

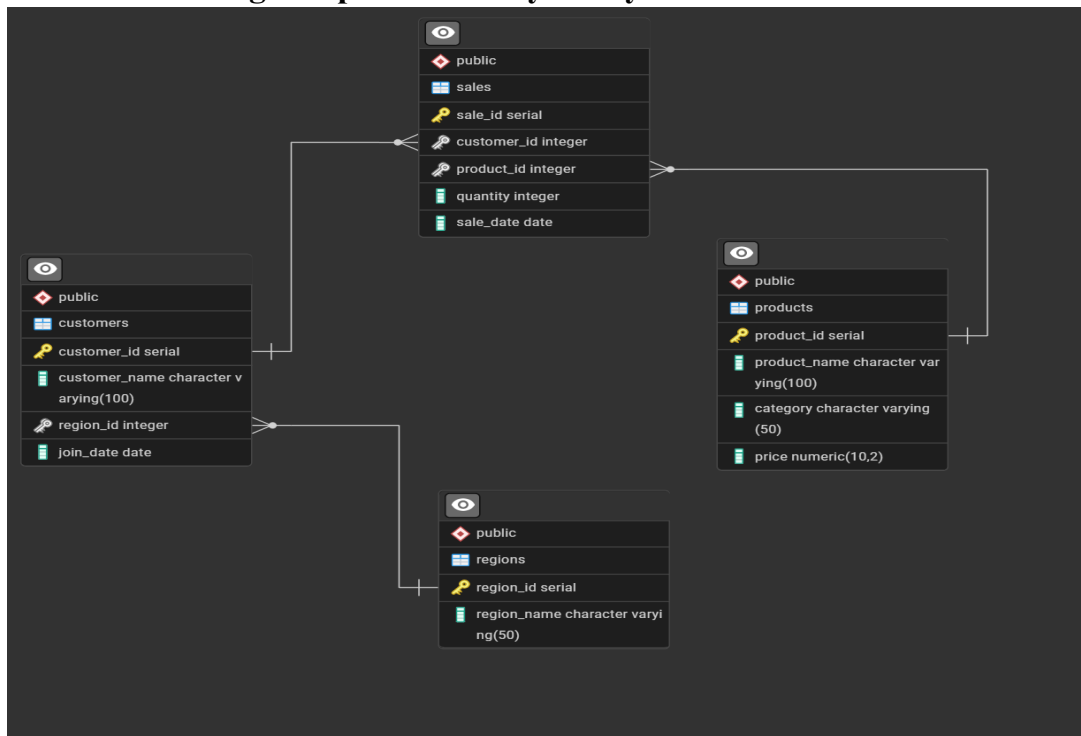
LagosExpress-Grocery FMCG Case Study – PostgreSQL Capstone Project (An Hypothetical Company)

Insights to the Analysis

Background:

LagosExpress-Grocery is an FMCG company that sells consumer goods (beverages, snacks, and toiletries) across various regions in Nigeria. The company wants to analyze its sales performance, customer distribution, and product profitability using PostgreSQL. They maintain 4 main tables: **customers**, **products**, **sales**, and **regions**.

ERD For our LagosExpress Grocery Analysis



Problem Questions and Solutions

Q1. Use a JOIN to find the total amount each customer spent.

SELECT

c.customer_name,

SUM(p.price * s.quantity) AS total_spent

FROM sales s

JOIN customers c

ON s.customer_id = c.customer_id

JOIN products p

ON s.product_id = p.product_id

GROUP BY c.customer_name

ORDER BY total_spent DESC;

Code:

The screenshot shows a PostgreSQL query editor interface. The top bar indicates the connection is 'LagosExpress_Grocery/postgres@PostgreSQL 17'. The query editor contains the following SQL code:

```
1 SELECT
2     c.customer_name,
3     SUM(p.price * s.quantity) AS total_spent
4 FROM
5     sales s
6 JOIN
7     customers c
8 ON s.customer_id = c.customer_id
9 JOIN
10    products p
11 ON s.product_id = p.product_id
12 GROUP BY
13     c.customer_name
14 ORDER BY
15     total_spent DESC;
16
```

Below the query editor, the 'Data Output' tab is active, showing the results of the query. The results are displayed in a table with two columns: 'customer_name' and 'total_spent'. The table shows 5 rows of data, with the first row being Mary Johnson with a total spent of 3500.00. The status bar at the bottom indicates 'Total rows: 5' and 'Query complete 00:00:00.434'.

	customer_name character varying (100)	total_spent numeric
1	Mary Johnson	3500.00

Code Result:

This screenshot shows the same PostgreSQL query editor interface as the previous one, but with more rows of results visible in the 'Data Output' tab. The SQL query is identical. The results table now shows 5 rows of data, with the first row being Mary Johnson with a total spent of 3500.00, and the last row being Fatima Bello with a total spent of 1200.00. The status bar at the bottom indicates 'Total rows: 5' and 'Query complete 00:00:00.434'.

	customer_name character varying (100)	total_spent numeric
1	Mary Johnson	3500.00
2	Musa Ibrahim	3500.00
3	Chinedu Okafor	2050.00
4	Grace Udo	1800.00
5	Fatima Bello	1200.00

Insight:

- Mary Johnson spent ₦3,500 (Spent the highest)
- Fatima Bello spent ₦1200.00 (Spent the lowest)
- Customers' total spending helps SwiftGrocer identify high-value clients.

Q2. Use a LEFT JOIN to find all customers, even those who haven't made any purchase.

SELECT

c.customer_name,

s.sale_id

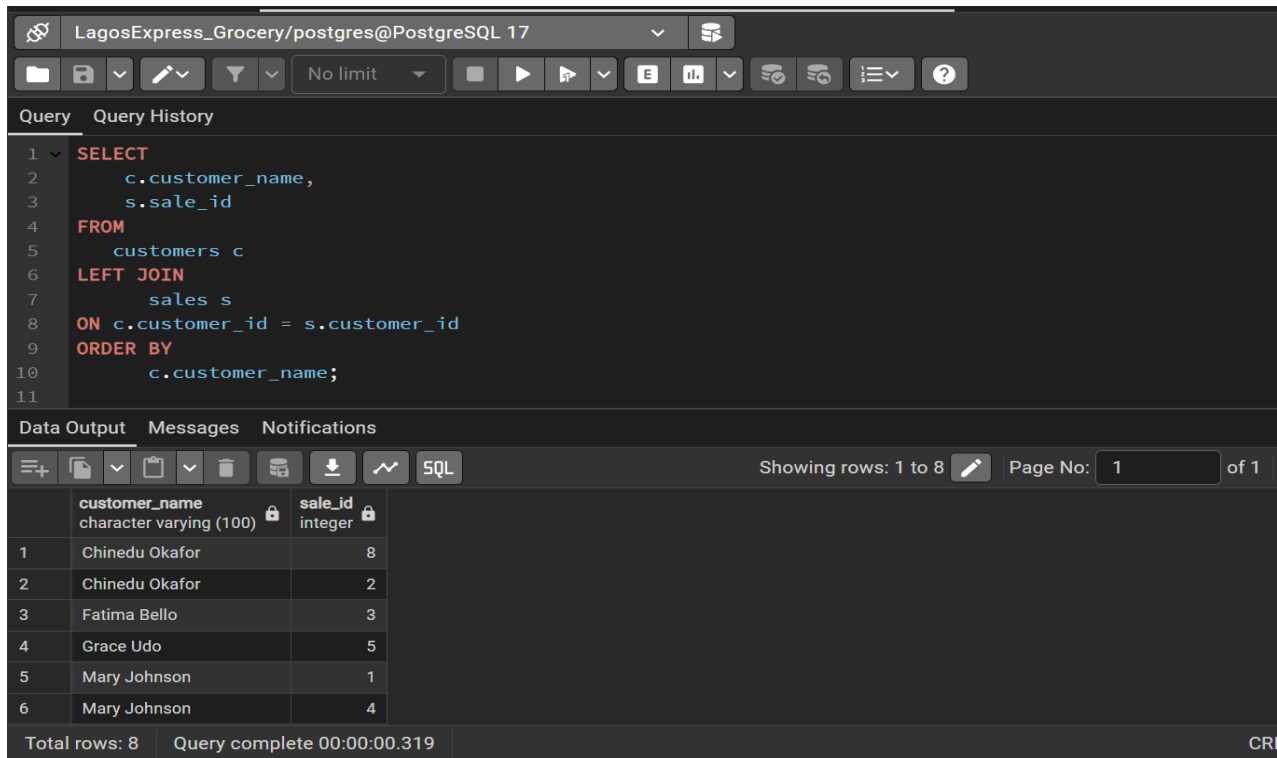
FROM customers c

LEFT JOIN sales s

ON c.customer_id = s.customer_id

ORDER BY c.customer_name;

Code:



The screenshot shows a PostgreSQL query editor interface. The query is as follows:

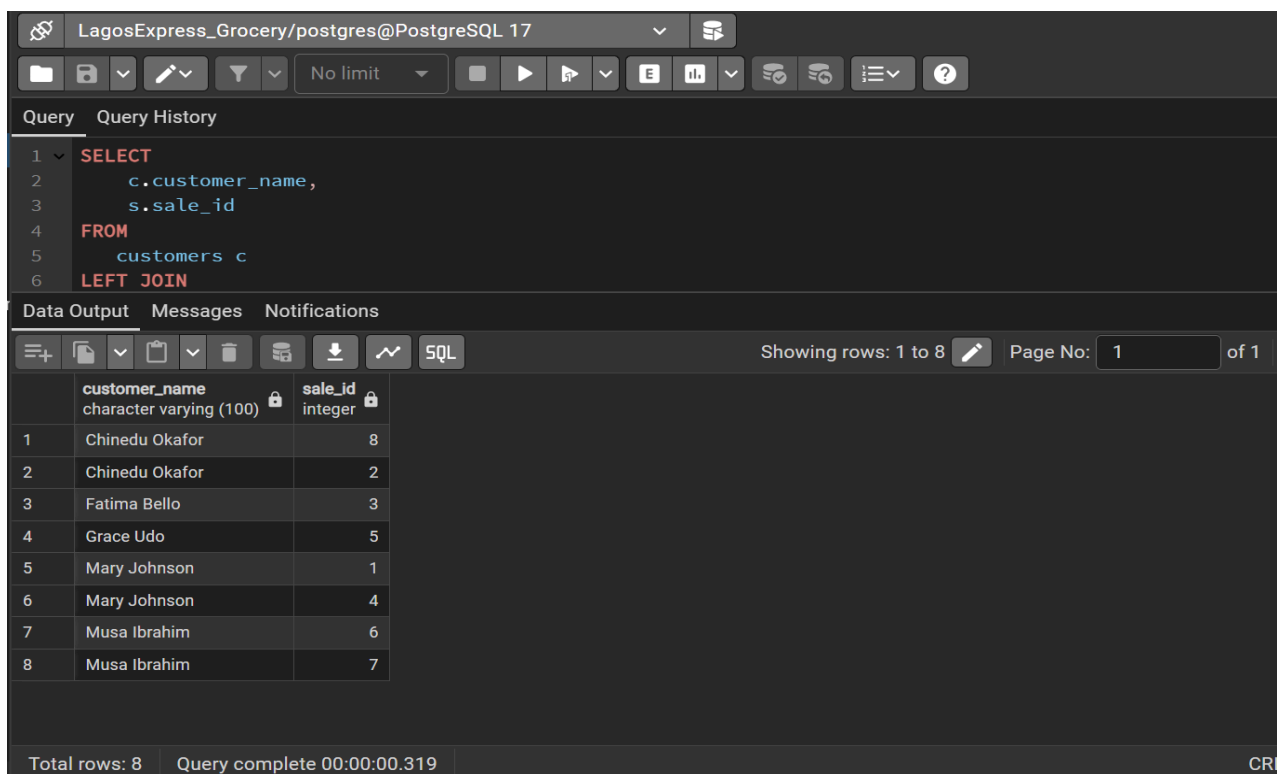
```
1 SELECT
2     c.customer_name,
3     s.sale_id
4 FROM
5     customers c
6 LEFT JOIN
7     sales s
8 ON c.customer_id = s.customer_id
9 ORDER BY
10    c.customer_name;
11
```

The results are displayed in a table with 8 rows:

	customer_name	sale_id
1	Chinedu Okafor	8
2	Chinedu Okafor	2
3	Fatima Bello	3
4	Grace Udo	5
5	Mary Johnson	1
6	Mary Johnson	4

Total rows: 8. Query complete 00:00:00.319.

Code Result:



The screenshot shows a PostgreSQL query editor interface. The query is as follows:

```
1 SELECT
2     c.customer_name,
3     s.sale_id
4 FROM
5     customers c
6 LEFT JOIN
7     sales s
8 ON c.customer_id = s.customer_id
9 ORDER BY
10    c.customer_name;
11
```

The results are displayed in a table with 8 rows:

	customer_name	sale_id
1	Chinedu Okafor	8
2	Chinedu Okafor	2
3	Fatima Bello	3
4	Grace Udo	5
5	Mary Johnson	1
6	Mary Johnson	4
7	Musa Ibrahim	6
8	Musa Ibrahim	7

Total rows: 8. Query complete 00:00:00.319.

Insight:

- If any **sale_id** is **NULL**, that customer hasn't made a purchase. Helps identify inactive customers.
- According to the result all customers made a purchase from the LagosExpress Grocery

Q3. Use a RIGHT JOIN to display all products and their sales (even if not sold).

SELECT

p.product_name,

s.quantity

FROM sales s

RIGHT JOIN products p

ON s.product_id = p.product_id

ORDER BY p.product_name;

Code:

The screenshot shows a PostgreSQL query editor interface. The top bar indicates the connection is 'LagosExpress_Grocery/postgres@PostgreSQL 17'. Below the toolbar, the 'Query' tab is active, displaying the following SQL query:

```
1 SELECT
2     p.product_name,
3     s.quantity
4 FROM
5     sales s
6 RIGHT JOIN
7     products p
8 ON s.product_id = p.product_id
9 ORDER BY
10    p.product_name;
```

The 'Data Output' tab is active, showing the results of the query. The results are displayed in a table with two columns: 'product_name' (character varying (100)) and 'quantity' (integer). The table contains 8 rows of data:

	product_name	quantity
1	AquaPure Water	12
2	AquaPure Water	15
3	ChocoBite	10
4	ChocoBite	7
5	FreshSoap	8
6	NutiMilk	6
Total rows: 8		

The bottom status bar shows 'Query complete 00:00:00.395' and 'Page No: 1 of 1'.

Code Result:

The screenshot shows a PostgreSQL query editor interface. At the top, the connection is 'LagosExpress_Grocery/postgres@PostgreSQL 17'. Below the toolbar, the query is as follows:

```
1 SELECT
2     p.product_name,
3     s.quantity
4 FROM
5     sales s
6 RIGHT JOIN
7     products p
```

The 'Data Output' tab is active, showing a table with 8 rows. The columns are 'product_name' (character varying (100)) and 'quantity' (integer). The data is as follows:

	product_name	quantity
1	AquaPure Water	12
2	AquaPure Water	15
3	ChocoBite	10
4	ChocoBite	7
5	FreshSoap	8
6	NutiMilk	6
7	Sparkle Cola	10
8	Sparkle Cola	5

At the bottom, the status bar indicates 'Total rows: 8' and 'Query complete 00:00:00.395'.

Insight:

- If any **quantity** is **NULL**, that product hasn't been sold. This is useful for stock control.
- At least one of every product in the LagosExpress Grocery was sold

Q4. Use AGGREGATION to find total sales revenue per category.

SELECT

p.category,

SUM(p.price * s.quantity) AS total_revenue

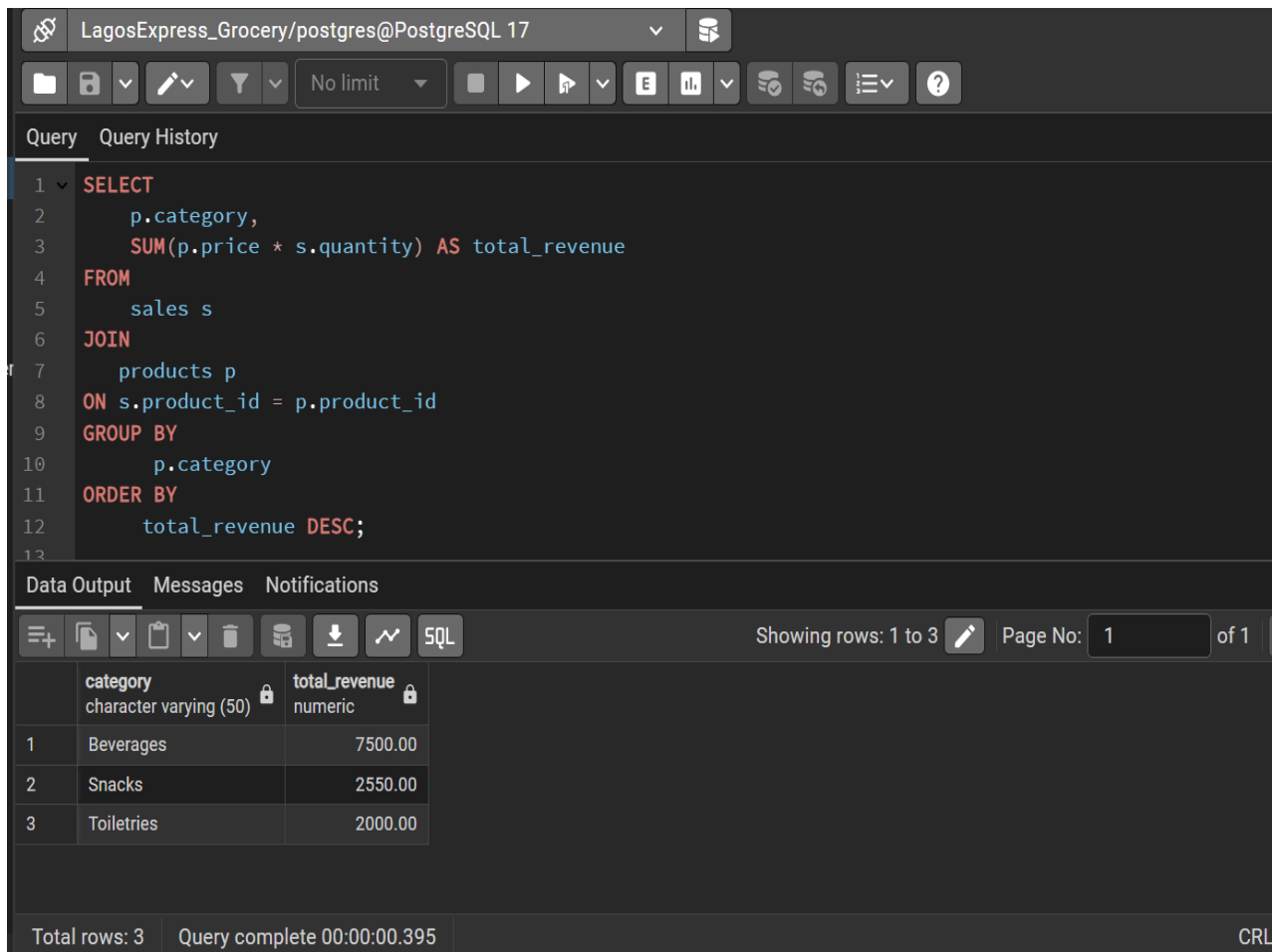
FROM sales s

JOIN products p ON s.product_id = p.product_id

GROUP BY p.category

ORDER BY total_revenue DESC;

Code and Result:



The screenshot shows a PostgreSQL query editor interface. At the top, the database connection is 'LagosExpress_Grocery/postgres@PostgreSQL 17'. Below the connection bar, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query. Below the query, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing the results of the query in a table format. The table has two columns: 'category' (character varying (50)) and 'total_revenue' (numeric). The results show three rows: 'Beverages' with a total revenue of 7500.00, 'Snacks' with 2550.00, and 'Toiletries' with 2000.00. At the bottom, a status bar indicates 'Total rows: 3' and 'Query complete 00:00:00.395'.

```
1 SELECT
2     p.category,
3     SUM(p.price * s.quantity) AS total_revenue
4 FROM
5     sales s
6 JOIN
7     products p
8 ON s.product_id = p.product_id
9 GROUP BY
10    p.category
11 ORDER BY
12    total_revenue DESC;
```

	category character varying (50)	total_revenue numeric
1	Beverages	7500.00
2	Snacks	2550.00
3	Toiletries	2000.00

Total rows: 3 Query complete 00:00:00.395 CRL

Insight:

- The beverages category led in sales.
- Management can focus marketing on best-selling categories.

Q5. Use DATE FUNCTIONS to find monthly sales totals for 2025.

```
SELECT
    DATE_TRUNC('month', sale_date) AS month,
    SUM(p.price * s.quantity) AS monthly_revenue
FROM sales s
JOIN products p
ON s.product_id = p.product_id
WHERE sale_date BETWEEN '2025-01-01' AND '2025-12-31'
GROUP BY month
ORDER BY monthly_revenue DESC;
```

Code:

LagosExpress_Grocery/postgres@PostgreSQL 17

No limit

Query Query History

```

1 SELECT
2     DATE_TRUNC('month', sale_date) AS month,
3     SUM(p.price * s.quantity) AS monthly_revenue
4 FROM
5     sales s
6 JOIN
7     products p
8 ON s.product_id = p.product_id
9 WHERE
10    sale_date
11        BETWEEN '2025-01-01' AND '2025-12-31'
12 GROUP BY
13     month
14 ORDER BY
15     monthly_revenue DESC;

```

Data Output Messages Notifications

Showing rows: 1 to 4 Page No: 1 of 1

	month timestamp with time zone 🔒	monthly_revenue numeric 🔒
1	2025-03-01 00:00:00+01	3800.00
2	2025-04-01 00:00:00+01	2550.00
3	2025-02-01 00:00:00+01	2000.00

Total rows: 4 Query complete 00:00:00.275

Code Result:

LagosExpress_Grocery/postgres@PostgreSQL 17

📁 🗄️ ⚙️ 🔍 No limit 🎛️ ▶️ 🖨️ E 📊 🔄 🔄 ☰ ?

Query Query History

```

1 SELECT
2     DATE_TRUNC('month', sale_date) AS month,
3     SUM(p.price * s.quantity) AS monthly_revenue
4 FROM
5     sales s
6 JOIN
7     products p
8 ON s.product_id = p.product_id
9 WHERE
10    sale_date
11        BETWEEN '2025-01-01' AND '2025-12-31'
12 GROUP BY
13    month
  
```

Data Output Messages Notifications

⌵ 🗄️ ⚙️ 🗑️ 🗄️ 📥 📈 SQL Showing rows: 1 to 4 Page No: 1 of 1

	month timestamp with time zone 🔒	monthly_revenue numeric 🔒
1	2025-03-01 00:00:00+01	3800.00
2	2025-04-01 00:00:00+01	2550.00
3	2025-02-01 00:00:00+01	2000.00
4	2025-01-01 00:00:00+01	1200.00

Total rows: 4 Query complete 00:00:00.275

Insight:

Shows sales growth or increase and decline month-by-month. This helps identify seasonal patterns.

LagosExpress Grocery experienced a decline in sales between March 2025 to April 2025

Q6. Use CASE STATEMENT to classify customers based on spending.

SELECT

c.customer_name,

SUM(p.price * s.quantity) AS total_spent,

CASE

WHEN SUM(p.price * s.quantity) > 3000 THEN 'Premium'

WHEN SUM(p.price * s.quantity) BETWEEN 1500 AND 3000 THEN

'Regular'

ELSE 'Low Value'

END AS customer_segment

FROM sales s

JOIN customers c

ON s.customer_id = c.customer_id

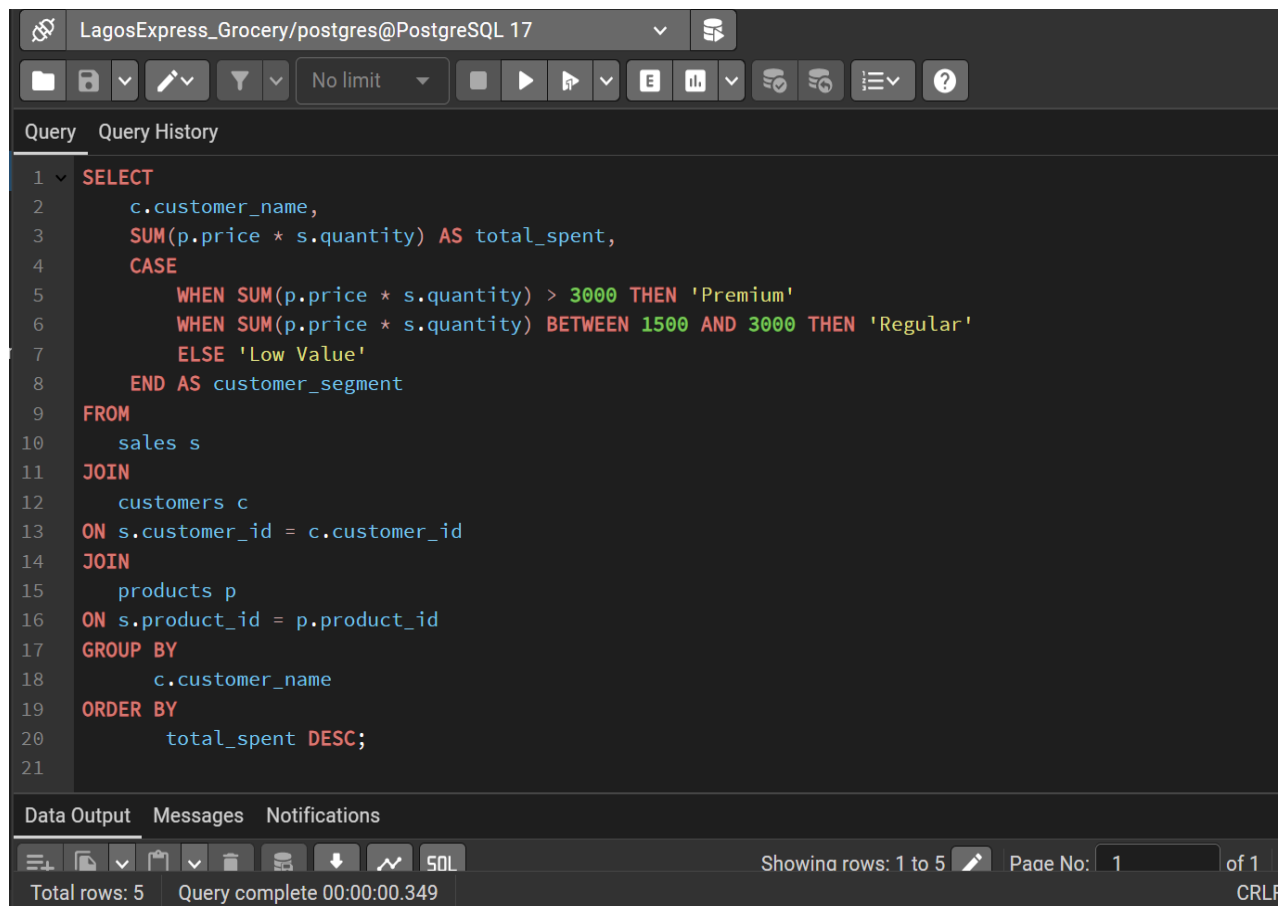
JOIN products p

ON s.product_id = p.product_id

GROUP BY c.customer_name

ORDER BY total_spent DESC;

Code:



The screenshot shows a PostgreSQL query editor interface. The top bar displays the connection name 'LagosExpress_Grocery/postgres@PostgreSQL 17'. Below the toolbar, the 'Query' tab is active, showing the following SQL code:

```
1 SELECT
2     c.customer_name,
3     SUM(p.price * s.quantity) AS total_spent,
4     CASE
5         WHEN SUM(p.price * s.quantity) > 3000 THEN 'Premium'
6         WHEN SUM(p.price * s.quantity) BETWEEN 1500 AND 3000 THEN 'Regular'
7         ELSE 'Low Value'
8     END AS customer_segment
9 FROM
10     sales s
11 JOIN
12     customers c
13 ON s.customer_id = c.customer_id
14 JOIN
15     products p
16 ON s.product_id = p.product_id
17 GROUP BY
18     c.customer_name
19 ORDER BY
20     total_spent DESC;
21
```

The bottom of the interface shows the 'Data Output' tab, which indicates 'Total rows: 5' and 'Query complete 00:00:00.349'. The status bar at the very bottom shows 'Showing rows: 1 to 5', 'Page No: 1', and 'of 1'.

Code Result:

The screenshot shows a PostgreSQL query editor interface. The top bar indicates the connection is 'LagosExpress_Grocery/postgres@PostgreSQL 17'. Below the toolbar, the 'Query' tab is active, displaying a SQL query. The query selects customer names, calculates total spent, and categorizes customers into segments based on their total spent. The 'Data Output' tab shows the results of the query, which are 5 rows. The status bar at the bottom indicates 'Total rows: 5' and 'Query complete 00:00:00.349'.

```
1 SELECT
2   c.customer_name,
3   SUM(p.price * s.quantity) AS total_spent,
4   CASE
5     WHEN SUM(p.price * s.quantity) > 3000 THEN 'Premium'
6     WHEN SUM(p.price * s.quantity) BETWEEN 1500 AND 3000 THEN 'Regular'
7     ELSE 'Low Value'
8   END AS customer_segment
9 FROM
10  sales s
11 JOIN
```

	customer_name character varying (100)	total_spent numeric	customer_segment text
1	Mary Johnson	3500.00	Premium
2	Musa Ibrahim	3500.00	Premium
3	Chinedu Okafor	2050.00	Regular
4	Grace Udo	1800.00	Regular
5	Fatima Bello	1200.00	Low Value

Total rows: 5 Query complete 00:00:00.349 CRLF

Insight:

- Classifies customers for loyalty programs
- *Mary Johnson* was a Premium Customer

Q7. Use WITH STATEMENT to find the best-performing region.

```
WITH region_sales AS (  
    SELECT  
        r.region_name,  
        SUM(p.price * s.quantity) AS total_sales  
    FROM sales s  
    JOIN customers c  
    ON s.customer_id = c.customer_id  
    JOIN regions r  
    ON c.region_id = r.region_id  
    JOIN products p  
    ON s.product_id = p.product_id  
    GROUP BY r.region_name  
)  
SELECT *  
FROM region_sales  
ORDER BY total_sales DESC;
```

Code:

LagosExpress_Grocery/postgres@PostgreSQL 17

Query

Query History

```
1 WITH region_sales AS (  
2     SELECT  
3         r.region_name,  
4         SUM(p.price * s.quantity) AS total_sales  
5     FROM  
6         sales s  
7     JOIN  
8         customers c  
9     ON s.customer_id = c.customer_id  
10    JOIN  
11        regions r  
12    ON c.region_id = r.region_id  
13    JOIN  
14        products p  
15    ON s.product_id = p.product_id  
16    GROUP BY  
17        r.region_name  
18 )  
19 SELECT *  
20 FROM  
21     region_sales  
22 ORDER BY total_sales DESC;
```

Data Output Messages Notifications

Total rows: 4 Query complete 00:00:00.276 CRLF

Code Result:

LagosExpress_Grocery/postgres@PostgreSQL 17

Query

Query History

```
1 WITH region_sales AS (  
2     SELECT  
3         r.region_name,  
4         SUM(p.price * s.quantity) AS total_sales  
5     FROM  
6         sales s  
7     JOIN  
8         customers c  
9     ON s.customer_id = c.customer_id  
10    JOIN  
11        regions r  
12    ON c.region_id = r.region_id  
13    JOIN
```

Data Output Messages Notifications

Showing rows: 1 to 4 Page No: 1 of 1

	region_name character varying (50)	total_sales numeric
1	North	4700.00
2	West	3500.00
3	South	2050.00
4	East	1800.00

Total rows: 4 Query complete 00:00:00.142 CRLF

Insight:

- Identifies which region contributes most to total revenue. This supports resource allocation.
- The best performing region to our sales was the North. Which is the Northern part of Lagos

Q8. Use UNION to list both product categories and customer regions (as a combined business domain list).

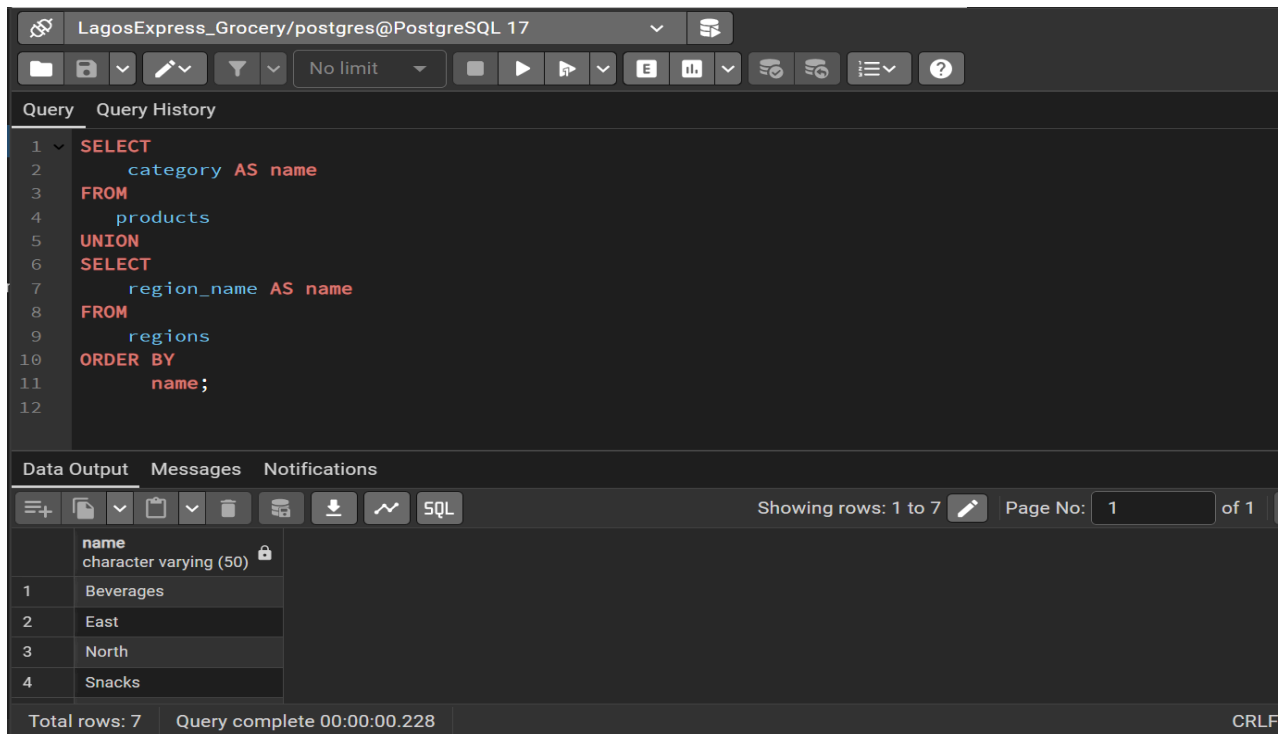
SELECT category AS name FROM products

UNION

SELECT region_name AS name FROM regions

ORDER BY name;

Code:



The screenshot shows a PostgreSQL query editor interface. The query is as follows:

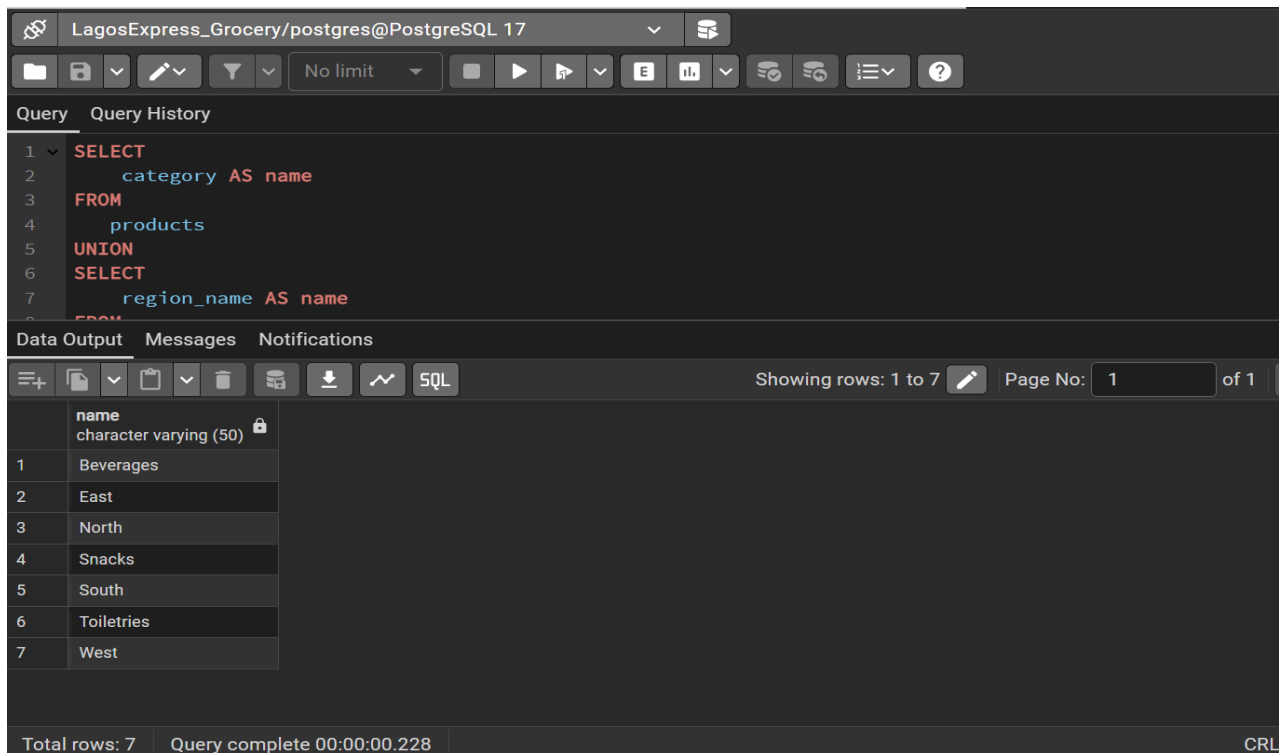
```
1 SELECT
2     category AS name
3 FROM
4     products
5 UNION
6 SELECT
7     region_name AS name
8 FROM
9     regions
10 ORDER BY
11     name;
12
```

The results are displayed in a table with the following data:

	name
1	Beverages
2	East
3	North
4	Snacks

The interface also shows "Total rows: 7" and "Query complete 00:00:00.228".

Code Result:



The screenshot shows the same PostgreSQL query editor interface with the same query. The results are displayed in a table with the following data:

	name
1	Beverages
2	East
3	North
4	Snacks
5	South
6	Toiletries
7	West

The interface also shows "Total rows: 7" and "Query complete 00:00:00.228".

Insight:

Demonstrates merging of different datasets. This is useful for creating dropdown menus or combined reports.

Q9. Use Logical Operators (AND/OR) to find sales of beverages above ₦1,000.

SELECT

```
c.customer_name,  
p.product_name,  
p.category,  
s.quantity,  
p.price * s.quantity AS total_value
```

FROM sales s

JOIN customers c

ON s.customer_id = c.customer_id

JOIN products p

ON s.product_id = p.product_id

WHERE p.category = 'Beverages' **AND** (p.price * s.quantity) > 1000;

Code:

The screenshot shows a PostgreSQL query editor interface. The top bar indicates the connection is 'LagosExpress_Grocery/postgres@PostgreSQL 17'. Below the toolbar, the query editor displays the following SQL query:

```
1 SELECT  
2     c.customer_name,  
3     p.product_name,  
4     p.category,  
5     s.quantity,  
6     p.price * s.quantity AS total_value  
7 FROM  
8     sales s  
9 JOIN  
10    customers c  
11   ON s.customer_id = c.customer_id  
12 JOIN  
13    products p  
14   ON s.product_id = p.product_id  
15 WHERE  
16     p.category = 'Beverages' AND (p.price * s.quantity) > 1000;  
17  
18
```

Below the query editor, the 'Data Output' tab is active, showing the results of the query. The results are displayed in a table with the following columns: customer_name, product_name, category, quantity, and total_value. The table shows 4 rows of data.

customer_name	product_name	category	quantity	total_value
Estimote Dalls	AquaDura Water	Beverages	12	1200.00

The bottom status bar indicates 'Total rows: 4' and 'Query complete 00:00:00.107'.

LagosExpress_Grocery/postgres@PostgreSQL 17

No limit

Query Query History

```

1 SELECT
2     c.customer_name,
3     p.product_name,
4     p.category,
5     s.quantity,
6     p.price * s.quantity AS total_value
7 FROM
8     sales s
9 JOIN
10    customers c
11 ON s.customer_id = c.customer_id
12 JOIN

```

Data Output Messages Notifications

Showing rows: 1 to 4 Page No: 1 of 1

	customer_name character varying (100)	product_name character varying (100)	category character varying (50)	quantity integer	total_value numeric
1	Fatima Bello	AquaPure Water	Beverages	12	1200.00
2	Grace Udo	NutiMilk	Beverages	6	1800.00
3	Musa Ibrahim	Sparkle Cola	Beverages	10	2000.00
4	Musa Ibrahim	AquaPure Water	Beverages	15	1500.00

Total rows: 4 Query complete 00:00:00.107 CRLF

Helps spot large beverage orders. This is good for upselling premium drinks.