SwiftCart Final Report DBMS Project

Kanishk Kumar Meena
2022233

TABLE OF CONTENTS

01

....

PROJECT SCOPE

Project Scope

This project aims to develop an online grocery store system named SwiftCart.com that provides users with a convenient website to browse through, select, and purchase general household items. The system will allow users to create accounts, browse through a variety of products, add items to their cart, and complete the purchase. Users can choose to have the groceries delivered to their location. The pricing model will include product prices, and additional charges may apply for delivery services. The system will also manage supplier information for efficient inventory management.

For implementing this project I will use MySQL database to manage user accounts, product information, and order details. The system will have entities for users, products, orders, suppliers, and delivery information. The application's front end is developed using HTML, CSS and JavaScript while the backend is using Python, MySQL, and Flask.

FUNCTIONS IMPLEMENTED

1. User Actions:

    a. Log In: Users can log in using their credentials.

    b. Sign Up: New users can create accounts.

    c. Sign Out

2. Shopping:

    a. Browse Products: Users can explore available grocery items.

    b. Add to Cart: Add products to a virtual shopping cart.

    c. View Cart: Review and modify items in the cart.

3. Placing Orders:

    a. Place Order: Users can submit orders for selected items.

    b. Order History: View past order details.

4. Pricing and Payment:

    a. Product Prices: Display prices for each product.

    b. Delivery Charges: Clearly communicate additional delivery costs.

FUNCTIONS IMPLEMENTED

5. Supplier Management:

   a. View Suppliers: Admin can view supplier information.

   b. Supplier Product Catalog: View products available from each supplier.

6. Interface Design:

   a. User-Friendly Design: Intuitive and responsive for easy use.

   b. Search Feature: Users can quickly find specific products.

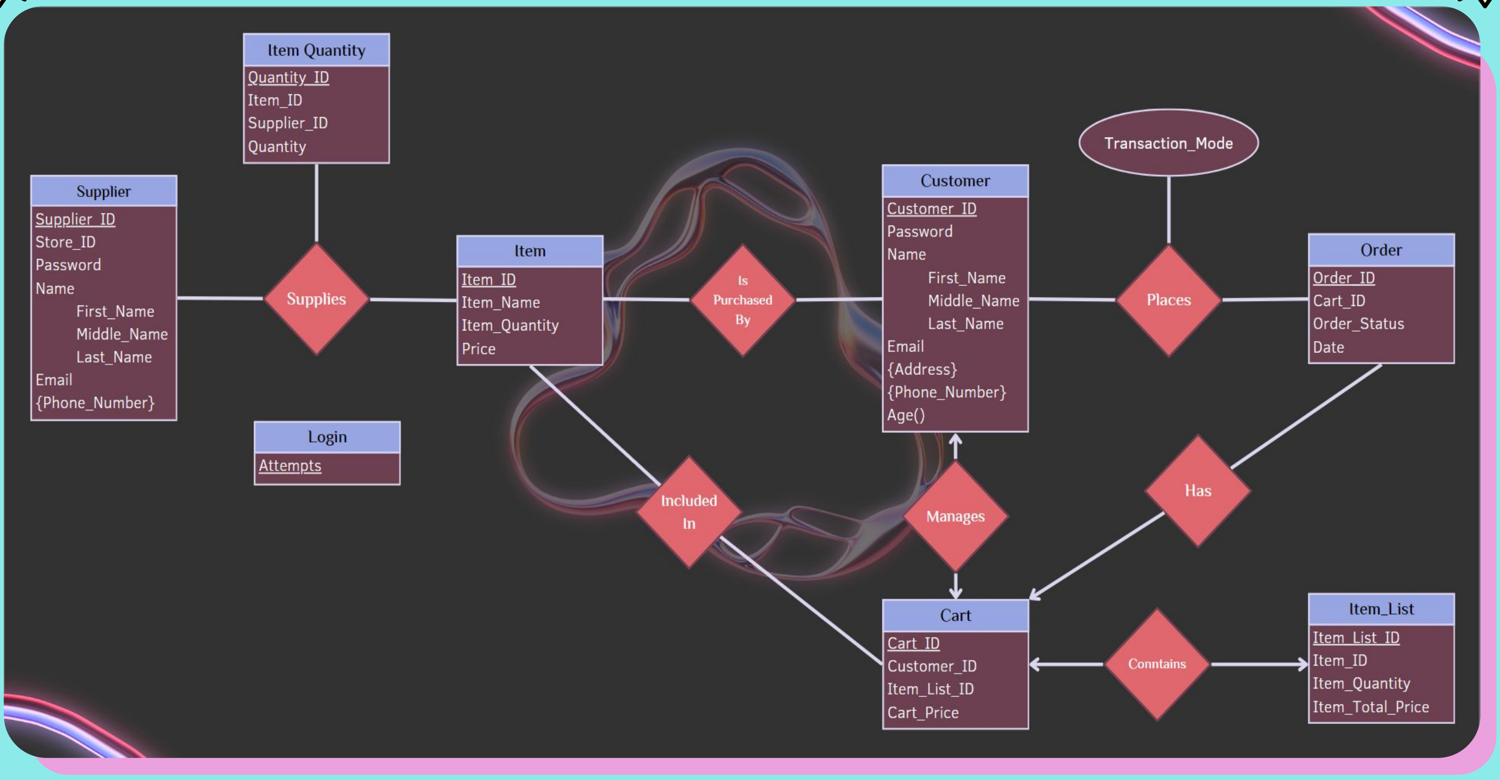
   c. Works on Various Devices: Accessible on desktops, tablets, and mobile phones.

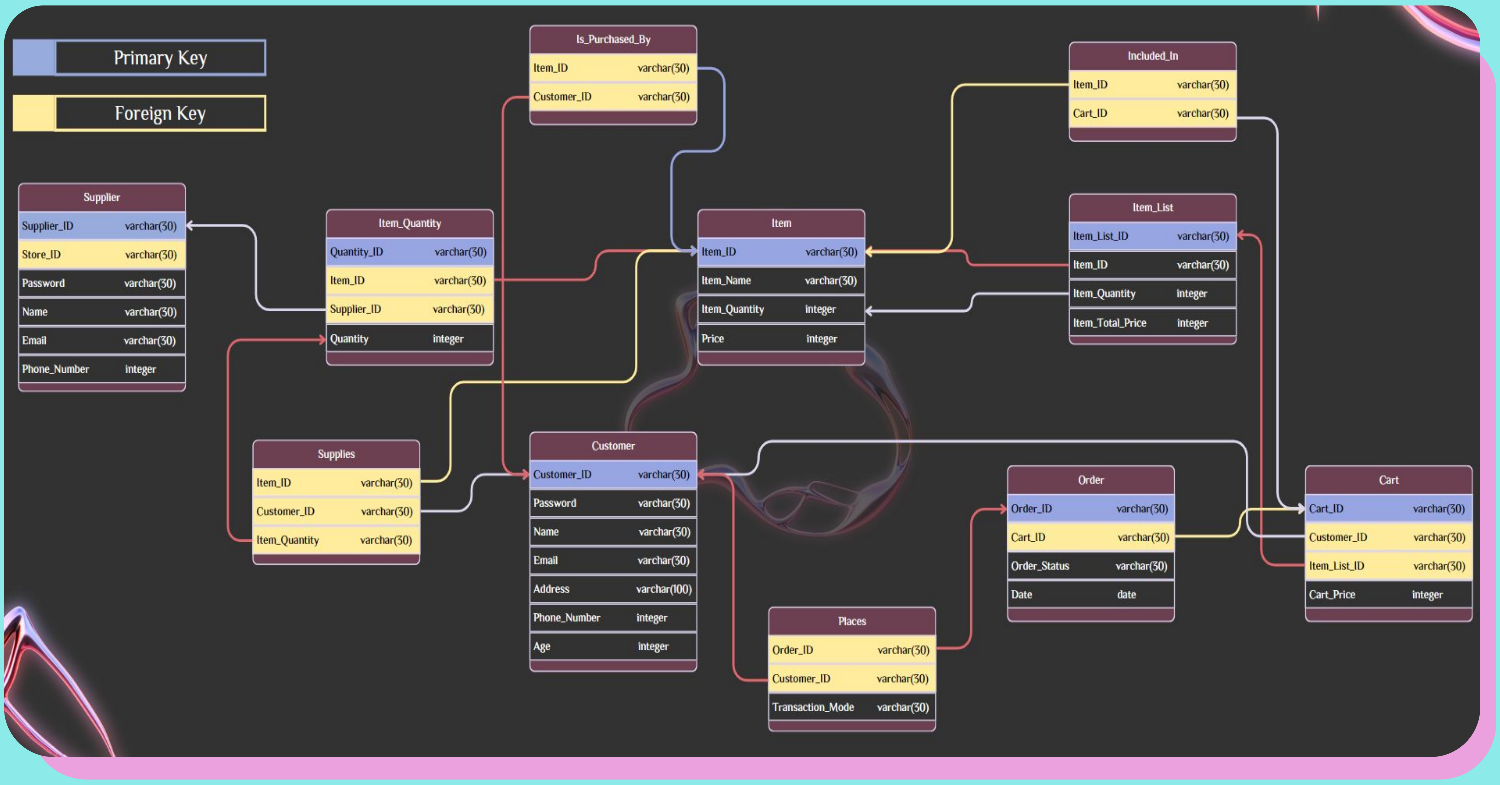
7. Scalability:

   System will handle increased users and transactions efficiently.

ER DIAGRAM

02





ER DIAGRAM & RELATIONAL MODEL

03

ENTITIES, ATTRIBUTES AND SCHEMA

ENTITIES, ATTRIBUTES AND SCHEMA

Supplier (Supplier_ID, Store_ID, Sup_Password, First_Name, Middle_Name, Last_Name, Email)

Supplier_ID	VARCHAR(10), NOT NULL, PRIMARY KEY
Store_ID	VARCHAR(10), NOT NULL
Sup_Password	VARCHAR(6), NOT NULL, UNIQUE KEY
First_Name	VARCHAR(15), NOT NULL
Middle_Name	VARCHAR(15), NOT NULL
Last_Name	VARCHAR(15), NOT NULL
Email	VARCHAR(30), NOT NULL, UNIQUE KEY

Customer (Customer_ID, Cus_Password, First_Name, Middle_Name, Last_Name, Email, Address, Age)

Customer_ID	VARCHAR(10), NOT NULL, PRIMARY KEY
Cus_Password	VARCHAR(6), NOT NULL, UNIQUE KEY
First_Name	VARCHAR(15), NOT NULL
Middle_Name	VARCHAR(15), NOT NULL
Last_Name	VARCHAR(15), NOT NULL
Email	VARCHAR(30), NOT NULL, UNIQUE KEY
Address	VARCHAR(255), NOT NULL
Age	INT, NOT NULL, CHECK (Age >= 18)

ENTITIES, ATTRIBUTES AND SCHEMA

Item (Item_ID, Item_Name, Item_Quantity, Price, Image_Address, I_Description)

Item_ID	VARCHAR(10), NOT NULL, PRIMARY KEY
Item_Name	VARCHAR(30), NOT NULL
Item_Quantity	INT, NOT NULL, UNIQUE KEY
Price	INT, NOT NULL
Image_Address	VARCHAR(30), NOT NULL
I_Description	VARCHAR(225), NOT NULL

Item_List (Item_List_ID, Item_ID, Item_Quantity, Item_Total_Price)

Item_List_ID	INT, NOT NULL, auto_increment, PRIMARY KEY
Item_ID	VARCHAR(10), NOT NULL, FOREIGN KEY REFERENCES Item
Item_Quantity	INT, NULL
Item_Total_Price	INT, NOT NULL

Item_Quantity (Item_ID, Supplier_ID)

Item_ID	VARCHAR(10), NOT NULL, FOREIGN KEY REFERENCES Item
Supplier_ID	VARCHAR(10), NOT NULL, FOREIGN KEY REFERENCES Supplier

ENTITIES, ATTRIBUTES AND SCHEMA

Supplies (Item_ID, Supplier_ID, Item_Quantity)

Item_ID	VARCHAR(10), NOT NULL, FOREIGN KEY REFERENCES Item
Supplier_ID	VARCHAR(10), NOT NULL, FOREIGN KEY REFERENCES Supplier
Item_Quantity	INT, NOT NULL

Is_Purchased_By (Item_ID, Customer_ID)

Item_ID	INT, NOT NULL, FOREIGN KEY REFERENCES Item
Customer_ID	VARCHAR(10), NOT NULL, FOREIGN KEY REFERENCES Customer

Cart (Cart_ID, Customer_ID, Item_List_ID, Cart_Price)

Cart_ID	INT, NOT NULL
Customer_ID	VARCHAR(10), NOT NULL, FOREIGN KEY REFERENCES Customer
Item_List_ID	INT, NOT NULL, FOREIGN KEY REFERENCES Item_list
Cart_Price	INT, NOT NULL

Orders (Order_ID, Cart_ID, Order_Date)

Order_ID	INT NOT NULL auto_increment, PRIMARY KEY
Cart_ID	INT, NOT NULL, FOREIGN KEY REFERENCES Cart
Order_Date	DATE

ENTITIES, ATTRIBUTES AND SCHEMA

Places (Order_ID, Customer_ID, Transaction_Mode)

Order_ID	INT, NOT NULL, FOREIGN KEY REFERENCES Order
Customer_ID	VARCHAR(10), NOT NULL, FOREIGN KEY REFERENCES Customer
Transaction_Mode	VARCHAR(30), NOT NULL

Included_IN (Item_ID, Cart_ID)

Item_ID	INT, NOT NULL, FOREIGN KEY REFERENCES Item
Cart_ID	INT, NOT NULL, FOREIGN KEY REFERENCES Cart

Supplier_Phone_Number (Supplier_ID, Phone_Number)

Supplier_ID	VARCHAR(10), NOT NULL, FOREIGN KEY REFERENCES Supplier
Phone_Number	VARCHAR(10), NOT NULL

Customer_Phone_Number (Customer_ID, Phone_Number)

Customer_ID	VARCHAR(10), NOT NULL, FOREIGN KEY REFERENCES Customer
Phone_Number	VARCHAR(10), NOT NULL

TRIGGERS AND TRANSACTIONS

04

TRIGGERS

1. Login Validation Trigger

Trigger to handle login attempts

```
DELIMITER $$
CREATE TRIGGER login_trigger
BEFORE INSERT ON login
FOR EACH ROW
BEGIN
   -- Declare a variable to hold the login message
   DECLARE login_message VARCHAR(255);
   -- Generate a login message based on the login attempts
   IF NEW.attempts = '0' THEN
      SET login_message = 'Try Different';
   ELSE
      SET login_message = CONCAT('Try Different (', NEW.attempts, ' attempts left)');
   END IF;
   -- Insert the login message into the login table
   INSERT INTO login (attempts) VALUES (login_message);
END$$
DELIMITER ;
```

2. Age Validation Trigger

Trigger to validate customer age before insertion

```
DELIMITER $$
CREATE TRIGGER before_insert_customer
BEFORE INSERT ON Customer
FOR EACH ROW
BEGIN
   -- If the age is less than 18, raise an error
   IF NEW.Age < 18 THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Age should be 18 or older';
   END IF;
END$$
DELIMITER;
```

TRANSACTIONS

NON-CONFLICTING TRANSACTIONS

1. Customer Purchase

```
-- T1: Customer adds items to cart and places order
START TRANSACTION;
INSERT INTO Cart (Customer_ID, Item_List_ID, Cart_Price) VALUES ('cus0000000', 1, 50);
INSERT INTO Orders (Cart_ID, Order_Date) VALUES (1, CURDATE());
INSERT INTO Places (Order_ID, Customer_ID, Transaction_Mode) VALUES (1, 'cus0000000',
'Online');
COMMIT;

-- T2: Deduct item quantities from inventory and update customer's purchased items
START TRANSACTION;
UPDATE Item_Quantity SET Item_Quantity = Item_Quantity - 1 WHERE Item_ID =
'itm0000001' AND Supplier_ID = 'sup0000001';
INSERT INTO Is_Purchased_By (Item_ID, Customer_ID) VALUES ('itm0000001', 'cus0000000');
COMMIT;
```

Transaction T1	Transaction T2
Read(Cart, Orders, Places)	
Write(Cart, Orders, Places)	
	Read(Item_Quantity)
Commit	
	Write(Item_Quantity, Is_Purchased_By)
	Commit

Explanation:
In Transaction T1, a customer adds items to the cart, places an order, and the order details are committed. Meanwhile, in Transaction T2, the item quantities are updated in the inventory based on the purchase made by the customer. These transactions are non-conflicting as they involve different tables and operations.

TRANSACTIONS

NON-CONFLICTING TRANSACTIONS

2. Supplier Restock

```
-- T1: Supplier updates item quantities
START TRANSACTION;
UPDATE Item_Quantity SET Item_Quantity = Item_Quantity + 50 WHERE Item_ID =
'itm0000001' AND Supplier_ID = 'sup0000001';
COMMIT;

-- T2: Update item quantity in inventory
START TRANSACTION;
UPDATE Item SET Item_Quantity = Item_Quantity + 50 WHERE Item_ID = 'itm0000001';
COMMIT;
```

Transaction T1	Transaction T2
	Read(Supplier_Phone_Number)
Read(Item_Quantity)	
Write(Item_Quantity)	
Commit	
	Write(Supplier_Phone_Number)
	Commit

Explanation:
Transaction T1 involves updating the item quantities by a supplier, committing the changes. Concurrently, Transaction T2 adds a supplier's phone number to the database. These transactions are non-conflicting as they operate on different tables and do not affect each other's data.

TRANSACTIONS

NON-CONFLICTING TRANSACTIONS

3. Customer Registration

```
-- T1: Insert new customer details
START TRANSACTION;
INSERT INTO Customer (Customer_ID, Cus_Password, First_Name, Last_Name, Email,
Address, Age)
VALUES ('C001', 'pswrd0', 'John', 'Doe', 'john@example.com', '123 Main St', 25);
COMMIT;

-- T2: Add customer's phone number
START TRANSACTION;
INSERT INTO Customer_Phone_Number (Customer_ID, Phone_Number) VALUES ('C001',
'8234567890');
COMMIT;
```

Transaction T1	Transaction T2
Read(Customer)	
Write(Customer)	
Commit	
	Read(Customer_Phone_Number)
	Write(Customer_Phone_Number)
	Commit

Explanation:
Transaction T1 inserts new customer details into the database and commits the changes. Concurrently, Transaction T2 adds the customer's phone number to the database. These transactions are non-conflicting as they operate on different tables and do not interfere with each other's data.

TRANSACTIONS

NON-CONFLICTING TRANSACTIONS

4. Supplier Registration

```
-- T1: Insert new supplier details
START TRANSACTION;
INSERT INTO Supplier (Supplier_ID, Store_ID, Sup_Password, First_Name, Last_Name,
Email)
VALUES ('sup0000000', 'str0000000', 'pas123', 'Jane', 'Smith', 'jane@example.com');
COMMIT;

-- T2: Add supplier's phone number
START TRANSACTION;
INSERT INTO Supplier_Phone_Number (Supplier_ID, Phone_Number) VALUES ('sup0000000',
'9777777710');
COMMIT;
```

Transaction T1	Transaction T2
Read(Supplier)	
Write(Supplier)	
Commit	
	Read (Supplier_Phone_Number)
	Write (Supplier_Phone_Number)
	Commit

Explanation:
Transaction T1 inserts new supplier details into the database and commits the changes. Concurrently, Transaction T2 adds the supplier's phone number to the database. These transactions are non-conflicting as they operate on different tables and do not interfere with each other's data.

TRANSACTIONS

CONFLICTING TRANSACTIONS

1. Customer Purchase Conflict

```
-- T1: Updating Cart for Customer 1
START TRANSACTION;
SELECT * FROM Cart WHERE Customer_ID = 'cus0000000' FOR UPDATE;
UPDATE Cart SET Cart_Price = 100 WHERE Customer_ID = 'cus0000000';
COMMIT;

-- T2: Updating Cart for Customer 2
START TRANSACTION;
SELECT * FROM Cart WHERE Customer_ID = 'cus0000001' FOR UPDATE;
UPDATE Cart SET Cart_Price = 150 WHERE Customer_ID = 'cus0000001';
COMMIT;
```

Transaction T1	Transaction T2
Read(Cart) {Exclusive Lock}	
	Read(Cart) {Exclusive Lock}
Write(Cart) {Exclusive Lock}	
Commit {Release Locks}	
	Write(Cart) {Exclusive Lock}
	Commit {Release Locks}

Explanation:
Transaction T1 and T2 both aim to update the cart table concurrently. T1 might be updating the cart for one customer, while T2 could be doing the same for another. However, both transactions may access and modify the same data simultaneously, potentially leading to inconsistencies or data corruption if not properly managed.
To prevent conflicts, T1 acquires an exclusive lock on the Cart table before updating it, ensuring that no other transaction can access or modify the data until T1 commits. T2, attempting to acquire its own exclusive lock, is blocked until T1 releases the lock. This mechanism ensures that only one transaction can modify the Cart table at a time, maintaining data integrity.

TRANSACTIONS

CONFLICTING TRANSACTIONS

2. Supplier Restock Conflict

```
-- T1: Restocking Items for Supplier 1
START TRANSACTION;
SELECT * FROM Item_Quantity WHERE Supplier_ID = 'sup0000001' FOR UPDATE;
UPDATE Item_Quantity SET Item_Quantity = Item_Quantity + 10 WHERE Supplier_ID =
'sup0000001';
COMMIT;
```

```
-- T2: Restocking Items for Supplier 2
START TRANSACTION;
SELECT * FROM Item_Quantity WHERE Supplier_ID = 'sup0000002' FOR UPDATE;
UPDATE Item_Quantity SET Item_Quantity = Item_Quantity + 20 WHERE Supplier_ID =
'sup0000002';
COMMIT;
```

Transaction T1	Transaction T2
Read(Item_Quantity) {Exclusive Lock}	
	Read (Item_Quantity) {Exclusive Lock}
Write(Item_Quantity) {Exclusive Lock}	
Commit {Release Locks}	
	Write (Item_Quantity) {Exclusive Lock}
	Commit {Release Locks}

Explanation:
Transactions T1 and T2 both involve updating the Item_Quantity table to restock items from different suppliers. However, as they operate concurrently, both transactions may attempt to modify the same item quantities simultaneously, posing a risk of data inconsistencies or incorrect updates.
To manage conflicts, T1 acquires an exclusive lock on the Item_Quantity table before proceeding with its update. This lock prevents T2 from accessing or modifying the data until T1 completes its operation. T2, waiting to acquire its exclusive lock, is blocked until T1 releases the lock. This locking mechanism ensures that only one transaction can modify the Item_Quantity table at a time, preserving data consistency.

USER GUIDE

05

USER GUIDE

LAUNCHING WEBSITE

- Inside 2022233_SwiftCart run the sc.sql in your MySQL.
- Then run main.py.
- At the terminal a link will appear.
- Ctrl+click on the link.
- It will redirect you to the website in your browser.

WELCOME PAGE

- There are three options.
- Login if you already have an account.
- Signup if you don't have account.
- Admin if you want to login as administrator.

ADMIN

- Login using this email and password.
  - swiftcart
  - 1234
- Click on login button.
- You will be redirected to admin page.
- Here you can see details customers, suppliers, cart and orders.

SIGNUP

- Enter all required fields.
- Age must be greater than 18.
- Middle and last name are optional.
- Then click on signup button.
- You will be redirected to login page.

USER GUIDE

LOGIN

- Enter your email address.
- Enter your password.
- Use this email and password if not Signup with new:
  - abc@123.com
  - abc123
- Then click on Login button to continue shopping.

HOME PAGE

- Here select the quantity of the items you want to order by adjusting their respective sliders.
- Then click on Add to Cart button present on top right of the website.
- Then click on Cart button on the navbar to go to cart.

CART

- Here you can see the items in your cart with respective total price and total cart price.
- Then click on Checkout button to place your order.
- Then you will be redirected to order confirmation page.

ORDER

- In this page Order Confirmation message will be displayed along with your address and estimated delivery time.
- Also a payment message will be displayed.

THANKS!

I WANT TO EXPRESS MY SINCERE GRATITUDE TO YOU FOR TAKING THE TIME TO EXPLORE MY DBMS PROJECT. YOUR INTEREST IN MY WORK MEANS A GREAT DEAL TO ME, AND I'M EXCITED TO SHARE MY PROJECT WITH YOU.

KANISHK KUMAR MEENA
2022233
Email : kanishk22233@iiitd.ac.in