San Jose State University

Online Clothing Store

Christopher Oh | 015008422 | christopher.oh@sjsu.edu
Diya Dalal | 017092257 | diya.dalal@sjsu.edu
Andranik Avagyan | 017423133 | andranik.avagyan@sjsu.edu
4 December 2025

Introduction

- **Project Overview:** Our project is an e-commerce platform that simulates the core functions of an online clothing site. Customers can log in, browse products from the database, add products to their cart, and purchase items. This project is relevant because online shopping is very popular currently.
- **Chosen Topic:** Our chosen topic is an e-commerce platform. It is a practical application that is commonly used.

Objectives

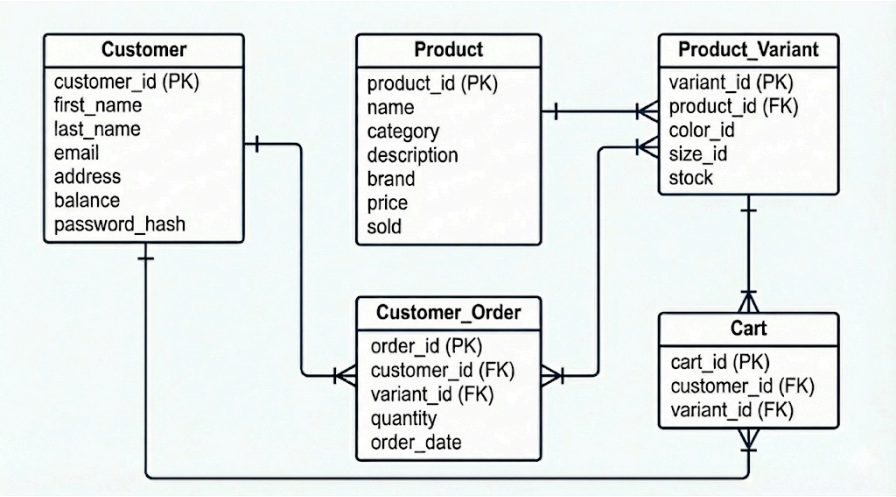
- **Primary Goals:** Our primary goal for our project was to develop a fully functioning e-commerce website using MySQL and JDBC, and a three-tier architecture. We wanted to implement logging in, browsing, and placing orders.
- **Key Features:** Our first feature is a customer login, through which new users get automatically added to the database. Then, we feature product browsing based on the items populated in our database, and automatic creation of new product variants based on what the customer selects. Finally, we have a cart feature in which you can place an order, and storage of customer's order information in a CustomerOrder table.

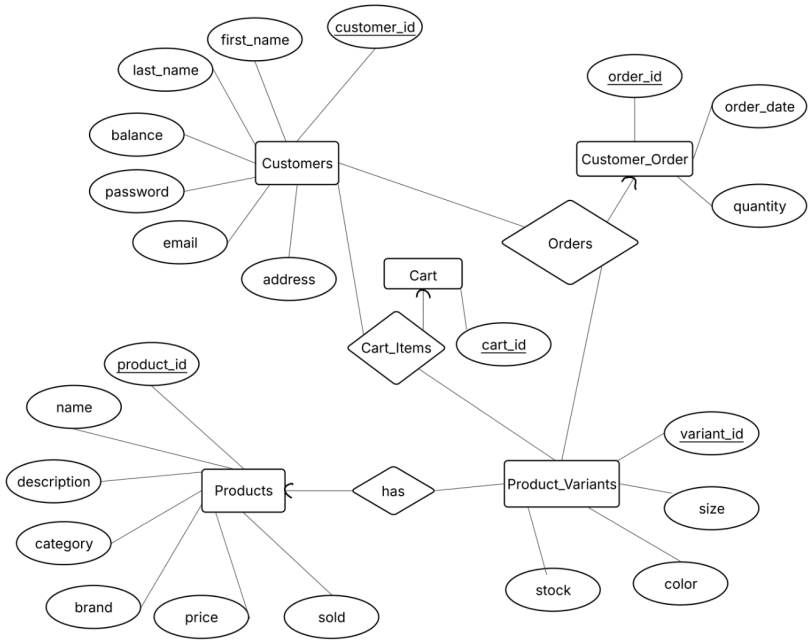
System Architecture

- **Three-Tier Architecture**: Our database layer is made up of MySQL database tables, which our project creates through JDBC. The logic layer is composed of Java controllers and DAO classes for request handling and database operations.
- **Technology Stack**: We used SpringBoot and Maven to build the project. The database uses MySQL, backend uses Java and JDBC, and the frontend uses HTML pages.
- **Database Connection**: The database connects to our Java project through JDBC. Through Maven, we added JDBC and MySQL dependencies. In order for a user to have the database configured on their device, they must run the DBInitializer.java file.

Database Design

- **Schema Diagram**:





- **Normalization**:

  Customer(<u>customer_id</u>, first_name, last_name, email, address, balance, password_hash)

  Product(<u>product_id</u>, name, category, description, brand, price, sold)

  Product_Variant(<u>variant_id</u>, product_id, size, color, stock)

  Cart(<u>cart_id</u>, customer_id, variant_id)

  Customer_Order(<u>order_id</u>, customer_id, variant_id, order_date)

- **Entity Descriptions**:

Customer - users who can purchase items from the e-store

Product - clothing items available

Product_Variant - specific size and colors of product to be purchased

Cart - list of product_variants the customer wishes to buy

Customer_Order - list of individual items purchased by a customer

- **Attributes and Constraints**:

Customer Table				
Attribute	SQL Type	Java Type	Description	Constraints
customer_id	INTEGER	int	autoincremented key for users	Primary key, autoincremented
first_name	VARCHAR(100)	String	User's first name	Not null
last_name	VARCHAR(100)	String	User's last name	Not null
email	VARCHAR(100)	String	User's email	Not null, unique
address	VARCHAR(255)	String	User' address	Not null
password_hash	VARCHAR(100)	String	Hashed password	Not null

Product				
Attribute	SQL Type	Java Type	Description	Constraints
product_id	INTEGER	int	autoincremented key for product	Primary key, autoincremented
name	VARCHAR(255)	String	Name of product	Not null
category	ENUM	enum	Type of clothing, such as shirt or pants	Not null
description	TEXT	String	Short description of product	Not null
brand	VARCHAR(100)	String	Clothing brand	Not null
price	DECIMAL(10,2)	double	Price of product	Not null, price > 0
sold	BOOLEAN	boolean	Whether item has been sold or not	Default to false

Product_Variant				
Attribute	SQL Type	Java Type	Description	Constraints
variant_id	INTEGER	int	autoincremented key for product	Primary key, autoincremented
product_id	INTEGER	int	What product this is a variant of	Foreign key to product_id in Product. Cascade on delete and update
color	ENUM	enum	Color of variant	Not null
size	ENUM	enum	Size of variant	Not null
stock	INTEGER	int	How many of this variant are in stock	Not null, stock > 0

Cart				
Attribute	SQL Type	Java Type	Description	Constraints
cart_id	INTEGER	int	autoincremented key for product	Primary key, autoincremented
customer_id	INTEGER	int	Who the cart belongs to	Foreign key to customer_id in Customer. Cascade on delete and update
variant_id	INTEGER	int	Specific variant added to customer’s cart	Foreign key to variant_id in Product_Variant. Cascade on delete and update

Customer_Order				
Attribute	SQL Type	Java Type	Description	Constraints
order_id	INTEGER	int	autoincremented key for product. Logs specific purchase of one item	Primary key, autoincremented

customer_id	INTEGER	int	Who bought this product variant	Foreign key to customer_id in Customer. Cascade on delete and update
variant_id	INTEGER	int	Specific variant bought	Foreign key to variant_id in Product_Variant. Cascade on delete and update
order_date	DATETIME	LocalDateTime	Time and date of purchase	Not null, default current timestamp

Functional Requirements

- **User Roles and Access**: Users of the system are customers. They can interact with the system by logging in, browsing products, adding items to their cart, and buying items to create orders.
- **Feature Descriptions**:
    - Adding new records: When a customer enters information on the login page, a new record is created in the database matching the information that was inputted.
    - Viewing existing data: The products browsing page is populated with clothing entries that we have already added to the database.
    - Updating records: When a customer buys products from their cart, the stock field in the Product_Variant table gets decreased based on how many items they ordered.
    - Deleting records: Customers can delete items from their cart by clicking the "remove" button. This will delete a record from the cart table.

Implementation Details

- **Code Structure**: The two main project folders are /java and /resources. In /java, the /store package has the DBInitializer class and controller classes. The /model package contains classes for getters and setters. The /enums folder lists out colors, sizes, and categories. The /dal package contains all the DAO classes and DAO implementation

classes. The resources folder contains /templates, which are HTML pages that the user views when running the project.

- **Important Code Snippets**:

Database Connection:

```
//Creating SQL database through JDBC
public class StoreDBInitializer {
    public void initialize() {
        String url = "jdbc:mysql://localhost:3306/";
        String user = "root";
        String password = "password"; //change to your SQL username + password when running

        //try-catch block to handle connection errors
        try (Connection conn = DriverManager.getConnection(url, user, password);
             Statement stmt = conn.createStatement()) {

            stmt.executeUpdate("CREATE DATABASE IF NOT EXISTS clothing_store");
            stmt.executeUpdate("USE clothing_store");

            //creating database tables through JDBC
            stmt.executeUpdate("""
                CREATE TABLE Customer (
```

Example CRUD Operations:

```
@Override
public Cart findById(int cartId) {
    String sql = "SELECT * FROM Cart WHERE cart_id = ?";
    List<Cart> result = jdbc.query(sql, mapper, cartId);
    return result.isEmpty() ? null : result.get(0);
}
```

```
@Override
public boolean delete(int orderId) {
    String sql = "DELETE FROM Customer_Order WHERE order_id = ?";
    return jdbc.update(sql, orderId) > 0;
}
```

- **Error Handling**: Our code uses try/catch blocks when dealing with database connections, giving us appropriate error messages if anything goes wrong.

Testing and Validation

We employed both manual and unit testing strategies to ensure system reliability and integrity. This approach enabled independent testing of the application tier before integration.

- **Test Cases**:

Test ID	Functional Area	Test Description	Expected Outcome
TC_CUST_01	Customer DAO	findById(int id)	Retrieve a customer by their valid ID. The returned Customer object should match the database record.

TC_CUST_02	Customer DAO	save(Customer customer)	Insert a new customer record. The operation should return true, and the new record should be retrievable.
TC_PROD_01	Product DAO	findById(int id)	Retrieve a product by its valid ID. The Product object, including its enum Category, should map correctly.
TC_PROD_02	Product DAO	findAll()	Retrieve all products. A list of all product records should be returned.
TC_PV_01	Product Variant DAO	findByProductId(int productId)	Retrieve all variants for a specific product ID. A list of corresponding ProductVariant objects should be returned.
TC_CART_01	Cart DAO	insert(Cart cart)	Add a product variant to a customer's cart.
TC_CART_02	Cart DAO	findByCustomerId(int customerId)	Retrieve all cart items for a specific customer.
TC_CART_03	Cart DAO	delete(int cartId)	Remove a single item from the cart.
TC_ORD_01	Order DAO	insert(CustomerOrder order)	Create a new order record for a customer buying a product variant.
TC_ORD_02	Order DAO	findByCustomerId(int customerId)	Retrieve the order history for a customer.

- **Sample Data**: We used SQL insert statements to populate the database. Customer: 15 customers. Product: 15 products. Product_Variant: One entry for each product. Customer_Order: Each existing customer had one order. Cart: Each customer has one product in their cart..

  Some sample entries are:

Customer	ID: 1, Name: "Alice Rivera", Email: "alice1@example.com"
Product	ID: 1, Name: "Cotton Tee", Category: "Shirt", Brand: "AlphaWear", Price: 19.99
Product_Variant	ID: 1, Product ID: 1, Color: 'Black', Size: 'S', Stock: 10
Cart	Customer ID: 1, Variant ID: 1
Customer_Order	Customer ID: 1, Variant ID: 1

- **Test Results**: Core functionalities have been tested both manually and with unit tests. Test cases helped to seamlessly integrate the presentation layer with the application layer.

Challenges and Solutions

- **Challenges Faced**: One of the most notable challenges we faced as a group was figuring out the table's schema. Our initial plans changed to become simpler and more

understandable. During this time, each group member had a unique approach, which widened our possibilities.

- **Solutions**: To solve our challenges, we each created a way to initialize the database and sent it to be reviewed by other members. After the review, we had several meetings to explain our ideas to each other and came up with a solution. A similar approach was used for our other challenges; we had weekly meetings and, during the week, chatted with each other to help and support each other to go forward.

Future Enhancements

- **Review:** In the future, we'd like to add a review feature. Customers will be able to leave a review of 1-5 stars and an option text description for both Product and Product_Variant. Other customers will also be able to select if they found the review helpful. Products and variants will display the average rating as well as the most helpful reviews with the option to view all of them.
- **Payment:** For the sake of this project, we just kept the concept of payment as a customer balance to keep the scope within reason. In the future, we'd like to keep a payment feature that tracks the payment method (such as credit card, debit, Paypal, etc) and whether the payment was successful or not.

Conclusion

- **Summary**: Overall, we successfully learned how to create a functional database in SQL, apply it to real-world situations, and connect a database to a web application. We were able to build a website that showcases database management and has a clean, easily navigable UI.
- **Reflections**: Our final project was a success, and accomplishes everything expected of a simple shopping experience. Since this was our first time creating a project of this scale, there are many opportunities to make the final product more complex, and add unique new features to it.

Appendices

- **Database Schema SQL And Sample Queries**:

```sql
CREATE TABLE Customer (
    customer_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    address VARCHAR(255) NOT NULL,
    balance DECIMAL(10, 2) DEFAULT 0,
    password_hash VARCHAR(100) NOT NULL
);
```

```sql
CREATE TABLE Product (
    product_id INT AUTO_INCREMENT PRIMARY KEY,
    name varchar(255) NOT NULL,
    category ENUM('Shirt', 'Blouse', 'Sweater',
    description TEXT,
    brand varchar(100),
    price DECIMAL(10, 2) NOT NULL,
    sold boolean DEFAULT FALSE,
    CHECK (price > 0)
);
```

```sql
CREATE TABLE Product_Variant (
    variant_id INT AUTO_INCREMENT PRIMARY KEY,
    product_id INT,
    color_id ENUM('Black', 'White', 'Grey', 'Red', 'Blue', 'Gree
    size_id ENUM('XXS', 'XS', 'S', 'M', 'L', 'XL', 'XXL'),
    stock INT,

    CONSTRAINT fk_product_product_variant
        FOREIGN KEY (product_id) REFERENCES Product(product_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,

    CHECK (stock >= 0)
);
```

```sql
CREATE TABLE Customer_Order (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT NOT NULL,
    variant_id INT NOT NULL,
    quantity INT NOT NULL,
    order_date DATETIME DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT fk_customer_order
        FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,

    CONSTRAINT fk_variant_order
        FOREIGN KEY (variant_id) REFERENCES Product_Variant(variant_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

```sql
INSERT INTO Cart (customer_id,
    variant_id) VALUES
    (1, 3),
    (2, 5),
```

```sql
CREATE TABLE Cart (
    cart_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT NOT NULL,
    variant_id INT NOT NULL,

    CONSTRAINT fk_customer_cart
        FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,

    CONSTRAINT fk_variant_cart
        FOREIGN KEY (variant_id) REFERENCES Product_Variant(variant_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

```sql
INSERT INTO Product_Variant (product_id,
    color_id, size_id, stock) VALUES
    (1, 'Black', 'M', 20),
    (2, 'Blue', 'L', 15),
```

```sql
INSERT INTO Customer_Order (customer_id,
    variant_id, quantity) VALUES
    (1, 1, 2),
    (2, 2, 1),
```

```sql
INSERT INTO Product (name, category, description, brand, price, sold) VALUES
    ('Everyday T-Shirt', 'Shirt', 'Comfy cotton tee', 'Nike', 9.99, FALSE),
    ('Fuzzy Sweatshirt', 'Hoodie', 'Warm fleece hoodie', 'Adidas', 29.99, FALSE),
```

```sql
INSERT INTO Customer (first_name, last_name, email, address, balance, password_hash) VALUES
    ('John', 'Doe', 'john.doe@gmail.com', '123 Main St', 120.50, '1234'),
    ('Jane', 'Smith', 'jane.smith@yahoo.com', '45 Forest Ave', 705.00, 'abcd'),
    ('Marie', 'Johnson', 'marie.johnson@gmail.com', '223 River Rd', 250.00, 'cats123'),
```

User Manual: The Github README contains a user manual.