STOREX

-Our Online Retail Store

Built as a course project for
CSE202: Fundamentals of Database Management Systems

Aman Tomar (2022061)

Harshil Handoo (2022206)

April 20,2024

Everything about Project

Project Scope:

The envisioned project represents a pioneering initiative in the realm of e-commerce, focusing on the creation of an advanced online retail application that redefines the shopping experience for consumers. By harnessing the power of technology, the application is designed to empower customers with the convenience of accessing an extensive range of products from the comfort of their homes. The interface of the application is carefully crafted to be user-friendly, facilitating a seamless and enjoyable browsing experience. Users can effortlessly navigate through a diverse selection of items, enabling them to make well-informed decisions.

The application's functionality extends beyond mere product exploration. Customers have the flexibility to explore available items, meticulously curate their selections, and effortlessly add desired products to virtual shopping carts. When users are ready to proceed to checkout, the application streamlines the payment process by seamlessly integrating the user's wallet as the preferred payment method. Ensuring a secure transaction, a secondary authentication step is implemented, thereby guaranteeing the deducting of the corresponding amount from the user's wallet.

Beyond the customer interface, the system is intricately designed to address the operational needs of the retail store's owner, referred to as the Admin. The Admin is equipped with an array of features, including inventory management, order fulfilment capabilities, and transaction tracking. This comprehensive system allows the Admin to monitor real-time product availability, supervise product deliveries to customers, and maintain meticulous transaction records. The utilization of essential customer information, such as names, customer IDs, and addresses, ensures a seamless and efficient product delivery process.

The product offerings within the application are intelligently categorized into various types, spanning groceries, sports items, cosmetics, and more. These categories are meticulously presented to users, fostering easy navigation and enabling users to make selections that align with their preferences. In addition to the product selection and purchase process, the application enhances the customer experience by incorporating an order tracking feature. This feature empowers customers to monitor the real-time status of their orders, providing visibility into order processing, shipment, and estimated delivery times.

Furthermore, the application adopts a customer-centric approach by implementing a robust rating and review system. Users have the opportunity to express their opinions by rating and leaving reviews for products they have purchased. This not only facilitates informed decision-making for other potential buyers but also contributes to an enriched and interactive shopping community within the application.
In addition to the existing functionalities, the envisioned project will incorporate a warehouse management system to cater to the procurement needs of the Admin. This warehouse functionality provides the Admin with the capability to efficiently manage and track the inventory of products available for sale.
Within the Admin interface, a dedicated section for warehouse management will be integrated. Here, the Admin can view the current stock levels of each product, receive alerts for low stock, and place orders to restock products as needed. The system will enable the Admin to interact with suppliers, track order statuses, and manage the inflow of products into the warehouse.

Functional Requirements:

Users must log in or create accounts using phone numbers or email addresses, ensuring authentication and database storage. Product availability is clearly displayed, and out-of-stock items are appropriately flagged. Each user has a dedicated cart, and products unavailable due to stock depletion cannot be added. Users utilize a wallet for purchases, with wallet increments subject to basic use authentication. Payments are possible only if the wallet has sufficient funds, with deducted amounts reflecting in both the user's account and product quantities.
Frequent customers have the opportunity to upgrade their membership status, unlocking various discounts and offers.

Constraints:

- Admin has full access to website functionalities.
- Admin can only add products to the store.
- Customers make purchases from the store.
- No product can exist without admin addition.

- Customers cannot exceed available product quantities in their carts.
- No product can exist without a category.
- Empty categories are not allowed.
- Customers can only purchase if they have sufficient wallet balance.
- Cart and wallet modifications require successful verification.

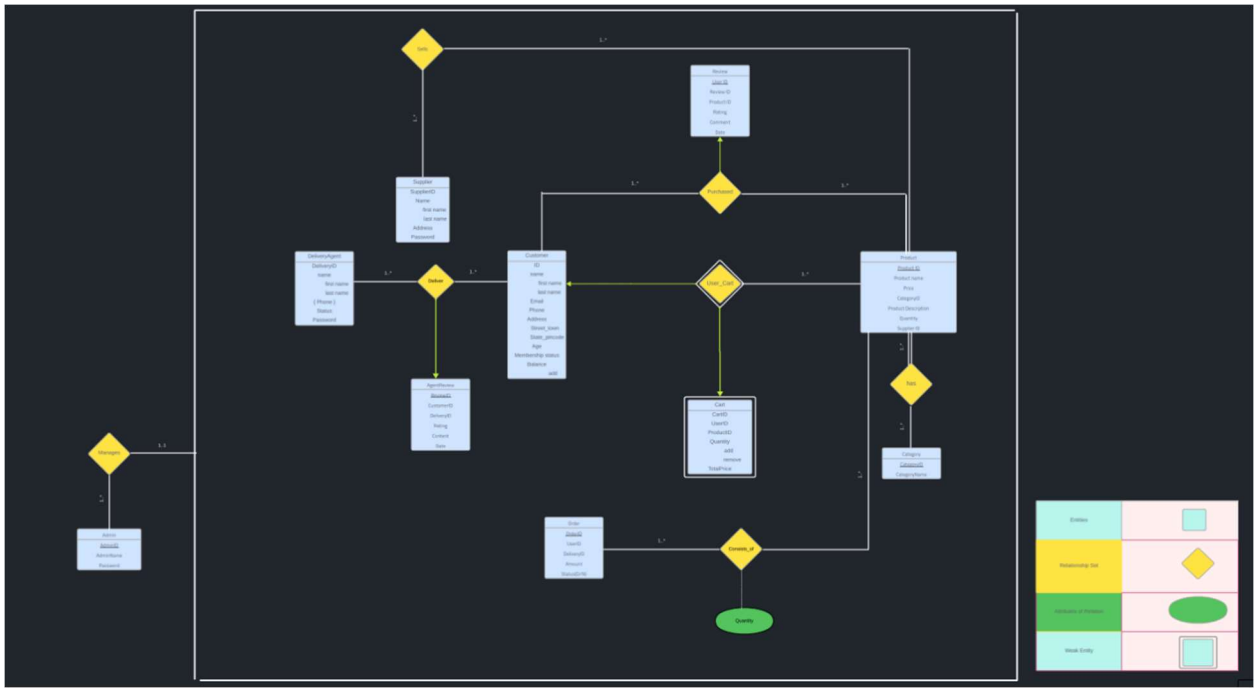
Technical Requirements:

We plan for STOREX to be a full-stack project with a backend and a frontend. According to the requirements, we plan to use the following tools and technologies:

- **MySQL Database**: MySQL will serve as the backend database management system, providing robust data storage and retrieval capabilities.
- **Python**: Python will be utilized as the primary programming language for backend development due to its versatility and extensive libraries.
- **Flask:** Flask, a lightweight and flexible micro-framework for Python, will be employed to build the backend server, facilitating rapid development and easy integration.
- **HTML:** Hypertext Markup Language (HTML) will be used for structuring the frontend user interface, enabling the creation of web pages with content and layout.
- **CSS:** Cascading Style Sheets (CSS) will complement HTML by providing styling and design elements to enhance the visual appeal and user experience of the frontend.

ER Model :

The Entity-Relationship (ER) Model acts as a blueprint for structuring databases, effectively capturing the intricate relationships between different entities and presenting them in a visually intuitive manner. It serves as a foundational tool for database design, facilitating a clear understanding of the data architecture and fostering efficient data management practices.



The ER Model stands as a cornerstone in database management, offering a structured framework to delineate intricate relationships and streamline data operations. Through its depiction of ternary relationships, such as those outlined above, it empowers businesses to efficiently manage customer interactions, optimize service delivery, and foster meaningful engagement with products and services. Ultimately, by leveraging the ER Model effectively, organizations can enhance their operational efficiency, improve customer satisfaction, and drive sustainable growth in an increasingly data-centric landscape.

Relational Model:

Some specific relations are as follows:

- Cart:
  (CustomerID - Taken from Customer Table, used as Primary Key)
- Order:
  (CustomerID - Taken from Customer Table, DeliveryID - Taken from DeliveryAgent Table, both used as Foreign Keys)
- Review:
  (CustomerID - Taken from Customer Table, ProductID - Taken from Product Table, both used as Foreign Keys)
- AgentReview:
  (CustomerID - Taken from Customer Table, DeliveryID - Taken from DeliveryAgent Table, both used as Foreign Keys)
- Sells:
  (SupplierID - Taken from Supplier Table, ProductID - Taken from Product Table, both combined to form the primary key for the table Sells)

The Relation Diagram

The Database Schema:

Tables Added or Renamed:

- Customer
- DeliveryAgent
- Supplier
- Category
- Product
- Cart
- Sells
- MakeOrder
- ProductReview
- AgentReview
- SystemAdmin

Handling Different Attributes:

Composite Attributes:

- Address (Customer, DeliveryAgent, Supplier)
- State_Pincode (Customer, DeliveryAgent, Supplier)

Multi-Valued Attributes:

None in the current schema.

Assumptions:

- Each entity (Customer, DeliveryAgent, Supplier, SystemAdmin) has a unique identifier (ID).
- Customers are uniquely identified by their email addresses.
- Products belong to specific categories and are provided by individual suppliers.
- Orders are associated with specific delivery agents for fulfillment.
- Each review, whether for a product or a delivery agent, is uniquely identified within its respective table.
- Each order may contain one or more products, with quantities specified in the MakeOrder table.

Data Generation & Population:

Data Generation:

Data was generated to populate the database tables. The generated data included:

- Randomly generated names, email addresses, phone numbers, and addresses for customers, delivery agents, suppliers, and administrators.
- Randomly assigned membership statuses and balances for customers.
- Randomly generated product names, prices, descriptions, quantities, and prices per product for products.
- Randomly assigned order statuses and dates for orders.
- Randomly generated review ratings and comments for product and delivery agent reviews.

Data Population:

All tables in the database were pre-populated with data to ensure integrity constraints were maintained for querying. The database was populated with the following number of rows of data:

- Customer: 10 rows
- DeliveryAgent: 10 rows
- Supplier: 10 rows
- Category: 10 rows
- Product: 10 rows
- Cart: 10 rows
- Sells: 10 rows
- MakeOrder: 10 rows
- ProductReview: 10 rows
- AgentReview: 10 rows
- SystemAdmin: 5 rows

Importance and Features of the Database Schema:

Our database schema serves as the backbone of our application, providing a structured framework for storing and managing data efficiently. With its well-defined tables and relationships, the schema ensures data integrity and consistency, enabling seamless operations across various aspects of our application.

Key features of our schema include:

- Comprehensive Data Organization: The schema organizes data into logical entities such as customers, products, orders, and reviews, facilitating easy retrieval and manipulation of information.
- Flexibility and Scalability: Designed with scalability in mind, the schema accommodates growth and expansion of our application by supporting the addition of new tables, attributes, and relationships as needed.
- Data Integrity Enforcement: Through the use of constraints, foreign key relationships, and triggers, the schema maintains data integrity, preventing inconsistencies and ensuring the reliability of our application.
- Enhanced User Experience: By organizing data logically and efficiently, the schema contributes to a smooth and intuitive user experience, allowing users to interact with our application seamlessly and effortlessly.
- In summary, our database schema plays a crucial role in the functionality, reliability, and performance of our application, providing a solid foundation for delivering a superior user experience and supporting future growth and development.

SQL QUERIES

We have classified queries based on use cases on the tables.

1. Show the average Rating of All products selled

```
-----------------------------------------------------------------
-----------------------------------------
SELECT ProductName, AVG(rating)
FROM product
JOIN MakeOrder ON Product.ProductID = MakeOrder.ProductID
JOIN ProductReview ON MakeOrder.productID =
ProductReview.productID
GROUP BY ProductName;
-----------------------------------------------------------------
-----------------------------------------
```

2. Showing the Product History of the Customer

```
-----------------------------------------------------------------
-----------------------------------------
SELECT
    MakeOrder.orderID, MakeOrder.OrderDate,
Makeorder.DeliveryDate,
    Product.ProductName, MakeOrder.Quantity, Product.Price,
(Makeorder.Quantity * Product.Price) AS total_price,
CASE
      WHEN (MakeOrder.Deliverydate IS NULL) = True THEN 'Not
Delivered'
      ELSE 'Delivered'
END AS delivered
FROM
    MakeOrder
INNER JOIN
    Product ON MakeOrder.ProductID = Product.ProductID and
MakeOrder.OrderStatus = 'Done'
WHERE
    MakeOrder.CustomerID = 11
ORDER BY
```

```
    Makeorder.OrderDate;
-----------------------------------------------------------------
-----------------------------------------
```

3. Total Sales for the Given Day

```
-----------------------------------------------------------------
-----------------------------------------
SELECT SUM(sub.total_price) AS total_sum
FROM (
    SELECT p.Price * m.Quantity AS total_price
    FROM Product p
    INNER JOIN MakeOrder m ON p.ProductID = m.ProductID and
m.OrderDate = '2024-03-01'
) AS sub;
-----------------------------------------------------------------
-----------------------------------------
```

4. Reduce Quantity of Product when Product is Added into cart for the Given Customer

```
-----------------------------------------------------------------
-----------------------------------------
UPDATE product p INNER JOIN cart c ON p.productID = c.productID
SET p.quantity = p.quantity - c.quantity
WHERE c.customerID = 50;
-----------------------------------------------------------------
-----------------------------------------
```

5.Total Price of the cart Items of the given Customer

```
-----------------------------------------------------------------
-----------------------------------------
SELECT SUM(TotalPrice) AS TotalOrderPrice
FROM Cart
WHERE CustomerID = 11;
-----------------------------------------------------------------
-----------------------------------------
```

6. This is for order's by GOLD member's

```
-----------------------------------------------------------------
-----------------------------------------
Select
    mo.OrderID,
    mo.OrderDate
FROM
    MakeOrder mo
JOIN
    Customer c ON mo.CustomerID = c.CustomerID
WHERE
    c.MembershipStatus = 'Gold'
    AND mo.OrderStatus = 'Done';

-----------------------------------------------------------------
-----------------------------------------
```

7. Select Top 3 Customer Based on Purchase Amount

```
-----------------------------------------------------------------
-----------------------------------------
SELECT
    mo.CustomerID,
    c.FirstName,
    c.LastName,
    SUM(p.Price * mo.Quantity) AS TotalPurchaseAmount
FROM
    MakeOrder mo
JOIN
    Customer c ON mo.CustomerID = c.CustomerID
JOIN
    Product p ON mo.ProductID = p.ProductID
WHERE
    mo.OrderStatus = 'Done'
GROUP BY
    mo.CustomerID,
    c.FirstName,
    c.LastName
ORDER BY
    TotalPurchaseAmount DESC
LIMIT 3;
-----------------------------------------------------------------
-----------------------------------------
```

8. Product for a certain order having rating above and equal to 3

```
-------------------------------------------------------------------
-----------------------------------------
SELECT
    p.ProductID, p.OrderID, r.Rating
FROM
    MakeOrder p
INNER JOIN
    ProductReview r
ON
    p.ProductID = r.productID and r.Rating >= 3;
-------------------------------------------------------------------
-----------------------------------------
```

9. Arranging all product of Some Specific category in descending order of rating, using like operator for searching a specific category.

```
-------------------------------------------------------------------
-----------------------------------------
SELECT
   ProductName, rating
FROM
   product
JOIN
   Category ON Category.CategoryID = Product.CategoryID
JOIN
   ProductReview ON product.productID = ProductReview.productID
WHERE
   ProductName LIKE 'Spices%'
Order By
   rating DESC;
-------------------------------------------------------------------
-----------------------------------------
```

10. Total profit per day(for the App store, This happens as from supplier we get product at less rate and sells them at higher rate)

```
-------------------------------------------------------------------
-----------------------------------------
```

```
SELECT
    SUM((p.Price - s.PricePerProduct) * m.Quantity) AS
TotalProfit
FROM
    MakeOrder m
INNER JOIN
    Product p ON m.ProductID = p.ProductID
INNER JOIN
    Sells s ON p.SupplierID = s.SupplierID AND p.ProductID =
s.ProductID
WHERE
    m.OrderDate = '2024-03-01';
```

---

11. Query to show all undelivered orders for a given customer.

```
SELECT
    mo.OrderID,
    mo.OrderDate AS OrderPlacedDate
FROM
    MakeOrder mo
WHERE
    mo.OrderStatus = 'Done' AND mo.DeliveryDate IS NULL
    AND mo.CustomerID = 18;
```

---

12. changing balance of Customer when Item is oredered(assuming customer has enough balance to buy certain product)

```
UPDATE Customer c
SET c.Balance = c.Balance - (
    SELECT (m.Quantity * p.Price) from Product p INNER join
MakeOrder m ON m.ProductID = p.ProductID and c.CustomerID =
m.CustomerID
) Where c.CustomerID > 0; -- this Where clause can be removed
```

---

13. This query adds more quantity of products for an existing product.

```
UPDATE Product SET Quantity = Quantity + 100 WHERE productID =
11;
```

---

14. This query deletes a product from a customer's cart.

```
DELETE FROM cart WHERE customerID = 11 AND productID = 11;
```

---

15. Add New Supplier.

```
INSERT INTO Supplier (SupplierID, SupplierName, Address,
State_Pincode, Hash_Password)
VALUES (21, '[supplier_name]', '[address]', 'DL 23838',
'[hashed_password]');
```

---

16. Search for Suppliers in a Specific State.

```
SELECT * FROM Supplier WHERE State_Pincode LIKE 'DL%'; --
Example: for Delhi
```

---

17. Count the Number of Suppliers.

```
SELECT COUNT(*) AS SupplierCount FROM Supplier;
```

---

18. Appoint Other Admins: Insert a new system administrator into the SystemAdmin table

```
INSERT INTO SystemAdmin (AdminId, FirstName, LastName,
Hash_Password)
VALUES (9, 'New', 'Admin', '[hashed_password]');
```

Triggers:

Triggers in SQL are special stored procedures that are automatically executed or fired when certain events occur in a database. These events can include data manipulation operations such as INSERT, UPDATE, or DELETE on specific tables. Triggers are useful for enforcing data integrity rules, implementing complex business logic, or auditing changes to data.

Trigger: ReduceBalance
Description: This trigger reduces the balance of a customer when a new order is made.

```
DROP TRIGGER IF EXISTS ReduceBalance;
DELIMITER $$
CREATE TRIGGER ReduceBalance AFTER INSERT ON MakeOrder
FOR EACH ROW
BEGIN
  UPDATE Customer SET Balance = Balance - (SELECT Price FROM Product WHERE
ProductID = NEW.ProductID) WHERE CustomerID = NEW.CustomerID;
END;
$$
DELIMITER ;
```

Trigger: Delivery
Description: This trigger updates the availability status of a delivery agent when a new order is made.

```
DROP TRIGGER IF EXISTS Delivery;
DELIMITER $$
CREATE TRIGGER Delivery AFTER INSERT ON MakeOrder
FOR EACH ROW
BEGIN
  UPDATE DeliveryAgent SET Availability = 1 WHERE DeliveryID = NEW.DeliveryID;
END;
$$
DELIMITER ;
```

Trigger Usage in Schema and Application:

In our schema and application, the two triggers serve specific purposes that enhance functionality and ensure smooth operation. Here's how they contribute:

Trigger: ReduceBalanceDescription:

- Maintaining Balance Integrity: By deducting the cost of purchased items from the customer's balance, this trigger ensures that the customer's balance accurately reflects their financial status after each transaction.
- Streamlined Transaction Processing: Customers don't need to manually update their balances after placing an order.

Trigger: DeliveryDescription:

- Real-Time Availability Management: By automatically marking a delivery agent as available when a new order is made, this trigger ensures that available agents can be assigned to fulfil orders promptly.
- Efficient Order Fulfillment: Ensuring that available agents are promptly assigned to new orders improves the efficiency of order delivery, leading to faster delivery times and improved customer satisfaction.

Overall, these triggers enhance the functionality and efficiency of our application by automating key processes, maintaining data integrity, and streamlining order fulfillment operations.

Non – conflicting transaction:

Non-conflicting transactions are those that work on different data items and/or the same data item, but one of them is a read operation.

Here, We are showing all 4 non – conflicting transactions in this schedule.

- T1 – Register Customer (one customer registering mainly making an entry in the customer table)
- T2 – Add product Supplier (Supplier adding product into an existing product, mainly increasing quantity of product)
- T3 – Customer Buying a Product (one Customer buying product, mainly adding product to order table and reading of balance if there is enough balance and then deduction in customer balance in customer table)
- T4 – customer adding money to their balance (one customer adding balance, mainly adding balance to their account and writing to the customer table balance field).

T1	T2	T3	T4
	Read (Product - Quantity)	Read (Balance)	Read (Balance)
Write (Details)	Product + X	Read (Product – Quantity)	Balance + x
	Write (Product + x)	Quantity - 1	Write (Balance)
Commit	Commit	Write (Quantity)	Commit
		Balance - X	
		Write (Balance)	
		Commit	

Here if one of the transaction aborts there will be no problem to other transaction it can safely rollback without changing consistency of database.

Conflicting transactions:

1. Consider the situation when two Customers buy the same product (these are conflicting as both are transactions on same data items and one of them is read)

T1	T2
Read (Product – Quantity)	Read (Product - Quantity)
Reduce quantity (Quantity – 1)	Reduce Quantity (Quantity – 1)
Write (Quantity)	Write (Quantity)
Read Balance	Read Balance
Reduce Balance	Reduce Balance
Commit	Commit

This schedule is Conflict Serializable in fact.

T1	T2
Read (Product – Quantity)	
Reduce quantity (Quantity – 1)	
Write (Quantity)	
	Read (Product - Quantity)
	Reduce Quantity (Quantity – 1)
	Write (Quantity)
Read Balance	
Reduce Balance	
Commit	Read Balance
	Reduce Balance
	Commit

Conflicting Transactions:

Supplier adding product quantity of product into the store and Customer buying them at the same time this can form a set of conflicting transaction as like customer reads x as the product quantity and decreased them but at the same time supplier also reads x and increased it by some Y(say) now product quantity in database in (x + Y) but customer has buyed one so the database is not consistent.
Initial Product Quantity – X
Initial Balance of Customer – Z
Supplier adding product quantity - Y

T1	T2
Read(X)	Read(X)
X = X + Y	X = X - 1
Write(X)	Write(X)
Commit	Read(Z)
	Z = Z - A
	Write(Z)
	Commit

This can be Conflict Serializable to:

T1	T2
Read(X)	
X = X + Y	
Write(X)	
Commit	Read(X)
	X = X – 1
	Write(X)
	Read(Z)
	Z = Z - A
	Write(Z)
	Commit

USER GUIDE

How to use STOREX website and our database.

Navigation through the application

The front end of the application is designed to be as intuitive as possible. Although there remain some minor bugs and unimplemented features, the application is fully functional and can be used by all stakeholders. The UI is very simple and easy to understand, and any user can navigate through the application with ease.

Usage as Customers :

Without creating an account: The application can be used without creating an account, but the user will not be able to access the full functionality of the application. Without creating an account, the user can view limited data. They can also browse through the product catalogue.

Creating an account:

To log into the system, the user can follow one of two steps:

1. Click on the Log-in button located at the top right of the navigation bar. From there, the user can log into an existing account.
2. Click on Register on the top left of the navigation bar and enter their details such as name, city, email, password, etc. Once they are registered, they can log in as a customer as per step 1.

Logging in:

After logging in, the user can add products to their cart. They can use the Cart button on the navigation bar to proceed to the checkout page. They can review their order before placing it, and if confirmed, they are redirected to the payment gateway and the order will be placed. On the profile page, the customer can view their personal information5, their order history, and the currently active (undelivered) orders and after placing orders they can give rating to the product and after delivering of product they can give rating to delivery agent as well.

Usage as Admins :

Logging In

The admin can log in to the admin panel by clicking on the Log -in as Admin button shown in the footer. This button is displayed on almost all pages in the footer, even on the user-login page. The admin can view various things like top customers, top products, sale in a day, list of delivery agents and suppliers.

How to use

Once logged in, the admin can use the admin panel to perform various tasks, such as viewing the stocks through the catalogue, viewing orders, order statistics, and viewing the sales statistics.

Usage as Suppliers and Delivery Agents :

Both suppliers and delivery agents have similar interfaces to users for logging in. They can log in by clicking on the Log In button located at the top right of the navigation bar. From there, they can perform the tasks specific to their role.

Supplier’s;

Register And Login:

Supplier can log into the system and can register simply using register and choosing supplier as the option, and then they can login using login button.

Supplier History:

After that on the home page for supplier can see how much they have supplied and how much profit they have made.

Adding product into the store:

Also they can add product using product Id which they have access internally and can fix the price for that.

Delivery Agent:

Register And Login:

Delivery Agent can register using register button and login using login button and selecting respective choice as delivery agent.

Active Orders:

They can see active orders they are placing, and can make them available by placing them using button.

Delivered items:

They can also see on the same page all delivered products.