```
-- Create and use the ecommerce database
CREATE DATABASE ecommerce;
USE ecommerce;

-- Create customers table
CREATE TABLE customers (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    address VARCHAR(255)
);

-- Create products table
CREATE TABLE products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(10, 2) CHECK (price > 0),
    description TEXT
);

-- Create orders table (initial version before normalization)
CREATE TABLE orders (
    id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    order_date DATE DEFAULT CURRENT_DATE,
    total_amount DECIMAL(10, 2),
    FOREIGN KEY (customer_id) REFERENCES customers(id)
);
```

```
-- Insert sample data into customers table
INSERT INTO customers (name, email, address) VALUES
('Alice', 'alice@example.com', '123 Elm Street'),
('Bob', 'bob@example.com', '456 Oak Avenue'),
('Charlie', 'charlie@example.com', '789 Maple Drive');

-- Insert sample data into products table
INSERT INTO products (name, price, description) VALUES
('Product A', 25.00, 'This is Product A'),
('Product B', 30.00, 'This is Product B'),
('Product C', 50.00, 'This is Product C'),
('Product D', 20.00, 'This is Product D');

-- Insert sample data into orders table
INSERT INTO orders (customer_id, order_date, total_amount) VALUES
(1, CURDATE(), 75.00),
(2, CURDATE() - INTERVAL 10 DAY, 160.00),
(1, CURDATE() - INTERVAL 35 DAY, 200.00),
(3, CURDATE() - INTERVAL 5 DAY, 180.00);

-- Task 1: Retrieve all customers who have placed an order in the last 30 days
SELECT DISTINCT c.*
FROM customers c
JOIN orders o ON c.id = o.customer_id
WHERE o.order_date >= CURDATE() - INTERVAL 30 DAY;

-- Task 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.id;

-- Task 3: Update the price of Product C to 45.00
UPDATE products
SET price = 45.00
WHERE name = 'Product C';

-- Task 4: Add a new column discount to the products table
ALTER TABLE products
ADD COLUMN discount DECIMAL(5, 2) DEFAULT 0.00;

-- Task 5: Retrieve the top 3 products with the highest price
SELECT * FROM products
ORDER BY price DESC
LIMIT 3;

-- Normalization: Create order_items table
CREATE TABLE order_items (
    id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT,
    product_id INT,
    quantity INT NOT NULL CHECK (quantity > 0),
    unit_price DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (order_id) REFERENCES orders(id),
    FOREIGN KEY (product_id) REFERENCES products(id)
);

-- Insert sample order items
```

```
INSERT INTO order_items (order_id, product_id, quantity, unit_price) VALUES
(1, 1, 2, 25.00),
(1, 2, 1, 30.00),
(2, 3, 1, 50.00),
(2, 4, 3, 20.00),
(3, 1, 4, 25.00),
(4, 2, 2, 30.00);

-- Now we can properly answer Task 6 (this should come after order_items is created)
-- Task 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';

-- Task 7: Join the orders and customers tables
SELECT c.name, o.order_date
FROM orders o
JOIN customers c ON o.customer_id = c.id;

-- Task 8: Retrieve orders with total amount > 150.00
SELECT * FROM orders
WHERE total_amount > 150.00;

-- Task 9: Normalization already implemented with order_items table
-- For proper normalization, we would:
-- 1. Remove total_amount from orders table
```

```
-- 2. Calculate order totals from order_items


-- Task 10: Retrieve the average total of all orders
SELECT AVG(total_amount) AS average_order_total
FROM orders;
```