Staphon Smith

EcoMart Relational Database Design

A. Scenario 2 - EcoMart

A1: Business Problem: The EcoMart company faces challenges in managing and analyzing large-scale sales data across various regions, countries, and product categories. The current manual process is time-consuming, prone to errors, and lacks the scalability to accommodate growing data volumes. A database solution is essential to streamline data storage, enable efficient querying, and generate actionable insights.

A2: Proposed Data Structure: The proposed data structure includes three core tables—Categories, Products, and Transactions—each designed to store specific attributes. These tables are interlinked through primary and foreign keys, ensuring data consistency and enabling detailed analysis.

A3: Justification for Database Solution: A relational database offers scalability, data integrity, and efficient query performance. It eliminates redundancy through normalization, supports large-scale data processing, and provides a robust platform for advanced analytics. By using PostgreSQL, EcoMart can implement a cost-effective, reliable, and scalable solution.

A4. Usage of Business Data: The business data will be used to analyze sales trends by region and product category, monitor stock levels, evaluate revenue performance, and optimize order fulfillment. It will also support decision-making through real-time reporting and analytics.

B. Logical Data Model

The logical data model represents the structure of the database, including entities, attributes, and relationships. It ensures data is organized efficiently, adheres to normalization standards, and supports seamless querying and analysis.

Entities in the model include Categories, Products, and Transactions. Each entity contains attributes tailored to its role, such as category_name, price, and order_date. Relationships between tables ensure data integration, such as the one-to-many relationship between Categories and Products.

Entities and Attributes:

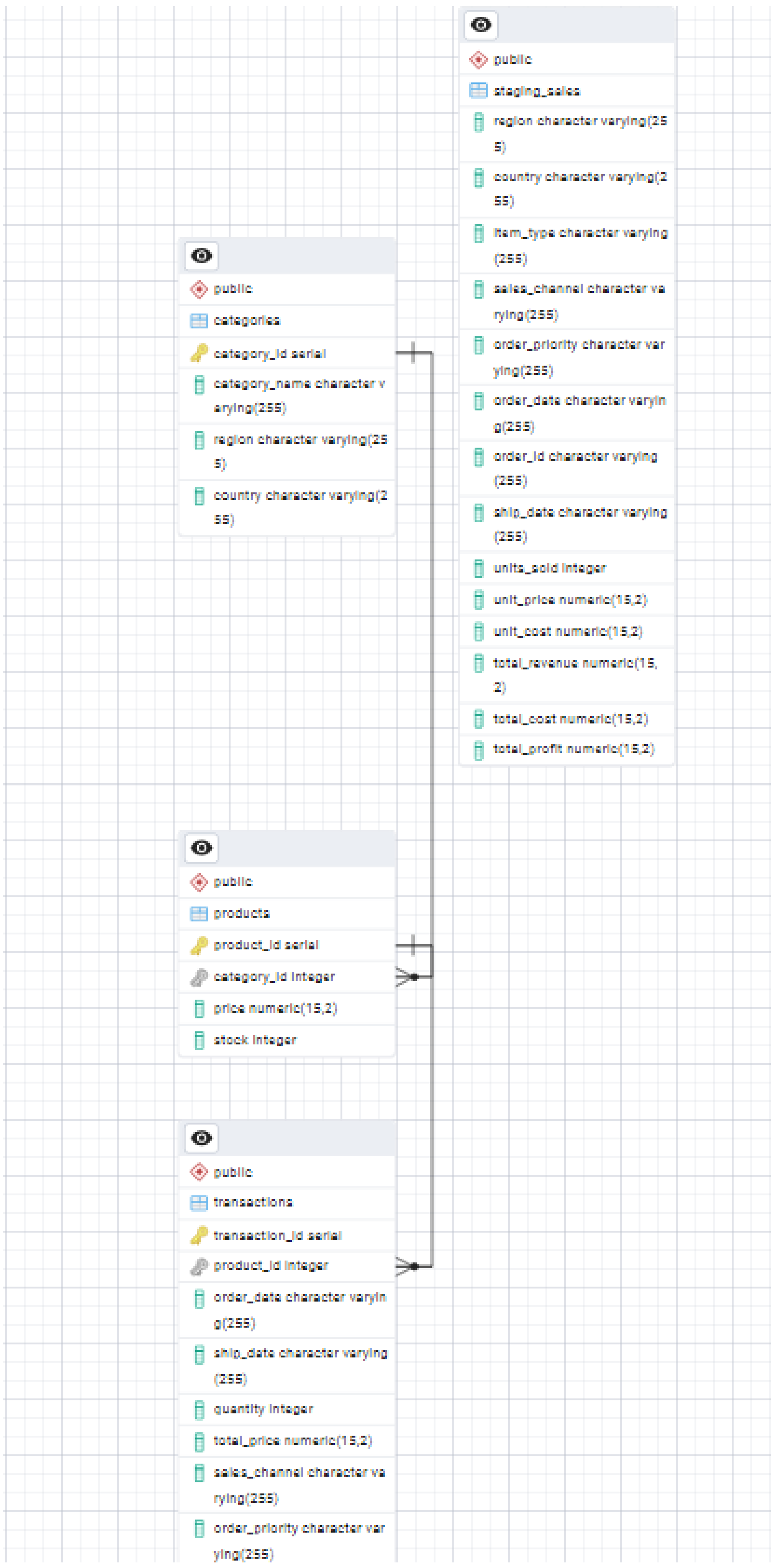
- *Categories*: category_id, category_name, region, and country.
- *Products*: product_id, category_id, price, and stock.
- *Transactions*: transaction_id, product_id, order_date, ship_date, quantity, and total_price.

Relationships:

- Categories → Products: One category can have multiple products.
- Products → Transactions: One product can have multiple transactions.

Normalization:
The design adheres to 3NF, ensuring no redundant data and maintaining data integrity.

Staphon Smith


public
staging_sales
region character varying(255)
country character varying(255)
item_type character varying(255)
sales_channel character varying(255)
order_priority character varying(255)
order_date character varying(255)
order_id character varying(255)
ship_date character varying(255)
units_sold integer
unit_price numeric(15,2)
unit_cost numeric(15,2)
total_revenue numeric(15,2)
total_cost numeric(15,2)
total_profit numeric(15,2)
public
categories
category_id serial
category_name character varying(255)
region character varying(255)
country character varying(255)
public
products
product_id serial
category_id integer
price numeric(15,2)
stock integer
public
transactions
transaction_id serial
product_id integer
order_date character varying(255)
ship_date character varying(255)
quantity integer
total_price numeric(15,2)
sales_channel character varying(255)
order_priority character varying(255)

C. Database Objects and Storage

The database consists of core objects like tables, indexes, and relationships designed to store and integrate data effectively. Tables represent entities, with attributes optimized for storage efficiency. For instance, “**NUMERIC**” is used for monetary values, while “**VARCHAR**” handles textual data. Indexes are automatically created for primary and foreign keys to optimize query performance.

D. Scalability

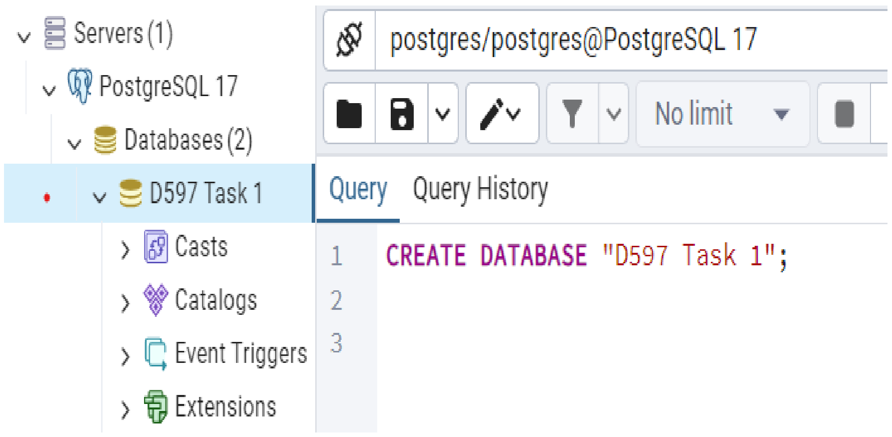
The database design addresses scalability concerns to handle increasing data volumes and user demands. Key strategies include indexing for optimized query performance, particularly on fields like “**order_date**” and “**region**”. Normalization minimizes redundancy, enhancing data consistency and storage efficiency. This makes data easier to scale than it was previously, and no need to go through thousands of rows of data.

E. Privacy and Security

Strong privacy and security measures are built into the database design to ensure compliance with regulations and protect sensitive information. Encryption safeguards data both at rest and in process, preventing unauthorized access. The company can also require audit logging and 2FA as an added layer of security precautions. Additionally, input validation is implemented to protect against SQL attacks, which will assist in maintaining the database’s integrity.

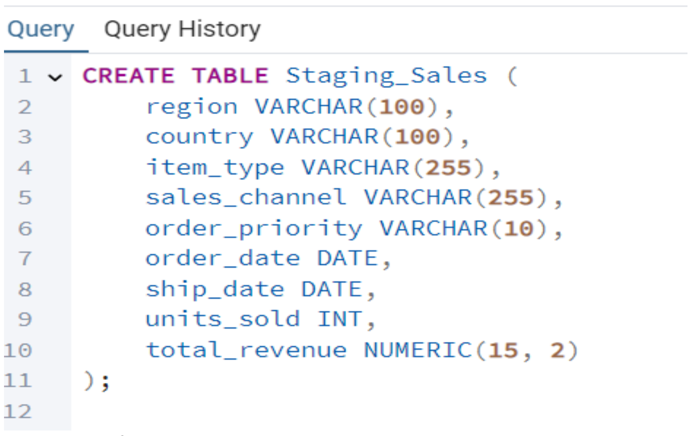
F. Implementation Through pgAdmin

F1: Creation of Database:

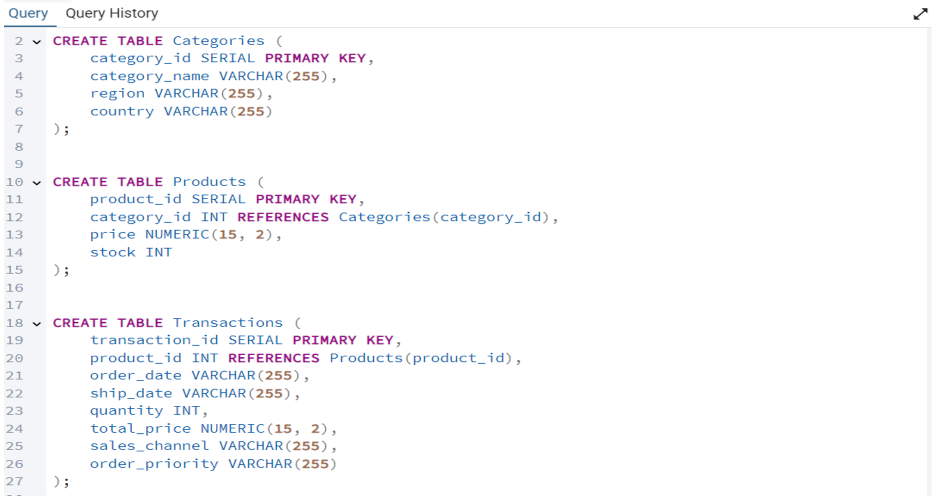


```sql
CREATE DATABASE "D597 Task 1";
```

F1: Creation of Tables:



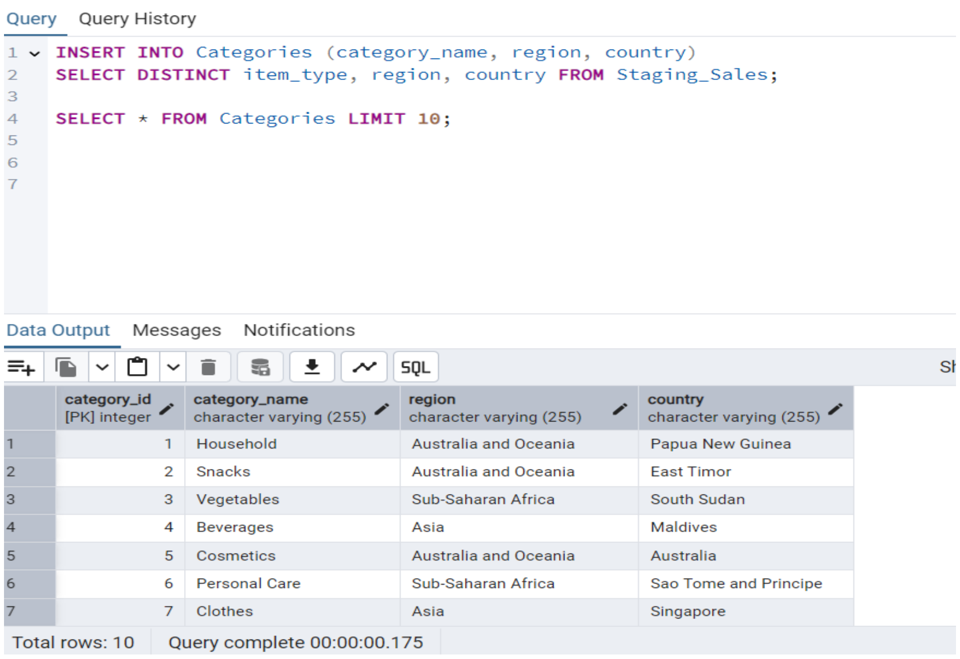
```sql
CREATE TABLE Staging_Sales (
    region VARCHAR(100),
    country VARCHAR(100),
    item_type VARCHAR(255),
    sales_channel VARCHAR(255),
    order_priority VARCHAR(10),
    order_date DATE,
    ship_date DATE,
    units_sold INT,
    total_revenue NUMERIC(15, 2)
);
```



```sql
CREATE TABLE Categories (
    category_id SERIAL PRIMARY KEY,
    category_name VARCHAR(255),
    region VARCHAR(255),
    country VARCHAR(255)
);


CREATE TABLE Products (
    product_id SERIAL PRIMARY KEY,
    category_id INT REFERENCES Categories(category_id),
    price NUMERIC(15, 2),
    stock INT
);


CREATE TABLE Transactions (
    transaction_id SERIAL PRIMARY KEY,
    product_id INT REFERENCES Products(product_id),
    order_date VARCHAR(255),
    ship_date VARCHAR(255),
    quantity INT,
    total_price NUMERIC(15, 2),
    sales_channel VARCHAR(255),
    order_priority VARCHAR(255)
);
```

Staphon Smith

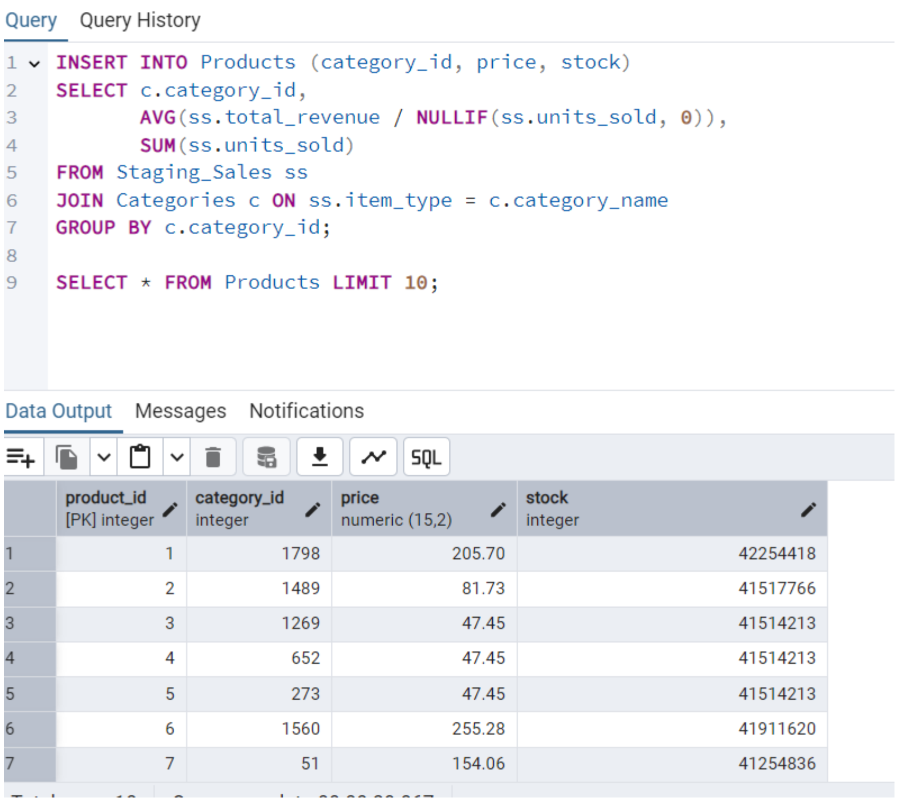
F2. Data import from “Staging Table” to respective tables:



```sql
INSERT INTO Categories (category_name, region, country)
SELECT DISTINCT item_type, region, country FROM Staging_Sales;

SELECT * FROM Categories LIMIT 10;
```

	category_id [PK] integer	category_name character varying (255)	region character varying (255)	country character varying (255)
1	1	Household	Australia and Oceania	Papua New Guinea
2	2	Snacks	Australia and Oceania	East Timor
3	3	Vegetables	Sub-Saharan Africa	South Sudan
4	4	Beverages	Asia	Maldives
5	5	Cosmetics	Australia and Oceania	Australia
6	6	Personal Care	Sub-Saharan Africa	Sao Tome and Principe
7	7	Clothes	Asia	Singapore

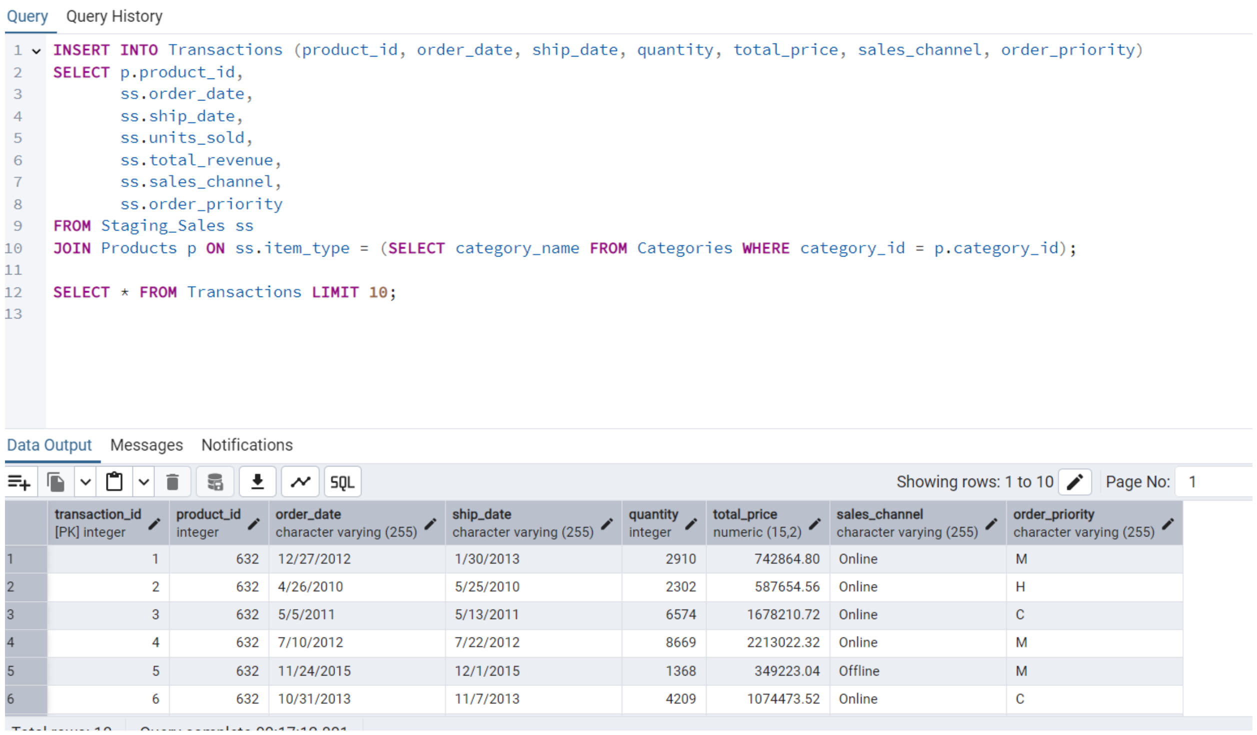
```
Total rows: 10    Query complete 00:00:00.175
```



```sql
INSERT INTO Products (category_id, price, stock)
SELECT c.category_id,
       AVG(ss.total_revenue / NULLIF(ss.units_sold, 0)),
       SUM(ss.units_sold)
FROM Staging_Sales ss
JOIN Categories c ON ss.item_type = c.category_name
GROUP BY c.category_id;

SELECT * FROM Products LIMIT 10;
```

	product_id [PK] integer	category_id integer	price numeric (15,2)	stock integer
1	1	1798	205.70	42254418
2	2	1489	81.73	41517766
3	3	1269	47.45	41514213
4	4	652	47.45	41514213
5	5	273	47.45	41514213
6	6	1560	255.28	41911620
7	7	51	154.06	41254836

Staphon Smith



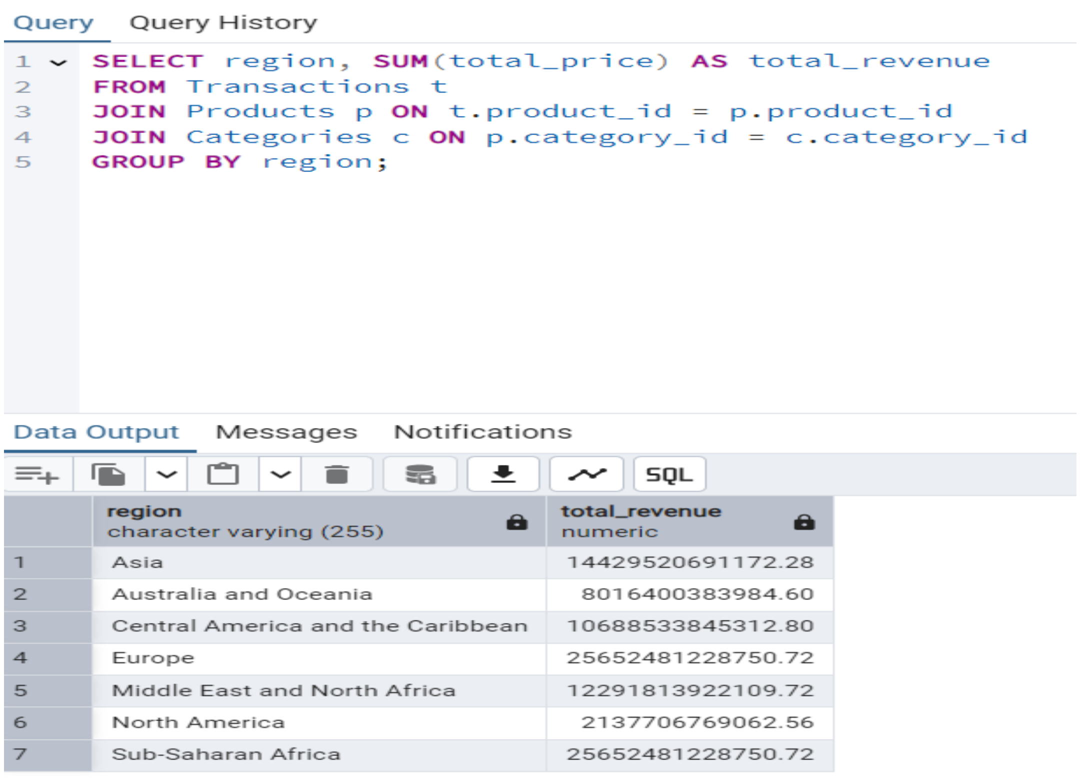
```sql
INSERT INTO Transactions (product_id, order_date, ship_date, quantity, total_price, sales_channel, order_priority)
SELECT p.product_id,
       ss.order_date,
       ss.ship_date,
       ss.units_sold,
       ss.total_revenue,
       ss.sales_channel,
       ss.order_priority
FROM Staging_Sales ss
JOIN Products p ON ss.item_type = (SELECT category_name FROM Categories WHERE category_id = p.category_id);

SELECT * FROM Transactions LIMIT 10;

```

	transaction_id [PK] integer	product_id integer	order_date character varying (255)	ship_date character varying (255)	quantity integer	total_price numeric (15,2)	sales_channel character varying (255)	order_priority character varying (255)
1	1	632	12/27/2012	1/30/2013	2910	742864.80	Online	M
2	2	632	4/26/2010	5/25/2010	2302	587654.56	Online	H
3	3	632	5/5/2011	5/13/2011	6574	1678210.72	Online	C
4	4	632	7/10/2012	7/22/2012	8669	2213022.32	Online	M
5	5	632	11/24/2015	12/1/2015	1368	349223.04	Offline	M
6	6	632	10/31/2013	11/7/2013	4209	1074473.52	Online	C

F3: Query Examples:

F3: Query 1 - Total Revenue by Region:

A query for *Total Revenue by Region* helps analyze which areas are performing the best or need improvement, so decisions about where to focus resources can be made. It's useful for planning strategies, budgeting, and forecasting future performance, while also identifying potential growth opportunities in certain regions. This type of query can also support compliance, reporting needs, and provide insights for comparing performance against competitors, helping the business stay on track with its goals.

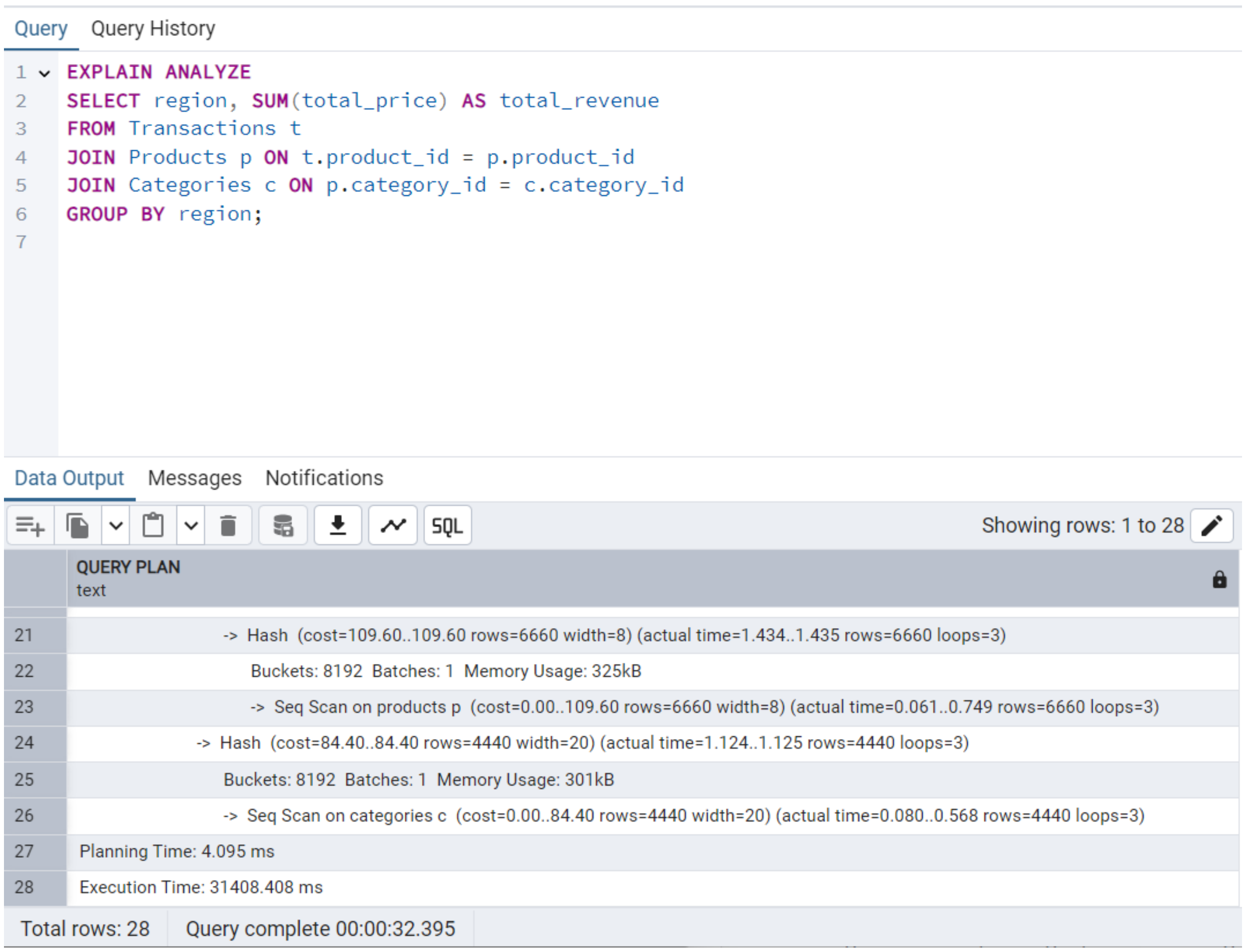


```sql
SELECT region, SUM(total_price) AS total_revenue
FROM Transactions t
JOIN Products p ON t.product_id = p.product_id
JOIN Categories c ON p.category_id = c.category_id
GROUP BY region;
```

	region character varying (255)	total_revenue numeric
1	Asia	14429520691172.28
2	Australia and Oceania	8016400383984.60
3	Central America and the Caribbean	10688533845312.80
4	Europe	25652481228750.72
5	Middle East and North Africa	12291813922109.72
6	North America	2137706769062.56
7	Sub-Saharan Africa	25652481228750.72

Staphon Smith

Before optimization:



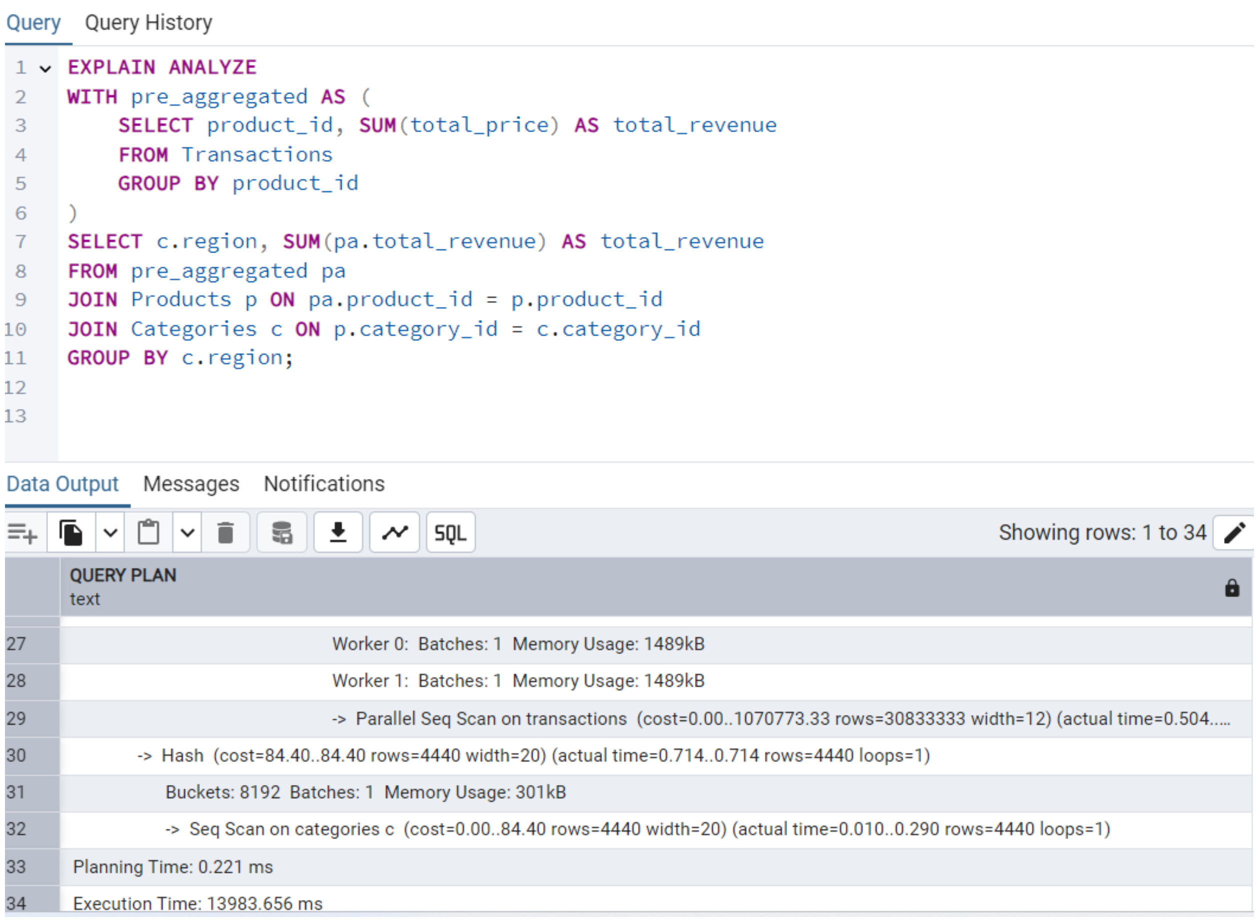
```sql
EXPLAIN ANALYZE
SELECT region, SUM(total_price) AS total_revenue
FROM Transactions t
JOIN Products p ON t.product_id = p.product_id
JOIN Categories c ON p.category_id = c.category_id
GROUP BY region;
```

```
21                    -> Hash  (cost=109.60..109.60 rows=6660 width=8) (actual time=1.434..1.435 rows=6660 loops=3)
22                          Buckets: 8192  Batches: 1  Memory Usage: 325kB
23                          -> Seq Scan on products p  (cost=0.00..109.60 rows=6660 width=8) (actual time=0.061..0.749 rows=6660 loops=3)
24                -> Hash  (cost=84.40..84.40 rows=4440 width=20) (actual time=1.124..1.125 rows=4440 loops=3)
25                      Buckets: 8192  Batches: 1  Memory Usage: 301kB
26                      -> Seq Scan on categories c  (cost=0.00..84.40 rows=4440 width=20) (actual time=0.080..0.568 rows=4440 loops=3)
27  Planning Time: 4.095 ms
28  Execution Time: 31408.408 ms
```

Total rows: 28   Query complete 00:00:32.395

Staphon Smith

After optimization:



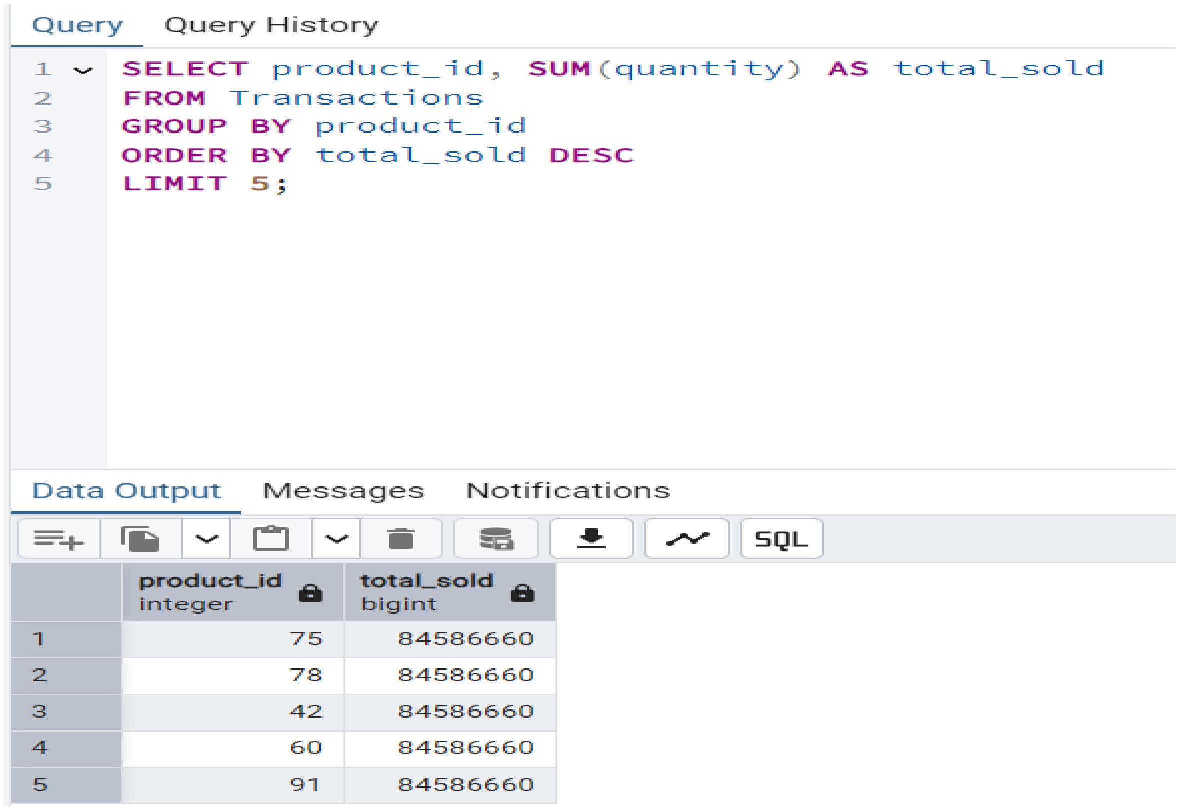
```
EXPLAIN ANALYZE
WITH pre_aggregated AS (
    SELECT product_id, SUM(total_price) AS total_revenue
    FROM Transactions
    GROUP BY product_id
)
SELECT c.region, SUM(pa.total_revenue) AS total_revenue
FROM pre_aggregated pa
JOIN Products p ON pa.product_id = p.product_id
JOIN Categories c ON p.category_id = c.category_id
GROUP BY c.region;
```

```
QUERY PLAN
text
27                          Worker 0:  Batches: 1  Memory Usage: 1489kB
28                          Worker 1:  Batches: 1  Memory Usage: 1489kB
29                          ->  Parallel Seq Scan on transactions  (cost=0.00..1070773.33 rows=30833333 width=12) (actual time=0.504.....
30              ->  Hash  (cost=84.40..84.40 rows=4440 width=20) (actual time=0.714..0.714 rows=4440 loops=1)
31                    Buckets: 8192  Batches: 1  Memory Usage: 301kB
32                    ->  Seq Scan on categories c  (cost=0.00..84.40 rows=4440 width=20) (actual time=0.010..0.290 rows=4440 loops=1)
33  Planning Time: 0.221 ms
34  Execution Time: 13983.656 ms
```

Runtime speed changed from 31408.4 ms to 13983.6 ms

F3: Query 2 - Top 5 Selling Products

A query for the *Top 5 Selling Products* helps identify which products are performing the best, giving insight into what customers prefer. This information is useful for making decisions about inventory, marketing, and production, ensuring resources are focused on the most profitable items. It also helps with forecasting future demand, planning promotions, and understanding trends. By knowing your top-performing products, you can drive more sales, increase profitability, and align strategies with what works best for the business.

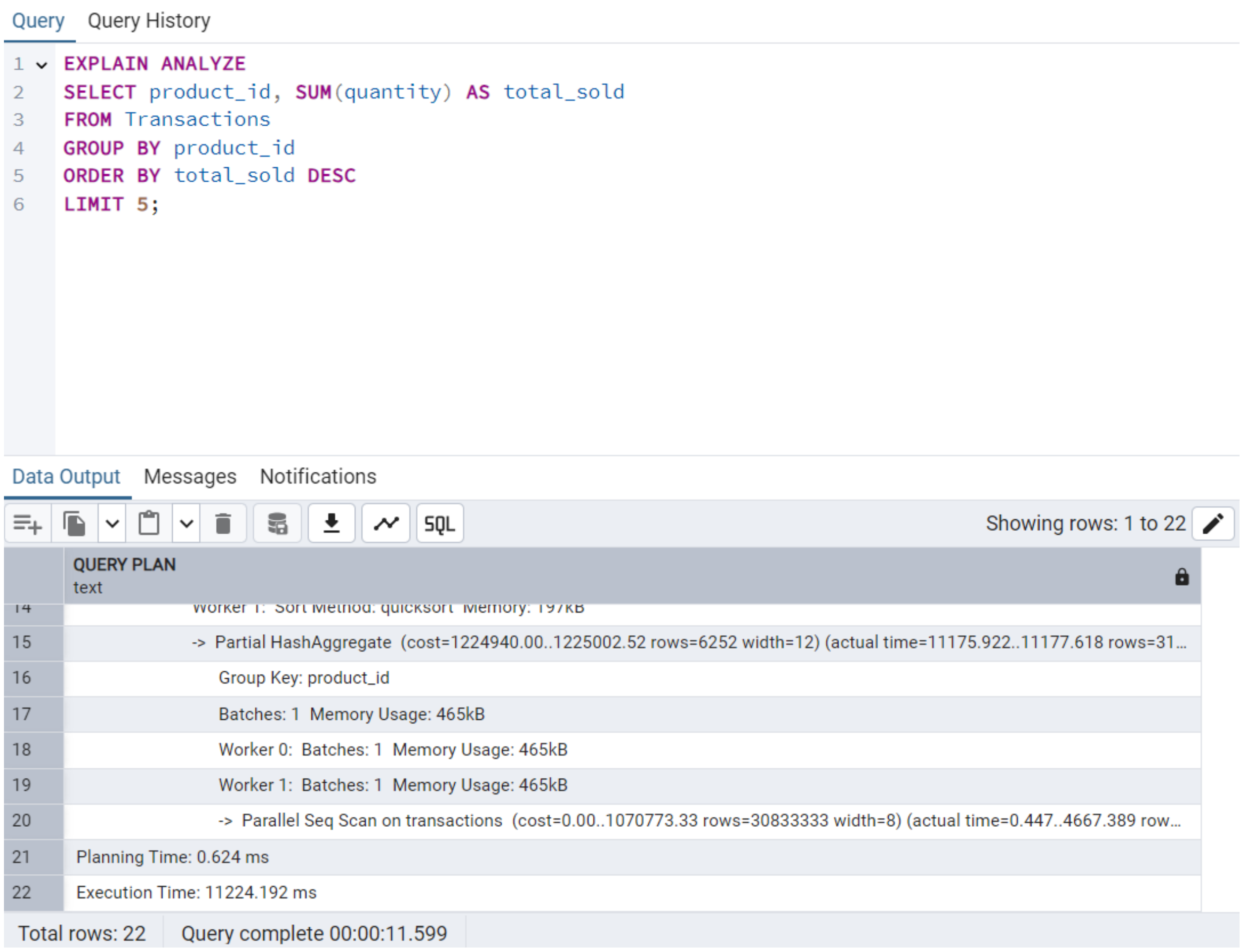


```
SELECT product_id, SUM(quantity) AS total_sold
FROM Transactions
GROUP BY product_id
ORDER BY total_sold DESC
LIMIT 5;
```

	product_id integer	total_sold bigint
1	75	84586660
2	78	84586660
3	42	84586660
4	60	84586660
5	91	84586660

Staphon Smith

Before Optimization:



Query　Query History

```
EXPLAIN ANALYZE
SELECT product_id, SUM(quantity) AS total_sold
FROM Transactions
GROUP BY product_id
ORDER BY total_sold DESC
LIMIT 5;
```

Data Output　Messages　Notifications

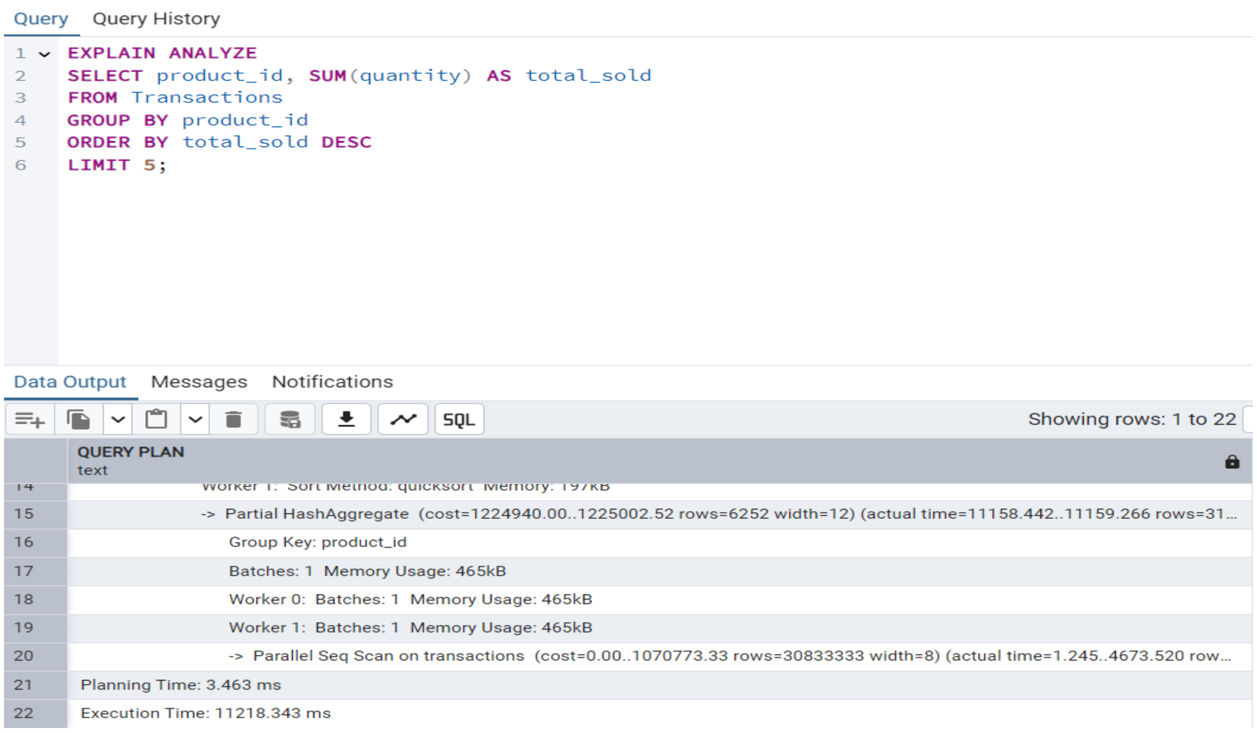
Showing rows: 1 to 22

```
QUERY PLAN
text
14                  Worker 1:  Sort Method: quicksort  Memory: 197kB
15                  ->  Partial HashAggregate  (cost=1224940.00..1225002.52 rows=6252 width=12) (actual time=11175.922..11177.618 rows=31...
16                        Group Key: product_id
17                        Batches: 1  Memory Usage: 465kB
18                        Worker 0:  Batches: 1  Memory Usage: 465kB
19                        Worker 1:  Batches: 1  Memory Usage: 465kB
20                        ->  Parallel Seq Scan on transactions  (cost=0.00..1070773.33 rows=30833333 width=8) (actual time=0.447..4667.389 row...
21  Planning Time: 0.624 ms
22  Execution Time: 11224.192 ms
```

Total rows: 22　Query complete 00:00:11.599

Staphon Smith

After optimization:



Query Query History

```
EXPLAIN ANALYZE
SELECT product_id, SUM(quantity) AS total_sold
FROM Transactions
GROUP BY product_id
ORDER BY total_sold DESC
LIMIT 5;
```

Data Output Messages Notifications

Showing rows: 1 to 22

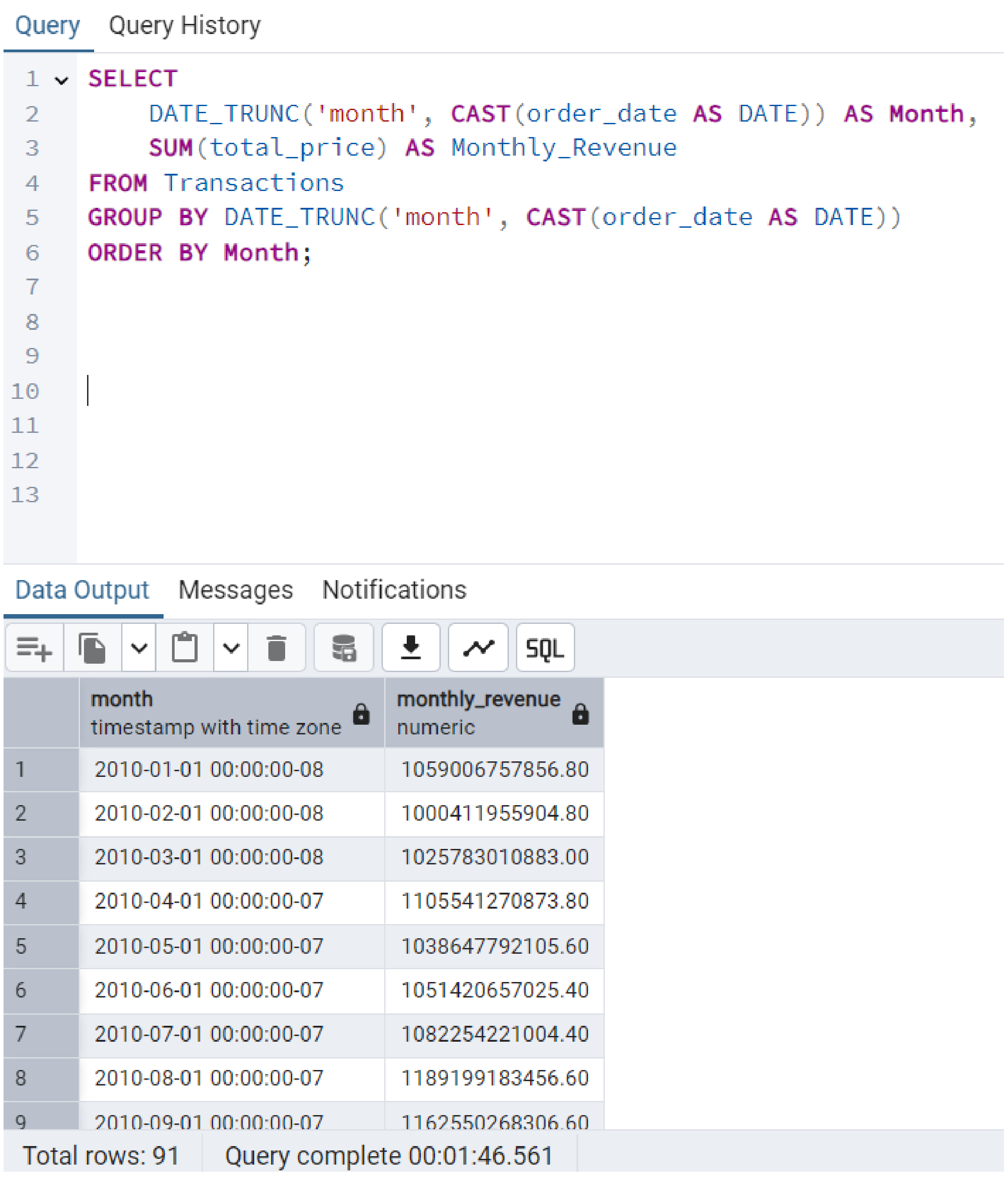
QUERY PLAN
text

```
14                Worker 1:  Sort Method: quicksort  Memory: 197kB
15                -> Partial HashAggregate  (cost=1224940.00..1225002.52 rows=6252 width=12) (actual time=11158.442..11159.266 rows=31...
16                    Group Key: product_id
17                    Batches: 1  Memory Usage: 465kB
18                    Worker 0:  Batches: 1  Memory Usage: 465kB
19                    Worker 1:  Batches: 1  Memory Usage: 465kB
20                    -> Parallel Seq Scan on transactions  (cost=0.00..1070773.33 rows=30833333 width=8) (actual time=1.245..4673.520 row...
21  Planning Time: 3.463 ms
22  Execution Time: 11218.343 ms
```

Runtime speed changed from 11224.1 ms to 11218.3 ms

F3: Query 3 - Analysis of Monthly Revenues

A query for the *Analysis of Monthly Revenues* helps track how the business is performing over time and spot trends or patterns in revenue. This information is useful for identifying peak sales periods, managing cash flow, and setting realistic goals for growth. It also helps in comparing month-to-month performance, forecasting future revenue, and planning strategies to address any dips or maximize high-performing months. Understanding monthly revenue trends is key to staying on top of the business's financial health and making informed decisions.



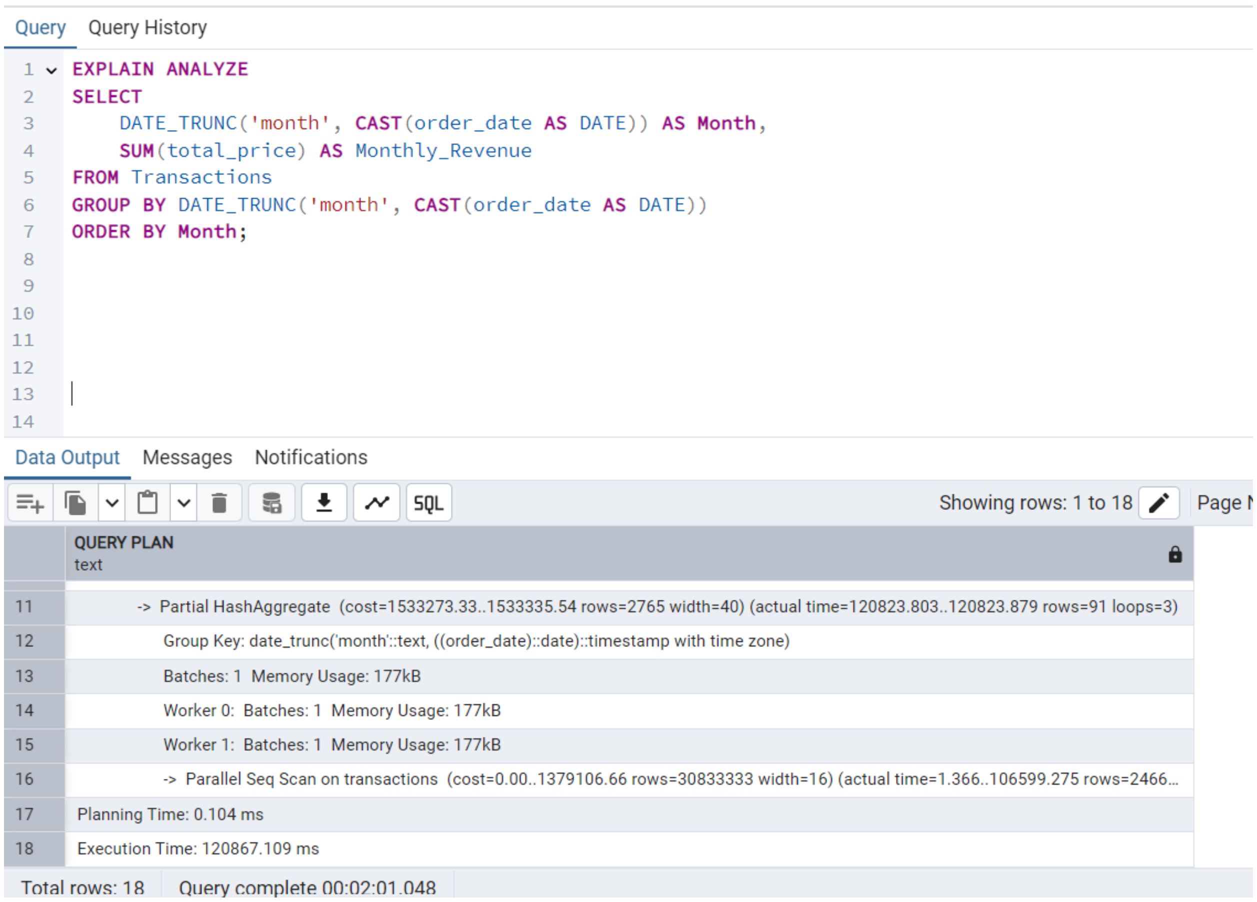
```
SELECT
    DATE_TRUNC('month', CAST(order_date AS DATE)) AS Month,
    SUM(total_price) AS Monthly_Revenue
FROM Transactions
GROUP BY DATE_TRUNC('month', CAST(order_date AS DATE))
ORDER BY Month;
```

	month timestamp with time zone	monthly_revenue numeric
1	2010-01-01 00:00:00-08	1059006757856.80
2	2010-02-01 00:00:00-08	1000411955904.80
3	2010-03-01 00:00:00-08	1025783010883.00
4	2010-04-01 00:00:00-07	1105541270873.80
5	2010-05-01 00:00:00-07	1038647792105.60
6	2010-06-01 00:00:00-07	1051420657025.40
7	2010-07-01 00:00:00-07	1082254221004.40
8	2010-08-01 00:00:00-07	1189199183456.60
9	2010-09-01 00:00:00-07	1162550268306.60

Total rows: 91   Query complete 00:01:46.561

Staphon Smith

Before optimization:



Query Query History

```
EXPLAIN ANALYZE
SELECT
    DATE_TRUNC('month', CAST(order_date AS DATE)) AS Month,
    SUM(total_price) AS Monthly_Revenue
FROM Transactions
GROUP BY DATE_TRUNC('month', CAST(order_date AS DATE))
ORDER BY Month;
```

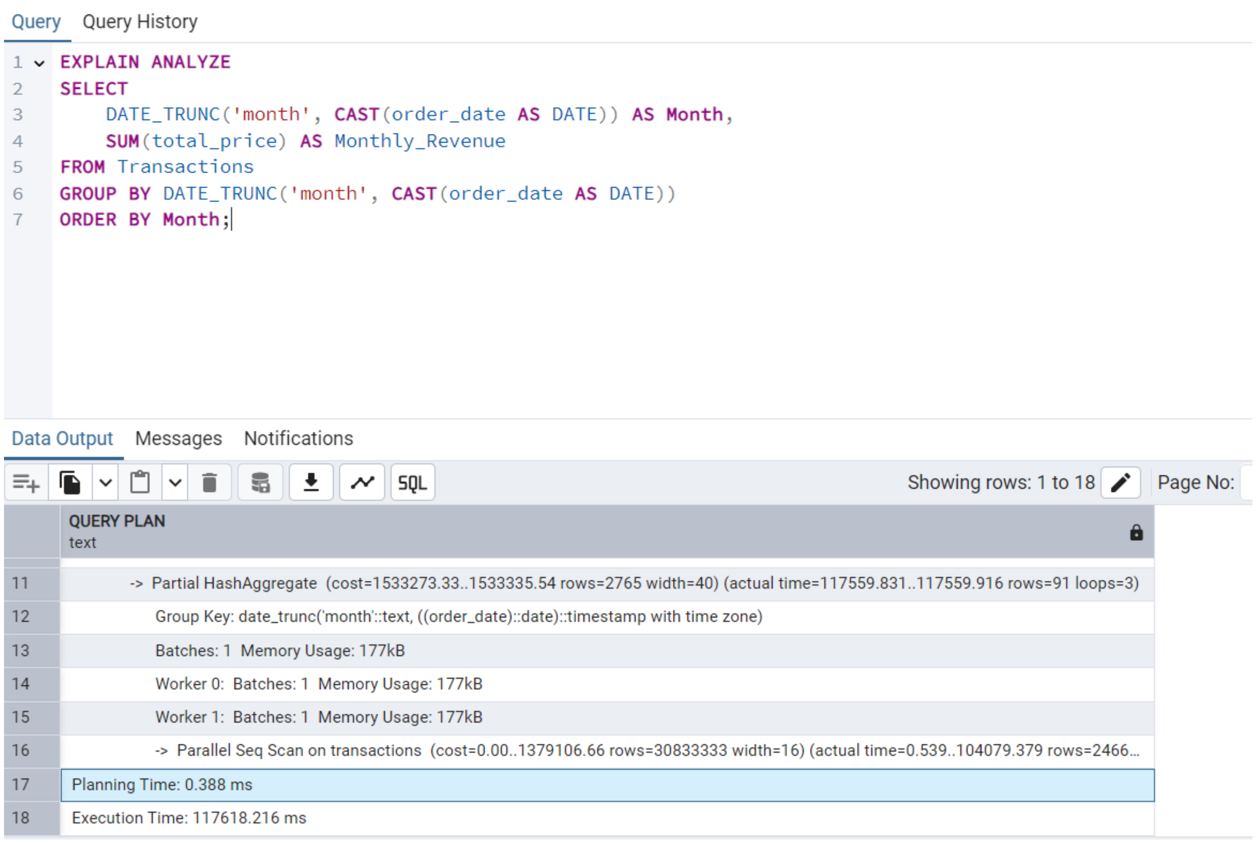
Data Output Messages Notifications

Showing rows: 1 to 18 Page

	QUERY PLAN text
11	-> Partial HashAggregate (cost=1533273.33..1533335.54 rows=2765 width=40) (actual time=120823.803..120823.879 rows=91 loops=3)
12	Group Key: date_trunc('month'::text, ((order_date)::date)::timestamp with time zone)
13	Batches: 1 Memory Usage: 177kB
14	Worker 0: Batches: 1 Memory Usage: 177kB
15	Worker 1: Batches: 1 Memory Usage: 177kB
16	-> Parallel Seq Scan on transactions (cost=0.00..1379106.66 rows=30833333 width=16) (actual time=1.366..106599.275 rows=2466...
17	Planning Time: 0.104 ms
18	Execution Time: 120867.109 ms

Total rows: 18 Query complete 00:02:01.048

After optimization:


Query Query History

```
EXPLAIN ANALYZE
SELECT
    DATE_TRUNC('month', CAST(order_date AS DATE)) AS Month,
    SUM(total_price) AS Monthly_Revenue
FROM Transactions
GROUP BY DATE_TRUNC('month', CAST(order_date AS DATE))
ORDER BY Month;
```

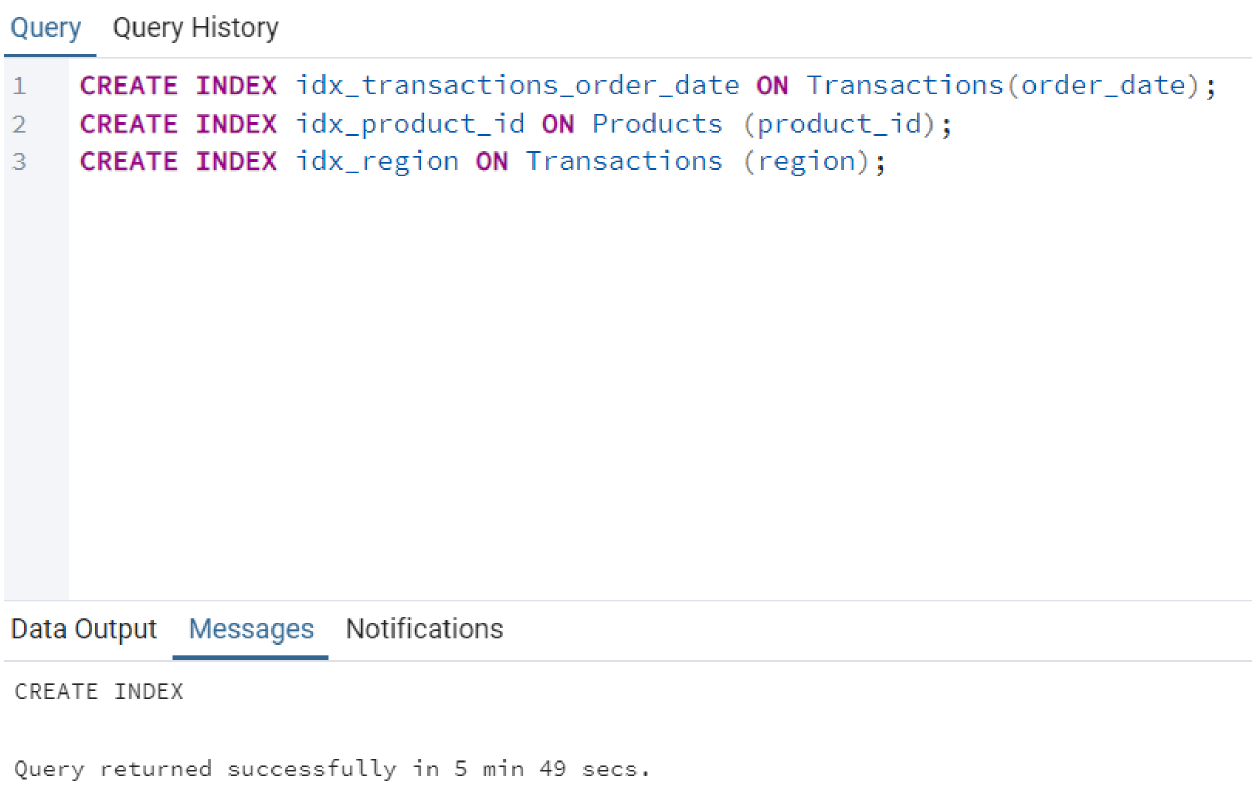
Data Output Messages Notifications

Showing rows: 1 to 18 Page No:

	QUERY PLAN text
11	-> Partial HashAggregate (cost=1533273.33..1533335.54 rows=2765 width=40) (actual time=117559.831..117559.916 rows=91 loops=3)
12	Group Key: date_trunc('month'::text, ((order_date)::date)::timestamp with time zone)
13	Batches: 1 Memory Usage: 177kB
14	Worker 0: Batches: 1 Memory Usage: 177kB
15	Worker 1: Batches: 1 Memory Usage: 177kB
16	-> Parallel Seq Scan on transactions (cost=0.00..1379106.66 rows=30833333 width=16) (actual time=0.539..104079.379 rows=2466...
17	Planning Time: 0.388 ms
18	Execution Time: 117618.216 ms

Runtime speed changed from 120867.1 ms to 117618.2 ms

F4: Optimization Techniques:



```sql
CREATE INDEX idx_transactions_order_date ON Transactions(order_date);
CREATE INDEX idx_product_id ON Products (product_id);
CREATE INDEX idx_region ON Transactions (region);
```

```
CREATE INDEX

Query returned successfully in 5 min 49 secs.
```

G: Panopto Video

H: Sources

No sources, web or otherwise, were used in the creation of the project and its follow up report.