Online Retail Sales Database Design

Project Report

Prepared by: srushti kottur
Date: July 2025

Table of Contents

1. Introduction

This project focuses on the design of a normalized SQL schema for an online retail sales platform. The goal is to design an efficient, scalable, and structured relational database that supports an e-commerce application with customers, products, orders, and payments.

2. Objectives

- Design a normalized SQL schema for an e-commerce platform.
- Identify and define key entities.
- Create an Entity Relationship (ER) diagram.
- Normalize the schema to the third normal form (3NF).
- Write DDL scripts to create tables with proper constraints.
- Populate the database with sample data.
- Write SQL queries and views for sales reports.

3. Entity Identification

The key entities identified in the system are:

- **Customer** – A registered user who can place orders.
- **Product** – Items available for purchase.
- **Order** – Purchase transaction details for a customer.
- **Order Item** – Items in an order (supports multi-product orders).
- **Payment** – Payment details for an order.

4. ER Diagram (Text-based version)

```
Table customers {
  customer_id INT PK
  name VARCHAR
  email VARCHAR UNIQUE
  phone VARCHAR
```

```
  address TEXT
}

Table products {
  product_id INT PK
  name VARCHAR
  description TEXT
  price DECIMAL
  stock_quantity INT
}

Table orders {
  order_id INT PK
  customer_id INT FK
  order_date DATE
  total_amount DECIMAL
  status VARCHAR
}

Table order_items {
  order_item_id INT PK
  order_id INT FK
  product_id INT FK
  quantity INT
  price DECIMAL
}

Table payments {
  payment_id INT PK
  order_id INT FK
  payment_date DATE
  amount DECIMAL
  payment_method VARCHAR
  payment_status VARCHAR
}

Ref: orders.customer_id > customers.customer_id
Ref: order_items.order_id > orders.order_id
Ref: order_items.product_id > products.product_id
Ref: payments.order_id > orders.order_id
```

5. Database Schema (MySQL DDL)

```
CREATE TABLE customers (
    customer_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(20),
    address TEXT
);

CREATE TABLE products (
    product_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    price DECIMAL(10,2) NOT NULL,
    stock_quantity INT DEFAULT 0
);

CREATE TABLE orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT NOT NULL,
    order_date DATE NOT NULL,
    total_amount DECIMAL(10,2) NOT NULL,
    status VARCHAR(50) DEFAULT 'Pending',
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

CREATE TABLE order_items (
    order_item_id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT NOT NULL,
    product_id INT NOT NULL,
    quantity INT NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

CREATE TABLE payments (
    payment_id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT NOT NULL,
    payment_date DATE NOT NULL,
```

```
    amount DECIMAL(10,2) NOT NULL,
    payment_method VARCHAR(50),
    payment_status VARCHAR(50),
    FOREIGN KEY (order_id) REFERENCES orders(order_id)
);
```

6. Data Normalization (3NF)

The schema follows **Third Normal Form (3NF)**:

- **1NF (First Normal Form):** No repeating groups or arrays; atomic values.
- **2NF (Second Normal Form):** Every non-key attribute is fully functionally dependent on the whole primary key.
- **3NF (Third Normal Form):** No transitive dependencies. All non-key attributes are directly dependent on the primary key.

7. Sample Data (INSERT Statements)

```
-- Customers
INSERT INTO customers (name, email, phone, address) VALUES
('Alice', 'alice@example.com', '1234567890', '123 Main St'),
('Bob', 'bob@example.com', '2345678901', '456 Oak Ave');

-- Products
INSERT INTO products (name, description, price, stock_quantity)
VALUES
('Laptop', 'High performance laptop', 1200.00, 10),
('Phone', 'Smartphone with good camera', 800.00, 20);

-- Orders
INSERT INTO orders (customer_id, order_date, total_amount, status)
VALUES
(1, '2025-07-20', 2000.00, 'Shipped'),
(2, '2025-07-21', 800.00, 'Pending');

-- Order Items
```

```
INSERT INTO order_items (order_id, product_id, quantity, price)
VALUES
(1, 1, 1, 1200.00),
(1, 2, 1, 800.00),
(2, 2, 1, 800.00);

-- Payments
INSERT INTO payments (order_id, payment_date, amount,
payment_method, payment_status) VALUES
(1, '2025-07-21', 2000.00, 'Credit Card', 'Completed'),
(2, '2025-07-22', 800.00, 'PayPal', 'Pending');
```

8. SQL Queries and Views

Total Sales Per Product

```
SELECT
    p.name AS product_name,
    SUM(oi.quantity) AS total_units_sold,
    SUM(oi.price * oi.quantity) AS total_revenue
FROM
    order_items oi
JOIN
    products p ON oi.product_id = p.product_id
GROUP BY
    p.name;
```

Orders with Customer Info

```
SELECT
    o.order_id,
    o.order_date,
    c.name AS customer_name,
    o.total_amount,
    o.status
FROM
    orders o
JOIN
```

```
    customers c ON o.customer_id = c.customer_id;
```

View for Completed Payments

```
CREATE VIEW completed_payments AS
SELECT
    p.payment_id,
    o.order_id,
    c.name AS customer_name,
    p.amount,
    p.payment_method,
    p.payment_status,
    p.payment_date
FROM
    payments p
JOIN
    orders o ON p.order_id = o.order_id
JOIN
    customers c ON o.customer_id = c.customer_id
WHERE
    p.payment_status = 'Completed';
```

9. Conclusion

This project successfully demonstrates a complete and normalized database design for an online retail platform. It supports various essential operations such as managing customers, products, and orders, along with payment tracking. By following best practices in SQL design and normalization, the database is efficient, consistent, and ready for real-world implementation.