Drop database if already exists

```
DROP DATABASE IF EXISTS ecommerce_analytics;

CREATE DATABASE ecommerce_analytics;

USE ecommerce_analytics;
```

Create Customers Table

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(50),
    city VARCHAR(50),
    signup_date DATE
);
```

Create Products Table

```
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(50),
    price DECIMAL(10,2)
);
```

Create Orders Table

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    total_amount DECIMAL(10,2),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

Create Order Items Table (Many-to-Many Relationship)

```
CREATE TABLE order_items (
    order_item_id INT PRIMARY KEY,
    order_id INT,
    product_id INT,
    quantity INT,
    subtotal DECIMAL(10,2),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

1.Calculate Total Revenue

```
SELECT SUM(total_amount) AS total_revenue
FROM orders;
```

2.Calculate Monthly Revenue (Time-based Aggregation)

```
SELECT
    DATE_FORMAT(order_date, '%Y-%m') AS month,
    SUM(total_amount) AS monthly_revenue
FROM orders
GROUP BY month
ORDER BY month;
```

3.Find Top Selling Products (By Quantity)

```
SELECT
    p.product_name,
    SUM(oi.quantity) AS total_quantity
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
GROUP BY p.product_name
ORDER BY total_quantity DESC;
```

4.Calculate Revenue by Product Category

```
SELECT
    p.category,
    SUM(oi.subtotal) AS category_revenue
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
GROUP BY p.category;
```

5.Identify Top Customers by Total Spending

```
SELECT
    c.customer_name,
    SUM(o.total_amount) AS total_spent
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
GROUP BY c.customer_name
ORDER BY total_spent DESC;
```

6.Calculate Average Order Value

```
SELECT AVG(total_amount) AS avg_order_value
FROM orders;
```

7.Filter Customers from Delhi

```
SELECT *

FROM customers

WHERE city = 'Delhi';
```

8.Find Highest Order Value

```
SELECT *

FROM orders

ORDER BY total_amount DESC

LIMIT 1;
```

9.Find Second Highest Order Value (Using Subquery)

```
SELECT MAX(total_amount)

FROM orders

WHERE total_amount < (SELECT MAX(total_amount) FROM orders);
```

10. Calculate Customer Lifetime Value (CLV)

```
SELECT

    c.customer_name,
```

```
    SUM(o.total_amount) AS lifetime_value
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
GROUP BY c.customer_name;
```

11.Rank Customers by Spending (Window Function)

```
SELECT
    c.customer_name,
    SUM(o.total_amount) AS total_spent,
    RANK() OVER (ORDER BY SUM(o.total_amount) DESC) AS spending_rank
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
GROUP BY c.customer_name;
```

12.Calculate Running Total Revenue (Cumulative Sales)

```
SELECT
    order_date,
    SUM(total_amount) OVER (ORDER BY order_date) AS running_total
FROM orders;
```

13.Create Index for Performance Optimization

```
CREATE INDEX idx_order_date ON orders(order_date);
```

14.Create View for Monthly Sales Report

```
CREATE VIEW monthly_sales AS
SELECT
   DATE_FORMAT(order_date, '%Y-%m') AS month,
   SUM(total_amount) AS revenue
FROM orders
GROUP BY month;
```

15.Create Stored Procedure to Get Orders of Specific Customer

```
DELIMITER //

CREATE PROCEDURE GetCustomerOrders(IN cust_id INT)
BEGIN
   SELECT * FROM orders
   WHERE customer_id = cust_id;
END //
DELIMITER ;
```