Ecommerce – SQL

CREATING DATABASE

```
create database codingchallenge
    -> ;
```

```
use codingchallenge;
```

CREATING TABLE CUSTOMERS

```
CREATE TABLE customers (
    ->     customerID INT PRIMARY KEY,
    ->     firstName VARCHAR(50),
    ->     lastName VARCHAR(50),
    ->     Email VARCHAR(100) UNIQUE,
    ->     address VARCHAR(255)
    -> );
```

INSERTING VALUES IN THE TABLE CUSTOMERS

```
INSERT INTO customers (customerID, firstName, lastName, Email, address) VALUES
    -> (1, 'John', 'Doe', 'johndoe@example.com', '123 Main St, City'),
    -> (2, 'Jane', 'Smith', 'janesmith@example.com', '456 Elm St, Town'),
    -> (3, 'Robert', 'Johnson', 'robert@example.com', '789 Oak St, Village'),
    -> (4, 'Sarah', 'Brown', 'sarah@example.com', '101 Pine St, Suburb'),
    -> (5, 'David', 'Lee', 'david@example.com', '234 Cedar St, District'),
    -> (6, 'Laura', 'Hall', 'laura@example.com', '567 Birch St, County'),
    -> (7, 'Michael', 'Davis', 'michael@example.com', '890 Maple St, State'),
    -> (8, 'Emma', 'Wilson', 'emma@example.com', '321 Redwood St, Country'),
    -> (9, 'William', 'Taylor', 'william@example.com', '432 Spruce St, Province'),
    -> (10, 'Olivia', 'Adams', 'olivia@example.com', '765 Fir St, Territory');
```

CREATING TABLE PRODUCTS

```
CREATE TABLE products (
   ->     product_id INT PRIMARY KEY,
   ->     name VARCHAR(100),
   ->     price DECIMAL(10,2),
   ->     description TEXT,
   ->     stockQuantity INT
   -> );
```

INSERTING VALUES IN THE TABLE PRODUCTS

```
insert into products (product_id,name,description,price,stockquantity) values
   -> (1,'Laptop','High-performance laptop',800.00,10),
   -> (2,'smartphone','Latest smartphone',600.00,15),
   -> (3,'Tablet','Portable tablet',300.00,20),
   -> (4,'Headphones','Noise-cancelling',150.00,30);
insert into products (product_id,name,description,price,stockquantity) values
   -> (5,'TV','4k smart tv',900.00,5),
   -> (6,'coffee maker','automatic coffee maker',50.00,25);
INSERT INTO products (product_id, name, description, price, stockQuantity) VALUES
   -> (7, 'Refrigerator', 'Energy-efficient', 700.00, 10),
   -> (8, 'Microwave Oven', 'Countertop microwave', 80.00, 15),
   -> (9, 'Blender', 'High-speed blender', 70.00, 20),
   -> (10, 'Vacuum Cleaner', 'Bagless vacuum cleaner', 120.00, 10);
```

CREATING TABLE CART

```
CREATE TABLE cart (
   ->     cart_id INT PRIMARY KEY,
   ->     customer_id INT,
   ->     product_id INT,
   ->     quantity INT,
   ->     FOREIGN KEY (customer_id) REFERENCES customers(customerID),
   ->     FOREIGN KEY (product_id) REFERENCES products(product_id)
   -> );
```

INSERTING VALUES IN TABLE CART

```
INSERT INTO cart (cart_id, customer_id, product_id, quantity) VALUES
   -> (1, 1, 1, 2),
   -> (2, 1, 3, 1),
   -> (3, 2, 2, 3),
   -> (4, 3, 4, 4),
   -> (5, 3, 5, 2),
   -> (6, 4, 6, 1),
   -> (7, 5, 1, 1),
   -> (8, 6, 10, 2),
   -> (9, 6, 9, 3),
   -> (10, 7, 7, 2);
```

CREATING TABLE ORDERS

```
CREATE TABLE orders (
   ->     order_id INT PRIMARY KEY,
   ->     customer_id INT,
   ->     order_date DATE,
   ->     total_price DECIMAL(10,2),
   ->     shipping_address VARCHAR(255),
   ->     FOREIGN KEY (customer_id) REFERENCES customers(customerID)
   -> );
```

INSERTING VALUES INTO ORDERS

```
INSERT INTO orders (order_id, customer_id, order_date, total_price) VALUES
   -> (1, 1, '2023-01-05', 1200.00),
   -> (2, 2, '2023-02-10', 900.00),
   -> (3, 3, '2023-03-15', 300.00),
   -> (4, 4, '2023-04-20', 150.00),
   -> (5, 5, '2023-05-25', 1800.00),
   -> (6, 6, '2023-06-30', 400.00),
   -> (7, 7, '2023-07-05', 700.00),
   -> (8, 8, '2023-08-10', 160.00),
   -> (9, 9, '2023-09-15', 140.00),
   -> (10, 10, '2023-10-20', 1400.00);
```

CREATING TABLE ORDER_ITEMS

```
CREATE TABLE order_items (
   ->     order_item_id INT PRIMARY KEY,
   ->     order_id INT,
   ->     product_id INT,
   ->     quantity INT,
   ->     FOREIGN KEY (order_id) REFERENCES orders(order_id),
   ->     FOREIGN KEY (product_id) REFERENCES products(product_id)
   -> );
ALTER TABLE order_items ADD COLUMN item_amount DECIMAL(10,2);
```

INSERTING VALUES IN ORDER_ITEMS

```
INSERT INTO order_items (order_item_id, order_id, product_id, quantity, item_amount) VALUES
   -> (2, 1, 3, 1, 300.00),
   -> (3, 2, 2, 3, 1800.00),
   -> (4, 3, 5, 2, 1800.00),
   -> (5, 4, 4, 4, 600.00),
   -> (6, 4, 6, 1, 50.00),
   -> (7, 5, 1, 1, 800.00),
   -> (8, 5, 2, 2, 1200.00),
   -> (9, 6, 10, 2, 240.00),
   -> (10, 6, 9, 3, 210.00);
```

SQL QUESTIONS

2. Remove all cart items for a specific customer.

```
DELETE FROM cart
WHERE customer_id = 3;
```

3. Retrieve Products Priced Below $100.

```
SELECT * FROM products
WHERE price < 100;
```

4. Find Products with Stock Quantity Greater Than 5.

```
SELECT * FROM products
WHERE stockQuantity > 5;
```

5. Retrieve Orders with Total Amount Between $500 and $1000.

```
SELECT * FROM orders
WHERE total_price BETWEEN 500 AND 1000;
```

6. Find Products which name end with letter ‘r’.

```
SELECT * FROM products
WHERE name LIKE '%r';
```

7. Retrieve Cart Items for Customer 5.

```
SELECT
   ->     cart.cart_id,
   ->     customers.customerID,
   ->     customers.firstName,
   ->     customers.lastName,
   ->     products.product_id,
   ->     products.name AS product_name,
   ->     cart.quantity
   -> FROM cart
   -> JOIN customers ON cart.customer_id = customers.customerID
   -> JOIN products ON cart.product_id = products.product_id
   -> WHERE cart.customer_id = 5;
```

8. Find Customers Who Placed Orders in 2023.

```
SELECT  c.*
FROM customers c
JOIN orders o ON c.customerID = o.customer_id
WHERE YEAR(o.order_date) = 2023;
```

9. Determine the Minimum Stock Quantity for Each Product Category.

```
select name,min(stockquantity)
   -> from products
   -> group by name;
```

10. Calculate the Total Amount Spent by Each Customer.

```
SELECT o.customer_id, c.firstName, c.lastName, SUM(o.total_price) AS total_spent
   -> FROM orders o
   -> JOIN customers c ON o.customer_id = c.customerID
   -> GROUP BY o.customer_id, c.firstName, c.lastName;
```

11. Find the Average Order Amount for Each Customer.

```
SELECT o.customer_id, c.firstName, c.lastName, AVG(o.total_price) AS average_order_value
   -> FROM orders o
   -> JOIN customers c ON o.customer_id = c.customerID
   -> GROUP BY o.customer_id;
```

12. Count the Number of Orders Placed by Each Customer.

```
SELECT orders.customer_id, customers.firstName, customers.lastName, COUNT(orders.order_id)
FROM orders
JOIN customers ON orders.customer_id = customers.customerID
GROUP BY orders.customer_id, customers.firstName, customers.lastName;
```

13. Find the Maximum Order Amount for Each Customer.

```
SELECT orders.customer_id, customers.firstName, customers.lastName, MAX(orders.total_price)

FROM orders

JOIN customers ON orders.customer_id = customers.customerID

GROUP BY orders.customer_id, customers.firstName, customers.lastName;
```

14. Get Customers Who Placed Orders Totaling Over $1000.

```
SELECT orders.customer_id, customers.firstName, customers.lastName, SUM(orders.total_price)

FROM orders

JOIN customers ON orders.customer_id = customers.customerID

GROUP BY orders.customer_id, customers.firstName, customers.lastName

HAVING SUM(orders.total_price) > 1000;
```

15. Subquery to Find Products Not in the Cart.

```
SELECT * FROM products

WHERE product_id NOT IN (SELECT product_id FROM cart);
```

16. Subquery to Find Customers Who Haven't Placed Orders.

```
SELECT * FROM customers

WHERE customerID NOT IN (SELECT customer_id FROM orders);
```

17. Subquery to Calculate the Percentage of Total Revenue for a Product.

```
SELECT order_items.product_id,

    (SUM(order_items.quantity * products.price) * 100) /

    (SELECT SUM(order_items.quantity * products.price) FROM order_items

     JOIN products ON order_items.product_id = products.product_id)

FROM order_items

JOIN products ON order_items.product_id = products.product_id as percentage

GROUP BY order_items.product_id;
```

18. Subquery to Find Products with Low Stock.

```
SELECT * FROM products
```

```
WHERE stockQuantity < (SELECT AVG(stockQuantity) FROM products);
```

19. Subquery to Find Customers Who Placed High-Value Orders.

```
SELECT * FROM customers

WHERE customerID IN (

   SELECT customer_id FROM orders

   WHERE total_price > (SELECT AVG(total_price) FROM orders)

);
```