**Report: E-Commerce Database SQL Queries**

**1. Customers Table**

CREATE TABLE customers (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL

);

INSERT INTO customers (name, email) VALUES

('Alice', 'alice@example.com'),

('Bob', 'bob@example.com'),

('Charlie', 'charlie@example.com');

Purpose: Stores customer details. Each customer has a unique ID and email.

2**. Products Table**

CREATE TABLE products (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

price DECIMAL(10,2) NOT NULL,

discount DECIMAL(5,2) DEFAULT 0.00

);

INSERT INTO products (name, price) VALUES

('Product A', 50.00),

('Product B', 30.00),

('Product C', 40.00),

('Product D', 60.00);

Purpose: Stores product details with price and discount.

**3. Orders Table**

CREATE TABLE orders (

id INT AUTO\_INCREMENT PRIMARY KEY,

customer\_id INT,

order\_date DATE NOT NULL,

total\_amount DECIMAL(10,2) NOT NULL,

FOREIGN KEY (customer\_id) REFERENCES customers(id)

);

INSERT INTO orders (customer\_id, order\_date, total\_amount) VALUES

(1, '2025-08-01', 80.00),

(2, '2025-08-10', 150.00),

(1, '2025-08-20', 200.00);

Purpose: Stores orders placed by customers. Each order is linked to one customer.

**4. Order Items Table (Normalized)**

CREATE TABLE order\_items (

id INT AUTO\_INCREMENT PRIMARY KEY,

order\_id INT,

product\_id INT,

quantity INT NOT NULL,

FOREIGN KEY (order\_id) REFERENCES orders(id),

FOREIGN KEY (product\_id) REFERENCES products(id)

);

INSERT INTO order\_items (order\_id, product\_id, quantity) VALUES

(1, 1, 1),

(1, 2, 1),

(2, 3, 2),

(3, 1, 1),

(3, 4, 2);

Purpose: Stores each product included in an order (with quantity).

**1. Retrieve all customers who have placed an order in the last 30 days.**

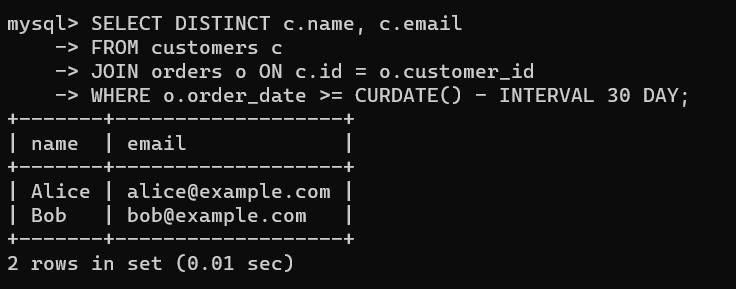
SELECT DISTINCT c.name, c.email

FROM customers c

JOIN orders o ON c.id = o.customer\_id

WHERE o.order\_date >= CURDATE() - INTERVAL 30 DAY;

**Explanation**: Finds unique customers who have at least one order in the last 30 days.



**2. Get the total amount of all orders placed by each customer.**

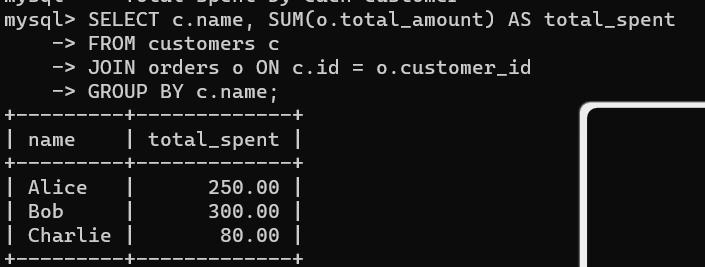
SELECT c.name, SUM(o.total\_amount) AS total\_spent

FROM customers c

JOIN orders o ON c.id = o.customer\_id

GROUP BY c.name;

**Explanation**: Groups orders by customer and calculates the total spent.



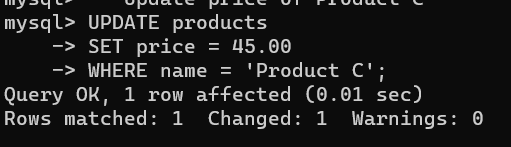
**3. Update the price of Product C to 45.00.**

UPDATE products

SET price = 45.00

WHERE name = 'Product C';

**Explanation**: Updates the price of the product named "Product C".

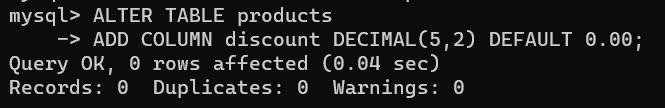


**4. Add a new column discount to the products table.**

ALTER TABLE products

ADD COLUMN discount DECIMAL(5,2) DEFAULT 0.00;

**Explanation**: Adds a new column for discount with default value 0.00.



**5. Retrieve the top 3 products with the highest price.**

SELECT name, price

FROM products

ORDER BY price DESC

LIMIT 3;

**Explanation**: Orders products by price (descending) and shows only the top 3.



**6. Get the names of customers who have ordered Product A.**

SELECT DISTINCT c.name

FROM customers c

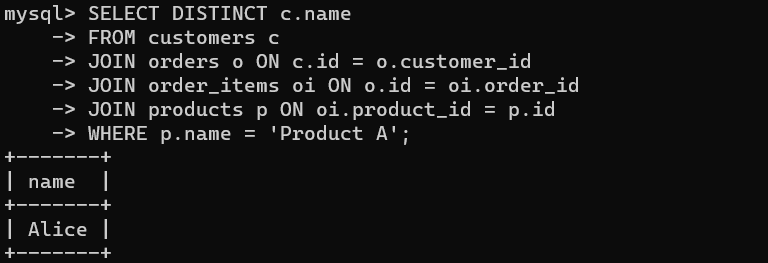
JOIN orders o ON c.id = o.customer\_id

JOIN order\_items oi ON o.id = oi.order\_id

JOIN products p ON oi.product\_id = p.id

WHERE p.name = 'Product A';

**Explanation**: Joins customers → orders → order\_items → products to find who ordered "Product A".



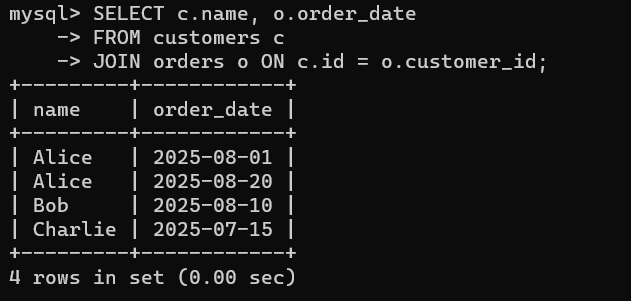
**7. Join the orders and customers tables to retrieve the customer's name and order date for each order.**

SELECT c.name, o.order\_date

FROM customers c

JOIN orders o ON c.id = o.customer\_id;

**Explanation**: Simple join to see which customer placed which order on what date.



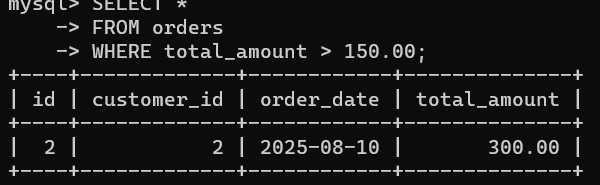
**8. Retrieve the orders with a total amount greater than 150.00.**

SELECT id, customer\_id, total\_amount

FROM orders

WHERE total\_amount > 150.00;

**Explanation**: Filters orders where the total is above 150.



**9. Normalize the database by creating a separate table for order items.**

CREATE TABLE order\_items (

id INT AUTO\_INCREMENT PRIMARY KEY,

order\_id INT,

product\_id INT,

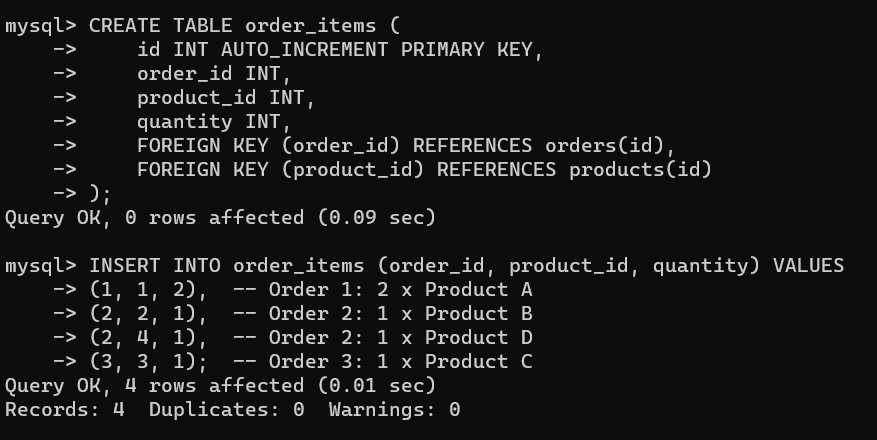
quantity INT,

FOREIGN KEY (order\_id) REFERENCES orders(id),

FOREIGN KEY (product\_id) REFERENCES products(id)

);

**Explanation**: Breaks down orders into individual products with quantities.



**10. Retrieve the average total of all orders.**

SELECT AVG(total\_amount) AS average\_order

FROM orders;

**Explanation**: Calculates the average total value across all orders.

