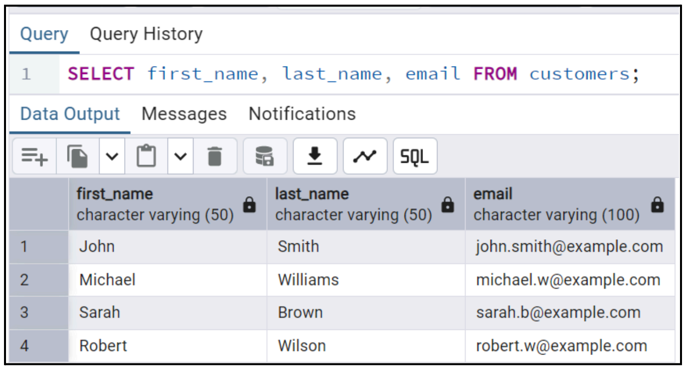
TASK-3

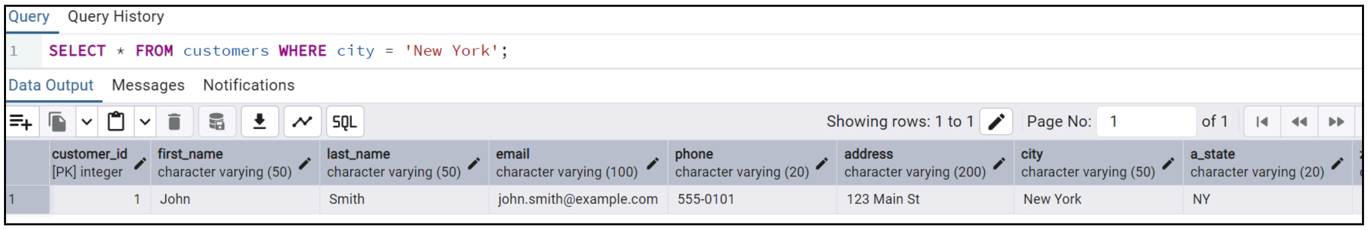
1. **SELECT – Retrieving Data:** statement is fundamental in SQL, used to retrieve data from a table. It can fetch all columns using SELECT * or only specific ones. For example, ***SELECT * FROM customers;*** retrieves all customer data, whereas ***SELECT first_name, last_name, email FROM customers;*** fetches only the names and email addresses, making the query more efficient and readable when not all data is required.



```sql
SELECT first_name, last_name, email FROM customers;
```

	first_name character varying (50)	last_name character varying (50)	email character varying (100)
1	John	Smith	john.smith@example.com
2	Michael	Williams	michael.w@example.com
3	Sarah	Brown	sarah.b@example.com
4	Robert	Wilson	robert.w@example.com

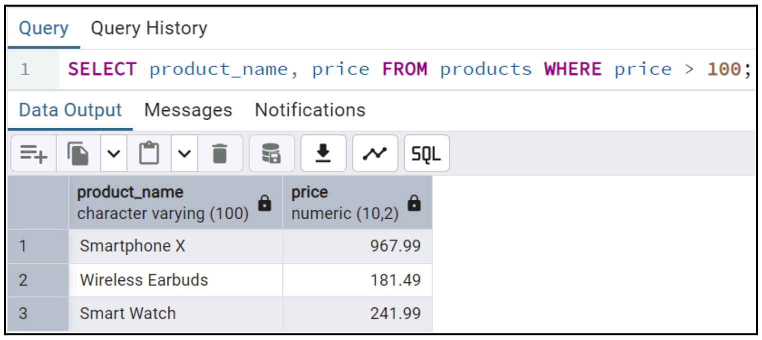
2. **WHERE –** Filtering Rows: clause filters records that meet specific conditions. It helps retrieve only relevant rows. For instance, ***SELECT * FROM customers WHERE city = 'New York';*** fetches customers living in New York, and ***SELECT product_name, price FROM products WHERE price > 100;*** filters out products priced above $100



```sql
SELECT * FROM customers WHERE city = 'New York';
```

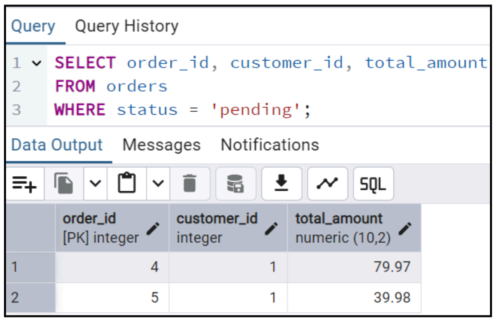
Showing rows: 1 to 1   Page No: 1 of 1

	customer_id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)	phone character varying (20)	address character varying (200)	city character varying (50)	a_state character varying (20)
1	1	John	Smith	john.smith@example.com	555-0101	123 Main St	New York	NY



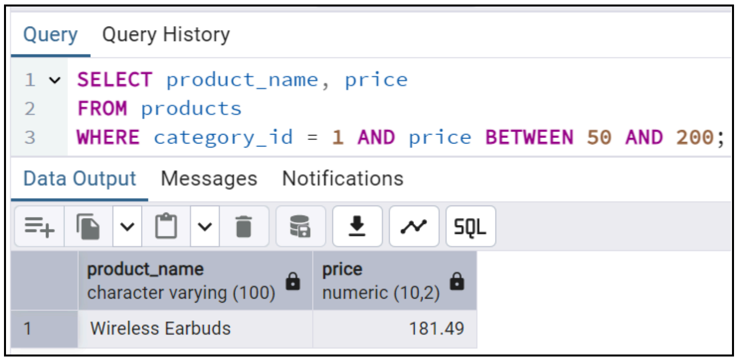
```sql
SELECT product_name, price FROM products WHERE price > 100;
```

	product_name character varying (100)	price numeric (10,2)
1	Smartphone X	967.99
2	Wireless Earbuds	181.49
3	Smart Watch	241.99



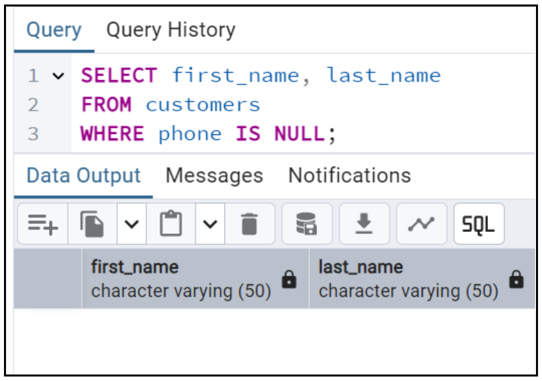
```sql
SELECT order_id, customer_id, total_amount
FROM orders
WHERE status = 'pending';
```

	order_id [PK] integer	customer_id integer	total_amount numeric (10,2)
1	4	1	79.97
2	5	1	39.98



```sql
SELECT product_name, price
FROM products
WHERE category_id = 1 AND price BETWEEN 50 AND 200;
```

	product_name character varying (100)	price numeric (10,2)
1	Wireless Earbuds	181.49



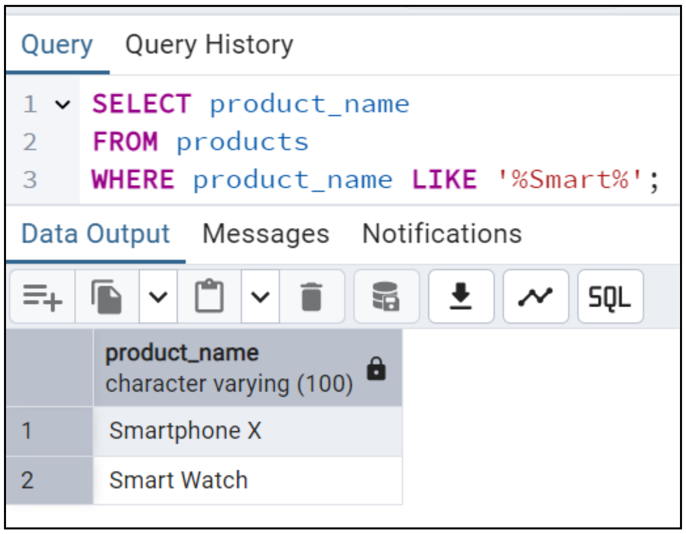
```sql
SELECT first_name, last_name
FROM customers
WHERE phone IS NULL;
```

	first_name character varying (50)	last_name character varying (50)

3. **LIKE Operator:** The LIKE operator is used for basic pattern matching in SQL queries. It supports two wildcard characters:

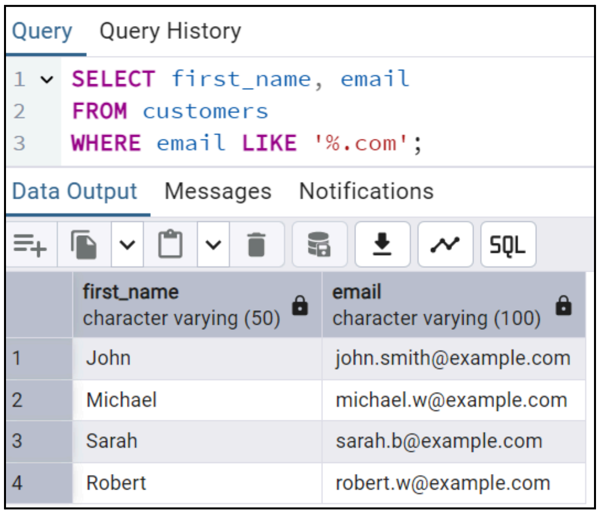
   i. % (percent): Matches zero or more characters.

   ii. _ (underscore): Matches exactly one character.



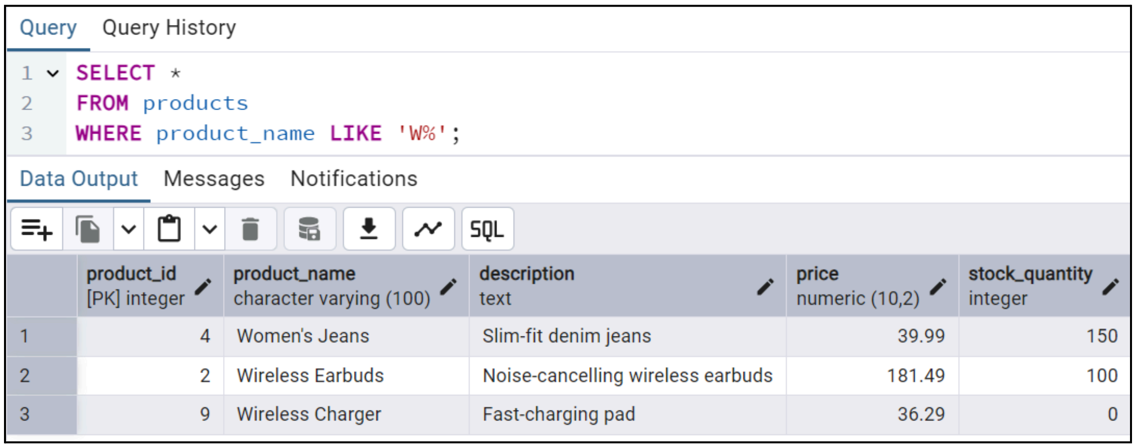
```sql
SELECT product_name
FROM products
WHERE product_name LIKE '%Smart%';
```

	product_name character varying (100)
1	Smartphone X
2	Smart Watch



```sql
SELECT first_name, email
FROM customers
WHERE email LIKE '%.com';
```

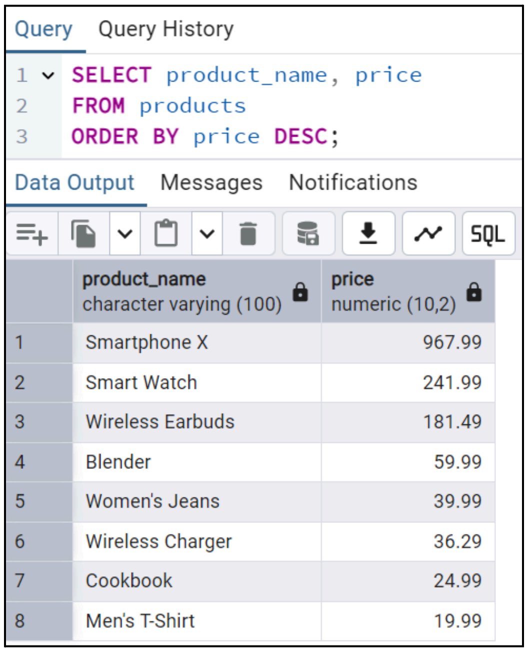
	first_name character varying (50)	email character varying (100)
1	John	john.smith@example.com
2	Michael	michael.w@example.com
3	Sarah	sarah.b@example.com
4	Robert	robert.w@example.com



```sql
SELECT *
FROM products
WHERE product_name LIKE 'W%';
```

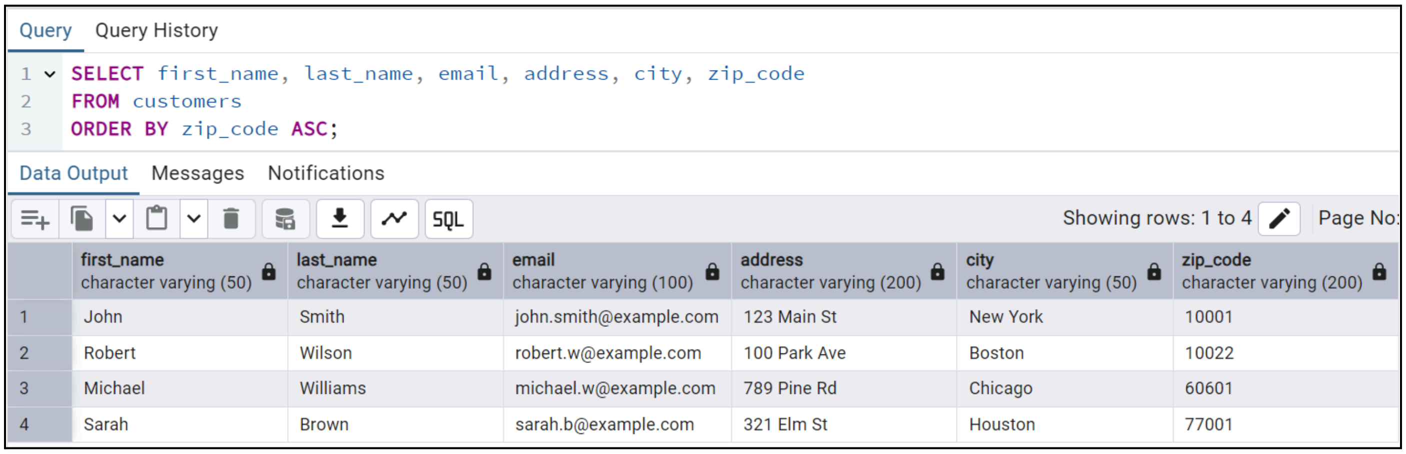
	product_id [PK] integer	product_name character varying (100)	description text	price numeric (10,2)	stock_quantity integer
1	4	Women's Jeans	Slim-fit denim jeans	39.99	150
2	2	Wireless Earbuds	Noise-cancelling wireless earbuds	181.49	100
3	9	Wireless Charger	Fast-charging pad	36.29	0

4. **ORDER BY – Sorting Results:** ORDER BY sorts the query result in ascending (ASC) or descending (DESC) order based on one or more columns. For example, to get the most expensive products first: SELECT product_name, price FROM products ORDER BY price DESC;. To sort orders by oldest date: SELECT order_id, order_date FROM orders ORDER BY order_date ASC;



```sql
SELECT product_name, price
FROM products
ORDER BY price DESC;
```

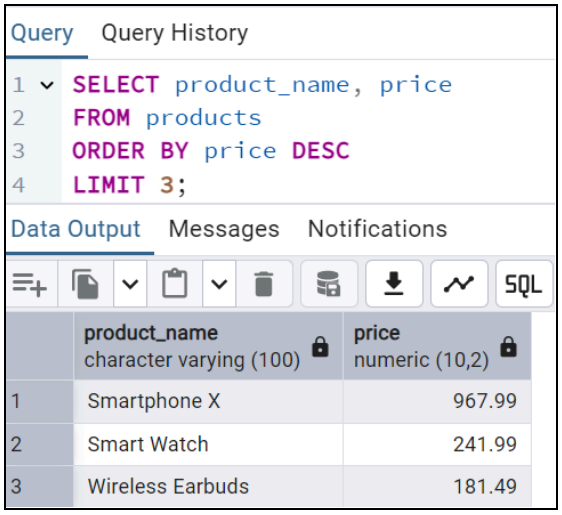
	product_name character varying (100)	price numeric (10,2)
1	Smartphone X	967.99
2	Smart Watch	241.99
3	Wireless Earbuds	181.49
4	Blender	59.99
5	Women's Jeans	39.99
6	Wireless Charger	36.29
7	Cookbook	24.99
8	Men's T-Shirt	19.99



```sql
SELECT first_name, last_name, email, address, city, zip_code
FROM customers
ORDER BY zip_code ASC;
```

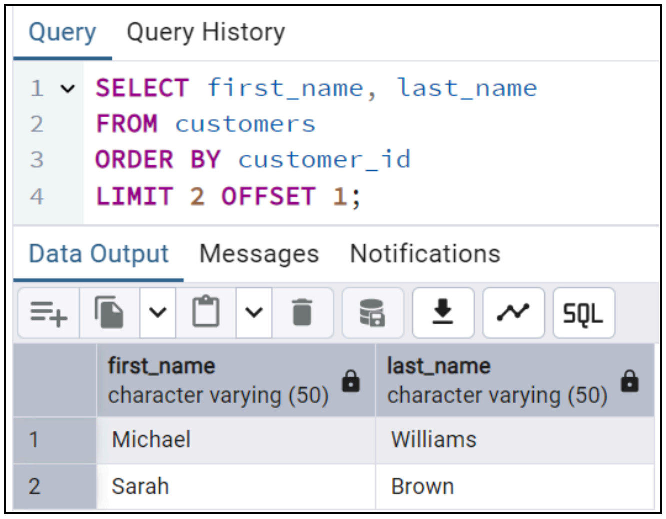
	first_name character varying (50)	last_name character varying (50)	email character varying (100)	address character varying (200)	city character varying (50)	zip_code character varying (200)
1	John	Smith	john.smith@example.com	123 Main St	New York	10001
2	Robert	Wilson	robert.w@example.com	100 Park Ave	Boston	10022
3	Michael	Williams	michael.w@example.com	789 Pine Rd	Chicago	60601
4	Sarah	Brown	sarah.b@example.com	321 Elm St	Houston	77001

5. **LIMIT and OFFSET – Pagination:** LIMIT restricts the number of records returned, while OFFSET skips a defined number of rows. This is useful for pagination. For instance, to get the top 3 expensive products: SELECT product_name, price FROM products ORDER BY price DESC LIMIT 3;. To get the second page of customers: SELECT first_name, last_name FROM customers ORDER BY customer_id LIMIT 2 OFFSET 1;



```sql
SELECT product_name, price
FROM products
ORDER BY price DESC
LIMIT 3;
```

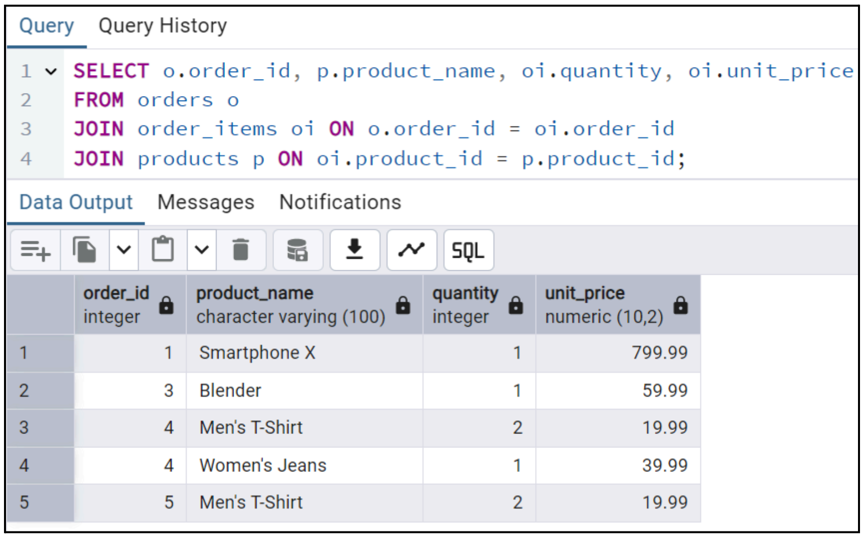
	product_name character varying (100)	price numeric (10,2)
1	Smartphone X	967.99
2	Smart Watch	241.99
3	Wireless Earbuds	181.49



```sql
SELECT first_name, last_name
FROM customers
ORDER BY customer_id
LIMIT 2 OFFSET 1;
```

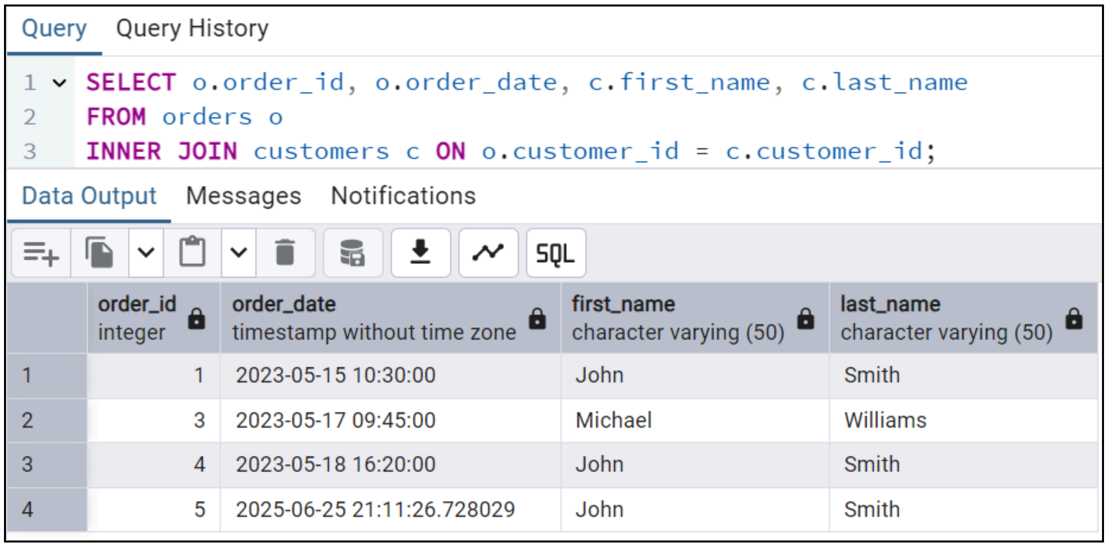
	first_name character varying (50)	last_name character varying (50)
1	Michael	Williams
2	Sarah	Brown

6. **JOIN – Combining Tables:** Joins allow data from multiple tables to be combined based on a related column. Examples include:
   - ➔ INNER JOIN (e.g., matching orders and customers)
   - ➔ LEFT JOIN (includes all customers even without orders)
   - ➔ RIGHT JOIN (includes all products even if not ordered)
   - ➔ FULL OUTER JOIN (includes all customers and products regardless of match)
   - ➔ CROSS JOIN (produces a Cartesian product of both tables)



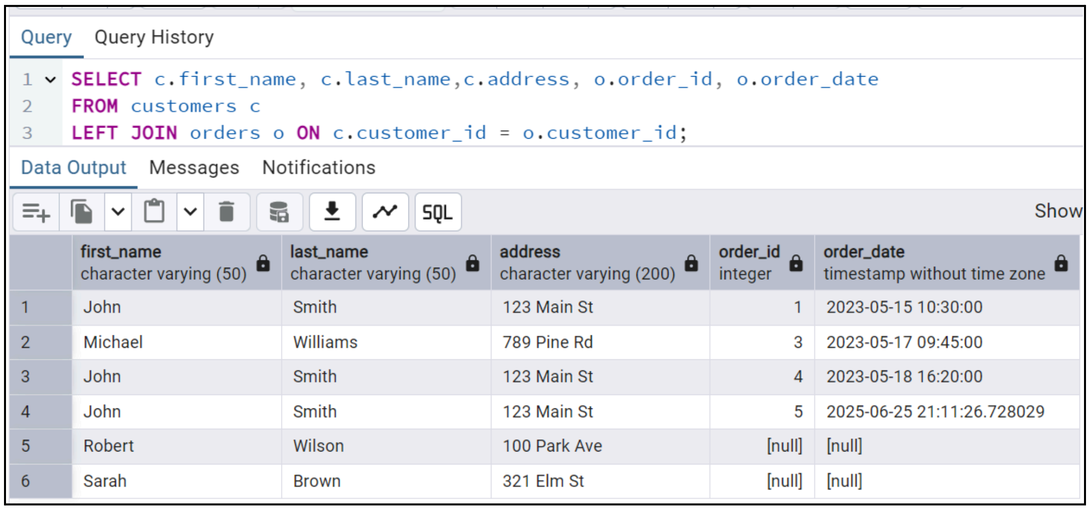
```sql
SELECT o.order_id, p.product_name, oi.quantity, oi.unit_price
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id;
```

	order_id integer	product_name character varying (100)	quantity integer	unit_price numeric (10,2)
1	1	Smartphone X	1	799.99
2	3	Blender	1	59.99
3	4	Men's T-Shirt	2	19.99
4	4	Women's Jeans	1	39.99
5	5	Men's T-Shirt	2	19.99



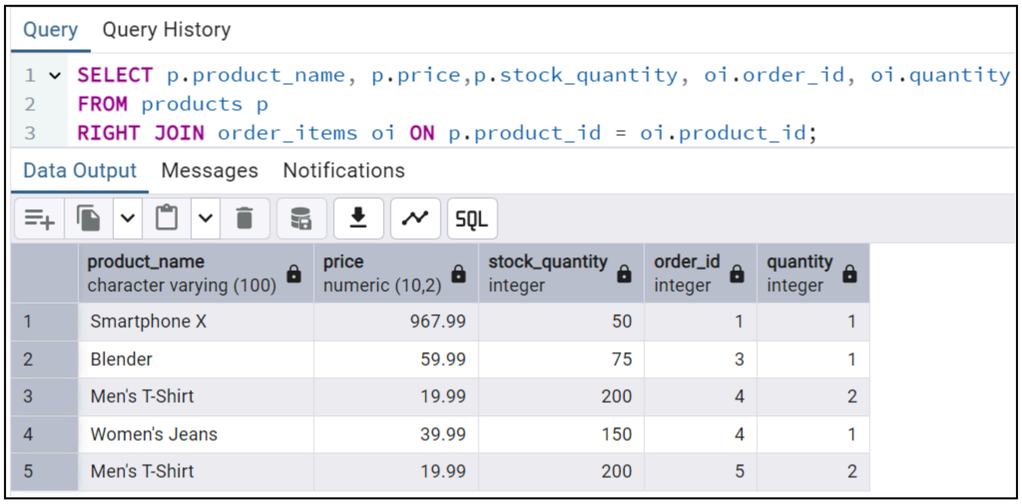
```sql
SELECT o.order_id, o.order_date, c.first_name, c.last_name
FROM orders o
INNER JOIN customers c ON o.customer_id = c.customer_id;
```

	order_id integer	order_date timestamp without time zone	first_name character varying (50)	last_name character varying (50)
1	1	2023-05-15 10:30:00	John	Smith
2	3	2023-05-17 09:45:00	Michael	Williams
3	4	2023-05-18 16:20:00	John	Smith
4	5	2025-06-25 21:11:26.728029	John	Smith



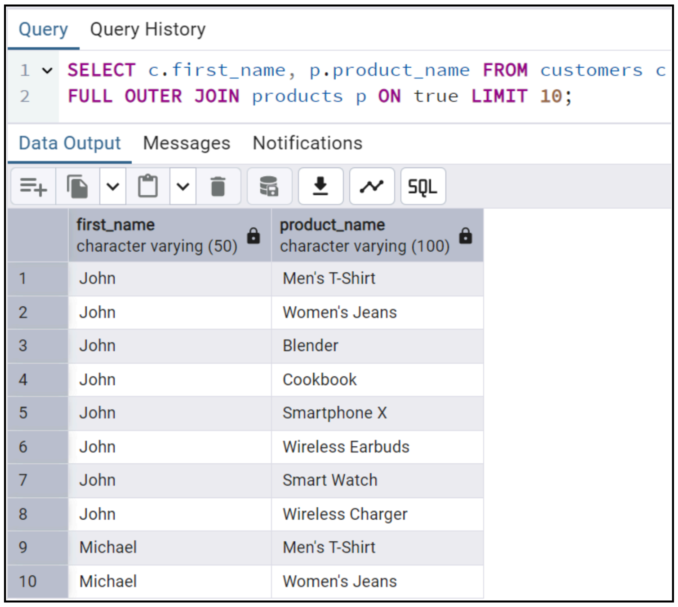
```sql
SELECT c.first_name, c.last_name,c.address, o.order_id, o.order_date
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id;
```

	first_name character varying (50)	last_name character varying (50)	address character varying (200)	order_id integer	order_date timestamp without time zone
1	John	Smith	123 Main St	1	2023-05-15 10:30:00
2	Michael	Williams	789 Pine Rd	3	2023-05-17 09:45:00
3	John	Smith	123 Main St	4	2023-05-18 16:20:00
4	John	Smith	123 Main St	5	2025-06-25 21:11:26.728029
5	Robert	Wilson	100 Park Ave	[null]	[null]
6	Sarah	Brown	321 Elm St	[null]	[null]



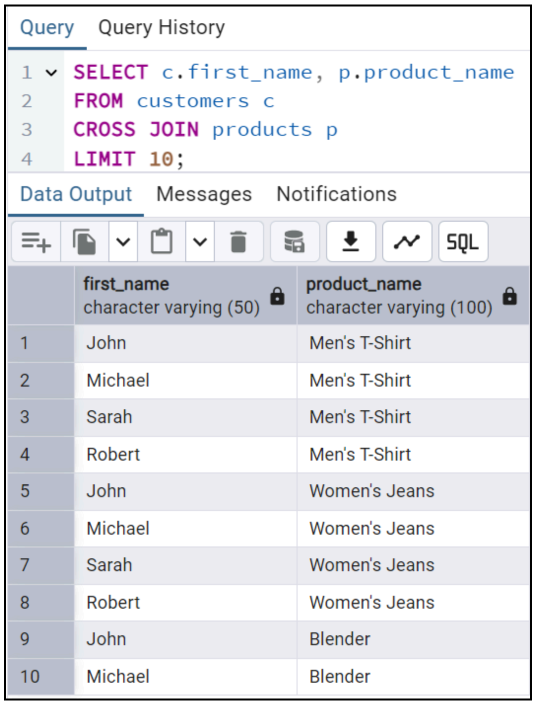
```sql
SELECT p.product_name, p.price,p.stock_quantity, oi.order_id, oi.quantity
FROM products p
RIGHT JOIN order_items oi ON p.product_id = oi.product_id;
```

	product_name character varying (100)	price numeric (10,2)	stock_quantity integer	order_id integer	quantity integer
1	Smartphone X	967.99	50	1	1
2	Blender	59.99	75	3	1
3	Men's T-Shirt	19.99	200	4	2
4	Women's Jeans	39.99	150	4	1
5	Men's T-Shirt	19.99	200	5	2



```sql
SELECT c.first_name, p.product_name FROM customers c
FULL OUTER JOIN products p ON true LIMIT 10;
```

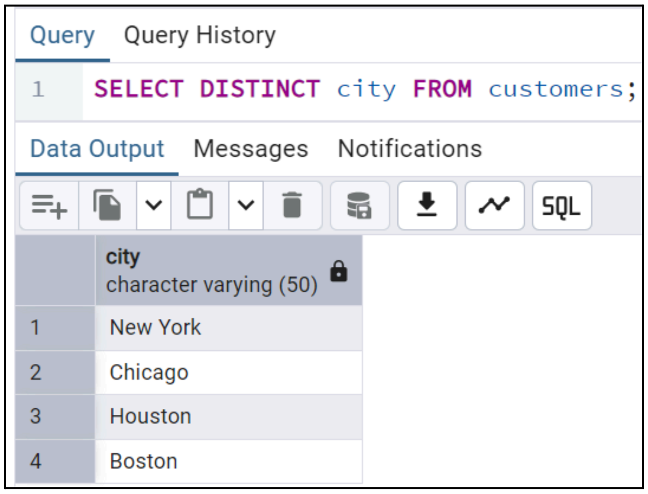
	first_name character varying (50)	product_name character varying (100)
1	John	Men's T-Shirt
2	John	Women's Jeans
3	John	Blender
4	John	Cookbook
5	John	Smartphone X
6	John	Wireless Earbuds
7	John	Smart Watch
8	John	Wireless Charger
9	Michael	Men's T-Shirt
10	Michael	Women's Jeans



```sql
SELECT c.first_name, p.product_name
FROM customers c
CROSS JOIN products p
LIMIT 10;
```

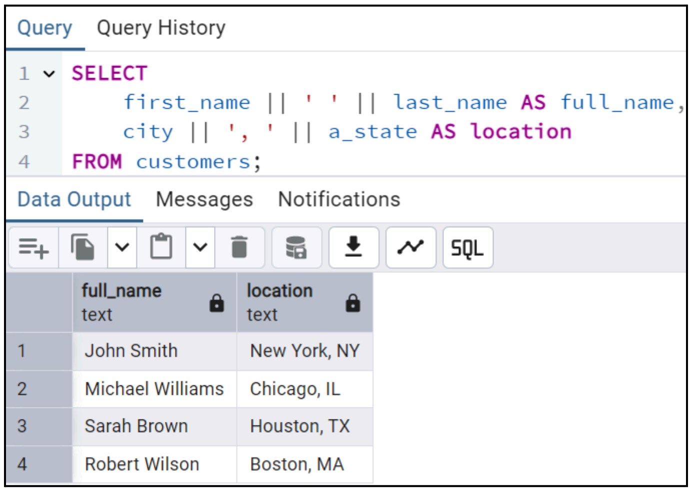
	first_name character varying (50)	product_name character varying (100)
1	John	Men's T-Shirt
2	Michael	Men's T-Shirt
3	Sarah	Men's T-Shirt
4	Robert	Men's T-Shirt
5	John	Women's Jeans
6	Michael	Women's Jeans
7	Sarah	Women's Jeans
8	Robert	Women's Jeans
9	John	Blender
10	Michael	Blender

7. **DISTINCT –** Removing Duplicates: is used to return unique values by eliminating duplicates. For example, SELECT DISTINCT city FROM customers; gives a list of cities with at least one customer, removing repeated entries



```sql
SELECT DISTINCT city FROM customers;
```

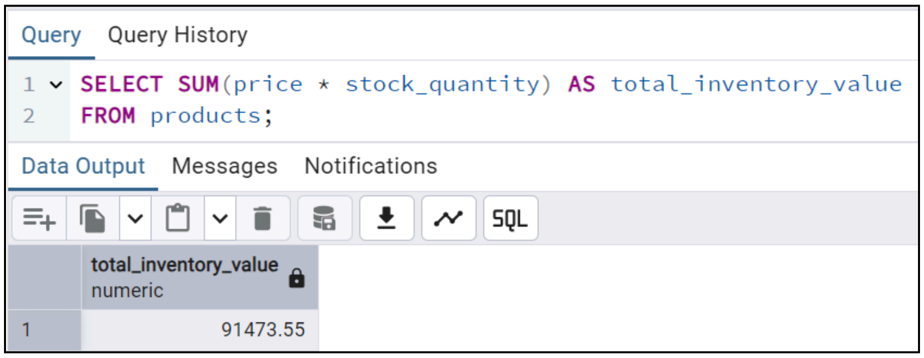
	city character varying (50)
1	New York
2	Chicago
3	Houston
4	Boston



```sql
SELECT
    first_name || ' ' || last_name AS full_name,
    city || ', ' || a_state AS location
FROM customers;
```

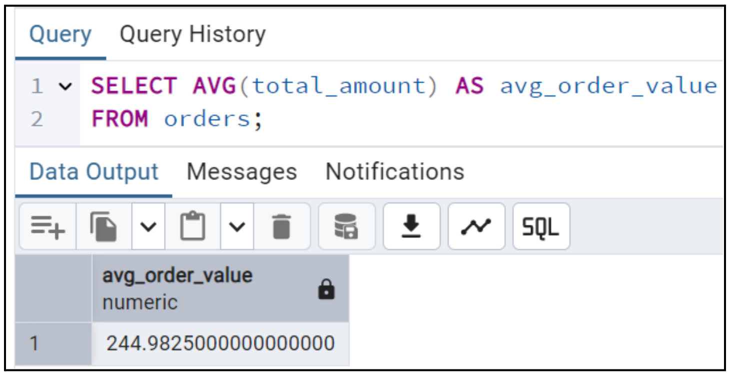
	full_name text	location text
1	John Smith	New York, NY
2	Michael Williams	Chicago, IL
3	Sarah Brown	Houston, TX
4	Robert Wilson	Boston, MA

8. **Aggregate Functions – Summarizing Data:** Functions like SUM, AVG, COUNT, MIN, MAX, and STDDEV summarize data. For instance, to compute the total inventory value: SELECT SUM(price * stock_quantity) FROM products;. To find the average order value: SELECT AVG(total_amount) FROM orders;. These are key in analytics and reporting.



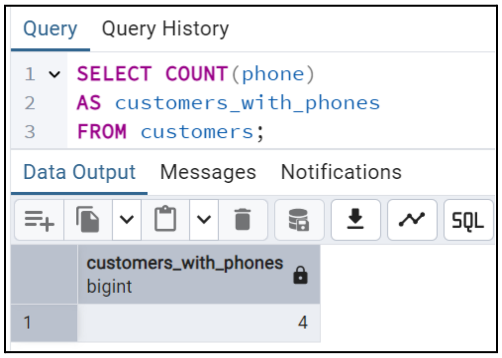
```sql
SELECT SUM(price * stock_quantity) AS total_inventory_value
FROM products;
```

```
  total_inventory_value
  numeric
1            91473.55
```



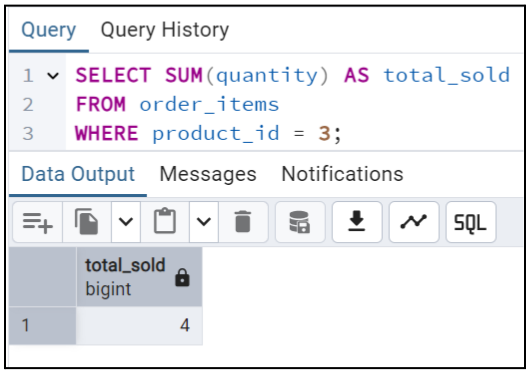
```sql
SELECT AVG(total_amount) AS avg_order_value
FROM orders;
```

```
  avg_order_value
  numeric
1 244.9825000000000000
```



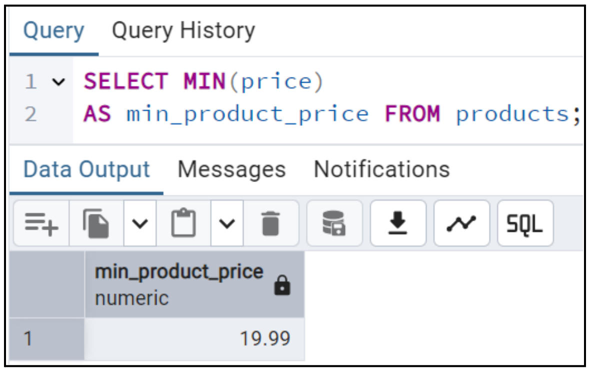
```sql
SELECT COUNT(phone)
AS customers_with_phones
FROM customers;
```

```
  customers_with_phones
  bigint
1                     4
```



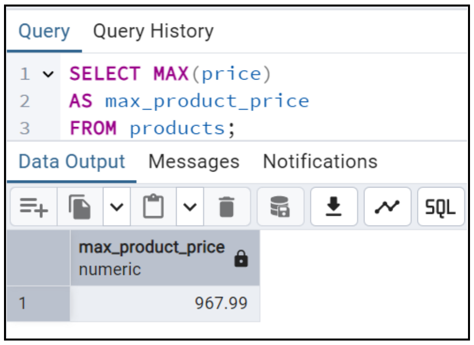
```sql
SELECT SUM(quantity) AS total_sold
FROM order_items
WHERE product_id = 3;
```

```
  total_sold
  bigint
1          4
```



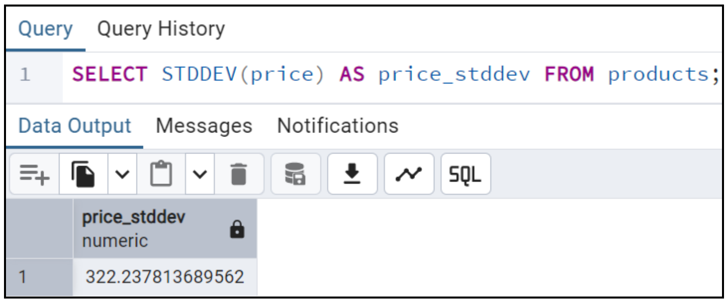
```sql
SELECT MIN(price)
AS min_product_price FROM products;
```

```
  min_product_price
  numeric
1             19.99
```



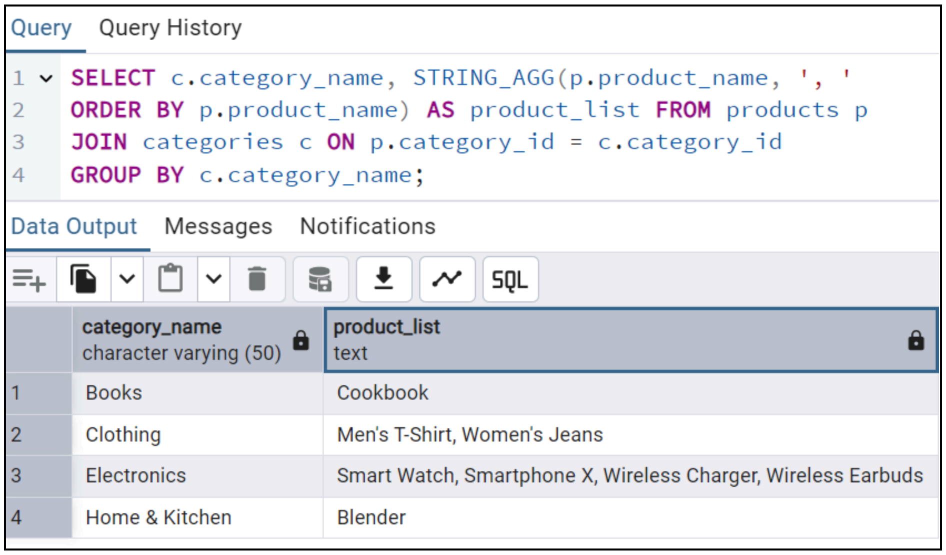
```sql
SELECT MAX(price)
AS max_product_price
FROM products;
```

```
  max_product_price
  numeric
1            967.99
```



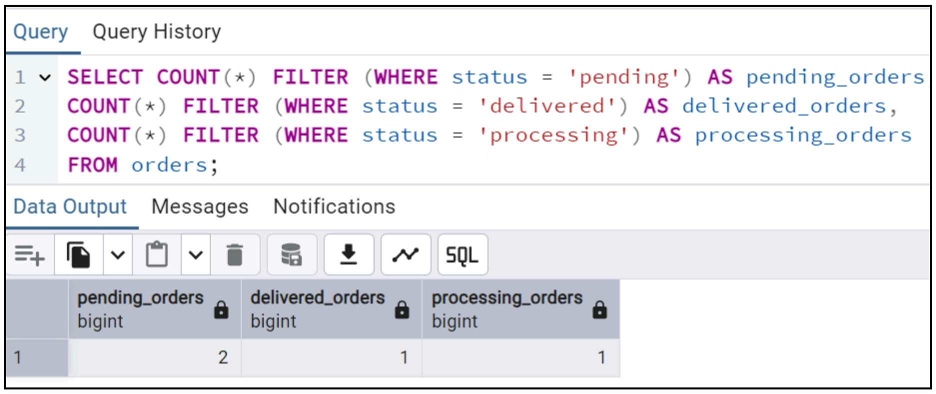
```sql
SELECT STDDEV(price) AS price_stddev FROM products;
```

```
  price_stddev
  numeric
1 322.237813689562
```



```sql
SELECT c.category_name, STRING_AGG(p.product_name, ', '
ORDER BY p.product_name) AS product_list FROM products p
JOIN categories c ON p.category_id = c.category_id
GROUP BY c.category_name;
```

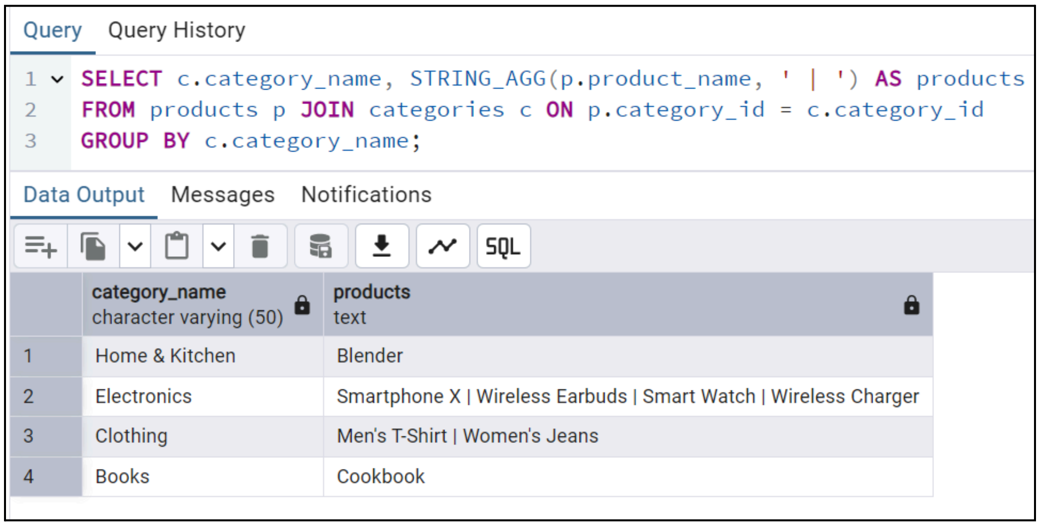
	category_name character varying (50)	product_list text
1	Books	Cookbook
2	Clothing	Men's T-Shirt, Women's Jeans
3	Electronics	Smart Watch, Smartphone X, Wireless Charger, Wireless Earbuds
4	Home & Kitchen	Blender



```sql
SELECT COUNT(*) FILTER (WHERE status = 'pending') AS pending_orders
COUNT(*) FILTER (WHERE status = 'delivered') AS delivered_orders,
COUNT(*) FILTER (WHERE status = 'processing') AS processing_orders
FROM orders;
```

	pending_orders bigint	delivered_orders bigint	processing_orders bigint
1	2	1	1

9. **GROUP BY – Grouping Data:** GROUP BY aggregates rows with the same values in specified columns into summary rows.



```sql
SELECT c.category_name, STRING_AGG(p.product_name, ' | ') AS products
FROM products p JOIN categories c ON p.category_id = c.category_id
GROUP BY c.category_name;
```

	category_name character varying (50)	products text
1	Home & Kitchen	Blender
2	Electronics	Smartphone X | Wireless Earbuds | Smart Watch | Wireless Charger
3	Clothing	Men's T-Shirt | Women's Jeans
4	Books	Cookbook