SQL Project: Advanced Relational Schema and Analytical Queries for an Online Store

-Lohitaaksh Bethaa

INDEX

1. Executive summary

This report delivers a detailed analysis of customer purchasing behavior within an online retail environment, based on data from over 100 customer transactions. The primary objective was to identify patterns and insights that can inform strategic decision-making across marketing, inventory management, and customer engagement functions.

Our analysis revealed that a small subset of loyal customers contributes disproportionately to overall revenue, highlighting the importance of targeted retention strategies. Additionally, seasonal fluctuations were observed, with significant sales spikes during holiday periods, suggesting the need for proactive inventory planning.

Top-selling products consistently belonged to specific categories, underlining the opportunity to expand those lines. The report also uncovers underperforming segments that could benefit from promotional campaigns or reevaluation.

Based on these findings, we recommend implementing a loyalty program for high-value customers, refining marketing efforts toward the most responsive customer segments, and optimizing inventory based on sales trends. These insights are intended to support more data-driven decisions and increase both customer satisfaction and profitability.

2. Introduction

Leveraging consumer behavior is essential to maximizing sales and improving customer retention in the crowded world of e-commerce. The goal of this report is to evaluate transactional data from the online store, with the expectation of revealing meaningful behaviors related to buying. Using structured query language (SQL) helped me maneuver through customer demographic information, frequency of orders, most popular products by revenue, and total contribution to revenue.

The analysis examines four primary tables: **customers**, **products**, **orders**, and **order_items,** and spans over 100 rows of realistic transactional data. The intention of this report is to provide findings that are actionable by the marketing and sales teams to improve their decision making. Stakeholders include the operations teams, customer success managers, and senior leadership.

3. Creation of Tables

Introduction to command:

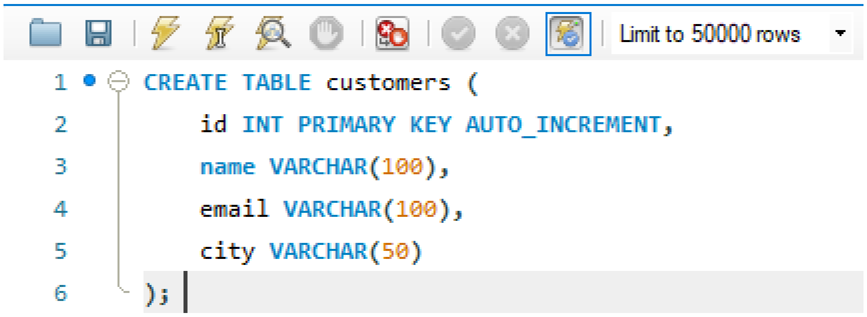
- Here we have used PRIMARY KEY and AUTO_INCREMENET functions in SQL to not allow duplication of characters for customer ID in customer table, order ID in ORDERS and ORDERS

ITEMS table and product ID in products table. The code is as follows.

- Rest all objects in the table are following traditional data types to support data processing.

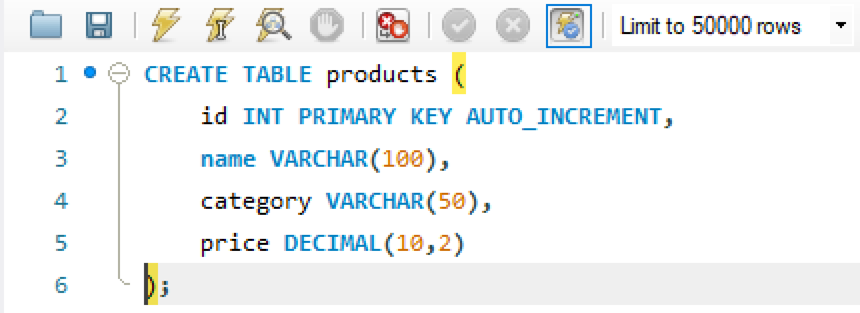
Commands:

Customer Table:



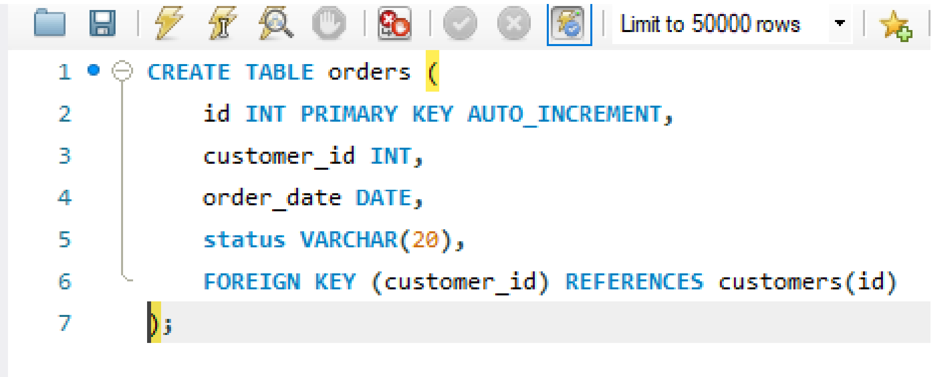
```
CREATE TABLE customers (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    email VARCHAR(100),
    city VARCHAR(50)
);
```

Product Table:



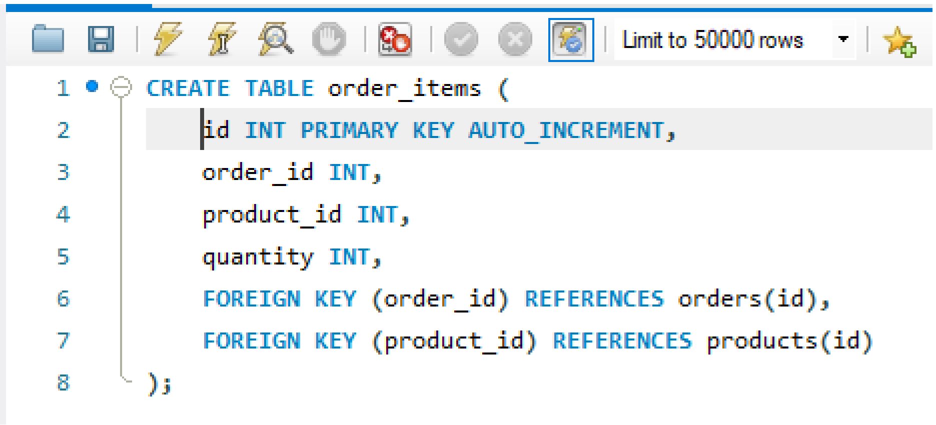
```
CREATE TABLE products (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    category VARCHAR(50),
    price DECIMAL(10,2)
);
```

Order Table:



```sql
CREATE TABLE orders (
    id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    order_date DATE,
    status VARCHAR(20),
    FOREIGN KEY (customer_id) REFERENCES customers(id)
);
```

Order Items Table:



```sql
CREATE TABLE order_items (
    id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    product_id INT,
    quantity INT,
    FOREIGN KEY (order_id) REFERENCES orders(id),
    FOREIGN KEY (product_id) REFERENCES products(id)
);
```

Logic and Reasoning:

The main approach for using the above methods is to design a normalized relational schema that supports data integrity, scalability, and analytical querying by separating products, customers, orders, and items into distinct tables with appropriate keys and constraints.

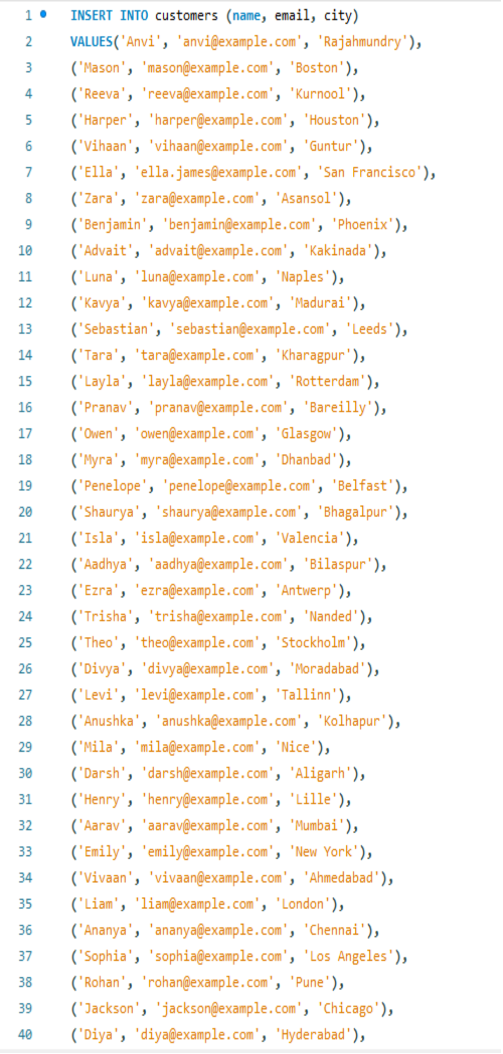
4. Data Insertion

Introduction to Commands:

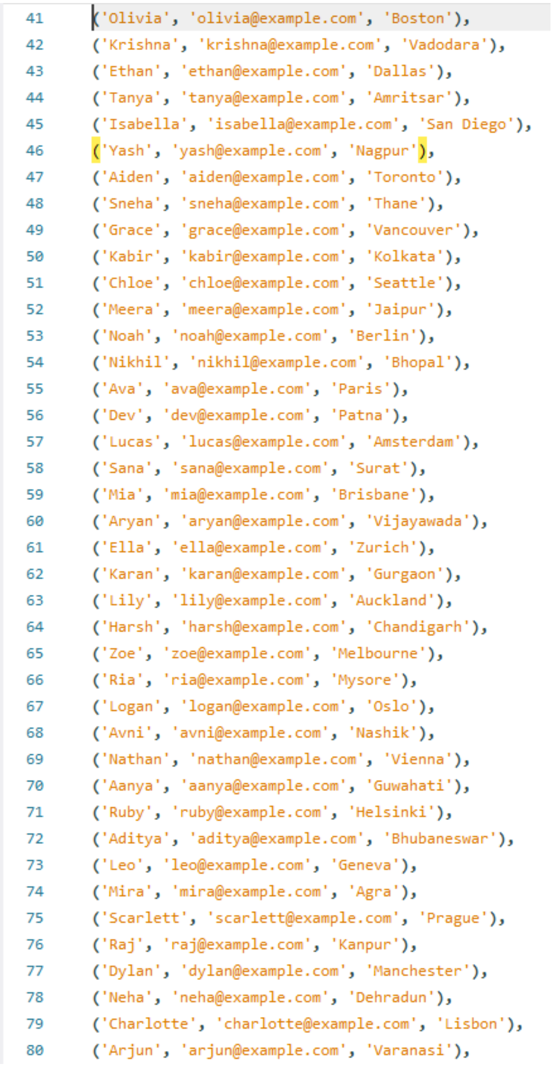
Here we have inserted data into the respective tables use INSERT INTO and VALUES statement.

Commands

Inserting in customer table:

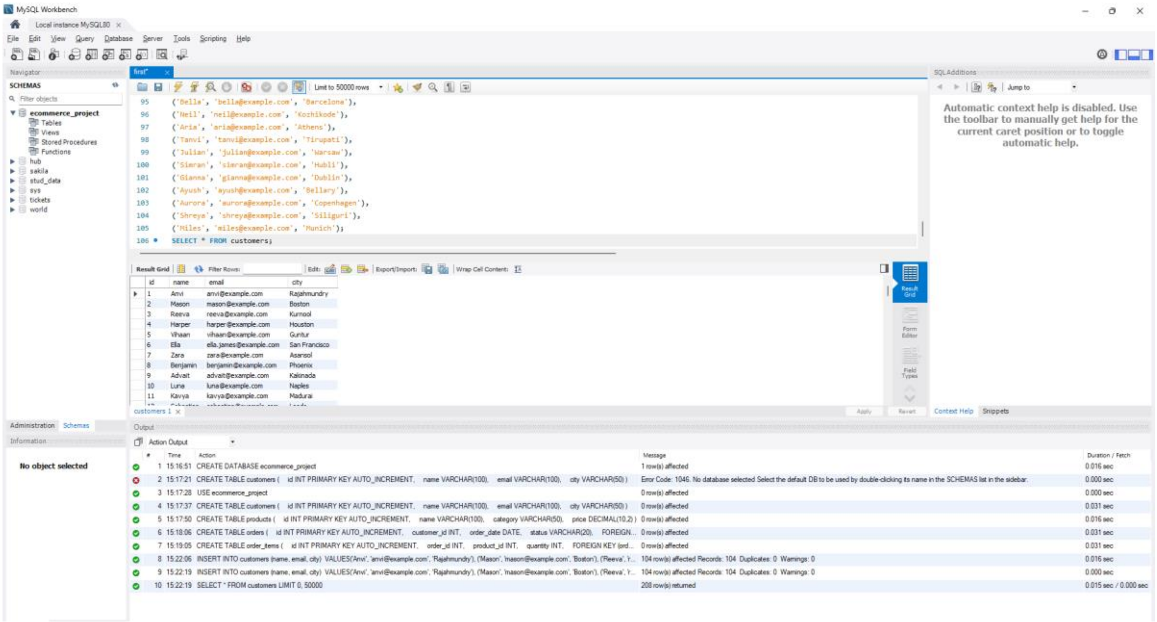


```sql
INSERT INTO customers (name, email, city)
VALUES('Anvi', 'anvi@example.com', 'Rajahmundry'),
('Mason', 'mason@example.com', 'Boston'),
('Reeva', 'reeva@example.com', 'Kurnool'),
('Harper', 'harper@example.com', 'Houston'),
('Vihaan', 'vihaan@example.com', 'Guntur'),
('Ella', 'ella.james@example.com', 'San Francisco'),
('Zara', 'zara@example.com', 'Asansol'),
('Benjamin', 'benjamin@example.com', 'Phoenix'),
('Advait', 'advait@example.com', 'Kakinada'),
('Luna', 'luna@example.com', 'Naples'),
('Kavya', 'kavya@example.com', 'Madurai'),
('Sebastian', 'sebastian@example.com', 'Leeds'),
('Tara', 'tara@example.com', 'Kharagpur'),
('Layla', 'layla@example.com', 'Rotterdam'),
('Pranav', 'pranav@example.com', 'Bareilly'),
('Owen', 'owen@example.com', 'Glasgow'),
('Myra', 'myra@example.com', 'Dhanbad'),
('Penelope', 'penelope@example.com', 'Belfast'),
('Shaurya', 'shaurya@example.com', 'Bhagalpur'),
('Isla', 'isla@example.com', 'Valencia'),
('Aadhya', 'aadhya@example.com', 'Bilaspur'),
('Ezra', 'ezra@example.com', 'Antwerp'),
('Trisha', 'trisha@example.com', 'Nanded'),
('Theo', 'theo@example.com', 'Stockholm'),
('Divya', 'divya@example.com', 'Moradabad'),
('Levi', 'levi@example.com', 'Tallinn'),
('Anushka', 'anushka@example.com', 'Kolhapur'),
('Mila', 'mila@example.com', 'Nice'),
('Darsh', 'darsh@example.com', 'Aligarh'),
('Henry', 'henry@example.com', 'Lille'),
('Aarav', 'aarav@example.com', 'Mumbai'),
('Emily', 'emily@example.com', 'New York'),
('Vivaan', 'vivaan@example.com', 'Ahmedabad'),
('Liam', 'liam@example.com', 'London'),
('Ananya', 'ananya@example.com', 'Chennai'),
('Sophia', 'sophia@example.com', 'Los Angeles'),
('Rohan', 'rohan@example.com', 'Pune'),
('Jackson', 'jackson@example.com', 'Chicago'),
('Diya', 'diya@example.com', 'Hyderabad'),
```



```sql
('Olivia', 'olivia@example.com', 'Boston'),
('Krishna', 'krishna@example.com', 'Vadodara'),
('Ethan', 'ethan@example.com', 'Dallas'),
('Tanya', 'tanya@example.com', 'Amritsar'),
('Isabella', 'isabella@example.com', 'San Diego'),
('Yash', 'yash@example.com', 'Nagpur'),
('Aiden', 'aiden@example.com', 'Toronto'),
('Sneha', 'sneha@example.com', 'Thane'),
('Grace', 'grace@example.com', 'Vancouver'),
('Kabir', 'kabir@example.com', 'Kolkata'),
('Chloe', 'chloe@example.com', 'Seattle'),
('Meera', 'meera@example.com', 'Jaipur'),
('Noah', 'noah@example.com', 'Berlin'),
('Nikhil', 'nikhil@example.com', 'Bhopal'),
('Ava', 'ava@example.com', 'Paris'),
('Dev', 'dev@example.com', 'Patna'),
('Lucas', 'lucas@example.com', 'Amsterdam'),
('Sana', 'sana@example.com', 'Surat'),
('Mia', 'mia@example.com', 'Brisbane'),
('Aryan', 'aryan@example.com', 'Vijayawada'),
('Ella', 'ella@example.com', 'Zurich'),
('Karan', 'karan@example.com', 'Gurgaon'),
('Lily', 'lily@example.com', 'Auckland'),
('Harsh', 'harsh@example.com', 'Chandigarh'),
('Zoe', 'zoe@example.com', 'Melbourne'),
('Ria', 'ria@example.com', 'Mysore'),
('Logan', 'logan@example.com', 'Oslo'),
('Avni', 'avni@example.com', 'Nashik'),
('Nathan', 'nathan@example.com', 'Vienna'),
('Aanya', 'aanya@example.com', 'Guwahati'),
('Ruby', 'ruby@example.com', 'Helsinki'),
('Aditya', 'aditya@example.com', 'Bhubaneswar'),
('Leo', 'leo@example.com', 'Geneva'),
('Mira', 'mira@example.com', 'Agra'),
('Scarlett', 'scarlett@example.com', 'Prague'),
('Raj', 'raj@example.com', 'Kanpur'),
('Dylan', 'dylan@example.com', 'Manchester'),
('Neha', 'neha@example.com', 'Dehradun'),
('Charlotte', 'charlotte@example.com', 'Lisbon'),
('Arjun', 'arjun@example.com', 'Varanasi'),
```

Output:



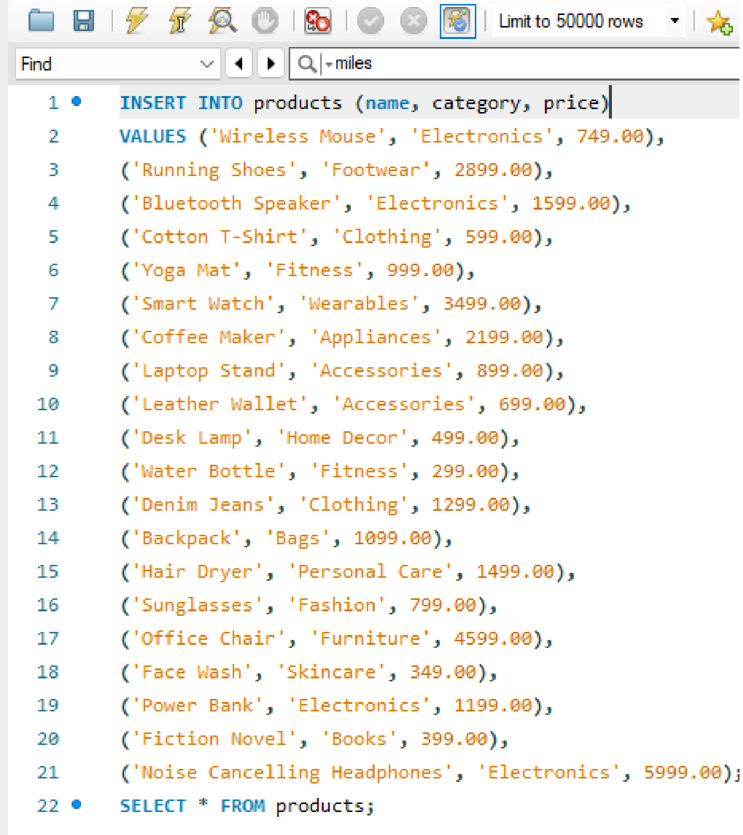
```sql
('Bella', 'bella@example.com', 'Barcelona'),
('Neil', 'neil@example.com', 'Kozhikode'),
('Aria', 'aria@example.com', 'Athens'),
('Tanvi', 'tanvi@example.com', 'Tirupati'),
('Julian', 'julian@example.com', 'Warsaw'),
('Simran', 'simran@example.com', 'Hubli'),
('Gianna', 'gianna@example.com', 'Dublin'),
('Ayush', 'ayush@example.com', 'Bellary'),
('Aurora', 'aurora@example.com', 'Copenhagen'),
('Shreya', 'shreya@example.com', 'Siliguri'),
('Miles', 'miles@example.com', 'Munich');
SELECT * FROM customers;
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

id	name	email	city
1	Anvi	anvi@example.com	Rajahmundry
2	Mason	mason@example.com	Boston
3	Reeva	reeva@example.com	Kurnool
4	Harper	harper@example.com	Houston
5	Vihaan	vihaan@example.com	Guntur
6	Ella	ella.james@example.com	San Francisco
7	Zara	zara@example.com	Asansol
8	Benjamin	benjamin@example.com	Phoenix
9	Advait	advait@example.com	Kakinada
10	Luna	luna@example.com	Naples
11	Kavya	kavya@example.com	Madurai

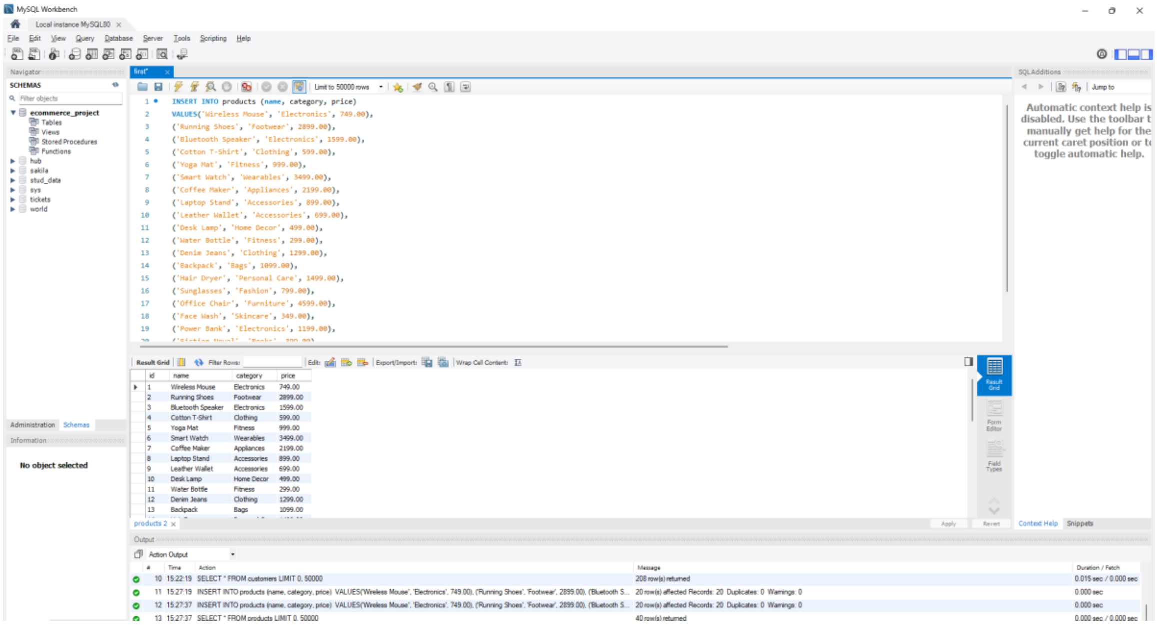
```
#   Time      Action                                                                                                                                          Message                                                                                                  Duration / Fetch
1   15:16:51  CREATE DATABASE ecommerce_project                                                                                                               1 row(s) affected                                                                                        0.016 sec
2   15:17:21  CREATE TABLE customers (    id INT PRIMARY KEY AUTO_INCREMENT,    name VARCHAR(100),    email VARCHAR(100),    city VARCHAR(50) )              Error Code: 1046. No database selected Select the default DB to be used by double-clicking its name in the SCHEMAS list in the sidebar.   0.000 sec
3   15:17:28  USE ecommerce_project                                                                                                                           0 row(s) affected                                                                                        0.000 sec
4   15:17:37  CREATE TABLE customers (    id INT PRIMARY KEY AUTO_INCREMENT,    name VARCHAR(100),    email VARCHAR(100),    city VARCHAR(50) )              0 row(s) affected                                                                                        0.031 sec
5   15:17:50  CREATE TABLE products (    id INT PRIMARY KEY AUTO_INCREMENT,    name VARCHAR(100),    category VARCHAR(50),    price DECIMAL(10,2) )          0 row(s) affected                                                                                        0.016 sec
6   15:18:06  CREATE TABLE orders (    id INT PRIMARY KEY AUTO_INCREMENT,    customer_id INT,    order_date DATE,    status VARCHAR(20),    FOREIGN...        0 row(s) affected                                                                                        0.031 sec
7   15:19:05  CREATE TABLE order_items (    id INT PRIMARY KEY AUTO_INCREMENT,    order_id INT,    product_id INT,    quantity INT,    FOREIGN KEY (ord...     0 row(s) affected                                                                                        0.031 sec
8   15:22:06  INSERT INTO customers (name, email, city)  VALUES('Anvi', 'anvi@example.com', 'Rajahmundry'), ('Mason', 'mason@example.com', 'Boston'), ('Reeva', 'r...   104 row(s) affected Records: 104  Duplicates: 0  Warnings: 0                              0.016 sec
9   15:22:19  INSERT INTO customers (name, email, city)  VALUES('Anvi', 'anvi@example.com', 'Rajahmundry'), ('Mason', 'mason@example.com', 'Boston'), ('Reeva', 'r...   104 row(s) affected Records: 104  Duplicates: 0  Warnings: 0                              0.000 sec
10  15:22:19  SELECT * FROM customers LIMIT 0, 50000                                                                                                          208 row(s) returned                                                                                      0.015 sec / 0.000 sec
```

Inserting into Product table:



```sql
INSERT INTO products (name, category, price)
VALUES ('Wireless Mouse', 'Electronics', 749.00),
('Running Shoes', 'Footwear', 2899.00),
('Bluetooth Speaker', 'Electronics', 1599.00),
('Cotton T-Shirt', 'Clothing', 599.00),
('Yoga Mat', 'Fitness', 999.00),
('Smart Watch', 'Wearables', 3499.00),
('Coffee Maker', 'Appliances', 2199.00),
('Laptop Stand', 'Accessories', 899.00),
('Leather Wallet', 'Accessories', 699.00),
('Desk Lamp', 'Home Decor', 499.00),
('Water Bottle', 'Fitness', 299.00),
('Denim Jeans', 'Clothing', 1299.00),
('Backpack', 'Bags', 1099.00),
('Hair Dryer', 'Personal Care', 1499.00),
('Sunglasses', 'Fashion', 799.00),
('Office Chair', 'Furniture', 4599.00),
('Face Wash', 'Skincare', 349.00),
('Power Bank', 'Electronics', 1199.00),
('Fiction Novel', 'Books', 399.00),
('Noise Cancelling Headphones', 'Electronics', 5999.00);
SELECT * FROM products;
```

Output:



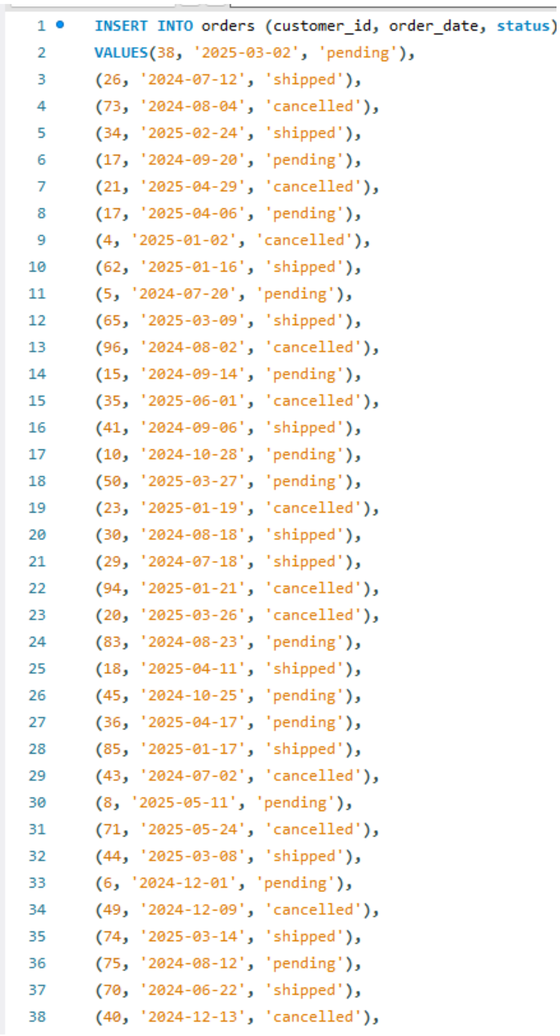
```
INSERT INTO products (name, category, price)
VALUES('Wireless Mouse', 'Electronics', 749.00),
('Running Shoes', 'Footwear', 2899.00),
('Bluetooth Speaker', 'Electronics', 1599.00),
('Cotton T-Shirt', 'Clothing', 599.00),
('Yoga Mat', 'Fitness', 999.00),
('Smart Watch', 'Wearables', 3499.00),
('Coffee Maker', 'Appliances', 2199.00),
('Laptop Stand', 'Accessories', 899.00),
('Leather Wallet', 'Accessories', 699.00),
('Desk Lamp', 'Home Decor', 499.00),
('Water Bottle', 'Fitness', 299.00),
('Denim Jeans', 'Clothing', 1299.00),
('Backpack', 'Bags', 1099.00),
('Hair Dryer', 'Personal Care', 1499.00),
('Sunglasses', 'Fashion', 799.00),
('Office Chair', 'Furniture', 4599.00),
('Face Wash', 'Skincare', 349.00),
('Power Bank', 'Electronics', 1199.00),
```

Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.

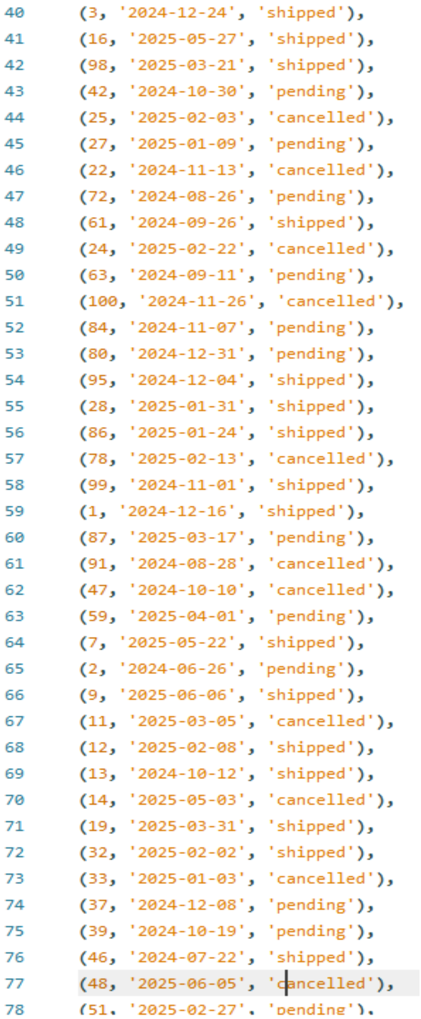
id	name	category	price
1	Wireless Mouse	Electronics	749.00
2	Running Shoes	Footwear	2899.00
3	Bluetooth Speaker	Electronics	1599.00
4	Cotton T-Shirt	Clothing	599.00
5	Yoga Mat	Fitness	999.00
6	Smart Watch	Wearables	3499.00
7	Coffee Maker	Appliances	2199.00
8	Laptop Stand	Accessories	899.00
9	Leather Wallet	Accessories	699.00
10	Desk Lamp	Home Decor	499.00
11	Water Bottle	Fitness	299.00
12	Denim Jeans	Clothing	1299.00
13	Backpack	Bags	1099.00

#	Time	Action	Message	Duration / Fetch
10	15:22:19	SELECT * FROM customers LIMIT 0, 50000	208 row(s) returned	0.015 sec / 0.000 sec
11	15:27:19	INSERT INTO products (name, category, price) VALUES('Wireless Mouse', 'Electronics', 749.00), ('Running Shoes', 'Footwear', 2899.00), ('Bluetooth S...	20 row(s) affected Records: 20 Duplicates: 0 Warnings: 0	0.000 sec
12	15:27:37	INSERT INTO products (name, category, price) VALUES('Wireless Mouse', 'Electronics', 749.00), ('Running Shoes', 'Footwear', 2899.00), ('Bluetooth S...	20 row(s) affected Records: 20 Duplicates: 0 Warnings: 0	0.000 sec
13	15:27:37	SELECT * FROM products LIMIT 0, 50000	40 row(s) returned	0.000 sec / 0.000 sec

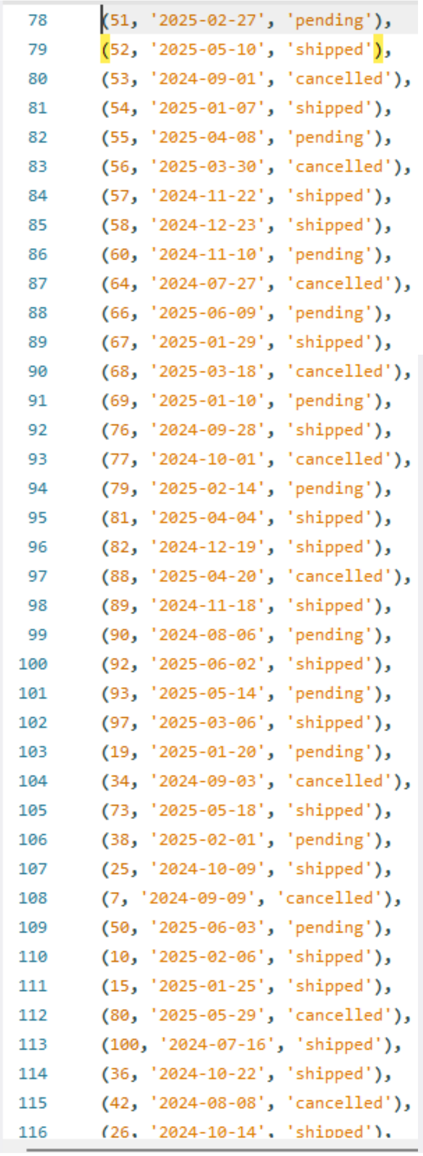
Inserting into Order table: (134 orders):



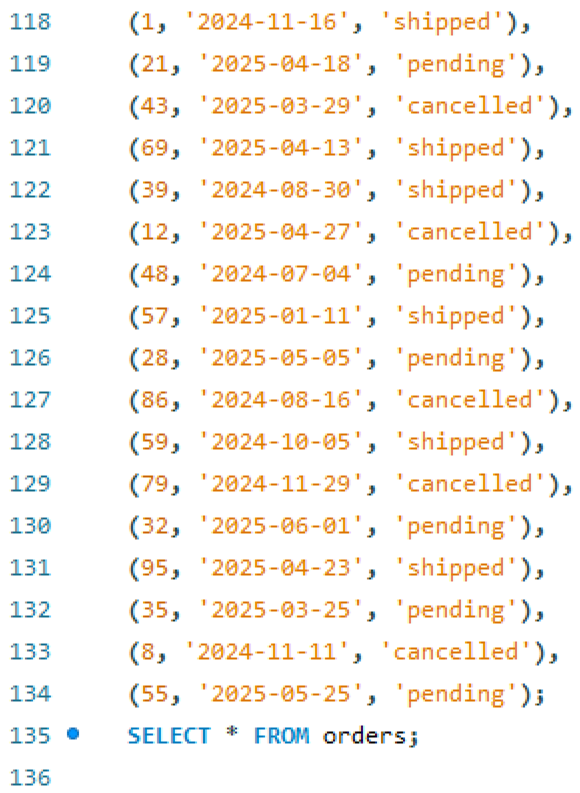
```
INSERT INTO orders (customer_id, order_date, status)
VALUES(38, '2025-03-02', 'pending'),
(26, '2024-07-12', 'shipped'),
(73, '2024-08-04', 'cancelled'),
(34, '2025-02-24', 'shipped'),
(17, '2024-09-20', 'pending'),
(21, '2025-04-29', 'cancelled'),
(17, '2025-04-06', 'pending'),
(4, '2025-01-02', 'cancelled'),
(62, '2025-01-16', 'shipped'),
(5, '2024-07-20', 'pending'),
(65, '2025-03-09', 'shipped'),
(96, '2024-08-02', 'cancelled'),
(15, '2024-09-14', 'pending'),
(35, '2025-06-01', 'cancelled'),
(41, '2024-09-06', 'shipped'),
(10, '2024-10-28', 'pending'),
(50, '2025-03-27', 'pending'),
(23, '2025-01-19', 'cancelled'),
(30, '2024-08-18', 'shipped'),
(29, '2024-07-18', 'shipped'),
(94, '2025-01-21', 'cancelled'),
(20, '2025-03-26', 'cancelled'),
(83, '2024-08-23', 'pending'),
(18, '2025-04-11', 'shipped'),
(45, '2024-10-25', 'pending'),
(36, '2025-04-17', 'pending'),
(85, '2025-01-17', 'shipped'),
(43, '2024-07-02', 'cancelled'),
(8, '2025-05-11', 'pending'),
(71, '2025-05-24', 'cancelled'),
(44, '2025-03-08', 'shipped'),
(6, '2024-12-01', 'pending'),
(49, '2024-12-09', 'cancelled'),
(74, '2025-03-14', 'shipped'),
(75, '2024-08-12', 'pending'),
(70, '2024-06-22', 'shipped'),
(40, '2024-12-13', 'cancelled'),
```



```
(3, '2024-12-24', 'shipped'),
(16, '2025-05-27', 'shipped'),
(98, '2025-03-21', 'shipped'),
(42, '2024-10-30', 'pending'),
(25, '2025-02-03', 'cancelled'),
(27, '2025-01-09', 'pending'),
(22, '2024-11-13', 'cancelled'),
(72, '2024-08-26', 'pending'),
(61, '2024-09-26', 'shipped'),
(24, '2025-02-22', 'cancelled'),
(63, '2024-09-11', 'pending'),
(100, '2024-11-26', 'cancelled'),
(84, '2024-11-07', 'pending'),
(80, '2024-12-31', 'pending'),
(95, '2024-12-04', 'shipped'),
(28, '2025-01-31', 'shipped'),
(86, '2025-01-24', 'shipped'),
(78, '2025-02-13', 'cancelled'),
(99, '2024-11-01', 'shipped'),
(1, '2024-12-16', 'shipped'),
(87, '2025-03-17', 'pending'),
(91, '2024-08-28', 'cancelled'),
(47, '2024-10-10', 'cancelled'),
(59, '2025-04-01', 'pending'),
(7, '2025-05-22', 'shipped'),
(2, '2024-06-26', 'pending'),
(9, '2025-06-06', 'shipped'),
(11, '2025-03-05', 'cancelled'),
(12, '2025-02-08', 'shipped'),
(13, '2024-10-12', 'shipped'),
(14, '2025-05-03', 'cancelled'),
(19, '2025-03-31', 'shipped'),
(32, '2025-02-02', 'shipped'),
(33, '2025-01-03', 'cancelled'),
(37, '2024-12-08', 'pending'),
(39, '2024-10-19', 'pending'),
(46, '2024-07-22', 'shipped'),
(48, '2025-06-05', 'cancelled'),
(51, '2025-02-27', 'pending'),
```

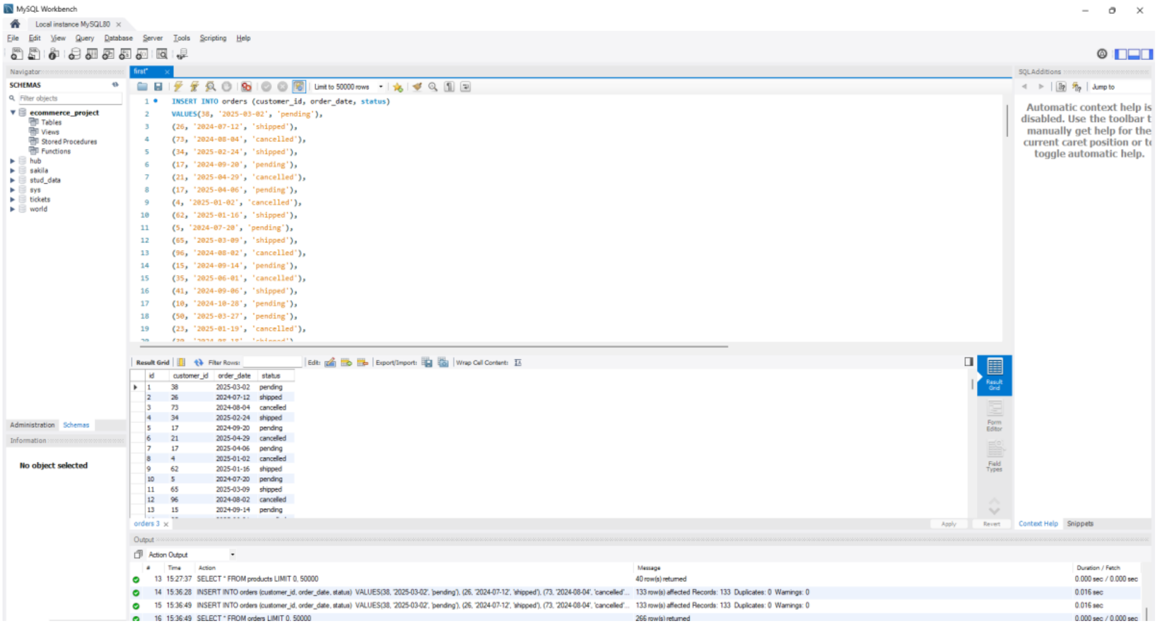


```
(51, '2025-02-27', 'pending'),
(52, '2025-05-10', 'shipped'),
(53, '2024-09-01', 'cancelled'),
(54, '2025-01-07', 'shipped'),
(55, '2025-04-08', 'pending'),
(56, '2025-03-30', 'cancelled'),
(57, '2024-11-22', 'shipped'),
(58, '2024-12-23', 'shipped'),
(60, '2024-11-10', 'pending'),
(64, '2024-07-27', 'cancelled'),
(66, '2025-06-09', 'pending'),
(67, '2025-01-29', 'shipped'),
(68, '2025-03-18', 'cancelled'),
(69, '2025-01-10', 'pending'),
(76, '2024-09-28', 'shipped'),
(77, '2024-10-01', 'cancelled'),
(79, '2025-02-14', 'pending'),
(81, '2025-04-04', 'shipped'),
(82, '2024-12-19', 'shipped'),
(88, '2025-04-20', 'cancelled'),
(89, '2024-11-18', 'shipped'),
(90, '2024-08-06', 'pending'),
(92, '2025-06-02', 'shipped'),
(93, '2025-05-14', 'pending'),
(97, '2025-03-06', 'shipped'),
(19, '2025-01-20', 'pending'),
(34, '2024-09-03', 'cancelled'),
(73, '2025-05-18', 'shipped'),
(38, '2025-02-01', 'pending'),
(25, '2024-10-09', 'shipped'),
(7, '2024-09-09', 'cancelled'),
(50, '2025-06-03', 'pending'),
(10, '2025-02-06', 'shipped'),
(15, '2025-01-25', 'shipped'),
(80, '2025-05-29', 'cancelled'),
(100, '2024-07-16', 'shipped'),
(36, '2024-10-22', 'shipped'),
(42, '2024-08-08', 'cancelled'),
(26, '2024-10-14', 'shipped'),
```



```
(1, '2024-11-16', 'shipped'),
(21, '2025-04-18', 'pending'),
(43, '2025-03-29', 'cancelled'),
(69, '2025-04-13', 'shipped'),
(39, '2024-08-30', 'shipped'),
(12, '2025-04-27', 'cancelled'),
(48, '2024-07-04', 'pending'),
(57, '2025-01-11', 'shipped'),
(28, '2025-05-05', 'pending'),
(86, '2024-08-16', 'cancelled'),
(59, '2024-10-05', 'shipped'),
(79, '2024-11-29', 'cancelled'),
(32, '2025-06-01', 'pending'),
(95, '2025-04-23', 'shipped'),
(35, '2025-03-25', 'pending'),
(8, '2024-11-11', 'cancelled'),
(55, '2025-05-25', 'pending');
SELECT * FROM orders;
```

Output:



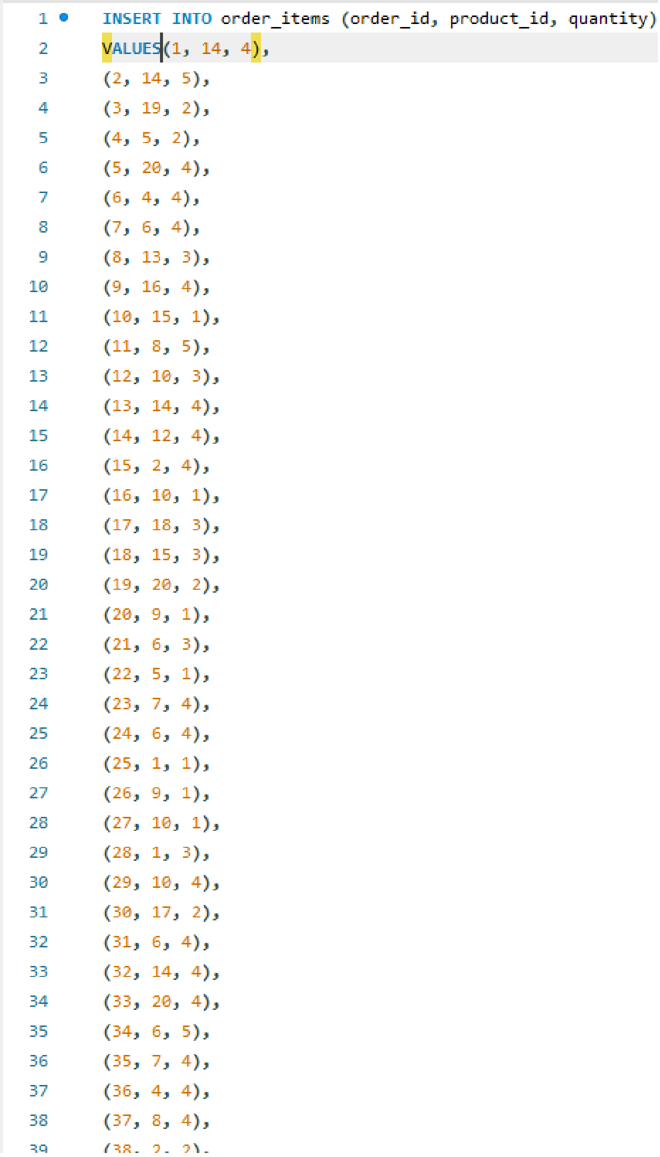
Automatic context help is disabled. Use the toolbar t manually get help for the current caret position or t toggle automatic help.

```sql
INSERT INTO orders (customer_id, order_date, status)
VALUES(38, '2025-03-02', 'pending'),
(26, '2024-07-12', 'shipped'),
(73, '2024-08-04', 'cancelled'),
(34, '2025-02-24', 'shipped'),
(17, '2024-09-20', 'pending'),
(21, '2025-04-29', 'cancelled'),
(17, '2025-04-06', 'pending'),
(4, '2025-01-02', 'cancelled'),
(62, '2025-01-16', 'shipped'),
(5, '2024-07-20', 'pending'),
(65, '2025-03-09', 'shipped'),
(96, '2024-08-02', 'cancelled'),
(15, '2024-09-14', 'pending'),
(35, '2025-06-01', 'cancelled'),
(41, '2024-09-06', 'shipped'),
(10, '2024-10-28', 'pending'),
(50, '2025-03-27', 'pending'),
(23, '2025-01-19', 'cancelled'),
```

id	customer_id	order_date	status
1	38	2025-03-02	pending
2	26	2024-07-12	shipped
3	73	2024-08-04	cancelled
4	34	2025-02-24	shipped
5	17	2024-09-20	pending
6	21	2025-04-29	cancelled
7	17	2025-04-06	pending
8	4	2025-01-02	cancelled
9	62	2025-01-16	shipped
10	5	2024-07-20	pending
11	65	2025-03-09	shipped
12	96	2024-08-02	cancelled
13	15	2024-09-14	pending

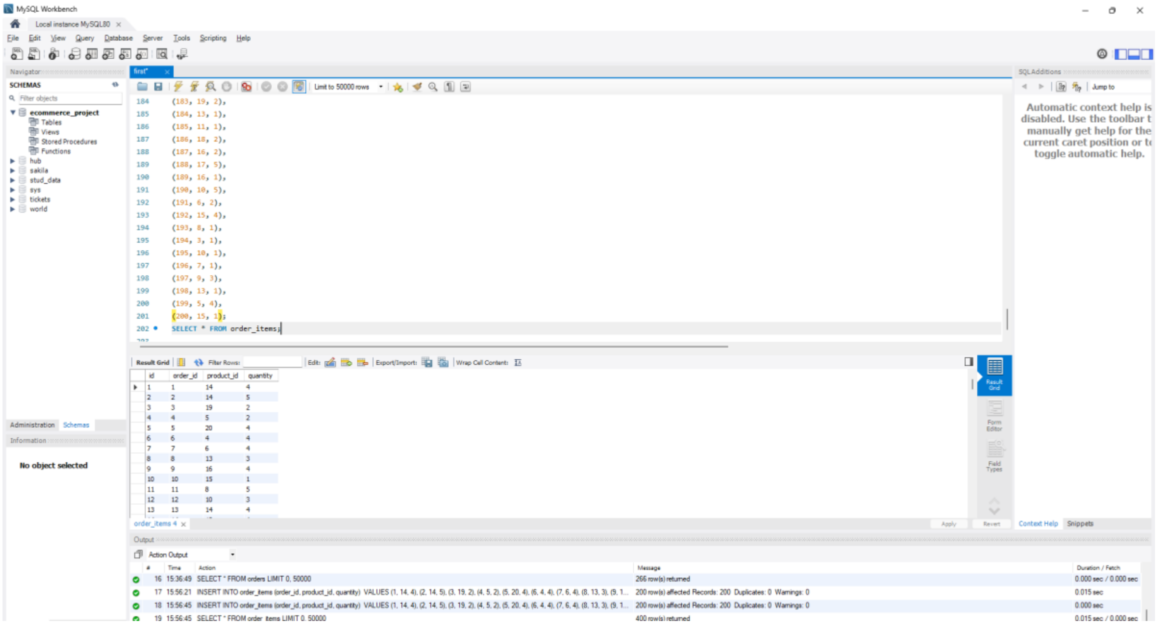
#	Time	Action	Message	Duration / Fetch
13	15:27:37	SELECT * FROM products LIMIT 0, 50000	40 row(s) returned	0.000 sec / 0.000 sec
14	15:36:28	INSERT INTO orders (customer_id, order_date, status) VALUES(38, '2025-03-02', 'pending'), (26, '2024-07-12', 'shipped'), (73, '2024-08-04', 'cancelled'...	133 row(s) affected Records: 133 Duplicates: 0 Warnings: 0	0.016 sec
15	15:36:49	INSERT INTO orders (customer_id, order_date, status) VALUES(38, '2025-03-02', 'pending'), (26, '2024-07-12', 'shipped'), (73, '2024-08-04', 'cancelled'...	133 row(s) affected Records: 133 Duplicates: 0 Warnings: 0	0.016 sec
16	15:36:49	SELECT * FROM orders LIMIT 0, 50000	266 row(s) returned	0.000 sec / 0.000 sec

Inserting into Order Items table:



```sql
INSERT INTO order_items (order_id, product_id, quantity)
VALUES(1, 14, 4),
(2, 14, 5),
(3, 19, 2),
(4, 5, 2),
(5, 20, 4),
(6, 4, 4),
(7, 6, 4),
(8, 13, 3),
(9, 16, 4),
(10, 15, 1),
(11, 8, 5),
(12, 10, 3),
(13, 14, 4),
(14, 12, 4),
(15, 2, 4),
(16, 10, 1),
(17, 18, 3),
(18, 15, 3),
(19, 20, 2),
(20, 9, 1),
(21, 6, 3),
(22, 5, 1),
(23, 7, 4),
(24, 6, 4),
(25, 1, 1),
(26, 9, 1),
(27, 10, 1),
(28, 1, 3),
(29, 10, 4),
(30, 17, 2),
(31, 6, 4),
(32, 14, 4),
(33, 20, 4),
(34, 6, 5),
(35, 7, 4),
(36, 4, 4),
(37, 8, 4),
(38, 2, 2),
```

Output:



```
(183, 19, 2),
(184, 13, 1),
(185, 11, 1),
(186, 18, 2),
(187, 16, 2),
(188, 17, 5),
(189, 16, 1),
(190, 10, 5),
(191, 6, 2),
(192, 15, 4),
(193, 8, 1),
(194, 3, 1),
(195, 10, 1),
(196, 7, 1),
(197, 9, 3),
(198, 13, 1),
(199, 5, 4),
(200, 15, 1);
SELECT * FROM order_items;
```

id	order_id	product_id	quantity
1	1	14	4
2	2	14	5
3	3	19	2
4	4	5	2
5	5	20	4
6	6	4	4
7	7	6	4
8	8	13	3
9	9	16	4
10	10	15	1
11	11	8	5
12	12	10	3
13	13	14	4

```
#   Time      Action                                                                                                                      Message                                                          Duration / Fetch
16  15:36:49  SELECT * FROM orders LIMIT 0, 50000                                                                                         266 row(s) returned                                              0.000 sec / 0.000 sec
17  15:56:21  INSERT INTO order_items (order_id, product_id, quantity) VALUES (1, 14, 4), (2, 14, 5), (3, 19, 2), (4, 5, 2), (5, 20, 4), (6, 4, 4), (7, 6, 4), (8, 13, 3), (9, 1...   200 row(s) affected Records: 200  Duplicates: 0  Warnings: 0   0.015 sec
18  15:56:45  INSERT INTO order_items (order_id, product_id, quantity) VALUES (1, 14, 4), (2, 14, 5), (3, 19, 2), (4, 5, 2), (5, 20, 4), (6, 4, 4), (7, 6, 4), (8, 13, 3), (9, 1...   200 row(s) affected Records: 200  Duplicates: 0  Warnings: 0   0.000 sec
19  15:56:45  SELECT * FROM order_items LIMIT 0, 50000                                                                                    400 row(s) returned                                              0.015 sec / 0.000 sec
```

Logic and Reasoning:

The thought was to incorporate populated realistic and varied entries across all tables to simulate real-world store transactions and enable meaningful analytics.

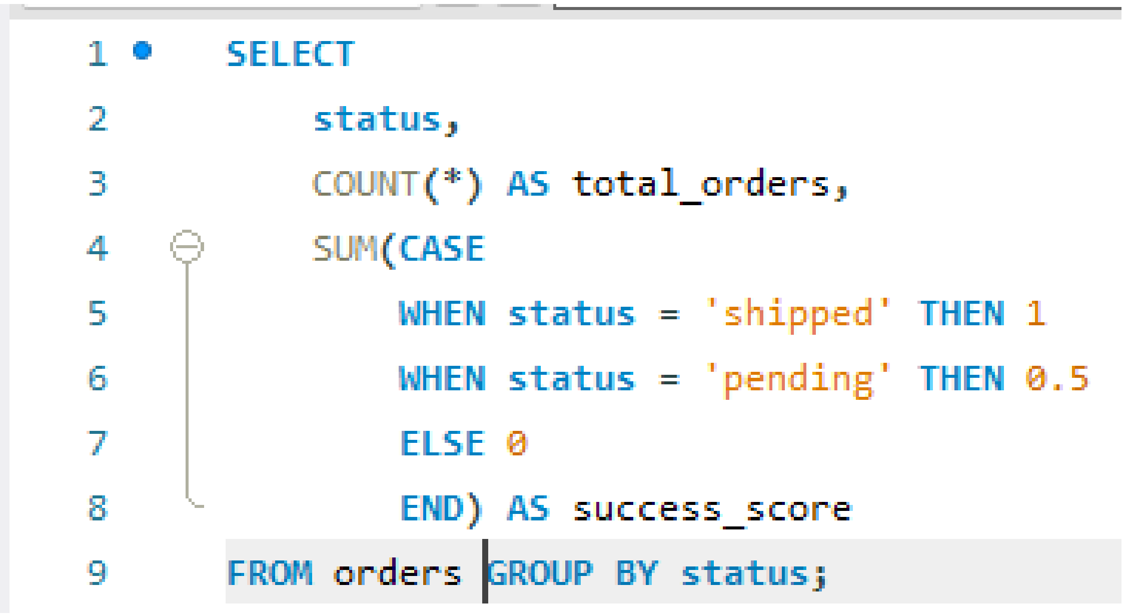
5. CASE Study 1: Order Status Insights

Introduction to Command:

- The task is to find and count as to how many orders are shipped, pending or have failed delivery. To do that we have used COUNT to find what order fall into the above 3 statuses.

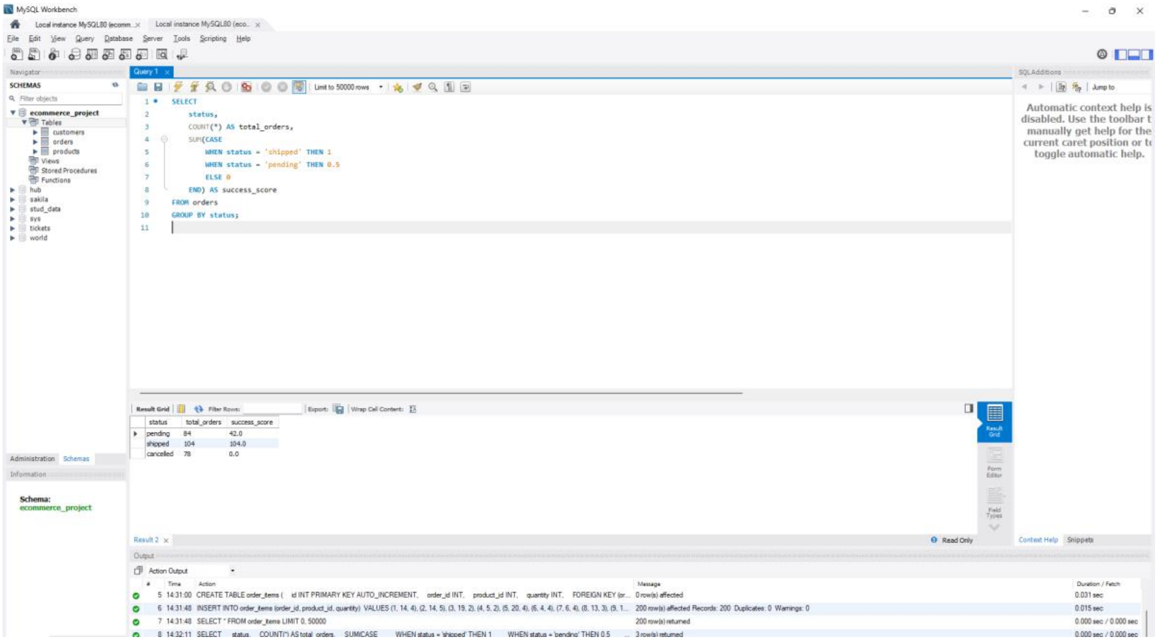
- Furter using SUM and assigning a new column called success score, we can find that success score of the orders corresponding to their status.

Code:



```sql
SELECT
    status,
    COUNT(*) AS total_orders,
    SUM(CASE
        WHEN status = 'shipped' THEN 1
        WHEN status = 'pending' THEN 0.5
        ELSE 0
        END) AS success_score
FROM orders GROUP BY status;
```

Output:



```
SELECT
    status,
    COUNT(*) AS total_orders,
    SUM(CASE
        WHEN status = 'shipped' THEN 1
        WHEN status = 'pending' THEN 0.5
        ELSE 0
    END) AS success_score
FROM orders
GROUP BY status;
```

status	total_orders	success_score
pending	84	42.0
shipped	104	104.0
cancelled	78	0.0

```
#  Time      Action                                                                                          Message                                               Duration / Fetch
5  14:31:00  CREATE TABLE order_items (  id INT PRIMARY KEY AUTO_INCREMENT,  order_id INT,  product_id INT,  quantity INT,  FOREIGN KEY (or...  0 row(s) affected  0.031 sec
6  14:31:48  INSERT INTO order_items (order_id, product_id, quantity) VALUES (1, 14, 4), (2, 14, 5), (3, 19, 2), (4, 5, 2), (5, 20, 4), (6, 4, 4), (7, 6, 4), (8, 13, 3), (9, 1...  200 row(s) affected Records: 200  Duplicates: 0  Warnings: 0  0.015 sec
7  14:31:48  SELECT * FROM order_items LIMIT 0, 50000                                                        200 row(s) returned                                   0.000 sec / 0.000 sec
8  14:32:11  SELECT   status,   COUNT(*) AS total_orders,   SUM(CASE    WHEN status = 'shipped' THEN 1    WHEN status = 'pending' THEN 0.5   ...  3 row(s) returned  0.000 sec / 0.000 sec
```

Logic and Reasoning

The logic behind this query is to evaluate the overall performance of orders by categorizing them based on their status and assigning a weighted success score to each category. It groups all records in the `orders` table by their status (such as 'shipped', 'pending', or 'cancelled') and counts how many orders fall into each group. To measure how successful each status type is, the query assigns a value of 1 to 'shipped' orders, 0.5 to 'pending' orders, and 0 to all others using a `CASE` statement. These values are then summed to produce a total "success score" for each status category. This allows the analysis to go beyond just counts and quantify how effectively orders are being fulfilled, providing a performance measure that reflects both volume and order outcome. The approach was to help categorize orders based on status flags, enabling performance tracking and operational insights on fulfillment efficiency.

CASE Study 2: CTE – Top Customers by Spend

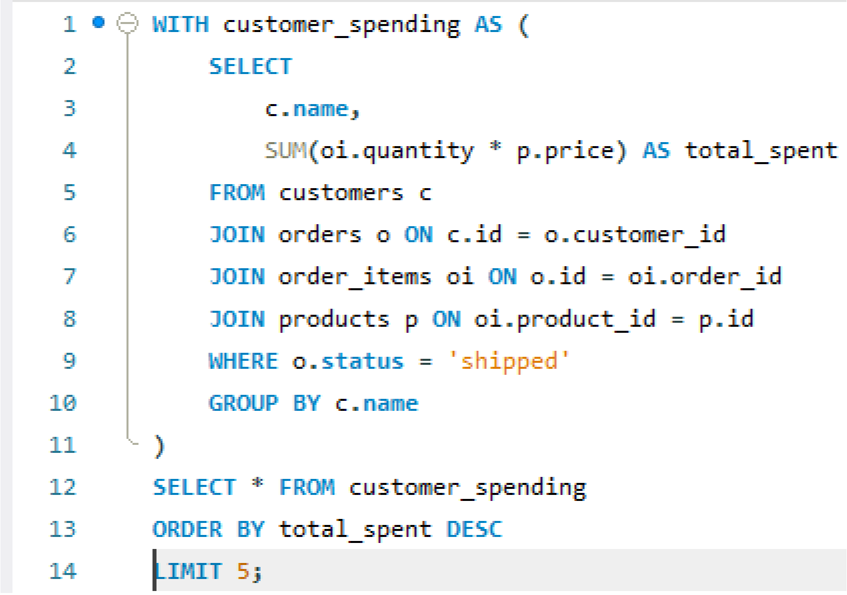
Introduction to command:

- This query is designed to identify the top 5 customers who have spent the most money on successfully shipped orders. It starts by calculating the total amount each customer has spent by multiplying the quantity of each product they ordered by the product's price.

- This calculation only includes orders that have been marked as "shipped," ensuring that only completed transactions are considered. The query gathers this information by joining the customers, orders, order items, and products tables. Once the total spending for each customer is calculated, it then ranks all customers in descending order based on how much they've spent and selects the top five.

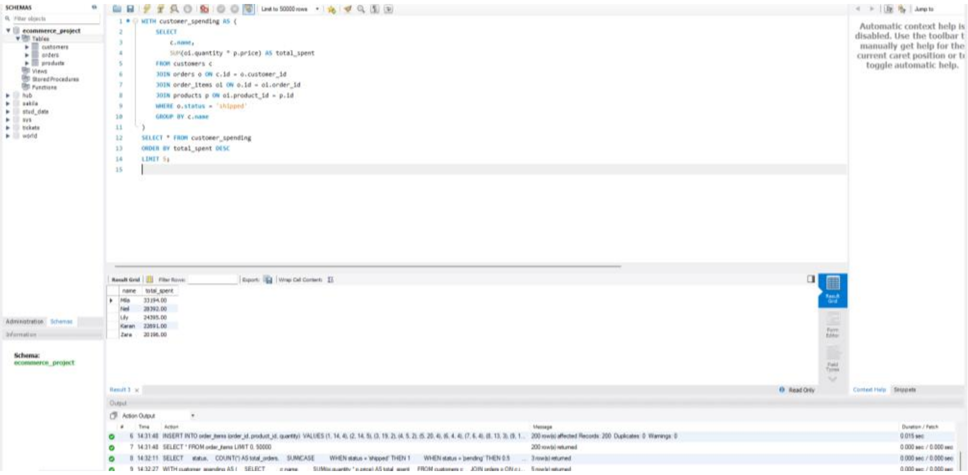
- The result shows the names of these top-spending customers along with how much they spent.

Command:



```sql
WITH customer_spending AS (
    SELECT
        c.name,
        SUM(oi.quantity * p.price) AS total_spent
    FROM customers c
    JOIN orders o ON c.id = o.customer_id
    JOIN order_items oi ON o.id = oi.order_id
    JOIN products p ON oi.product_id = p.id
    WHERE o.status = 'shipped'
    GROUP BY c.name
)
SELECT * FROM customer_spending
ORDER BY total_spent DESC
LIMIT 5;
```

Output:



Schema: ecommerce_project

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

name	total_spent
Mia	33194.00
Neil	28392.00
Lily	24395.00
Karan	22891.00
Zara	20196.00

#	Time	Action	Message	Duration / Fetch
6	14:31:48	INSERT INTO order_items (order_id, product_id, quantity) VALUES (1, 14, 4), (2, 14, 5), (3, 19, 2), (4, 5, 2), (5, 20, 4), (6, 4, 4), (7, 6, 4), (8, 13, 3), (9, 1...	200 row(s) affected Records: 200 Duplicates: 0 Warnings: 0	0.015 sec
7	14:31:48	SELECT * FROM order_items LIMIT 0, 50000	200 row(s) returned	0.000 sec / 0.000 sec
8	14:32:11	SELECT status, COUNT(*) AS total_orders, SUM(CASE WHEN status = 'shipped' THEN 1 WHEN status = 'pending' THEN 0.5 ...	3 row(s) returned	0.000 sec / 0.000 sec
9	14:32:27	WITH customer_spending AS (SELECT c.name, SUM(oi.quantity * p.price) AS total_spent FROM customers c JOIN orders o ON c.i...	5 row(s) returned	0.000 sec / 0.000 sec

Logic and Reasoning:

The logic behind the second question is to find the five customers who spent the most on orders that were successfully shipped. The calculation here is the total amount spent on each customer, which is just the product's price times the quantity of that product they purchased. All the data is joined across customers, orders, order_items, and products and filtered so that we only include orders whose status is 'shipped' to capture completed transactions that lead to sales. The results are grouped by customer name, and then the sum of the total amount spent for each customer is calculated and sorted by descending order. Then it simply displays only the five customers with the most amount spent to show the most important customers to the businesses sales. Therefore, we incorporate a Common Table Expression as a mechanism to modularize logic a bit more, as well to simplify bringing back high-value customers into focus for our queries.

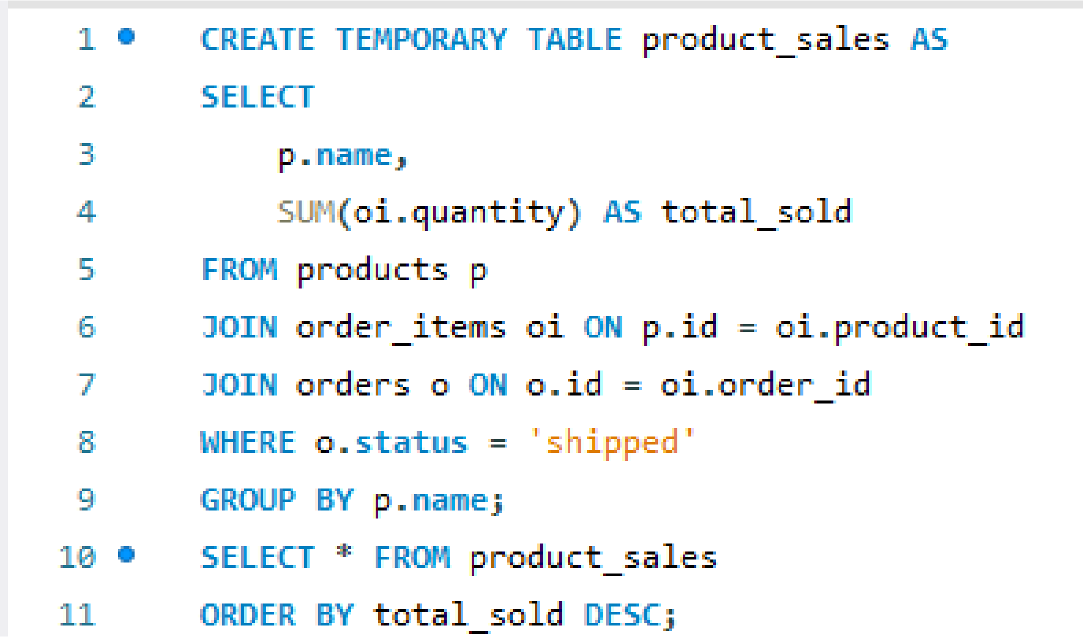
CASE Statement 3: Product Popularity

Introduction to Command:

- To identify the highest-selling products based on completed transactions, a **temporary** table called `product_sales` is created. It calculates the total number of units sold for each product by joining the `products`, `order_items`, and `orders` tables, while only including orders with a status of "shipped" to ensure the data reflects actual fulfilled sales.
- The total quantities are grouped by product name. Once the table is built, all product sales data is retrieved and sorted in

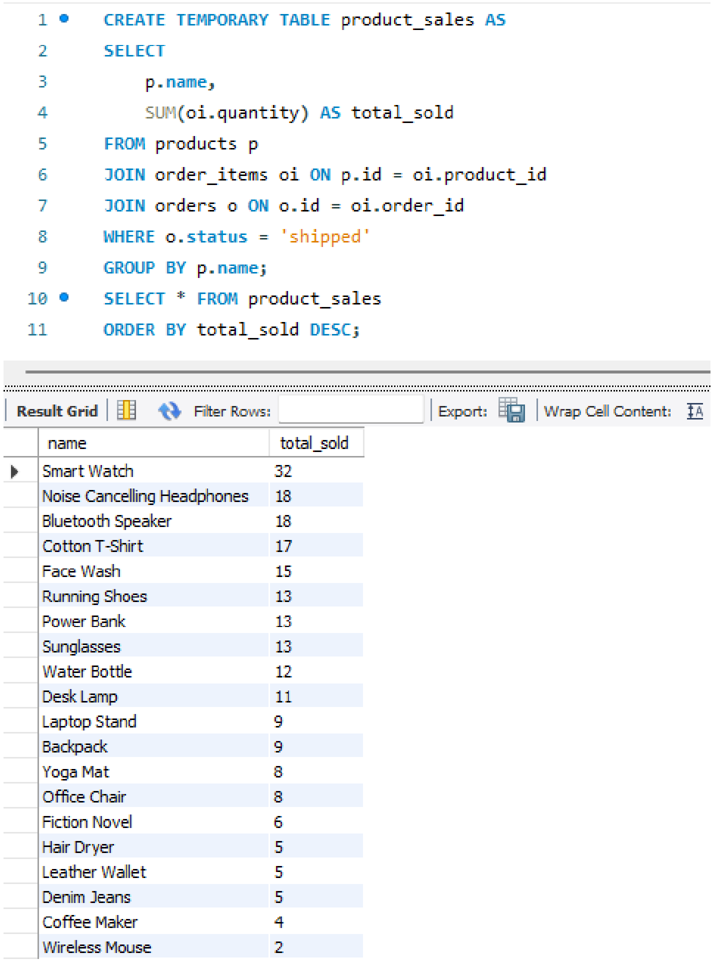
descending order of units sold, providing a clear overview of which products performed best in terms of shipped sales.

Command:



```
CREATE TEMPORARY TABLE product_sales AS
SELECT
    p.name,
    SUM(oi.quantity) AS total_sold
FROM products p
JOIN order_items oi ON p.id = oi.product_id
JOIN orders o ON o.id = oi.order_id
WHERE o.status = 'shipped'
GROUP BY p.name;
SELECT * FROM product_sales
ORDER BY total_sold DESC;
```

Command:



```
CREATE TEMPORARY TABLE product_sales AS
SELECT
    p.name,
    SUM(oi.quantity) AS total_sold
FROM products p
JOIN order_items oi ON p.id = oi.product_id
JOIN orders o ON o.id = oi.order_id
WHERE o.status = 'shipped'
GROUP BY p.name;
SELECT * FROM product_sales
ORDER BY total_sold DESC;
```

name	total_sold
Smart Watch	32
Noise Cancelling Headphones	18
Bluetooth Speaker	18
Cotton T-Shirt	17
Face Wash	15
Running Shoes	13
Power Bank	13
Sunglasses	13
Water Bottle	12
Desk Lamp	11
Laptop Stand	9
Backpack	9
Yoga Mat	8
Office Chair	8
Fiction Novel	6
Hair Dryer	5
Leather Wallet	5
Denim Jeans	5
Coffee Maker	4
Wireless Mouse	2

Logic and Reasoning:

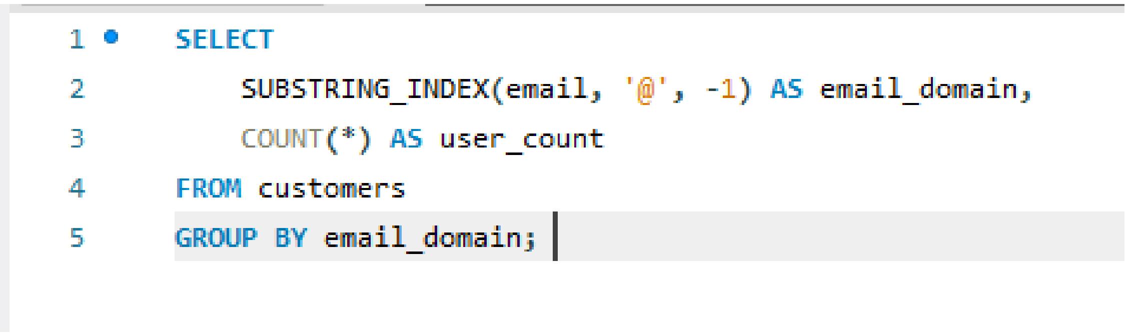
The rationale for the third query is to find the products with the most units sold, but only for orders that were shipped. The first step of this query is to join the **products, order_items,** and **orders** table, connecting the product information to the appropriate orders. The query only selects orders that are in a 'shipped' state to ensure we only take sales that were completed. It sums the total quantity for each product sold using **SUM(oi.quantity)** and groups the results by product name. The query then creates a temporary table, named **product_sales,** to store the results. Finally, it returns all records from the temporary table in descending order based on total units sold. This type of report would allow you to easily identify products that were successful based upon actual shipped sales.

CASE Study 4: String Functions – Email Domain Insights

Introduction to Command:

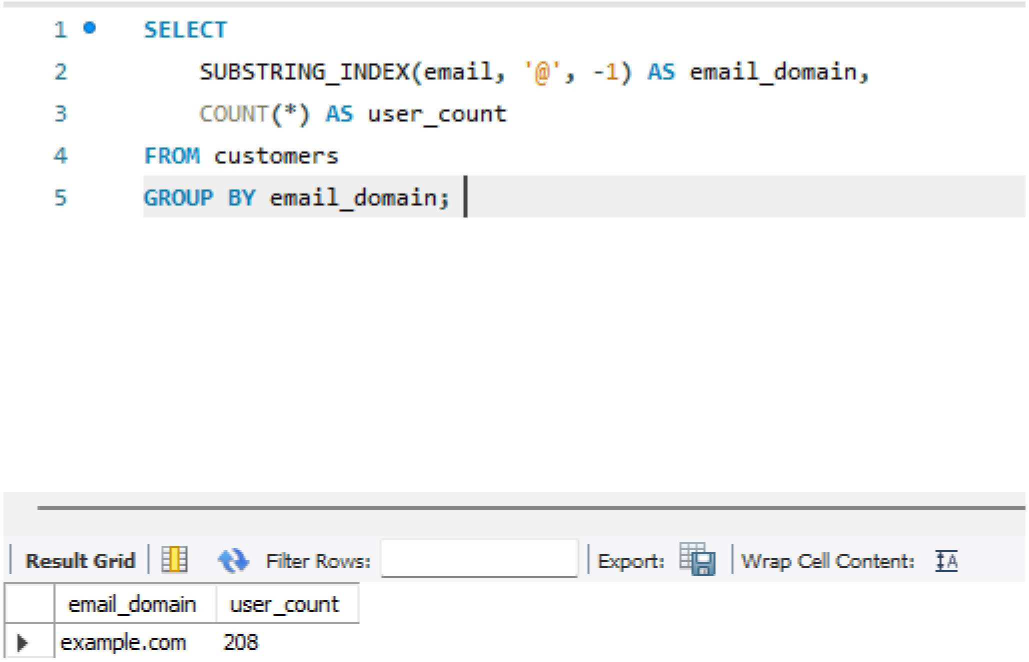
This query analyzes customer email addresses to determine how many users are associated with each email domain (like gmail.com, yahoo.com, etc.). It extracts the domain part of each email by using the **SUBSTRING_INDEX** function, which takes the portion of the email address after the @ symbol. The results are then grouped by these extracted domains, and a count is made of how many customers belong to each one. This provides a simple breakdown of customer distribution by email provider.

Command:



```
SELECT
    SUBSTRING_INDEX(email, '@', -1) AS email_domain,
    COUNT(*) AS user_count
FROM customers
GROUP BY email_domain;
```

Output:



```
SELECT
    SUBSTRING_INDEX(email, '@', -1) AS email_domain,
    COUNT(*) AS user_count
FROM customers
GROUP BY email_domain;
```

email_domain	user_count
example.com	208

Logic and Reasoning:

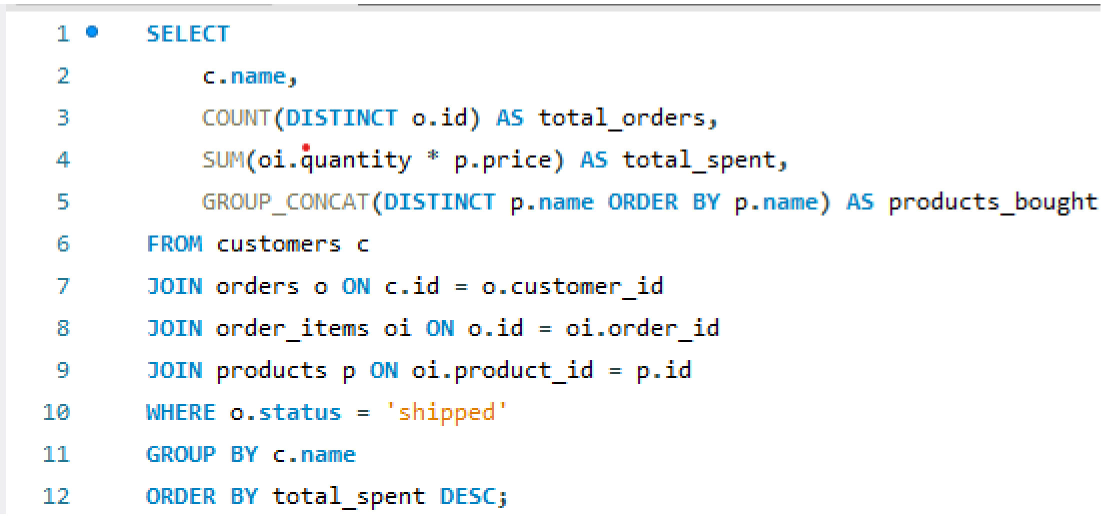
String parsing helps identify user base distribution across email providers; useful for targeted campaigns and customer profiling.

CASE Study 5: Order Summary Report per Customer

Introduction to Command:

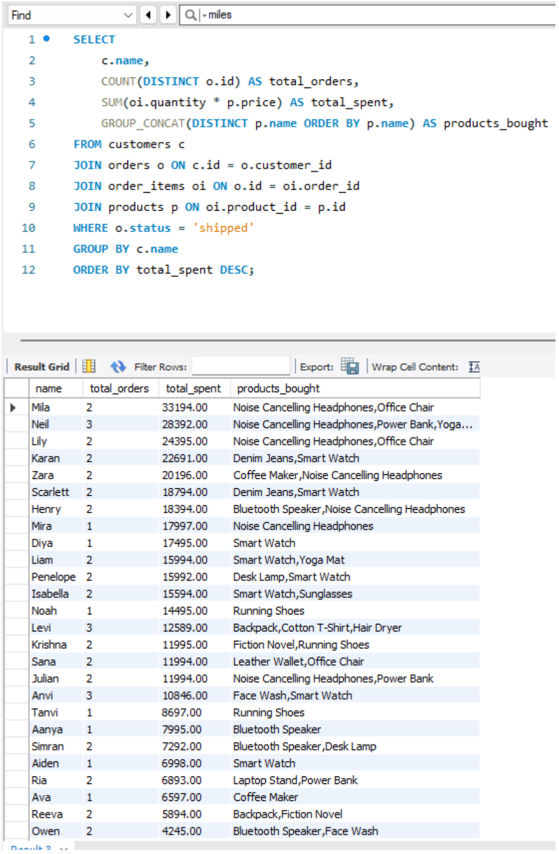
- To summarize customer purchasing behavior based on completed orders, the data combines customer details with their order and product information. For each customer, it calculates the number of distinct orders they've placed, the total amount they've spent (by multiplying product price with quantity), and a sorted list of unique products they’ve bought.
- The analysis only includes orders marked as "shipped" to ensure accuracy in spending and product data. The final results are grouped by customer name and sorted in descending order of total spending, highlighting the most valuable and active customers.

Code:



```
SELECT
    c.name,
    COUNT(DISTINCT o.id) AS total_orders,
    SUM(oi.quantity * p.price) AS total_spent,
    GROUP_CONCAT(DISTINCT p.name ORDER BY p.name) AS products_bought
FROM customers c
JOIN orders o ON c.id = o.customer_id
JOIN order_items oi ON o.id = oi.order_id
JOIN products p ON oi.product_id = p.id
WHERE o.status = 'shipped'
GROUP BY c.name
ORDER BY total_spent DESC;
```

Output:



```
SELECT
    c.name,
    COUNT(DISTINCT o.id) AS total_orders,
    SUM(oi.quantity * p.price) AS total_spent,
    GROUP_CONCAT(DISTINCT p.name ORDER BY p.name) AS products_bought
FROM customers c
JOIN orders o ON c.id = o.customer_id
JOIN order_items oi ON o.id = oi.order_id
JOIN products p ON oi.product_id = p.id
WHERE o.status = 'shipped'
GROUP BY c.name
ORDER BY total_spent DESC;
```

name	total_orders	total_spent	products_bought
Mila	2	33194.00	Noise Cancelling Headphones,Office Chair
Neil	3	28392.00	Noise Cancelling Headphones,Power Bank,Yoga...
Lily	2	24395.00	Noise Cancelling Headphones,Office Chair
Karan	2	22691.00	Denim Jeans,Smart Watch
Zara	2	20196.00	Coffee Maker,Noise Cancelling Headphones
Scarlett	2	18794.00	Denim Jeans,Smart Watch
Henry	2	18394.00	Bluetooth Speaker,Noise Cancelling Headphones
Mira	1	17997.00	Noise Cancelling Headphones
Diya	1	17495.00	Smart Watch
Liam	2	15994.00	Smart Watch,Yoga Mat
Penelope	2	15992.00	Desk Lamp,Smart Watch
Isabella	2	15594.00	Smart Watch,Sunglasses
Noah	1	14495.00	Running Shoes
Levi	3	12589.00	Backpack,Cotton T-Shirt,Hair Dryer
Krishna	2	11995.00	Fiction Novel,Running Shoes
Sana	2	11994.00	Leather Wallet,Office Chair
Julian	2	11994.00	Noise Cancelling Headphones,Power Bank
Anvi	3	10846.00	Face Wash,Smart Watch
Tanvi	1	8697.00	Running Shoes
Aanya	1	7995.00	Bluetooth Speaker
Simran	2	7292.00	Bluetooth Speaker,Desk Lamp
Aiden	1	6998.00	Smart Watch
Ria	2	6893.00	Laptop Stand,Power Bank
Ava	1	6597.00	Coffee Maker
Reeva	2	5894.00	Backpack,Fiction Novel
Owen	2	4245.00	Bluetooth Speaker,Face Wash

Logic and Reasoning:

Summarizes customer behavior and order trends, useful for loyalty programs, retention strategies, and performance dashboards.