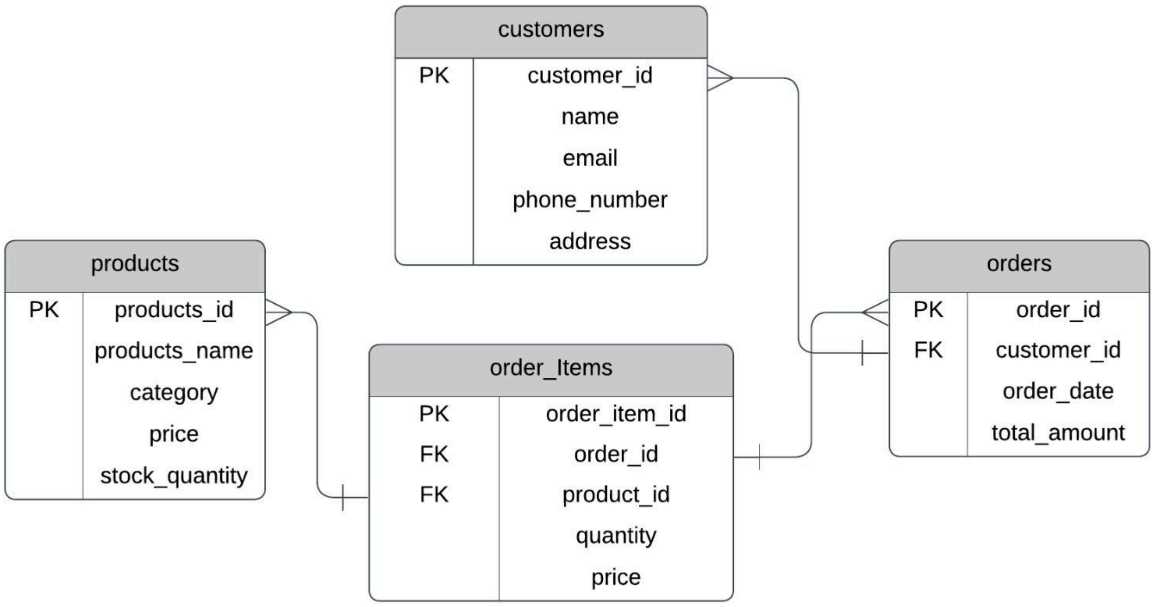
SQL E-Commerce Data Analysis Project Report



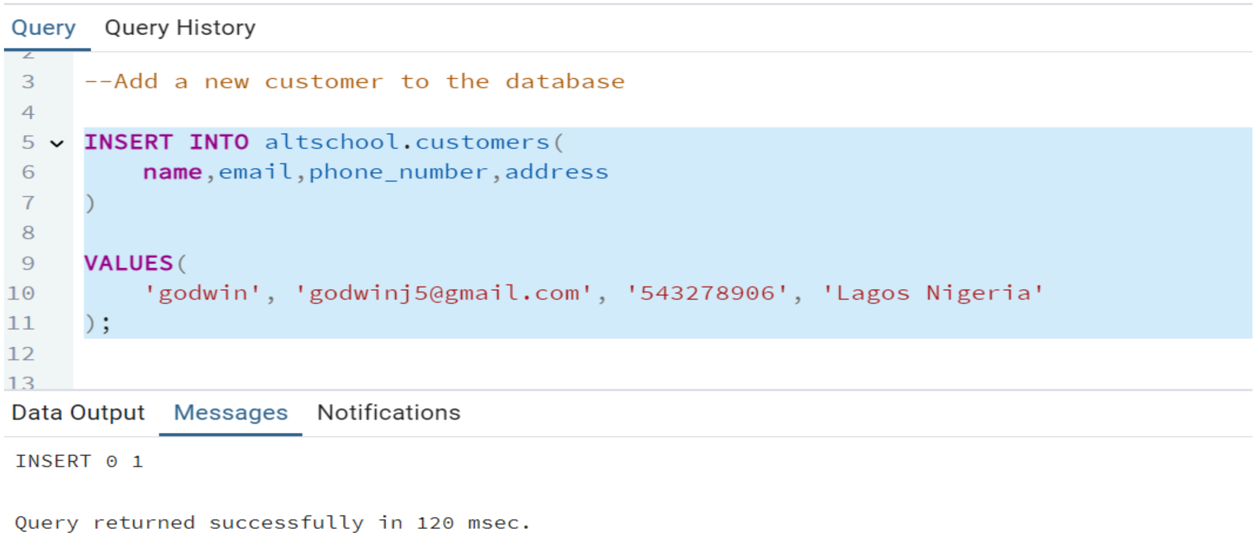
Schema design

The database schema consists of four interconnected tables: customers, orders, products, and order_items. It is designed to model an e-commerce platform that tracks customers, their orders, the products being sold, and the details of each order.

- **Customers:** Stores unique customer details like name, email, phone number, and address. It ensures data integrity with unique constraints on email and phone numbers.
- **Orders:** Tracks orders placed by customers, including the order date and total amount. It links to the customers table via a foreign key and uses ON DELETE CASCADE to maintain referential integrity.
- **Products:** Manages inventory, storing product names, categories, prices, and stock quantities. Each product has a unique ID for easy identification.
- **Order Items:** Captures the details of each item in an order, such as product, quantity, and price. It links to both orders and products tables and inherits their cascading delete behavior.

The schema is normalized to avoid redundancy, ensures data consistency through constraints, and supports scalability for future growth. It's efficient for querying and reporting on revenue, customer insights, and product trends.

Query to insert new customer

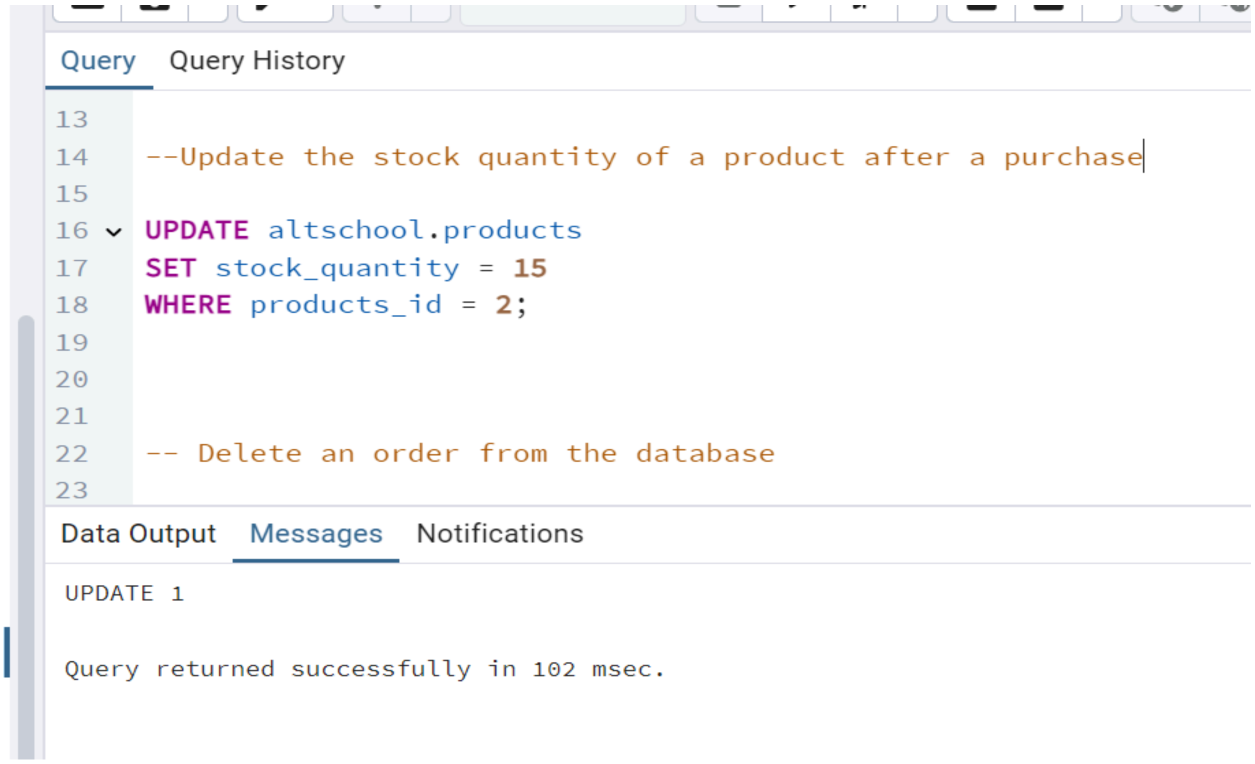

Query | Query History

```sql
--Add a new customer to the database

INSERT INTO altschool.customers(
    name,email,phone_number,address
)

VALUES(
    'godwin', 'godwinj5@gmail.com', '543278906', 'Lagos Nigeria'
);
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 120 msec.
```

Query to update stock quantity

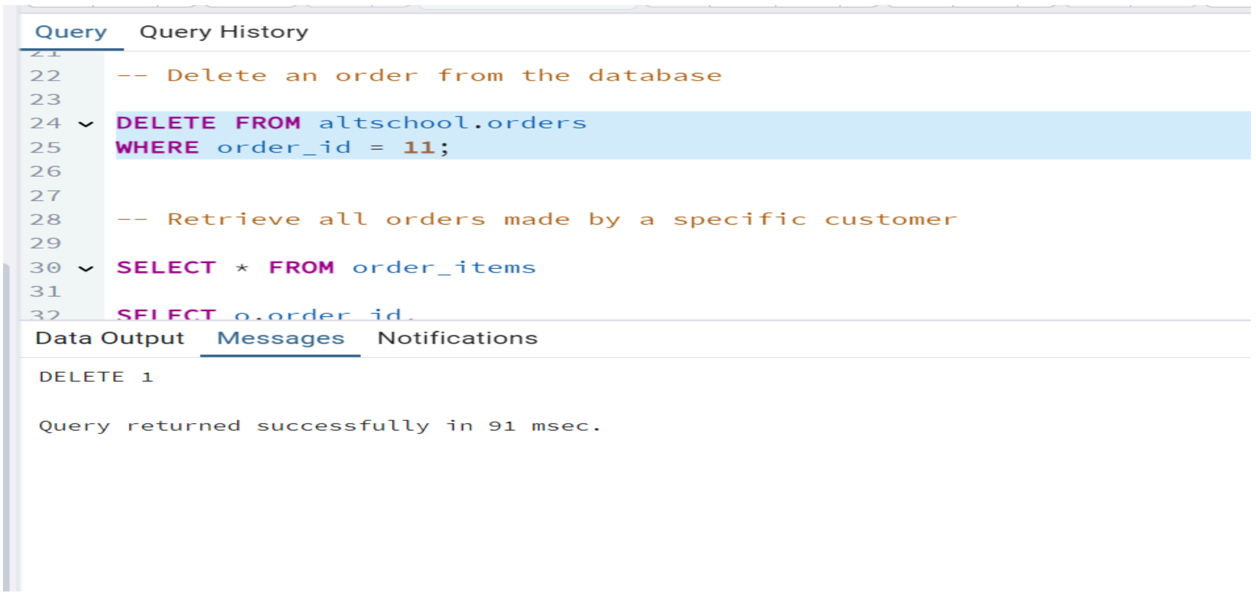

Query | Query History

```sql
--Update the stock quantity of a product after a purchase

UPDATE altschool.products
SET stock_quantity = 15
WHERE products_id = 2;



-- Delete an order from the database

```

Data Output | Messages | Notifications

```
UPDATE 1

Query returned successfully in 102 msec.
```

Query to delete an order



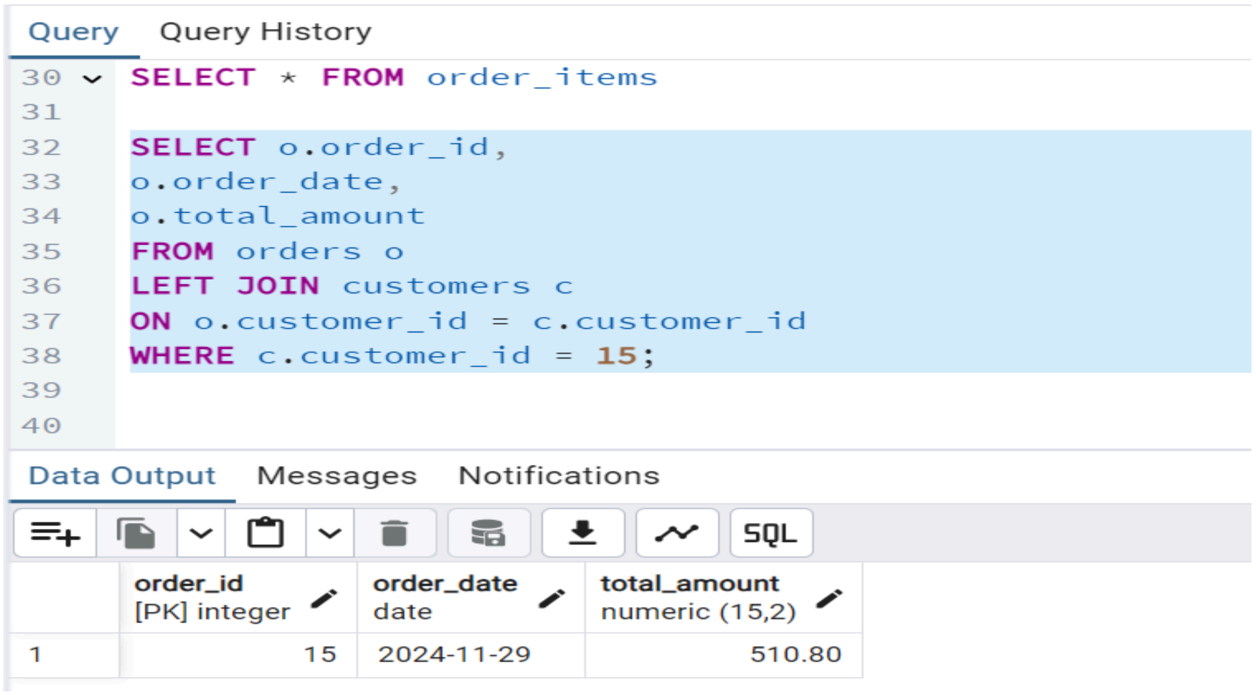
Query Query History

```
-- Delete an order from the database

DELETE FROM altschool.orders
WHERE order_id = 11;


-- Retrieve all orders made by a specific customer

SELECT * FROM order_items

SELECT o.order_id,
```

Data Output Messages Notifications

```
DELETE 1

Query returned successfully in 91 msec.
```

Query to retrieve orders



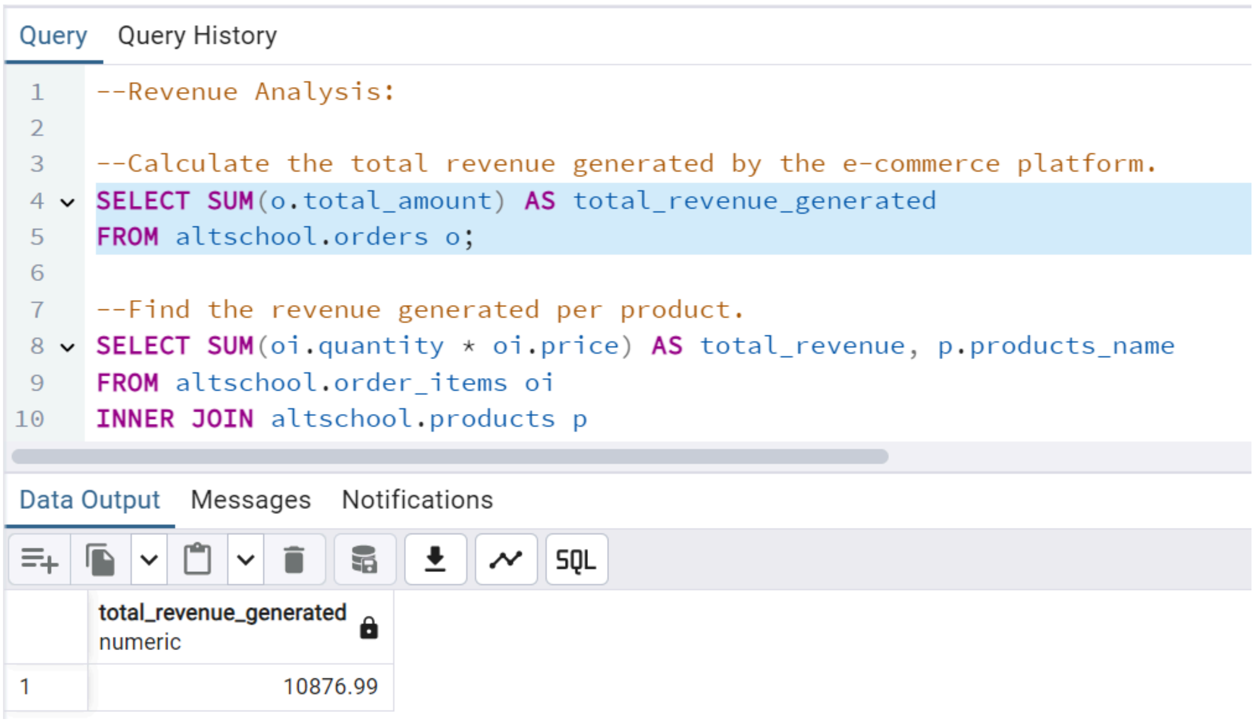
Query Query History

```
SELECT * FROM order_items

SELECT o.order_id,
o.order_date,
o.total_amount
FROM orders o
LEFT JOIN customers c
ON o.customer_id = c.customer_id
WHERE c.customer_id = 15;
```

Data Output Messages Notifications

	order_id [PK] integer	order_date date	total_amount numeric (15,2)
1	15	2024-11-29	510.80

Aggregate functions

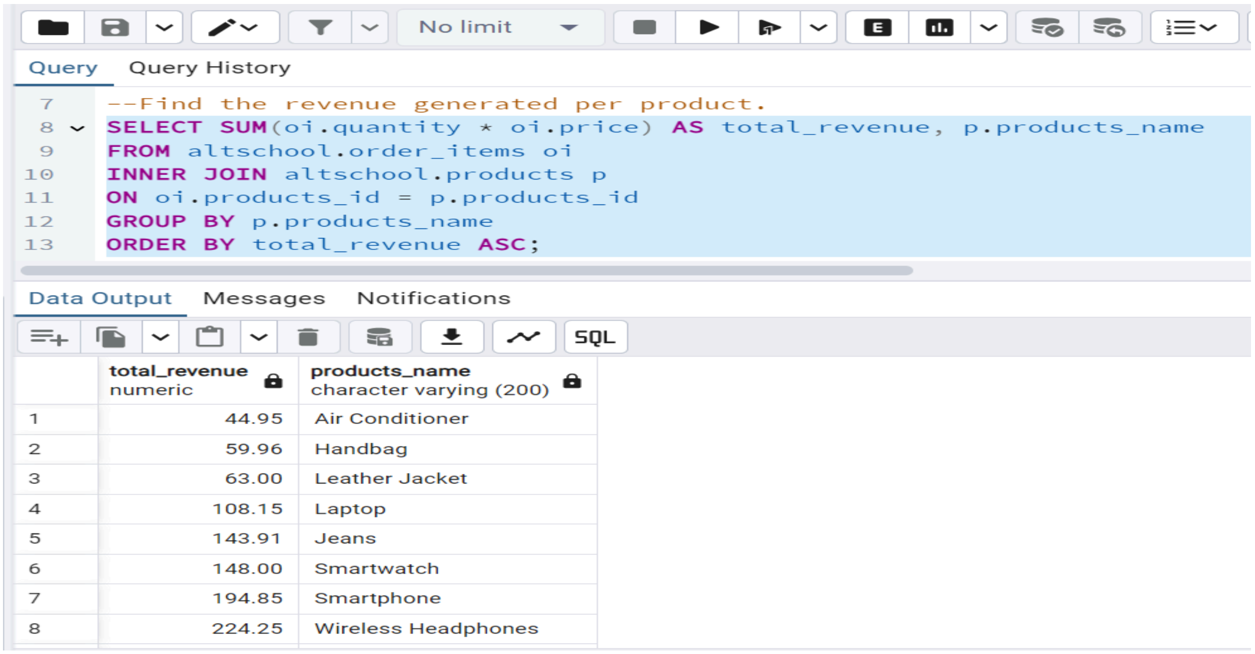


```sql
--Revenue Analysis:

--Calculate the total revenue generated by the e-commerce platform.
SELECT SUM(o.total_amount) AS total_revenue_generated
FROM altschool.orders o;

--Find the revenue generated per product.
SELECT SUM(oi.quantity * oi.price) AS total_revenue, p.products_name
FROM altschool.order_items oi
INNER JOIN altschool.products p
```

	total_revenue_generated numeric
1	10876.99

Total revenue generated was 10,876.

Revenue per product

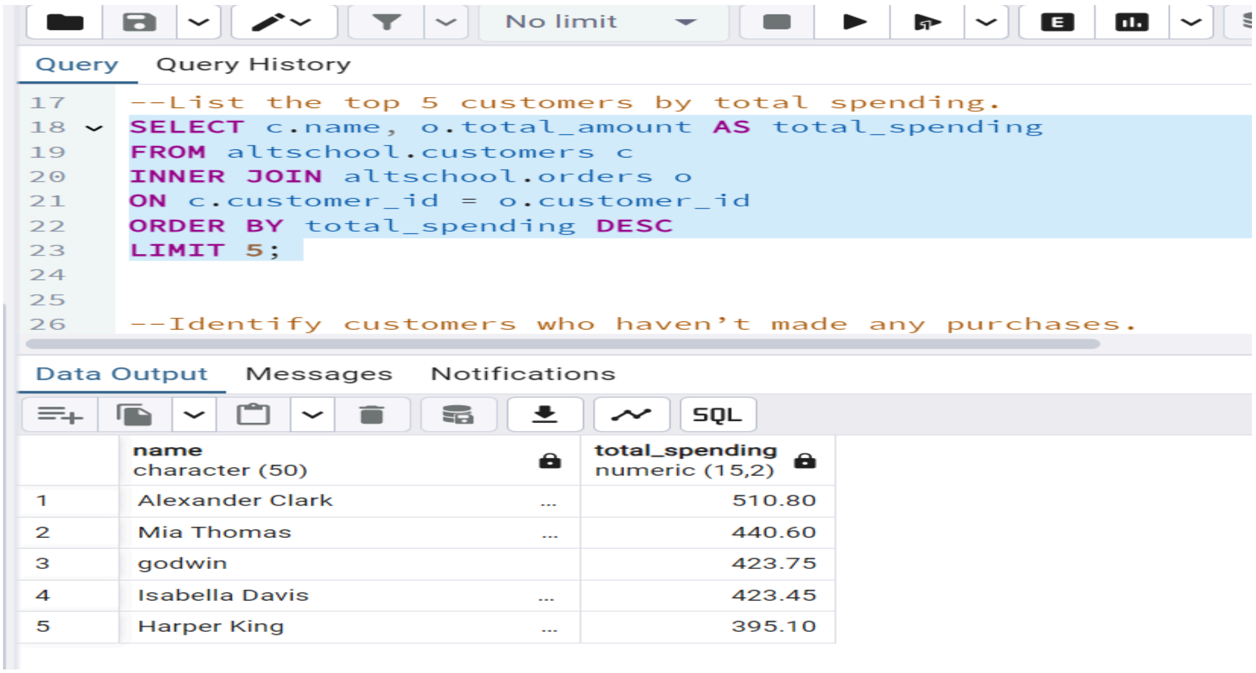


```sql
--Find the revenue generated per product.
SELECT SUM(oi.quantity * oi.price) AS total_revenue, p.products_name
FROM altschool.order_items oi
INNER JOIN altschool.products p
ON oi.products_id = p.products_id
GROUP BY p.products_name
ORDER BY total_revenue ASC;
```

	total_revenue numeric	products_name character varying (200)
1	44.95	Air Conditioner
2	59.96	Handbag
3	63.00	Leather Jacket
4	108.15	Laptop
5	143.91	Jeans
6	148.00	Smartwatch
7	194.85	Smartphone
8	224.25	Wireless Headphones

Wireless headphones, smartphones and smartwatches were the top 3 products with the highest revenue.

Top 5 customers

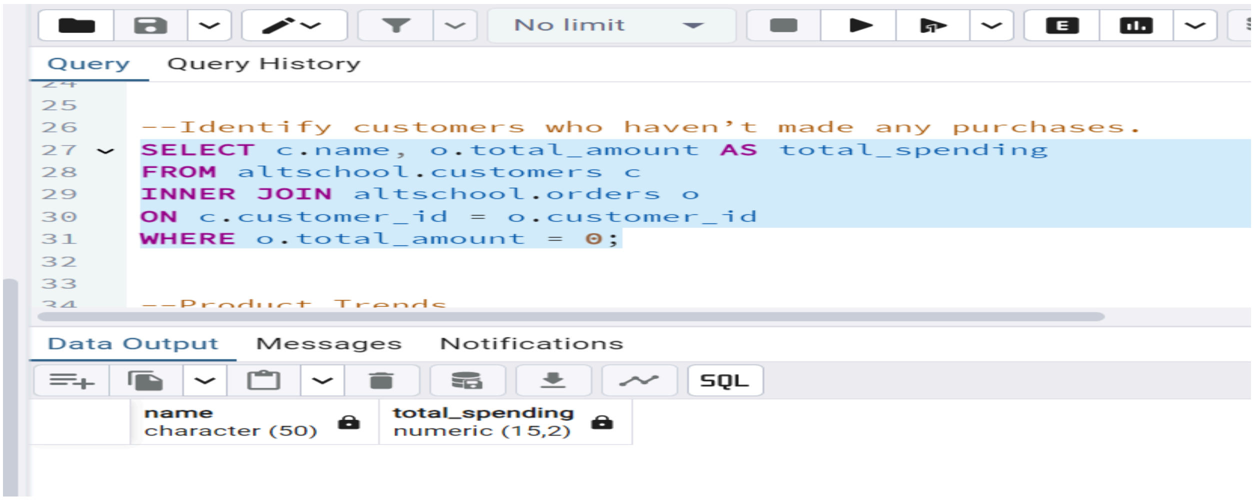


```
--List the top 5 customers by total spending.
SELECT c.name, o.total_amount AS total_spending
FROM altschool.customers c
INNER JOIN altschool.orders o
ON c.customer_id = o.customer_id
ORDER BY total_spending DESC
LIMIT 5;


--Identify customers who haven’t made any purchases.
```

	name character (50)	total_spending numeric (15,2)
1	Alexander Clark	510.80
2	Mia Thomas	440.60
3	godwin	423.75
4	Isabella Davis	423.45
5	Harper King	395.10

The top 5 customers with the highest spending are Alexander, Mia, Godwin, Isabella and Harper.

Customers without purchase

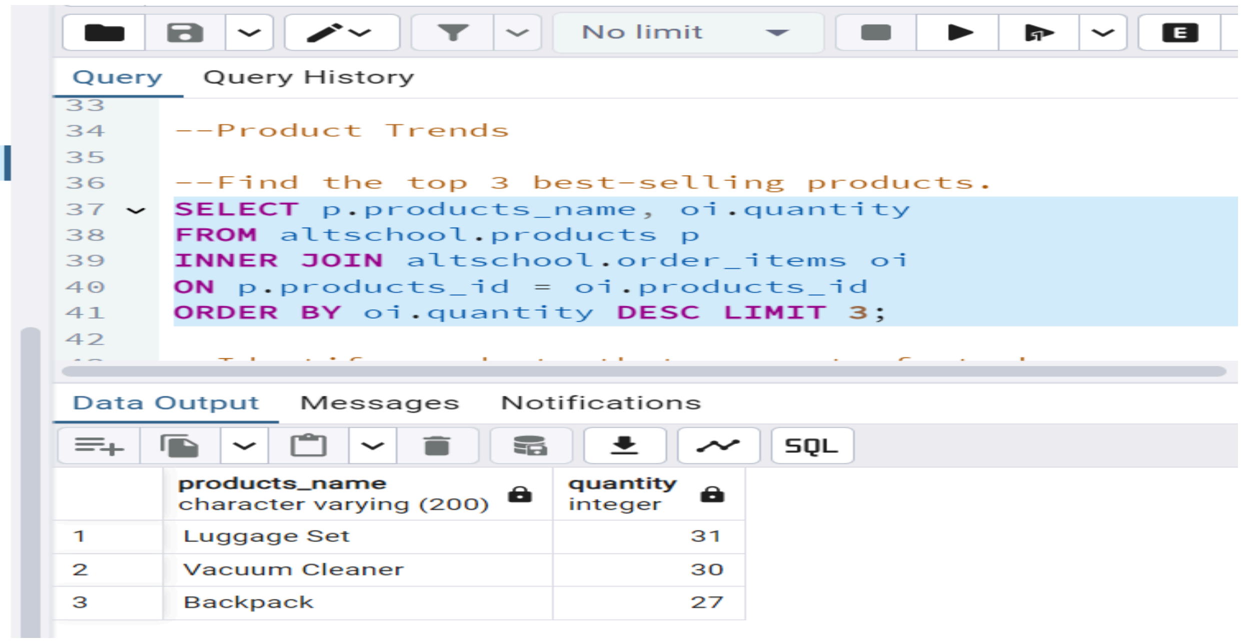


```

--Identify customers who haven’t made any purchases.
SELECT c.name, o.total_amount AS total_spending
FROM altschool.customers c
INNER JOIN altschool.orders o
ON c.customer_id = o.customer_id
WHERE o.total_amount = 0;


--Product Trends
```

name character (50)	total_spending numeric (15,2)

All customers made purchases.

Top 3 products sold



```
--Product Trends

--Find the top 3 best-selling products.
SELECT p.products_name, oi.quantity
FROM altschool.products p
INNER JOIN altschool.order_items oi
ON p.products_id = oi.products_id
ORDER BY oi.quantity DESC LIMIT 3;
```

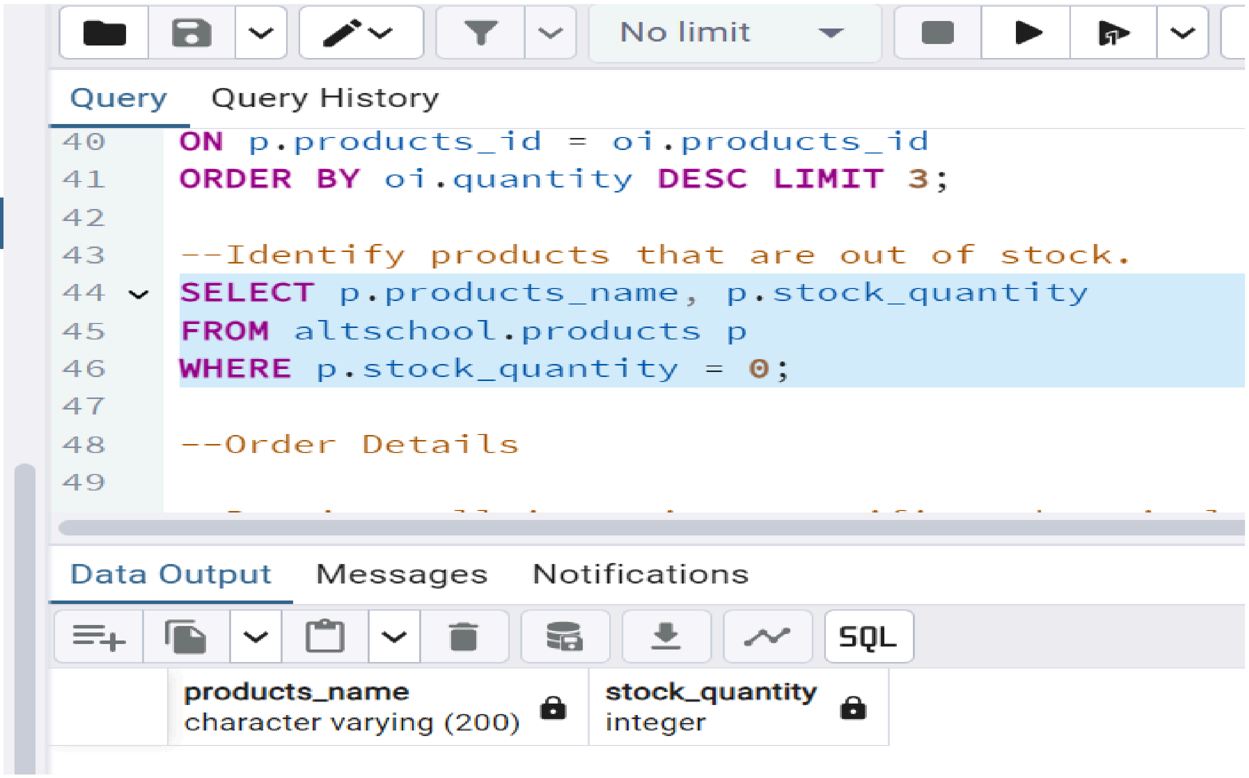
No limit

Data Output | Messages | Notifications

	products_name character varying (200)	quantity integer
1	Luggage Set	31
2	Vacuum Cleaner	30
3	Backpack	27

Luggage set, Vacuum cleaner and backpack were the top 3 products that were purchased.

Query to find products out of stock



```
ON p.products_id = oi.products_id
ORDER BY oi.quantity DESC LIMIT 3;

--Identify products that are out of stock.
SELECT p.products_name, p.stock_quantity
FROM altschool.products p
WHERE p.stock_quantity = 0;

--Order Details
```

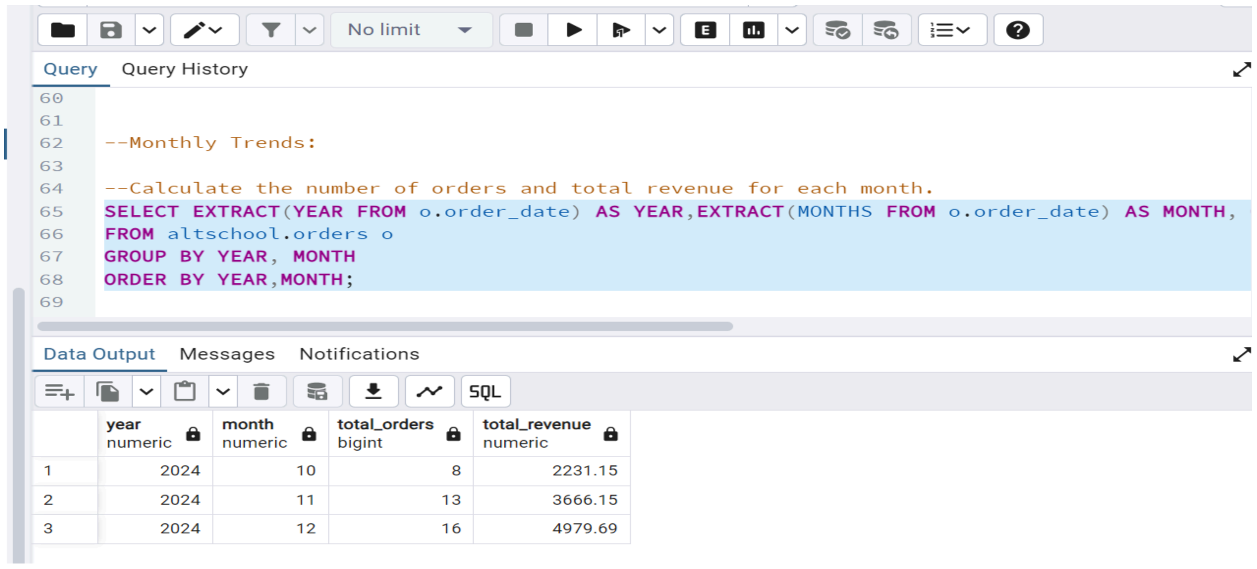
No limit

Data Output | Messages | Notifications

	products_name character varying (200)	stock_quantity integer

All products are in stock.

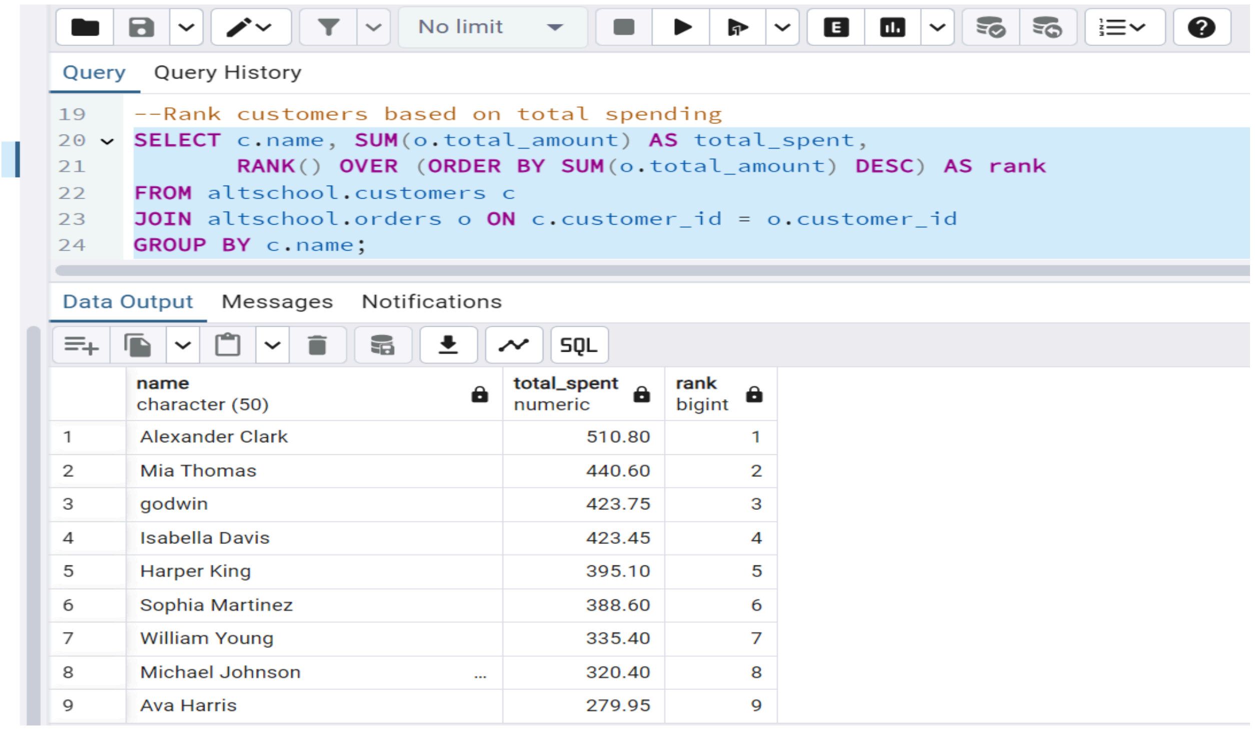
Revenue generated per month



```
--Monthly Trends:

--Calculate the number of orders and total revenue for each month.
SELECT EXTRACT(YEAR FROM o.order_date) AS YEAR,EXTRACT(MONTHS FROM o.order_date) AS MONTH,
FROM altschool.orders o
GROUP BY YEAR, MONTH
ORDER BY YEAR,MONTH;
```

	year numeric	month numeric	total_orders bigint	total_revenue numeric
1	2024	10	8	2231.15
2	2024	11	13	3666.15
3	2024	12	16	4979.69

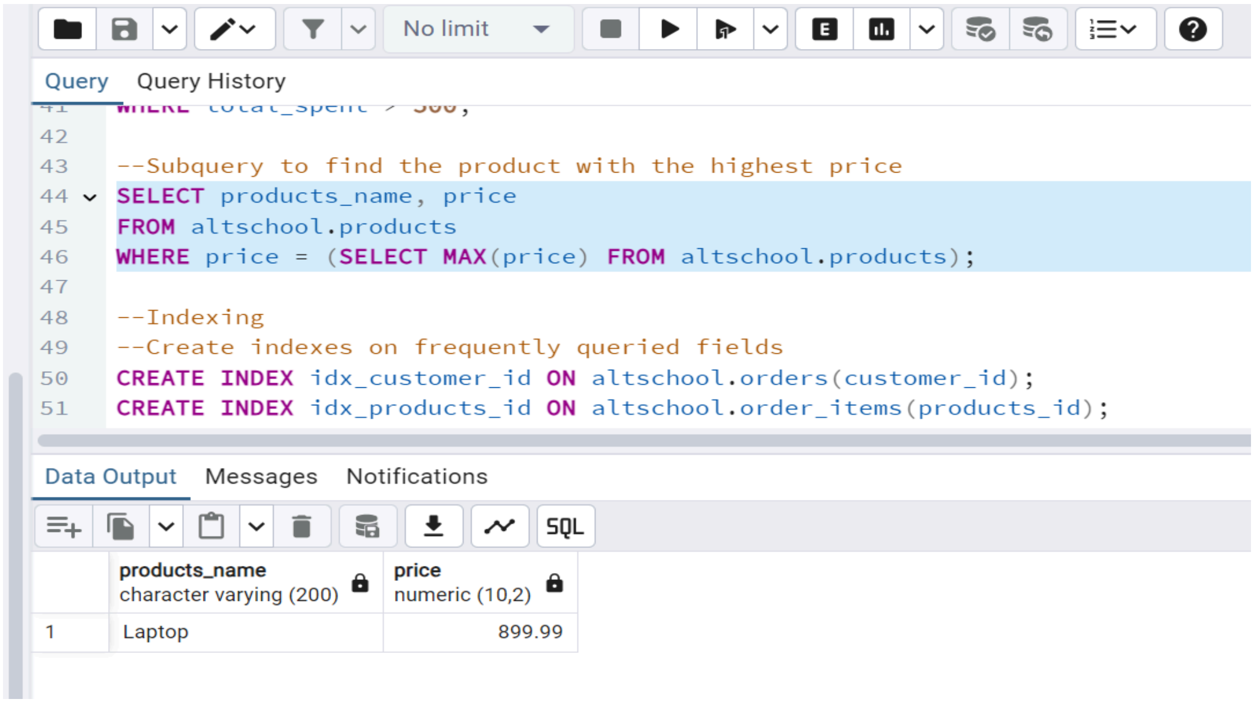
Query to rank customers



```
--Rank customers based on total spending
SELECT c.name, SUM(o.total_amount) AS total_spent,
       RANK() OVER (ORDER BY SUM(o.total_amount) DESC) AS rank
FROM altschool.customers c
JOIN altschool.orders o ON c.customer_id = o.customer_id
GROUP BY c.name;
```

	name character (50)	total_spent numeric	rank bigint
1	Alexander Clark	510.80	1
2	Mia Thomas	440.60	2
3	godwin	423.75	3
4	Isabella Davis	423.45	4
5	Harper King	395.10	5
6	Sophia Martinez	388.60	6
7	William Young	335.40	7
8	Michael Johnson ...	320.40	8
9	Ava Harris	279.95	9

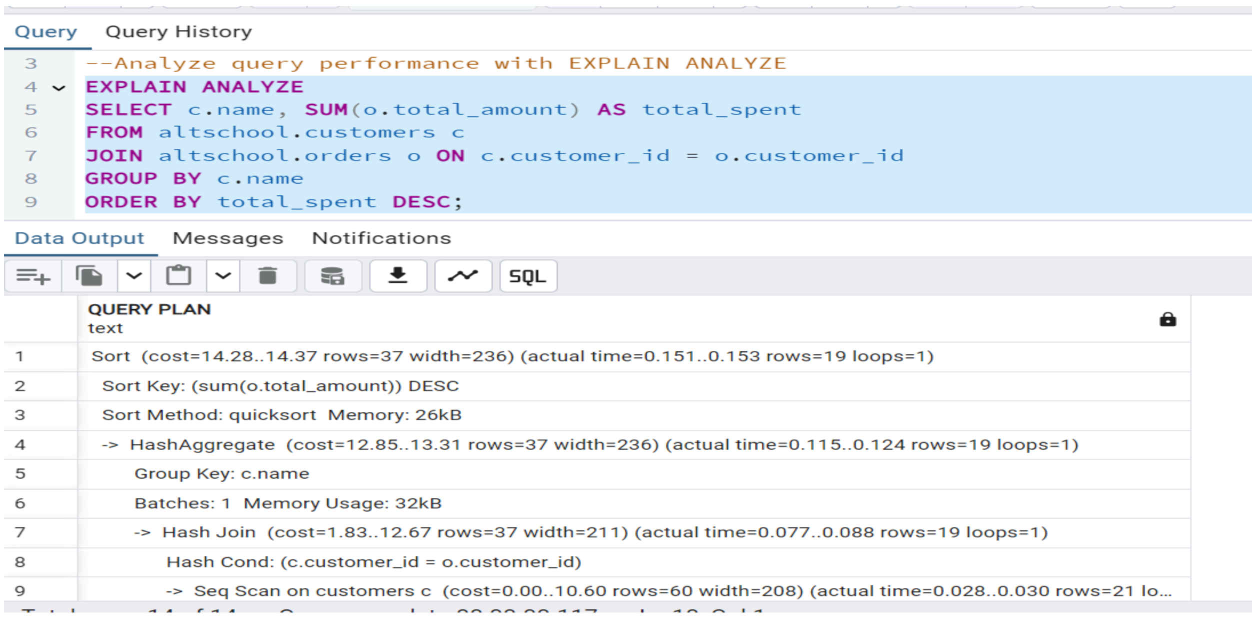
Subquery to find products with the highest price



```sql
WHERE total_spent > 500;

--Subquery to find the product with the highest price
SELECT products_name, price
FROM altschool.products
WHERE price = (SELECT MAX(price) FROM altschool.products);

--Indexing
--Create indexes on frequently queried fields
CREATE INDEX idx_customer_id ON altschool.orders(customer_id);
CREATE INDEX idx_products_id ON altschool.order_items(products_id);
```

	products_name character varying (200)	price numeric (10,2)
1	Laptop	899.99

Query to analyze query performance using EXPLAIN ANAALYZE



```sql
--Analyze query performance with EXPLAIN ANALYZE
EXPLAIN ANALYZE
SELECT c.name, SUM(o.total_amount) AS total_spent
FROM altschool.customers c
JOIN altschool.orders o ON c.customer_id = o.customer_id
GROUP BY c.name
ORDER BY total_spent DESC;
```

```
   QUERY PLAN
   text
1  Sort  (cost=14.28..14.37 rows=37 width=236) (actual time=0.151..0.153 rows=19 loops=1)
2    Sort Key: (sum(o.total_amount)) DESC
3    Sort Method: quicksort  Memory: 26kB
4    ->  HashAggregate  (cost=12.85..13.31 rows=37 width=236) (actual time=0.115..0.124 rows=19 loops=1)
5          Group Key: c.name
6          Batches: 1  Memory Usage: 32kB
7          ->  Hash Join  (cost=1.83..12.67 rows=37 width=211) (actual time=0.077..0.088 rows=19 loops=1)
8                Hash Cond: (c.customer_id = o.customer_id)
9                ->  Seq Scan on customers c  (cost=0.00..10.60 rows=60 width=208) (actual time=0.028..0.030 rows=21 lo...
```