E-Commerce Database Analysis using PostgreSQL

Internship Report
By: RAGAVARSHINI ALAGARSAMY
Company: ELEVATE LABS
Date: 08th AUG 2025

1. Introduction

This report documents the analysis of an e-commerce database using PostgreSQL. The database contains tables such as customers, categories, products, orders, and order_items. The objective of this project is to execute SQL queries for retrieving, analyzing, and optimizing business data.

2. Tools Used

- PostgreSQL 17

- pgAdmin 4

- Dataset provided by the company

3. Dataset Preparation

Before importing the dataset, existing data was truncated to avoid primary key conflicts. The CSV files were then imported using the following commands:

TRUNCATE TABLE table_name RESTART IDENTITY CASCADE;
\copy table_name FROM 'file_path.csv' DELIMITER ',' CSV HEADER;

Data was verified using:
SELECT * FROM table_name LIMIT 5;

4. Queries and Outputs

Each query is presented with its SQL code and the corresponding output screenshot.

Query 1

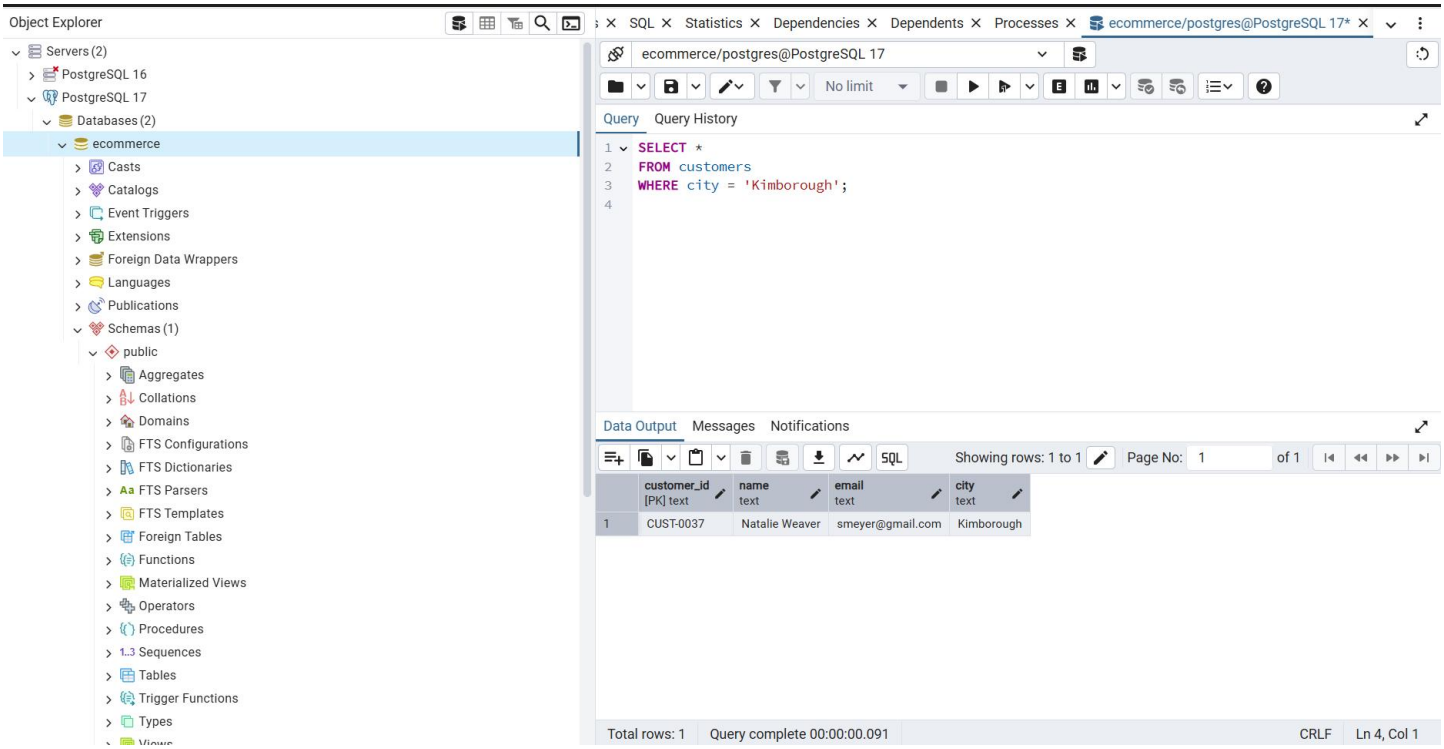
1. List all customers from a specific city

SELECT *

FROM customers

WHERE city = 'Kimborough';

OUTPUT:



Query 2

2. Top 10 most expensive products

SELECT product_id, product_name, price

FROM products

ORDER BY price DESC

LIMIT ;

OUTPUT:

ecommerce/postgres@PostgreSQL 17

No limit

Query | Query History

```
SELECT product_id, product_name, price
FROM products
ORDER BY price DESC
LIMIT 10;
```

Data Output | Messages | Notifications

Showing rows: 1 to 10 | Page No: 1 of 1

	product_id [PK] text	product_name text	price numeric (10,2)
1	PROD-00054	Drive	499.23
2	PROD-00060	Wish	485.33
3	PROD-00020	Marriage	482.21
4	PROD-00070	Land	480.71
5	PROD-00057	Soon	478.99
6	PROD-00071	Suddenly	473.64
7	PROD-00003	Media	469.44
8	PROD-00019	People	462.45
9	PROD-00027	Hundred	460.34
10	PROD-00004	Sort	459.61

Query 3

3. Total revenue per category

SELECT c.category_id, c.category_name, SUM(oi.price * oi.quantity) AS total_revenue

FROM order_items oi

JOIN products p ON oi.product_id = p.product_id

JOIN categories c ON p.category_id = c.category_id

GROUP BY c.category_id, c.category_name

ORDER BY total_revenue DESC;

OUTPUT:

SQL × Statistics × Dependencies × Dependents × Processes × ecommerce/postgres@PostgreSQL 17* ×

ecommerce/postgres@PostgreSQL 17

No limit

Query Query History

```
SELECT c.category_id, c.category_name, SUM(oi.price * oi.quantity) AS total_revenue
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
JOIN categories c ON p.category_id = c.category_id
GROUP BY c.category_id, c.category_name
ORDER BY total_revenue DESC;
```

Data Output Messages Notifications

Showing rows: 1 to 10 Page No: 1 of 1

	category_id [PK] text	category_name text	total_revenue numeric
1	CAT-010	Involve	53563.49
2	CAT-008	Onto	45331.95
3	CAT-001	Production	40745.31
4	CAT-007	Others	40530.12
5	CAT-005	Little	39184.53
6	CAT-004	Stuff	36657.91
7	CAT-002	Democratic	32607.00
8	CAT-003	Agent	30112.78
9	CAT-009	Best	29040.47
10	CAT-006	Event	21150.26

Total rows: 10 Query complete 00:00:00.097 CRLF Ln 6, Col 29

Query 4

4. Customers with more than 5 orders

SELECT customer_id, COUNT(order_id) AS total_orders

FROM orders

GROUP BY customer_id

HAVING COUNT(order_id) > 5

ORDER BY total_orders DESC;

OUTPUT:

SQL × Statistics × Dependencies × Dependents × Processes × ecommerce/postgres@PostgreSQL 17* ×

ecommerce/postgres@PostgreSQL 17

No limit

Query Query History

```
1 SELECT customer_id, COUNT(order_id) AS total_orders
2 FROM orders
3 GROUP BY customer_id
4 HAVING COUNT(order_id) > 5
5 ORDER BY total_orders DESC;
```

Data Output Messages Notifications

Showing rows: 1 to 11 Page No: 1 of 1

	customer_id text	total_orders bigint
1	CUST-0033	7
2	CUST-0023	7
3	CUST-0049	7
4	CUST-0047	6
5	CUST-0050	6
6	CUST-0020	6
7	CUST-0029	6
8	CUST-0044	6
9	CUST-0021	6
10	CUST-0019	6
11	CUST-0013	6

Successfully run. Total query runtime: 129 msec. 11 rows affected.

Total rows: 11 Query complete 00:00:00.129 CRLF Ln 5, Col 28

Query 5

5. Order details (INNER JOIN example)

SELECT o.order_id, o.order_date, c.name AS customer_name, p.product_name, oi.quantity, oi.price

FROM orders o

JOIN customers c ON o.customer_id = c.customer_id

JOIN order_items oi ON o.order_id = oi.order_id

JOIN products p ON oi.product_id = p.product_id

ORDER BY o.order_date DESC

LIMIT 20;

OUTPUT:

SQL × Statistics × Dependencies × Dependents × Processes × ecommerce/postgres@PostgreSQL 17* ×

ecommerce/postgres@PostgreSQL 17

No limit

Query Query History

```
SELECT o.order_id, o.order_date, c.name AS customer_name, p.product_name, oi.quantity, oi.price
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
ORDER BY o.order_date DESC
LIMIT 16;
```

Data Output Messages Notifications

Showing rows: 1 to 16 Page No: 1 of 1

	order_id text	order_date date	customer_name text	product_name text	quantity integer	price numeric (10,2)
1	ORD-00149	2025-08-02	Catherine Jordan	Expert	3	205.00
2	ORD-00149	2025-08-02	Catherine Jordan	Yard	4	194.66
3	ORD-00149	2025-08-02	Catherine Jordan	Read	1	436.49
4	ORD-00082	2025-07-27	Paul Porter	Safe	4	205.92
5	ORD-00082	2025-07-27	Paul Porter	Usually	5	40.23
6	ORD-00090	2025-07-26	Ashlee Gilbert	Act	1	179.36
7	ORD-00090	2025-07-26	Ashlee Gilbert	Stand	3	56.60
8	ORD-00116	2025-07-25	Beverly Gonzales	Learn	4	432.14
9	ORD-00116	2025-07-25	Beverly Gonzales	Forward	3	116.12
10	ORD-00009	2025-07-23	Anthony Pearson	According	5	126.45
11	ORD-00047	2025-07-16	Jacob Schaefer	Marriage	5	13.02
12	ORD-00177	2025-07-07	Linda Clark	Fight	3	70.78
13	ORD-00177	2025-07-07	Linda Clark	Behavior	2	192.67
14	ORD-00177	2025-07-07	Linda Clark	Should		

✓ Successfully run. Total query runtime: 76 msec. 16 rows affected.

Total rows: 16 Query complete 00:00:00.076 CRLF Ln 7, Col 9

Query 6

6. All customers and their orders (LEFT JOIN example)

SELECT c.customer_id, c.name, o.order_id, o.total_amount

FROM customers c

LEFT JOIN orders o ON c.customer_id = o.customer_id

ORDER BY c.customer_id;

OUTPUT:

ecommerce/postgres@PostgreSQL 17

No limit

Query | Query History

```
1 SELECT c.customer_id, c.name, o.order_id, o.total_amount
2 FROM customers c
3 LEFT JOIN orders o ON c.customer_id = o.customer_id
4 ORDER BY c.customer_id;
```

Data Output | Messages | Notifications

Showing rows: 1 to 201 | Page No: 1 of 1

	customer_id text	name text	order_id text	total_amount numeric (12,2)
1	CUST-0001	Catherine Jordan	ORD-00019	1474.46
2	CUST-0001	Catherine Jordan	ORD-00149	707.96
3	CUST-0001	Catherine Jordan	ORD-00029	1460.94
4	CUST-0001	Catherine Jordan	ORD-00076	1606.81
5	CUST-0001	Catherine Jordan	ORD-00127	341.39
6	CUST-0002	Evan Thomas	ORD-00134	1593.29
7	CUST-0002	Evan Thomas	ORD-00064	1062.01
8	CUST-0002	Evan Thomas	ORD-00103	86.86
9	CUST-0003	Leslie Fritz	ORD-00166	1692.90
10	CUST-0003	Leslie Fritz	ORD-00063	1693.98
11	CUST-0003	Leslie Fritz	ORD-00102	895.55
12	CUST-0003	Leslie Fritz	ORD-00192	109.16
13	CUST-0003	Leslie Fritz	ORD-00163	1767.87
14	CUST-0004	Nicholas Warner	ORD-00096	417.09

Total rows: 201 | Query complete 00:00:00.083 | CRLF | Ln 4, Col 24

Query 7

7. Products that have never been ordered (RIGHT JOIN example)

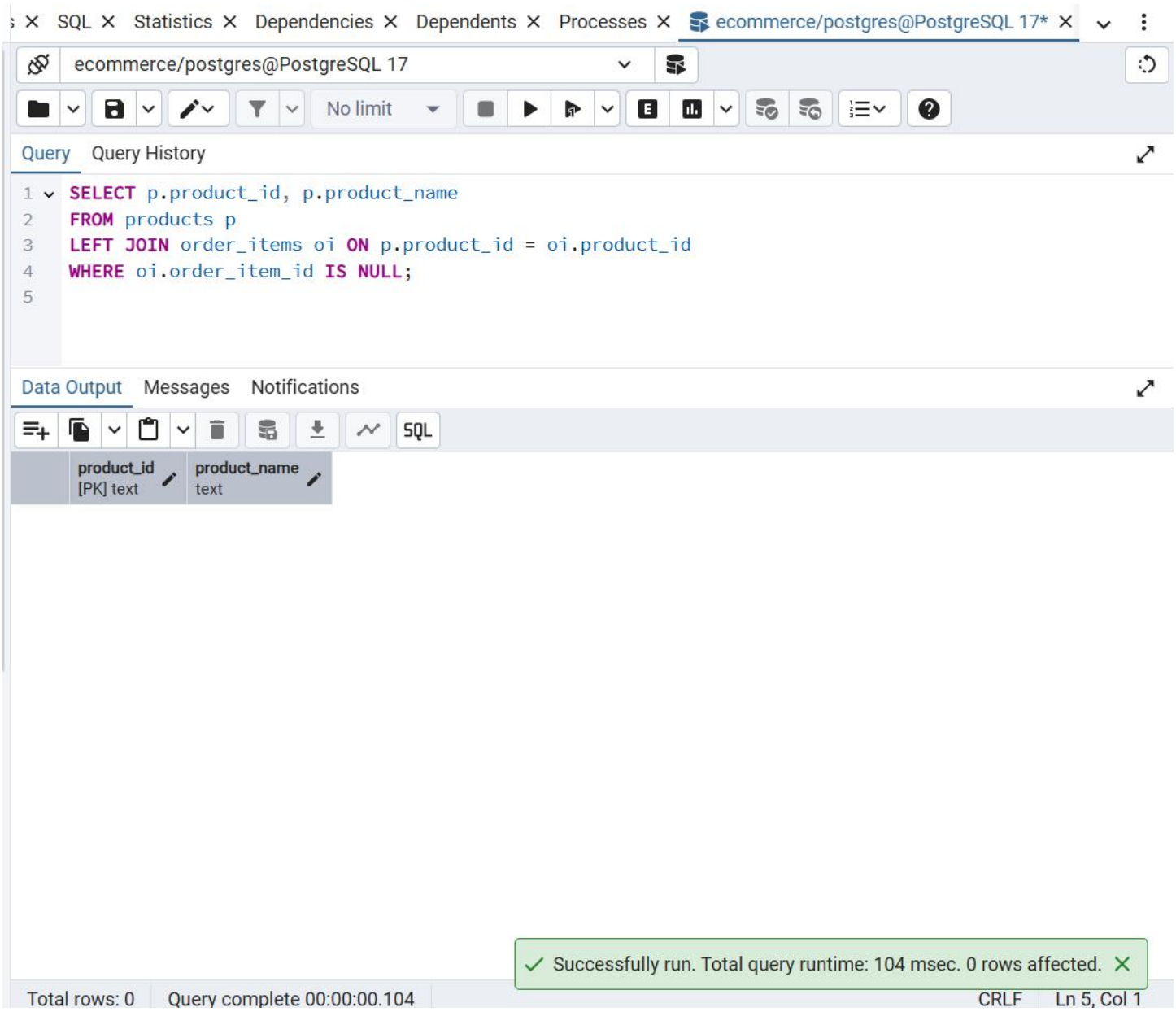
SELECT p.product_id, p.product_name

FROM products p

LEFT JOIN order_items oi ON p.product_id = oi.product_id

WHERE oi.order_item_id IS NULL;

OUTPUT:



Query 8

8. Subquery – Top 5 customers by total spend

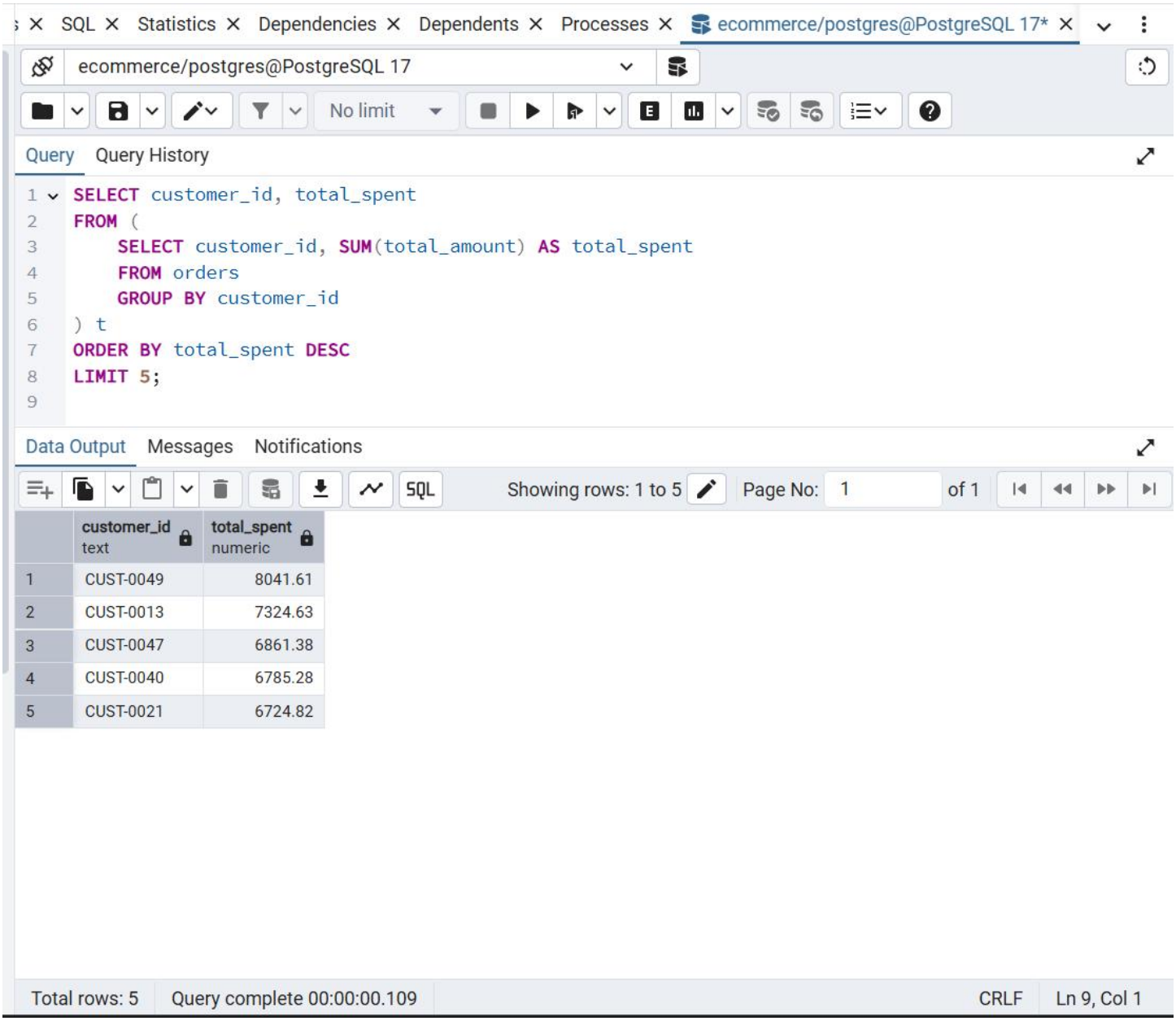
```
SELECT customer_id, total_spent
FROM (
    SELECT customer_id, SUM(total_amount) AS total_spent
    FROM orders
    GROUP BY customer_id
```

) t

ORDER BY total_spent DESC

LIMIT 5;

OUTPUT:



Query 9

9. Create a view for monthly revenue

CREATE OR REPLACE VIEW monthly_revenue AS

SELECT DATE_TRUNC('month', order_date) AS month, SUM(total_amount) AS revenue

FROM orders

GROUP BY month

ORDER BY month;

-- To see the view:

SELECT * FROM monthly_revenue;

OUTPUT:

SQL × Statistics × Dependencies × Dependents × Processes × ecommerce/postgres@PostgreSQL 17* ×

ecommerce/postgres@PostgreSQL 17

No limit

Query Query History

```
1  CREATE OR REPLACE VIEW monthly_revenue AS
2  SELECT DATE_TRUNC('month', order_date) AS month, SUM(total_amount) AS revenue
3  FROM orders
4  GROUP BY month
5  ORDER BY month;
6
7  -- To see the view:
8  SELECT * FROM monthly_revenue;
9
```

Data Output Messages Notifications

Showing rows: 1 to 25 Page No: 1 of 1

	month timestamp with time zone	revenue numeric
1	2023-08-01 00:00:00+05:30	7130.76
2	2023-09-01 00:00:00+05:30	7385.51
3	2023-10-01 00:00:00+05:30	9313.63
4	2023-11-01 00:00:00+05:30	8508.91
5	2023-12-01 00:00:00+05:30	9601.24
6	2024-01-01 00:00:00+05:30	8934.91
7	2024-02-01 00:00:00+05:30	10247.75
8	2024-03-01 00:00:00+05:30	17959.83
9	2024-04-01 00:00:00+05:30	13910.54
10	2024-05-01 00:00:00+05:30	6609.82
11	2024-06-01 00:00:00+05:30	2374.99
12	2024-07-01 00:00:00+05:30	6506.80
13	2024-08-01 00:00:00+05:30	6601.34

✓ Successfully run. Total query runtime: 105 msec. 25 rows affected.

Total rows: 25 Query complete 00:00:00.105 CRLF Ln 8, Col 31

Query 10

10. Index + EXPLAIN ANALYZE

```
-- Create index on orders table

CREATE INDEX idx_orders_customer_date ON orders(customer_id, order_date);



-- Analyze query performance

EXPLAIN ANALYZE

SELECT * FROM orders

WHERE customer_id = 'CUST-0005'

AND order_date > '2024-01-01';
```

OUTPUT:

SQL × Statistics × Dependencies × Dependents × Processes × ecommerce/postgres@PostgreSQL 17* ×

ecommerce/postgres@PostgreSQL 17

No limit

Query | Query History

```
-- Create index on orders table
CREATE INDEX idx_orders_customer_date ON orders(customer_id, order_date);

-- Analyze query performance
EXPLAIN ANALYZE
SELECT * FROM orders
WHERE customer_id = 'CUST-0005'
AND order_date > '2024-01-01';
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1 of 1

	QUERY PLAN text
1	Seq Scan on orders (cost=0.00..5.00 rows=3 width=30) (actual time=0.020..0.035 rows=3 loops=...
2	Filter: ((order_date > '2024-01-01'::date) AND (customer_id = 'CUST-0005'::text))
3	Rows Removed by Filter: 197
4	Planning Time: 0.507 ms
5	Execution Time: 0.051 ms

✓ Successfully run. Total query runtime: 79 msec. 5 rows affected.

Total rows: 5 | Query complete 00:00:00.079 | CRLF | Ln 8, Col 31

Query 11

11. Top 5 Customers by Lifetime Value

SELECT customer_id, SUM(total_amount) AS total_spent

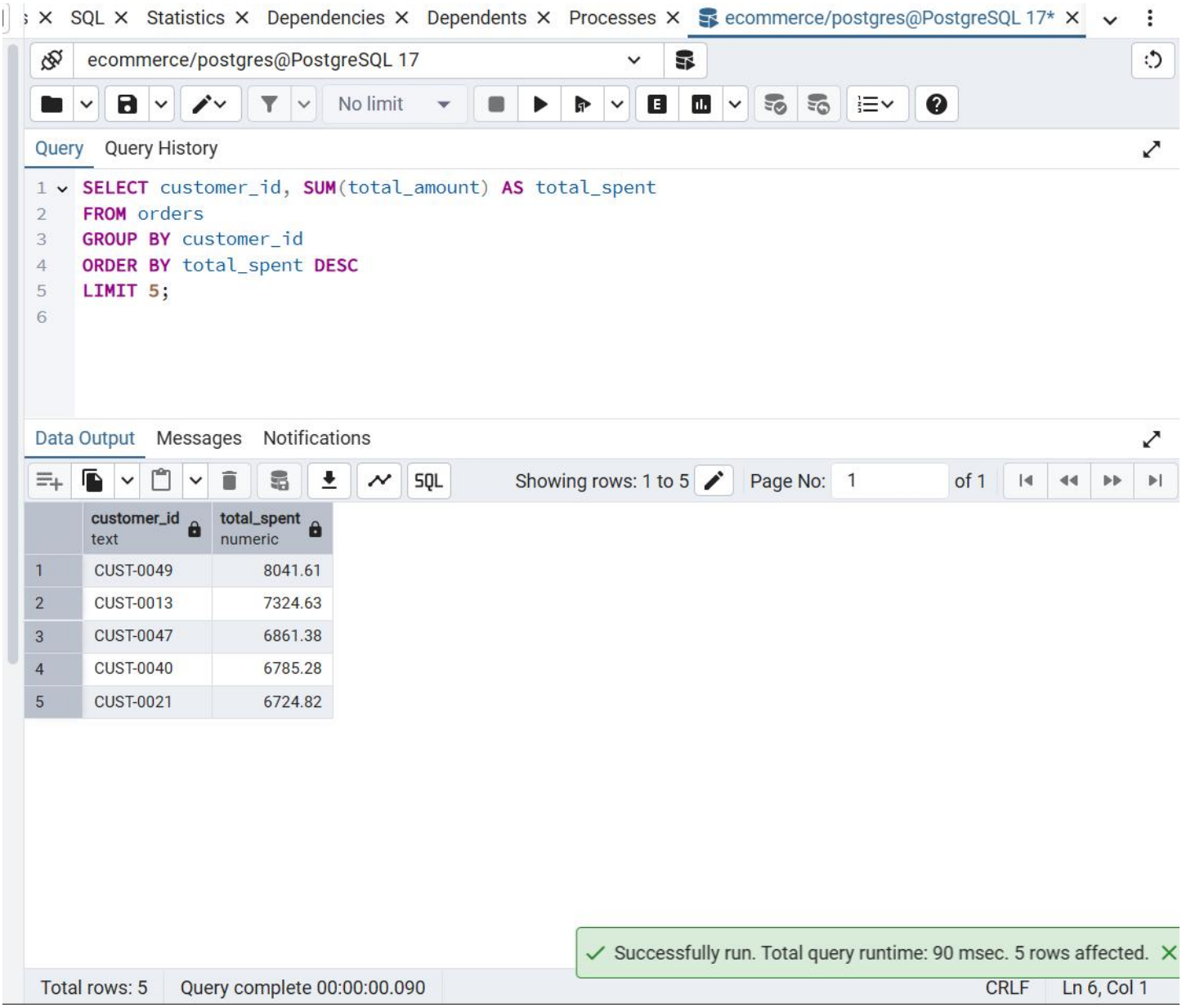
FROM orders

GROUP BY customer_id

ORDER BY total_spent DESC

LIMIT 5;

OUTPUT:



Query 12

12. Monthly Revenue Trend (shows business growth)

SELECT DATE_TRUNC('month', order_date) AS month, SUM(total_amount) AS revenue

FROM orders

GROUP BY month

ORDER BY month;

OUTPUT:

SQL × Statistics × Dependencies × Dependents × Processes × ecommerce/postgres@PostgreSQL 17* ×

ecommerce/postgres@PostgreSQL 17

No limit

Query Query History

```
SELECT DATE_TRUNC('month', order_date) AS month, SUM(total_amount) AS revenue
FROM orders
GROUP BY month
ORDER BY month;
```

Data Output Messages Notifications

Showing rows: 1 to 25 Page No: 1 of 1

	month timestamp with time zone	revenue numeric
1	2023-08-01 00:00:00+05:30	7130.76
2	2023-09-01 00:00:00+05:30	7385.51
3	2023-10-01 00:00:00+05:30	9313.63
4	2023-11-01 00:00:00+05:30	8508.91
5	2023-12-01 00:00:00+05:30	9601.24
6	2024-01-01 00:00:00+05:30	8934.91
7	2024-02-01 00:00:00+05:30	10247.75
8	2024-03-01 00:00:00+05:30	17959.83
9	2024-04-01 00:00:00+05:30	13910.54
10	2024-05-01 00:00:00+05:30	6609.82
11	2024-06-01 00:00:00+05:30	2374.99
12	2024-07-01 00:00:00+05:30	6506.80
13	2024-08-01 00:00:00+05:30	6601.34

Total rows: 25 Query complete 00:00:00.096 CRLF Ln 5, Col 1

Query 13

13 Most Profitable Category

SELECT c.category_name, SUM(oi.price * oi.quantity) AS profit

FROM order_items oi

JOIN products p ON oi.product_id = p.product_id

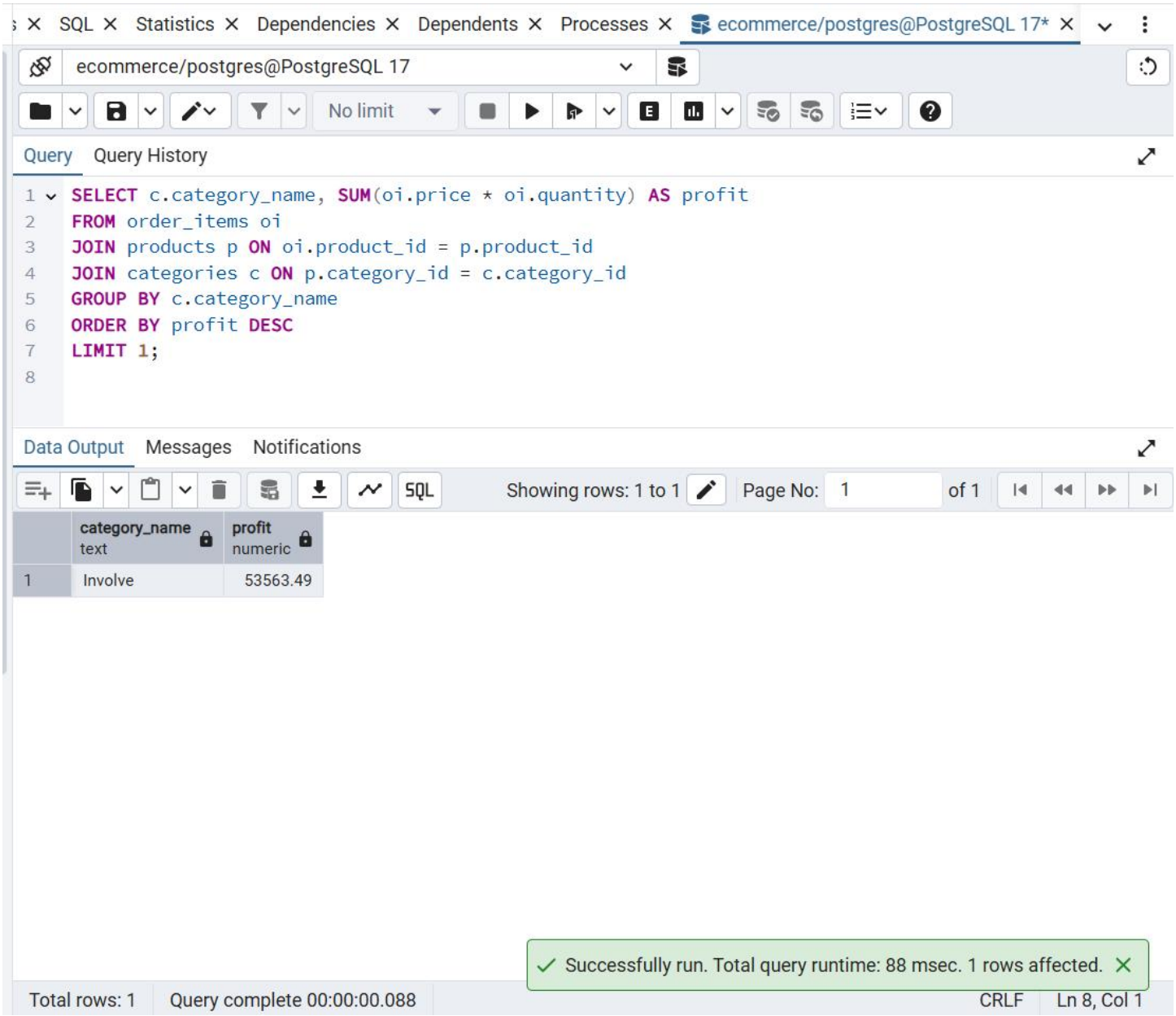
JOIN categories c ON p.category_id = c.category_id

GROUP BY c.category_name

ORDER BY profit DESC

LIMIT 1;

OUTPUT:



5. Observations

From the analysis, we observed trends in customer purchasing patterns, monthly revenues, and top spenders. For example, certain months such as March and April showed higher revenues, and a few customers contributed to a large percentage of sales.

6. Conclusion

Through this project, I gained hands-on experience in SQL queries, joins, views, indexing, and data analysis in PostgreSQL. These skills are essential for real-world database management and business intelligence.