Enterprise-Level SQL Project: Scalable E-Commerce Database System

1. Database Schema Design (Normalization & Relationships)

A well-structured database follows **3NF (Third Normal Form)** to eliminate redundancy while ensuring efficient data retrieval.

Database Schema Overview:

- **Customers** (Stores user details)
- **Products** (Inventory management)
- **Orders** (Stores transactions)
- **Order Details** (Handles many-to-many relationships)
- **Payments** (Tracks payment methods)
- **Reviews** (User-generated ratings and feedback)

Database Schema (DDL - Table Creation)

sql

CopyEdit

```
-- Step 1: Create Database

CREATE DATABASE ECommerceDB;

USE ECommerceDB;


-- Step 2: Customers Table

CREATE TABLE Customers (

    customer_id INT PRIMARY KEY AUTO_INCREMENT,

    first_name VARCHAR(50) NOT NULL,

    last_name VARCHAR(50) NOT NULL,

    email VARCHAR(100) UNIQUE NOT NULL,
```

```
    phone VARCHAR(15),
    address TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Step 3: Products Table
CREATE TABLE Products (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    category VARCHAR(50),
    price DECIMAL(10,2) CHECK (price > 0),
    stock_quantity INT CHECK (stock_quantity >= 0),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Step 4: Orders Table
CREATE TABLE Orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    total_amount DECIMAL(10,2),
    status ENUM('Pending', 'Shipped', 'Delivered', 'Cancelled') DEFAULT 'Pending',
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id) ON DELETE
CASCADE
);
```

```
-- Step 5: Order Details Table (Many-to-Many Relationship)
CREATE TABLE OrderDetails (
    order_detail_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    product_id INT,
    quantity INT CHECK (quantity > 0),
    subtotal DECIMAL(10,2),
    FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES Products(product_id) ON DELETE CASCADE
);

-- Step 6: Payments Table
CREATE TABLE Payments (
    payment_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    payment_method ENUM('Credit Card', 'Debit Card', 'PayPal', 'Crypto') NOT NULL,
    payment_status ENUM('Successful', 'Failed', 'Pending') DEFAULT 'Pending',
    amount DECIMAL(10,2) NOT NULL,
    transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE
);
```

```
-- Step 7: Reviews Table
CREATE TABLE Reviews (
    review_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    product_id INT,
    rating INT CHECK (rating BETWEEN 1 AND 5),
    review_text TEXT,
    review_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES Products(product_id) ON DELETE CASCADE
);
```

---

2. Advanced SQL Queries for Data Analysis & Insights

 Query 1: Find Top 5 Customers by Total Spend

This query uses **JOINs, Aggregation, and ORDER BY** to identify high-value customers.

```
sql
CopyEdit
SELECT c.customer_id,
       CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
       SUM(o.total_amount) AS total_spent
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id
```

```
ORDER BY total_spent DESC
LIMIT 5;
```

Query 2: Identify Low Stock Products Using CTEs

Common Table Expressions (**CTEs**) improve query readability.

```
sql
CopyEdit
WITH LowStock AS (
   SELECT product_id, name, stock_quantity
   FROM Products
   WHERE stock_quantity < 10
)
SELECT * FROM LowStock;
```

Query 3: Monthly Sales Trends Using Window Functions

Calculating **running total revenue** per month using **PARTITION BY**.

```
sql
CopyEdit
SELECT DATE_FORMAT(order_date, '%Y-%m') AS month,
      SUM(total_amount) AS monthly_revenue,
      SUM(SUM(total_amount)) OVER (ORDER BY DATE_FORMAT(order_date,
'%Y-%m')) AS running_total
FROM Orders
GROUP BY month;
```

Query 4: Stored Procedure for Automated Order Summary Reports

Stored Procedures automate **data reporting**.

```
sql

CopyEdit

DELIMITER //

CREATE PROCEDURE GetOrderSummary(IN customerId INT)

BEGIN

   SELECT o.order_id,

        o.order_date,

        SUM(od.subtotal) AS total_spent

   FROM Orders o

   JOIN OrderDetails od ON o.order_id = od.order_id

   WHERE o.customer_id = customerId

   GROUP BY o.order_id;

END //

DELIMITER;
```

---

3. Database Performance Optimization & Best Practices

Indexing for Faster Queries

```
sql

CopyEdit

-- Index on customer email for faster lookups

CREATE INDEX idx_email ON Customers(email);
```

-- Composite Index on Orders for better performance

CREATE INDEX idx_orders ON Orders(customer_id, order_date);

Security & Role-Based Access Control (RBAC)

sql

CopyEdit

-- Create Read-Only User

CREATE USER 'readonly_user'@'localhost' IDENTIFIED BY 'SecurePassword!';

GRANT SELECT ON ECommerceDB.* TO 'readonly_user'@'localhost';

-- Create Admin User with Full Privileges

CREATE USER 'admin_user'@'localhost' IDENTIFIED BY 'SuperSecure!';

GRANT ALL PRIVILEGES ON ECommerceDB.* TO 'admin_user'@'localhost';