.682000000001
7	473	Classic Vest, L	$ 7234.682000000001
8	604	Road-750 Black, 44	$ 69897.18239999999
9	605	Road-750 Black, 48	$ 69700.84199999999
10	232	Long-Sleeve Logo Jersey, L	$ 6929.389600000001
11	229	Long-Sleeve Logo Jersey, M	$ 6667.903200000002
12	584	Road-750 Black, 58	$ 64399.65119999999

Fig. 13. SQL-11 : results

VI. QUERY EXECUTION ANALYSIS

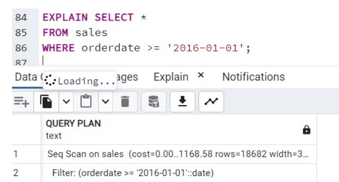
A. Query 1

```sql
EXPLAIN SELECT *
FROM sales
WHERE orderdate >= '2016-01-01';
```

Listing 13. Query 1

Result:

The query performs a full table scan, reading every row in the "sales" table sequentially. The Filter condition indicates that only rows where "orderdate" is greater than or equal to '2016-01-01' are included in the result set. The estimated cost is 1168.58 units.



```
84 EXPLAIN SELECT *
85 FROM sales
86 WHERE orderdate >= '2016-01-01';
87
```

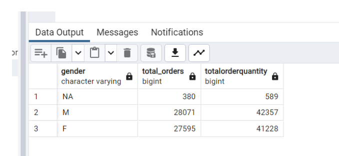
	QUERY PLAN text
1	Seq Scan on sales (cost=0.00..1168.58 rows=18682 width=3...
2	Filter: (orderdate >= '2016-01-01'::date)

Fig. 14. Explaining execution of Query1

In an attempt to reduce the cost of operation,Indexing is applied on order date for faster retrieval of data.

```sql
CREATE INDEX idx_order_date ON sales(orderdate);
```

Listing 14. Indexing Query



Data Output Messages Notifications

	gender character varying	total_orders bigint	totalorderquantity bigint
1	NA	380	589
2	M	28071	42357
3	F	27595	41228

Fig. 15. Results after implementing indexing

After Indexing :

The query uses a Bitmap Heap Scan, which is more specific and efficient than a sequential scan. The Bitmap Index Scan indicates that an index on "orderdate" (idx_order_date) is used for optimization. The estimated cost is reduced to a range between 217.08 and 918.60 units.

B. Query 2

```sql
EXPLAIN select ps.productsubcategorykey,ps.
    subcategoryname,count(s.ordernumber) as
    Totalorders,sum(s.orderquantity)
    quantity_ordered from
products
INNER join product_subcategory ps ON products.
    productsubcategorykey = ps.productsubcategorykey
inner join sales s  ON
products.productkey=s.productkey
group by ps.productsubcategorykey,ps.subcategoryname
order by  ps.productsubcategorykey
```

Listing 15. Query 2

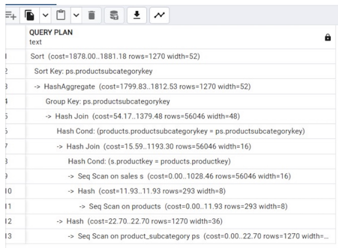


Fig. 16. Explaining execution of Query2

Limit Sorting:
Sorting can be expensive, especially for large result sets. If you don't need the entire result set sorted, you might consider limiting the number of rows you are retrieving or using an index to cover the sorting needs.

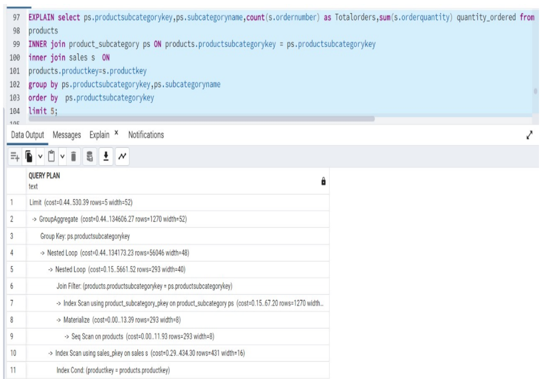


Fig. 17. Explanation after Limit sorting

After implementing limit sorting, great reduction in cost value can be observed.

C. Query 3

```
EXPLAIN select p.productkey, "Product_Name"."
    ProductName",count(s.ordernumber) as Totalorders
    ,sum(s.orderquantity) quantity_ordered from
sales s
inner join products p ON
s.productkey=p.productkey
inner join "Product_Name" ON "Product_Name"."
    ProductKey" = p.productkey
group by p.productkey,"Product_Name"."ProductName"
order by Totalorders desc;
```

Listing 16. Query 3

The above mentioned query is nested sub query which uses three tables to produce desired results. the cost function is in range of 6893.50 to 7033.61.

Inspecting the join procedures:

It can be observed that aggregation is performed at the end which leads to higher computation cost. lets perform aggregation at initial stage and perform join later.
From flow diagram we can conclude that only two tables are enough to produce the desired result. The following is the optimized SQL query.

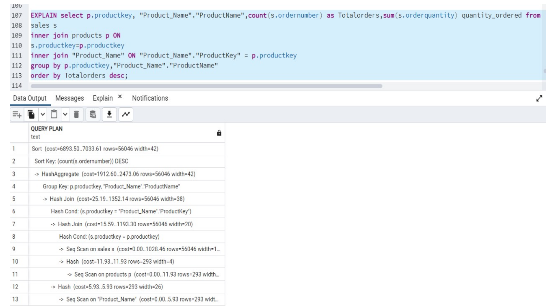


Fig. 18. Explanation of Query 3

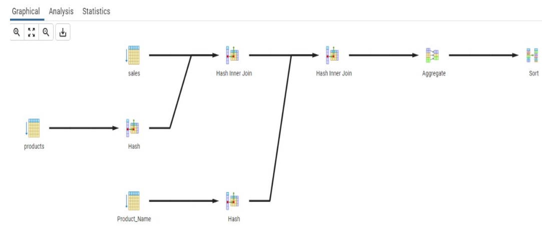


Fig. 19. Execution procedure of Query 3

```
select sa.ProductKey,"Product_Name"."ProductName",sa
    .Totalorders,sa.quantity_ordered from (SELECT
    Productkey,count(ordernumber) as Totalorders,sum
    (orderquantity) quantity_ordered from sales
Group by productkey) as sa,"Product_Name"
where "Product_Name"."ProductKey"=sa.ProductKey
order by sa.Totalorders desc
```

Listing 17. Optimized Query 3

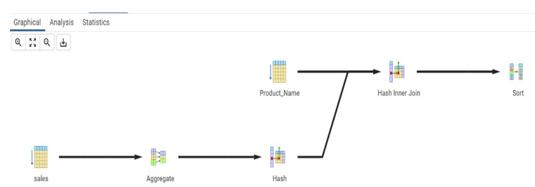


Fig. 20. Execution procedure of optimized query

we can see the cost function in range of 1464.30 to 1464.63

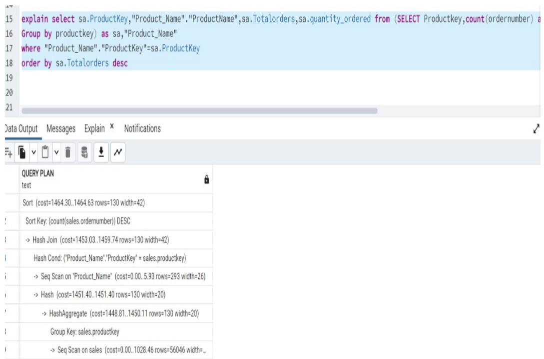


Fig. 21. Explanation of Optimized Query 3

VII. Contribution

All team members continued to collaborate and communicate effectively throughout the project. Having brainstorming sessions, team meetings, and idea exchange allowed for a more coherent process. We made effective use of each team member's strengths to solve problems and learned the corresponding skill while solving guarantying in a comprehensive approach to problem-solving.

VIII. References

1) Data Source
2) Lecture Slides
3) DATABASE SYSTEMS The Complete Book Second Edition by Hector Garcia-Molina,Jeffrey D. Ullman,Jennifer Widom