

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

-Lohitaaksh Bethaa

---

# INDEX

S.no.	Topic	Page no.
1.	Introduction	3
2.	Creation of Tables	
3.	Inserting Values into tables	5
4.	Case Study 1	x
5.	Case Study 2	x
6.	Case Study 3	x
7.	Case Study 4	x
8.	Case Study 5	x

## 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:

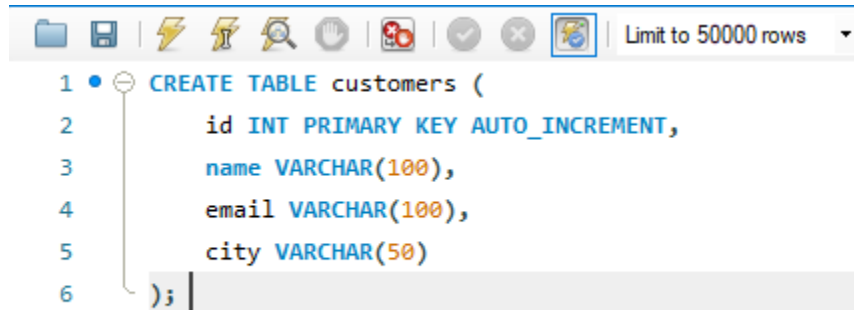
- Here we have used PRIMARY KEY and AUTO\_INCREMENT functions in SQL to not allow duplication of characters for customer ID in customer table, order ID in ORDERS and ORDERS

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.

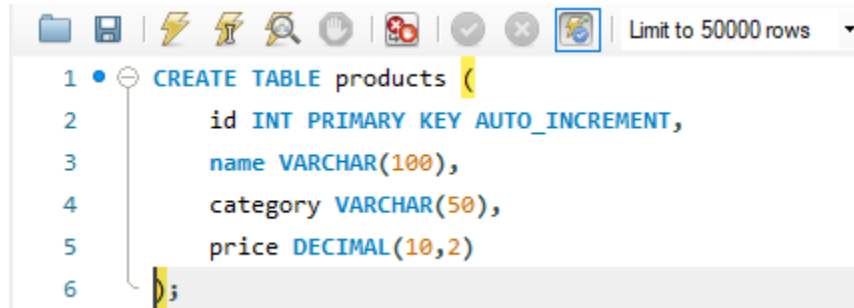
### Commands:

#### Customer Table:



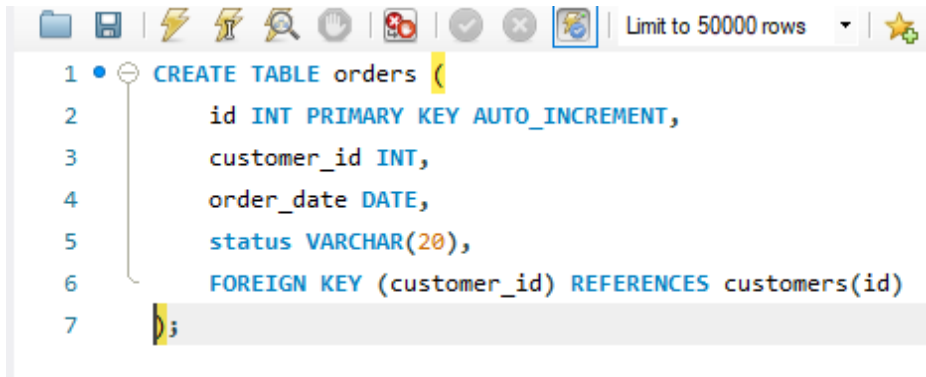
```
1 CREATE TABLE customers (  
2     id INT PRIMARY KEY AUTO_INCREMENT,  
3     name VARCHAR(100),  
4     email VARCHAR(100),  
5     city VARCHAR(50)  
6 );
```

#### Product Table:



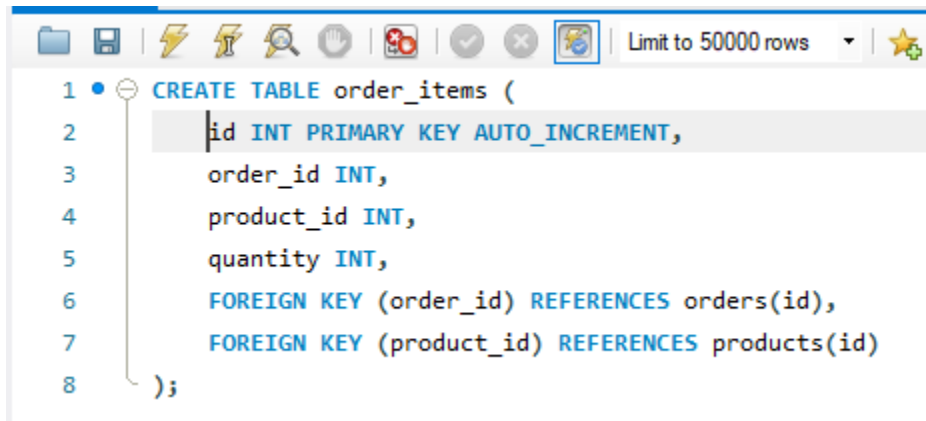
```
1 CREATE TABLE products (  
2     id INT PRIMARY KEY AUTO_INCREMENT,  
3     name VARCHAR(100),  
4     category VARCHAR(50),  
5     price DECIMAL(10,2)  
6 );
```

#### Order Table:



```
1 • CREATE TABLE orders (  
2     id INT PRIMARY KEY AUTO_INCREMENT,  
3     customer_id INT,  
4     order_date DATE,  
5     status VARCHAR(20),  
6     FOREIGN KEY (customer_id) REFERENCES customers(id)  
7 );
```

### Order Items Table:



```
1 • CREATE TABLE order_items (  
2     id INT PRIMARY KEY AUTO_INCREMENT,  
3     order_id INT,  
4     product_id INT,  
5     quantity INT,  
6     FOREIGN KEY (order_id) REFERENCES orders(id),  
7     FOREIGN KEY (product_id) REFERENCES products(id)  
8 );
```

### 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:

```
1 • INSERT INTO customers (name, email, city)
2 VALUES('Anvi', 'anvi@example.com', 'Rajahmundry'),
3 ('Mason', 'mason@example.com', 'Boston'),
4 ('Reeva', 'reeva@example.com', 'Kurnool'),
5 ('Harper', 'harper@example.com', 'Houston'),
6 ('Vihaan', 'vihaan@example.com', 'Guntur'),
7 ('Ella', 'ella.james@example.com', 'San Francisco'),
8 ('Zara', 'zara@example.com', 'Asansol'),
9 ('Benjamin', 'benjamin@example.com', 'Phoenix'),
10 ('Advait', 'advait@example.com', 'Kakinada'),
11 ('Luna', 'luna@example.com', 'Naples'),
12 ('Kavya', 'kavya@example.com', 'Madurai'),
13 ('Sebastian', 'sebastian@example.com', 'Leeds'),
14 ('Tara', 'tara@example.com', 'Kharagpur'),
15 ('Layla', 'layla@example.com', 'Rotterdam'),
16 ('Pranav', 'pranav@example.com', 'Bareilly'),
17 ('Owen', 'owen@example.com', 'Glasgow'),
18 ('Myra', 'myra@example.com', 'Dhanbad'),
19 ('Penelope', 'penelope@example.com', 'Belfast'),
20 ('Shaurya', 'shaurya@example.com', 'Bhagalpur'),
21 ('Isla', 'isla@example.com', 'Valencia'),
22 ('Aadhya', 'aadhya@example.com', 'Bilaspur'),
23 ('Ezra', 'ezra@example.com', 'Antwerp'),
24 ('Trisha', 'trisha@example.com', 'Nanded'),
25 ('Theo', 'theo@example.com', 'Stockholm'),
26 ('Divya', 'divya@example.com', 'Moradabad'),
27 ('Levi', 'levi@example.com', 'Tallinn'),
28 ('Anushka', 'anushka@example.com', 'Kolhapur'),
29 ('Mila', 'mila@example.com', 'Nice'),
30 ('Darsh', 'darsh@example.com', 'Aligarh'),
31 ('Henry', 'henry@example.com', 'Lille'),
32 ('Aarav', 'aarav@example.com', 'Mumbai'),
33 ('Emily', 'emily@example.com', 'New York'),
34 ('Vivaan', 'vivaan@example.com', 'Ahmedabad'),
35 ('Liam', 'liam@example.com', 'London'),
36 ('Ananya', 'ananya@example.com', 'Chennai'),
37 ('Sophia', 'sophia@example.com', 'Los Angeles'),
38 ('Rohan', 'rohan@example.com', 'Pune'),
39 ('Jackson', 'jackson@example.com', 'Chicago'),
40 ('Diya', 'diya@example.com', 'Hyderabad'),
41 ('Olivia', 'olivia@example.com', 'Boston'),
42 ('Krishna', 'krishna@example.com', 'Vadodara'),
43 ('Ethan', 'ethan@example.com', 'Dallas'),
44 ('Tanya', 'tanya@example.com', 'Amritsar'),
45 ('Isabella', 'isabella@example.com', 'San Diego'),
46 ('Yash', 'yash@example.com', 'Nagpur'),
47 ('Aiden', 'aiden@example.com', 'Toronto'),
48 ('Sneha', 'sneha@example.com', 'Thane'),
49 ('Grace', 'grace@example.com', 'Vancouver'),
50 ('Kabin', 'kabin@example.com', 'Kolkata'),
51 ('Chloe', 'chloe@example.com', 'Seattle'),
52 ('Meera', 'meera@example.com', 'Jaipur'),
53 ('Noah', 'noah@example.com', 'Berlin'),
54 ('Nikhil', 'nikhil@example.com', 'Bhopal'),
55 ('Ava', 'ava@example.com', 'Paris'),
56 ('Dev', 'dev@example.com', 'Patna'),
57 ('Lucas', 'lucas@example.com', 'Amsterdam'),
58 ('Sana', 'sana@example.com', 'Surat'),
59 ('Mia', 'mia@example.com', 'Brisbane'),
60 ('Aryan', 'aryan@example.com', 'Vijayawada'),
61 ('Ella', 'ella@example.com', 'Zurich'),
62 ('Karan', 'karan@example.com', 'Gurgaon'),
63 ('Lily', 'lily@example.com', 'Auckland'),
64 ('Harsh', 'harsh@example.com', 'Chandigarh'),
65 ('Zoe', 'zoe@example.com', 'Melbourne'),
66 ('Ria', 'ria@example.com', 'Mysore'),
67 ('Logan', 'logan@example.com', 'Oslo'),
68 ('Avni', 'avni@example.com', 'Nashik'),
69 ('Nathan', 'nathan@example.com', 'Vienna'),
70 ('Aanya', 'aanya@example.com', 'Guwahati'),
71 ('Ruby', 'ruby@example.com', 'Helsinki'),
72 ('Aditya', 'aditya@example.com', 'Bhubaneswar'),
73 ('Leo', 'leo@example.com', 'Geneva'),
74 ('Mira', 'mira@example.com', 'Agra'),
75 ('Scarlett', 'scarlett@example.com', 'Prague'),
76 ('Raj', 'raj@example.com', 'Kanpur'),
77 ('Dylan', 'dylan@example.com', 'Manchester'),
78 ('Neha', 'neha@example.com', 'Dehradun'),
79 ('Charlotte', 'charlotte@example.com', 'Lisbon'),
80 ('Arjun', 'arjun@example.com', 'Varanasi'),
```

## Output:

The screenshot shows the MySQL Workbench interface. The top toolbar includes icons for file operations, editing, and database management. The left sidebar shows the 'Schemas' tree with 'ecommerce\_project' selected. The main window displays the SQL editor with a query that inserts data into the 'customers' table. Below the editor, the 'Result Grid' shows the output of the query, which is a list of customer records. The bottom pane shows the 'Action Output' tab, which displays the execution log of the queries, including the creation of the database, tables, and the insertion of data.

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	Hason	hason@example.com	Boston
3	Reeva	reeva@example.com	Kurnool
4	Harper	harper@example.com	Houston
5	Vivian	vivian@example.com	Gurur
6	Ella	ella.james@example.com	San Francisco
7	Zara	zara@example.com	Kurnool
8	Benjamin	benjamin@example.com	Thomson
9	Advent	advent@example.com	Kabirabad
10	Luna	luna@example.com	Hoplet
11	Kavya	kavya@example.com	Nidurua

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: 1064. No database selected. Select the default DB to be used by double-clicking to 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, customer_id INT, 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 KEY (customer_id) REFERENCES customers (id))	0 row(s) affected	0.031 sec
7 15:19:09	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))	0 row(s) affected	0.031 sec
8 15:20:36	INSERT INTO customers (name, email, city) VALUES ('Anvi', 'anvi@example.com', 'Rajahmundry'), ('Hason', 'hason@example.com', 'Boston'), ('Reeva', 'reeva@example.com', 'Kurnool')	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'), ('Hason', 'hason@example.com', 'Boston'), ('Reeva', 'reeva@example.com', 'Kurnool')	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:

The screenshot shows the MySQL Workbench interface with the SQL editor. The query is as follows:

```
1 • INSERT INTO products (name, category, price)
2 VALUES ('Wireless Mouse', 'Electronics', 749.00),
3 ('Running Shoes', 'Footwear', 2899.00),
4 ('Bluetooth Speaker', 'Electronics', 1599.00),
5 ('Cotton T-Shirt', 'Clothing', 599.00),
6 ('Yoga Mat', 'Fitness', 999.00),
7 ('Smart Watch', 'Wearables', 3499.00),
8 ('Coffee Maker', 'Appliances', 2199.00),
9 ('Laptop Stand', 'Accessories', 899.00),
10 ('Leather Wallet', 'Accessories', 699.00),
11 ('Desk Lamp', 'Home Decor', 499.00),
12 ('Water Bottle', 'Fitness', 299.00),
13 ('Denim Jeans', 'Clothing', 1299.00),
14 ('Backpack', 'Bags', 1099.00),
15 ('Hair Dryer', 'Personal Care', 1499.00),
16 ('Sunglasses', 'Fashion', 799.00),
17 ('Office Chair', 'Furniture', 4599.00),
18 ('Face Wash', 'Skincare', 349.00),
19 ('Power Bank', 'Electronics', 1199.00),
20 ('Fiction Novel', 'Books', 399.00),
21 ('Noise Cancelling Headphones', 'Electronics', 5999.00);
22 • SELECT * FROM products;
```



# Output:

**MySQL Workbench**

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigation: Schemas, Filter objects, Filter results

**Schemas**

- ecommerce\_project
  - Tables
  - Views
  - Stored Procedures
  - Functions
  - Sub
  - Schema
  - Schema Data
  - sys
  - Information
  - world

**SQL Editor**

```
1 INSERT INTO products (name, category, price)
2 VALUES('Wireless Mouse', 'Electronics', 749.00),
3 ('Running Shoes', 'Footwear', 2899.00),
4 ('Bluetooth Speaker', 'Electronics', 1599.00),
5 ('Cotton T-shirt', 'Clothing', 599.00),
6 ('Yoga Mat', 'Fitness', 999.00),
7 ('Smart Watch', 'Wearables', 3499.00),
8 ('Coffee Maker', 'Appliances', 2199.00),
9 ('Laptop Stand', 'Accessories', 899.00),
10 ('Leather Wallet', 'Accessories', 499.00),
11 ('Desk Lamp', 'Home Decor', 499.00),
12 ('Water Bottle', 'Fitness', 299.00),
13 ('Denim Jeans', 'Clothing', 1299.00),
14 ('Backpack', 'Bags', 1899.00),
15 ('Hair Dryer', 'Personal Care', 1499.00),
16 ('Sunglasses', 'Fashion', 799.00),
17 ('Office Chair', 'Furniture', 4599.00),
18 ('Face Mask', 'Skincare', 349.00),
19 ('Power Bank', 'Electronics', 1199.00),
20 ('Smart Home Hub', 'Smart Home', 1999.00);
```

**Results Grid**

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	499.00
10	Desk Lamp	Home Decor	499.00
11	Water Bottle	Fitness	299.00
12	Denim Jeans	Clothing	1299.00
13	Backpack	Bags	1899.00

**Output**

Time	Action	Message	Duration / Fetch
10:15:22.19	SELECT * FROM customers LIMIT 0, 50000	208 rows 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 rows(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 rows(s) affected Records: 20 Duplicates: 0 Warnings: 0	0.000 sec
13:15:27.37	SELECT * FROM products LIMIT 0, 50000	40 rows(s) returned	0.000 sec / 0.000 sec

## Inserting into Order table: (134 orders):

```
1 • INSERT INTO orders (customer_id, order_date, status)
2 VALUES(38, '2025-03-02', 'pending'),
3 (26, '2024-07-12', 'shipped'),
4 (73, '2024-08-04', 'cancelled'),
5 (34, '2025-02-24', 'shipped'),
6 (17, '2024-09-20', 'pending'),
7 (21, '2025-04-29', 'cancelled'),
8 (17, '2025-04-06', 'pending'),
9 (4, '2025-01-02', 'cancelled'),
10 (62, '2025-01-16', 'shipped'),
11 (5, '2024-07-20', 'pending'),
12 (65, '2025-03-09', 'shipped'),
13 (96, '2024-08-02', 'cancelled'),
14 (15, '2024-09-14', 'pending'),
15 (35, '2025-06-01', 'cancelled'),
16 (41, '2024-09-06', 'shipped'),
17 (10, '2024-10-28', 'pending'),
18 (50, '2025-03-27', 'pending'),
19 (23, '2025-01-19', 'cancelled'),
20 (30, '2024-08-18', 'shipped'),
21 (29, '2024-07-18', 'shipped'),
22 (94, '2025-01-21', 'cancelled'),
23 (20, '2025-03-26', 'cancelled'),
24 (83, '2024-08-23', 'pending'),
25 (18, '2025-04-11', 'shipped'),
26 (45, '2024-10-25', 'pending'),
27 (36, '2025-04-17', 'pending'),
28 (85, '2025-01-17', 'shipped'),
29 (43, '2024-07-02', 'cancelled'),
30 (8, '2025-05-11', 'pending'),
31 (71, '2025-05-24', 'cancelled'),
32 (44, '2025-03-08', 'shipped'),
33 (6, '2024-12-01', 'pending'),
34 (49, '2024-12-09', 'cancelled'),
35 (74, '2025-03-14', 'shipped'),
36 (75, '2024-08-12', 'pending'),
37 (70, '2024-06-22', 'shipped'),
38 (40, '2024-12-13', 'cancelled'),
40 (3, '2024-12-24', 'shipped'),
41 (16, '2025-05-27', 'shipped'),
42 (98, '2025-03-21', 'shipped'),
43 (42, '2024-10-30', 'pending'),
44 (25, '2025-02-03', 'cancelled'),
45 (27, '2025-01-09', 'pending'),
46 (22, '2024-11-13', 'cancelled'),
47 (72, '2024-08-26', 'pending'),
48 (61, '2024-09-26', 'shipped'),
49 (24, '2025-02-22', 'cancelled'),
50 (63, '2024-09-11', 'pending'),
51 (100, '2024-11-26', 'cancelled'),
52 (84, '2024-11-07', 'pending'),
53 (80, '2024-12-31', 'pending'),
54 (95, '2024-12-04', 'shipped'),
55 (28, '2025-01-31', 'shipped'),
56 (86, '2025-01-24', 'shipped'),
57 (78, '2025-02-13', 'cancelled'),
58 (99, '2024-11-01', 'shipped'),
59 (1, '2024-12-16', 'shipped'),
60 (87, '2025-03-17', 'pending'),
61 (91, '2024-08-28', 'cancelled'),
62 (47, '2024-10-10', 'cancelled'),
63 (59, '2025-04-01', 'pending'),
64 (7, '2025-05-22', 'shipped'),
65 (2, '2024-06-26', 'pending'),
66 (9, '2025-06-06', 'shipped'),
67 (11, '2025-03-05', 'cancelled'),
68 (12, '2025-02-08', 'shipped'),
69 (13, '2024-10-12', 'shipped'),
70 (14, '2025-05-03', 'cancelled'),
71 (19, '2025-03-31', 'shipped'),
72 (32, '2025-02-02', 'shipped'),
73 (33, '2025-01-03', 'cancelled'),
74 (37, '2024-12-08', 'pending'),
75 (39, '2024-10-19', 'pending'),
76 (46, '2024-07-22', 'shipped'),
77 (48, '2025-06-05', 'cancelled'),
78 (51, '2025-02-27', 'pending'),
```

78	(51, '2025-02-27', 'pending'),	
79	(52, '2025-05-10', 'shipped'),	
80	(53, '2024-09-01', 'cancelled'),	
81	(54, '2025-01-07', 'shipped'),	
82	(55, '2025-04-08', 'pending'),	
83	(56, '2025-03-30', 'cancelled'),	
84	(57, '2024-11-22', 'shipped'),	
85	(58, '2024-12-23', 'shipped'),	
86	(60, '2024-11-10', 'pending'),	
87	(64, '2024-07-27', 'cancelled'),	
88	(66, '2025-06-09', 'pending'),	
89	(67, '2025-01-29', 'shipped'),	
90	(68, '2025-03-18', 'cancelled'),	118
91	(69, '2025-01-10', 'pending'),	119
92	(76, '2024-09-28', 'shipped'),	120
93	(77, '2024-10-01', 'cancelled'),	121
94	(79, '2025-02-14', 'pending'),	122
95	(81, '2025-04-04', 'shipped'),	123
96	(82, '2024-12-19', 'shipped'),	124
97	(88, '2025-04-20', 'cancelled'),	125
98	(89, '2024-11-18', 'shipped'),	126
99	(90, '2024-08-06', 'pending'),	127
100	(92, '2025-06-02', 'shipped'),	128
101	(93, '2025-05-14', 'pending'),	129
102	(97, '2025-03-06', 'shipped'),	130
103	(19, '2025-01-20', 'pending'),	131
104	(34, '2024-09-03', 'cancelled'),	132
105	(73, '2025-05-18', 'shipped'),	133
106	(38, '2025-02-01', 'pending'),	134
107	(25, '2024-10-09', 'shipped'),	135
108	(7, '2024-09-09', 'cancelled'),	136
109	(50, '2025-06-03', 'pending'),	
110	(10, '2025-02-06', 'shipped'),	
111	(15, '2025-01-25', 'shipped'),	
112	(80, '2025-05-29', 'cancelled'),	
113	(100, '2024-07-16', 'shipped'),	
114	(36, '2024-10-22', 'shipped'),	
115	(42, '2024-08-08', 'cancelled'),	
116	(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:

The screenshot displays the MySQL Workbench interface. The top toolbar includes icons for File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. The left sidebar shows the 'SCHEMAS' tab with a tree view of the 'ecommerce\_project' database, including tables, views, stored procedures, functions, and triggers. The main editor window contains a SQL query: 

```
1 INSERT INTO orders (customer_id, order_date, status)
2 VALUES(10, '2025-03-02', 'pending'),
3 (26, '2024-07-12', 'shipped'),
4 (73, '2024-08-04', 'cancelled'),
5 (14, '2025-02-24', 'shipped'),
6 (17, '2024-09-20', 'pending'),
7 (21, '2025-04-29', 'cancelled'),
8 (17, '2025-04-06', 'pending'),
9 (4, '2025-01-02', 'cancelled'),
10 (62, '2025-01-16', 'shipped'),
11 (5, '2024-07-20', 'pending'),
12 (65, '2025-03-09', 'shipped'),
13 (96, '2024-08-02', 'cancelled'),
14 (15, '2024-09-14', 'pending'),
15 (15, '2024-06-01', 'cancelled'),
16 (61, '2024-09-08', 'shipped'),
17 (10, '2024-10-28', 'pending'),
18 (10, '2025-03-27', 'pending'),
19 (23, '2025-01-19', 'cancelled'),
20 FROM products LIMIT 0.50000;
```

 The 'Result Grid' tab shows the results of the query, displaying columns 'id', 'customer\_id', 'order\_date', and 'status'. The 'Output' tab at the bottom shows the execution log, including the time taken for each statement and the number of rows affected. The log indicates that the first statement (INSERT) took 0.000 sec and affected 133 rows. The second statement (SELECT) took 0.016 sec and affected 133 rows. The third statement (SELECT) took 0.000 sec and affected 266 rows.

id	customer_id	order_date	status
1	10	2025-03-02	pending
2	26	2024-07-12	shipped
3	73	2024-08-04	cancelled
4	14	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
14	15	2024-06-01	cancelled
15	61	2024-09-08	shipped
16	10	2024-10-28	pending
17	10	2025-03-27	pending
18	23	2025-01-19	cancelled

Time	Action	Message	Duration / Pct
13 15:27:37	SELECT * FROM products LIMIT 0.50000	40 rows returned	0.000 sec / 0.000 sec
14 15:36:28	INSERT INTO orders (customer_id, order_date, status) VALUES(10, '2025-03-02', 'pending'), (26, '2024-07-12', 'shipped'), (73, '2024-08-04', 'cancelled'), (14, '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'), (15, '2024-06-01', 'cancelled'), (61, '2024-09-08', 'shipped'), (10, '2024-10-28', 'pending'), (10, '2025-03-27', 'pending'), (23, '2025-01-19', 'cancelled') FROM products LIMIT 0.50000	133 rows affected Records: 133 Duplicates: 0 Warnings: 0	0.016 sec
15 15:36:49	SELECT * FROM orders LIMIT 0.50000	266 rows returned	0.000 sec / 0.000 sec

## Inserting into Order Items table:

```
1 • INSERT INTO order_items (order_id, product_id, quantity)
2 VALUES(1, 14, 4),
3 (2, 14, 5),
4 (3, 19, 2),
5 (4, 5, 2),
6 (5, 20, 4),
7 (6, 4, 4),
8 (7, 6, 4),
9 (8, 13, 3),
10 (9, 16, 4),
11 (10, 15, 1),
12 (11, 8, 5),
13 (12, 10, 3),
14 (13, 14, 4),
15 (14, 12, 4),
16 (15, 2, 4),
17 (16, 10, 1),
18 (17, 18, 3),
19 (18, 15, 3),
20 (19, 20, 2),
21 (20, 9, 1),
22 (21, 6, 3),
23 (22, 5, 1),
24 (23, 7, 4),
25 (24, 6, 4),
26 (25, 1, 1),
27 (26, 9, 1),
28 (27, 10, 1),
29 (28, 1, 3),
30 (29, 10, 4),
31 (30, 17, 2),
32 (31, 6, 4),
33 (32, 14, 4),
34 (33, 20, 4),
35 (34, 6, 5),
36 (35, 7, 4),
37 (36, 4, 4),
38 (37, 8, 4),
39 (38, 2, 2).
```

## Output:

MySQL Workbench

Local instance 19525230

File Edit View Query Database Server Tools Scripting Help

Navigation

Schemas

Filter objects

ecommerce\_project

Tables

Views

Stored Procedures

Functions

Sub

Schema

Schema Data

sys

Database

world

154 (183, 15, 2)

155 (184, 13, 1)

156 (185, 11, 1)

157 (186, 18, 2)

158 (187, 16, 2)

159 (188, 17, 2)

160 (189, 16, 1)

161 (190, 18, 3)

162 (191, 6, 2)

163 (192, 15, 4)

164 (193, 8, 1)

165 (194, 3, 1)

166 (195, 10, 1)

167 (196, 7, 1)

168 (197, 9, 2)

169 (198, 13, 1)

170 (199, 5, 4)

171 (200, 15, 1)

SELECT \* FROM order\_items

Result Grid

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 15 4

10 10 15 1

11 11 8 5

12 12 10 3

13 13 14 4

order\_items 4 x

Output

Action Output

Time Action Message

16 19:36:49 SELECT \* FROM order\_items LIMIT 0, 50000 266 rows returned 0.000 sec / 0.000 sec

17 19:56:21 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, 15, 4), (10, 15, 1), (11, 8, 5), (12, 10, 3), (13, 14, 4) 200 rows affected Records: 200 Duplicates: 0 Warnings: 0 0.015 sec

18 19:56:45 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, 15, 4), (10, 15, 1), (11, 8, 5), (12, 10, 3), (13, 14, 4) 200 rows affected Records: 200 Duplicates: 0 Warnings: 0 0.000 sec

19 19:56:45 SELECT \* FROM order\_items LIMIT 0, 50000 400 rows 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

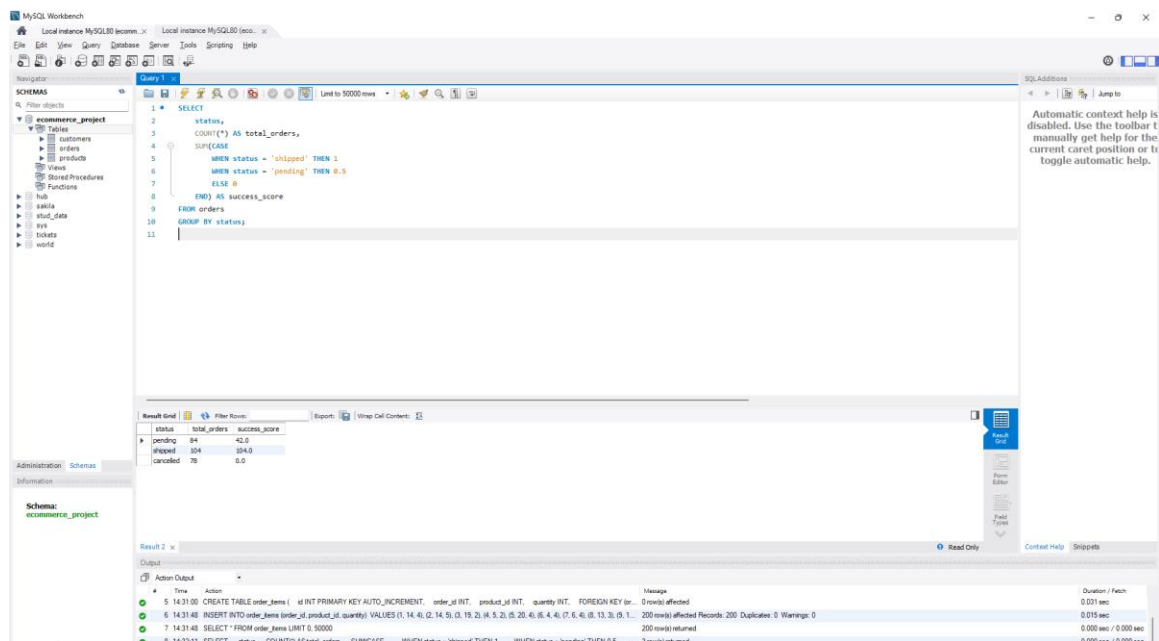
## Introduction to Command:

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

## Code:

```
1  ●  SELECT
2      status,
3      COUNT(*) AS total_orders,
4      SUM(CASE
5          WHEN status = 'shipped' THEN 1
6          WHEN status = 'pending' THEN 0.5
7          ELSE 0
8          END) AS success_score
9  FROM orders GROUP BY status;
```

## Output:



The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows the 'e-commerce\_project' database selected. The central editor contains a SQL query that calculates a success score for orders based on their status. The query uses a CASE statement to assign values: 1 for 'shipped', 0.5 for 'pending', and 0 for all other statuses. These values are summed and grouped by status to produce the success score. The bottom pane shows the results of the query, which are displayed in a table format. The table has three columns: 'status', 'total\_orders', and 'success\_score'. The results show that there are 84 shipped orders with a success score of 42.0, 104 pending orders with a success score of 52.0, and 70 cancelled orders with a success score of 0.0. The bottom pane also shows the execution log, indicating that the query was executed successfully and returned 3 rows.

```
1 SELECT
2   status,
3   COUNT(*) AS total_orders,
4   SUM(CASE
5     WHEN status = 'shipped' THEN 1
6     WHEN status = 'pending' THEN 0.5
7     ELSE 0
8   END) AS success_score
9 FROM orders
10 GROUP BY status;
```

status	total_orders	success_score
shipped	84	42.0
pending	104	52.0
cancelled	70	0.0

## 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.



## 4. 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:

```
1 • WITH customer_spending AS (  
2     SELECT  
3         c.name,  
4         SUM(oi.quantity * p.price) AS total_spent  
5     FROM customers c  
6     JOIN orders o ON c.id = o.customer_id  
7     JOIN order_items oi ON o.id = oi.order_id  
8     JOIN products p ON oi.product_id = p.id  
9     WHERE o.status = 'shipped'  
10    GROUP BY c.name  
11 )  
12 SELECT * FROM customer_spending  
13 ORDER BY total_spent DESC  
14 LIMIT 5;
```

## Output:

The screenshot shows a database IDE with a schema on the left and a SQL editor in the center. The SQL query is the same as the one in the 'Command' section. Below the editor, the 'Result Grid' shows the output of the query. The output is a table with three columns: name, total\_spent, and a third column that is empty. The data rows are as follows:

name	total_spent	
John	22195.00	
John	22195.00	
John	22195.00	
John	22195.00	
John	22195.00	

Below the result grid, the 'Output' pane shows the execution details of the query, including the SQL text and the execution time.

## 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.

## 5. 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:

```
1 • CREATE TEMPORARY TABLE product_sales AS
2   SELECT
3       p.name,
4       SUM(oi.quantity) AS total_sold
5   FROM products p
6   JOIN order_items oi ON p.id = oi.product_id
7   JOIN orders o ON o.id = oi.order_id
8   WHERE o.status = 'shipped'
9   GROUP BY p.name;
10 • SELECT * FROM product_sales
11   ORDER BY total_sold DESC;
```

## Command:

```
1 • CREATE TEMPORARY TABLE product_sales AS
2   SELECT
3     p.name,
4     SUM(oi.quantity) AS total_sold
5   FROM products p
6   JOIN order_items oi ON p.id = oi.product_id
7   JOIN orders o ON o.id = oi.order_id
8   WHERE o.status = 'shipped'
9   GROUP BY p.name;
10 • SELECT * FROM product_sales
11   ORDER BY total_sold DESC;
```

Result Grid			Filter Rows:	Export:	Wrap Cell Content:
	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.

## 6. 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:

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

## Output:

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

Result Grid			Filter Rows:	Export:	Wrap Cell Content:
	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.

## **7. 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:

```
1 • SELECT
2     c.name,
3     COUNT(DISTINCT o.id) AS total_orders,
4     SUM(oi.quantity * p.price) AS total_spent,
5     GROUP_CONCAT(DISTINCT p.name ORDER BY p.name) AS products_bought
6 FROM customers c
7 JOIN orders o ON c.id = o.customer_id
8 JOIN order_items oi ON o.id = oi.order_id
9 JOIN products p ON oi.product_id = p.id
10 WHERE o.status = 'shipped'
11 GROUP BY c.name
12 ORDER BY total_spent DESC;
```

## Output:

Find

```
1 • SELECT
2     c.name,
3     COUNT(DISTINCT o.id) AS total_orders,
4     SUM(oi.quantity * p.price) AS total_spent,
5     GROUP_CONCAT(DISTINCT p.name ORDER BY p.name) AS products_bought
6 FROM customers c
7 JOIN orders o ON c.id = o.customer_id
8 JOIN order_items oi ON o.id = oi.order_id
9 JOIN products p ON oi.product_id = p.id
10 WHERE o.status = 'shipped'
11 GROUP BY c.name
12 ORDER BY total_spent DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Contents: |

	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.

## **Analysis**