DBMS Mini Project Report

Mobile Store Management System

K J Prajwal Rai

PES1UG21CS254

Krishna Modala

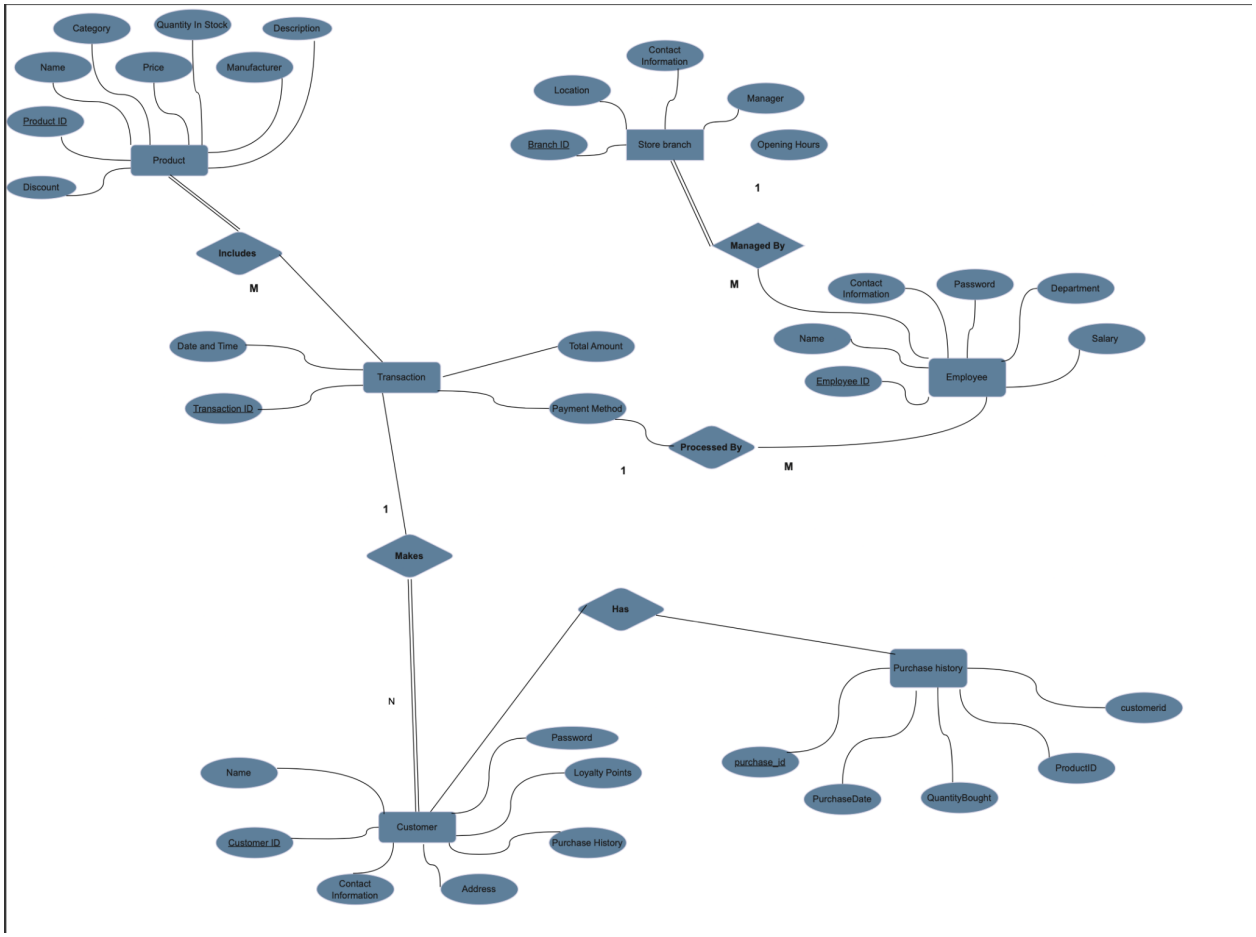
PES1UG21CS288

Table of contents:

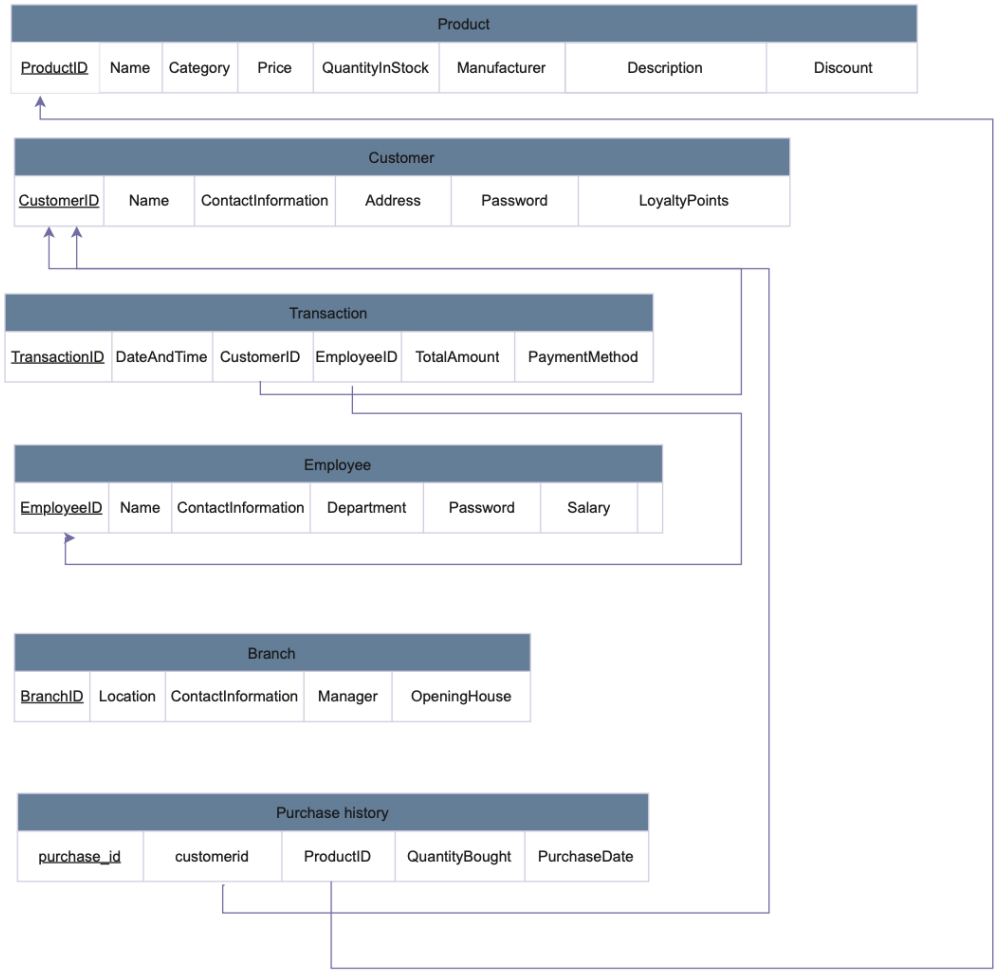
ABSTRACT

This project involves the design and implementation of a comprehensive database management system for a retail business. The system includes tables for customers, employees, products, store branches, and transactions, fostering seamless interaction between various entities. Key features include product management, purchase history tracking, and transaction recording. The project leverages MySQL and Python's Streamlit library for the backend and frontend, respectively. Through a user-friendly interface, customers can explore products, view purchase history, manage a shopping cart, and make transactions. The system also ensures data integrity and security. The report details the database structure, implementation, and frontend design, showcasing the successful integration of relational database concepts into a practical retail management solution.

ER Diagram:


Category
Quantity In Stock
Description
Name
Price
Manufacturer
Product ID
Product
Discount
Contact Information
Location
Manager
Branch ID
Store branch
Opening Hours
1
Includes
M
Managed By
M
Contact Information
Password
Department
Name
Salary
Employee ID
Employee
Date and Time
Total Amount
Transaction
Transaction ID
Payment Method
Processed By
1
M
1
Makes
Has
Purchase history
customerid
N
Password
Name
Loyalty Points
purchase_id
ProductID
PurchaseDate
QuantityBought
Customer
Customer ID
Purchase History
Contact Information
Address

Relational Schema:



DDL SQL Commands:

CUSTOMER:

```
DROP TABLE IF EXISTS `customer`;
CREATE TABLE `customer` (
 `customerid` varchar(15) NOT NULL,
 `Name` varchar(255) NOT NULL,
 `ContactInformation` varchar(50) DEFAULT NULL,
 `Address` varchar(255) DEFAULT NULL,
 `LoyaltyPoints` int DEFAULT '0',
 `Password` varchar(20) DEFAULT NULL,
 PRIMARY KEY (`customerid`),
 CONSTRAINT `customer_chk_1` CHECK ((`LoyaltyPoints` >= 0))
)
```

EMPLOYEE:

```
DROP TABLE IF EXISTS `employee`;
CREATE TABLE `employee` (
 `employeeid` varchar(15) NOT NULL,
 `Name` varchar(255) NOT NULL,
 `ContactInformation` varchar(50) DEFAULT NULL,
 `Department` varchar(50) NOT NULL,
 `Salary` decimal(10,2) NOT NULL,
 `Password` varchar(20) DEFAULT NULL,
 PRIMARY KEY (`employeeid`),
 CONSTRAINT `employee_chk_1` CHECK ((`Salary` >= 0))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

PRODUCT:

```
DROP TABLE IF EXISTS `product`;
CREATE TABLE `product` (
 `ProductID` varchar(10) NOT NULL,
 `Name` varchar(255) NOT NULL,
 `Category` varchar(50) NOT NULL,
 `Price` decimal(10,2) NOT NULL,
 `QuantityInStock` int NOT NULL,
 `Manufacturer` varchar(50) NOT NULL,
 `Description` text,
 `Discount` decimal(10,2) NOT NULL DEFAULT '0.00',
 PRIMARY KEY (`ProductID`),
 UNIQUE KEY `UQ_Product_Name` (`Name`),
 CONSTRAINT `product_chk_1` CHECK ((`Price` >= 0)),
 CONSTRAINT `product_chk_2` CHECK ((`QuantityInStock` >= 0))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

STORE BRANCH:

```
DROP TABLE IF EXISTS `storebranch`;
CREATE TABLE `storebranch` (
 `storebranchid` varchar(50) NOT NULL,
 `Location` varchar(255) NOT NULL,
 `ContactInformation` varchar(50) DEFAULT NULL,
 `Manager` varchar(255) NOT NULL,
 `OpeningHours` varchar(100) NOT NULL,
 PRIMARY KEY (`storebranchid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

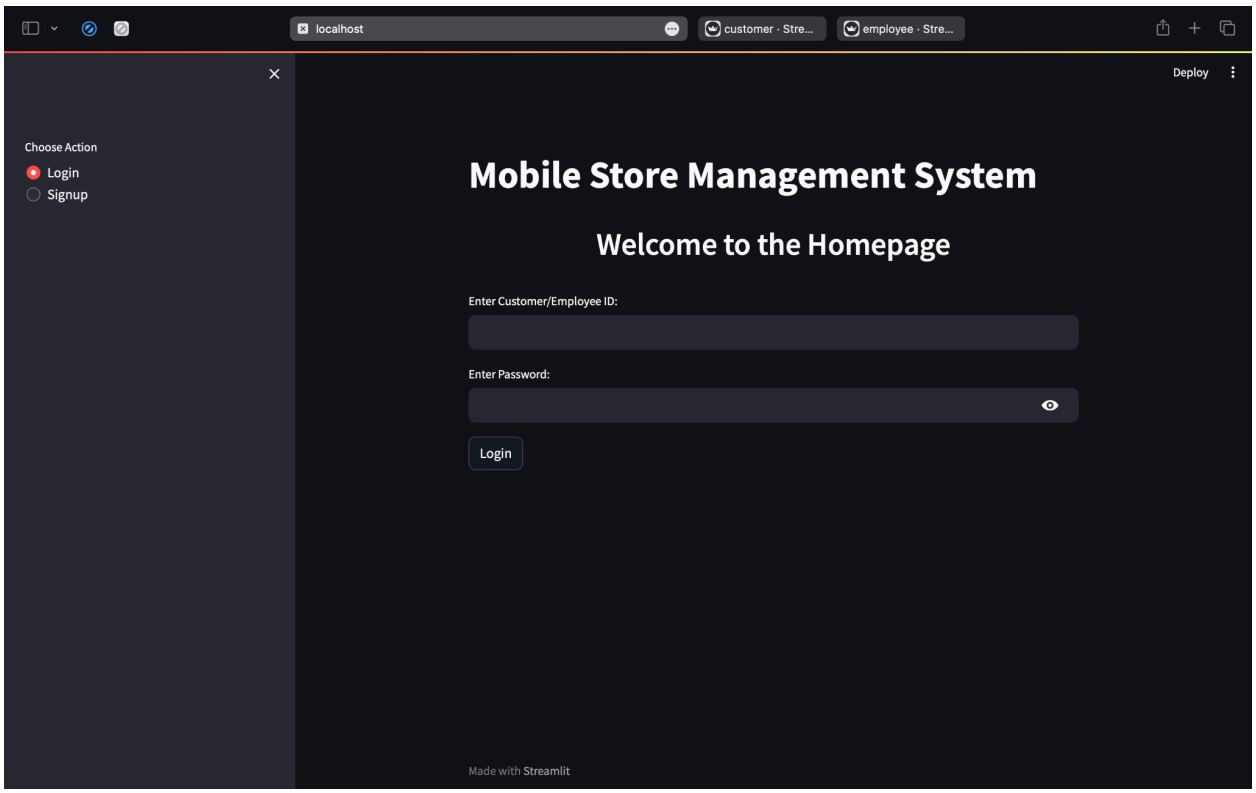
TRANSACTION:

```
DROP TABLE IF EXISTS `transaction`;
CREATE TABLE `transaction` (
 `TransactionID` int NOT NULL,
 `DateAndTime` datetime NOT NULL,
 `customerid` varchar(15) DEFAULT NULL,
 `employeeid` varchar(15) DEFAULT NULL,
 `TotalAmount` decimal(10,2) NOT NULL,
 `PaymentMethod` varchar(50) NOT NULL,
 PRIMARY KEY (`TransactionID`),
 CONSTRAINT `transaction_chk_1` CHECK ((`TotalAmount` >= 0))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
DROP TABLE IF EXISTS `purchase_history`;
CREATE TABLE `purchase_history` (
  `purchase_id` int NOT NULL AUTO_INCREMENT,
  `customerid` varchar(15) NOT NULL,
  `ProductID` varchar(10) NOT NULL,
  `QuantityBought` int NOT NULL,
  `PurchaseDate` datetime NOT NULL,
  PRIMARY KEY (`purchase_id`),
  KEY `fk_purchase_history_customer` (`customerid`),
  KEY `fk_purchase_history_product` (`ProductID`),
  CONSTRAINT `fk_purchase_history_customer` FOREIGN KEY (`customerid`)
REFERENCES `customer` (`customerid`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fk_purchase_history_product` FOREIGN KEY (`ProductID`)
REFERENCES `product` (`ProductID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

CRUD operations Screenshots:

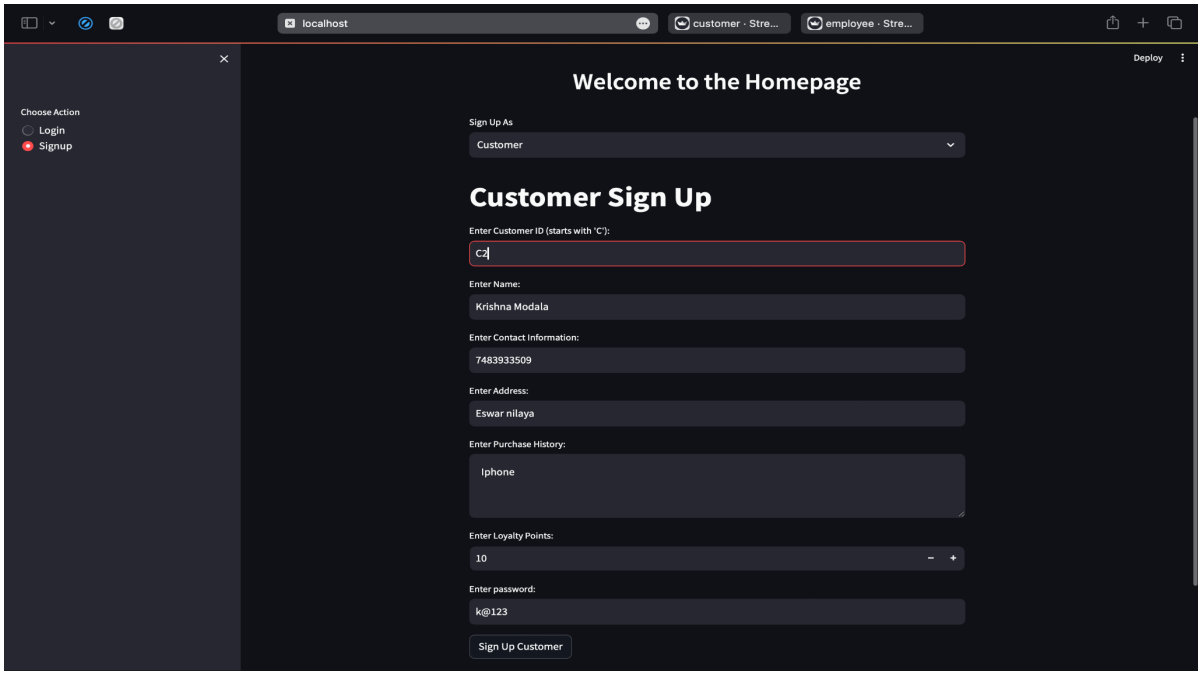
Login page:



Sign up page:

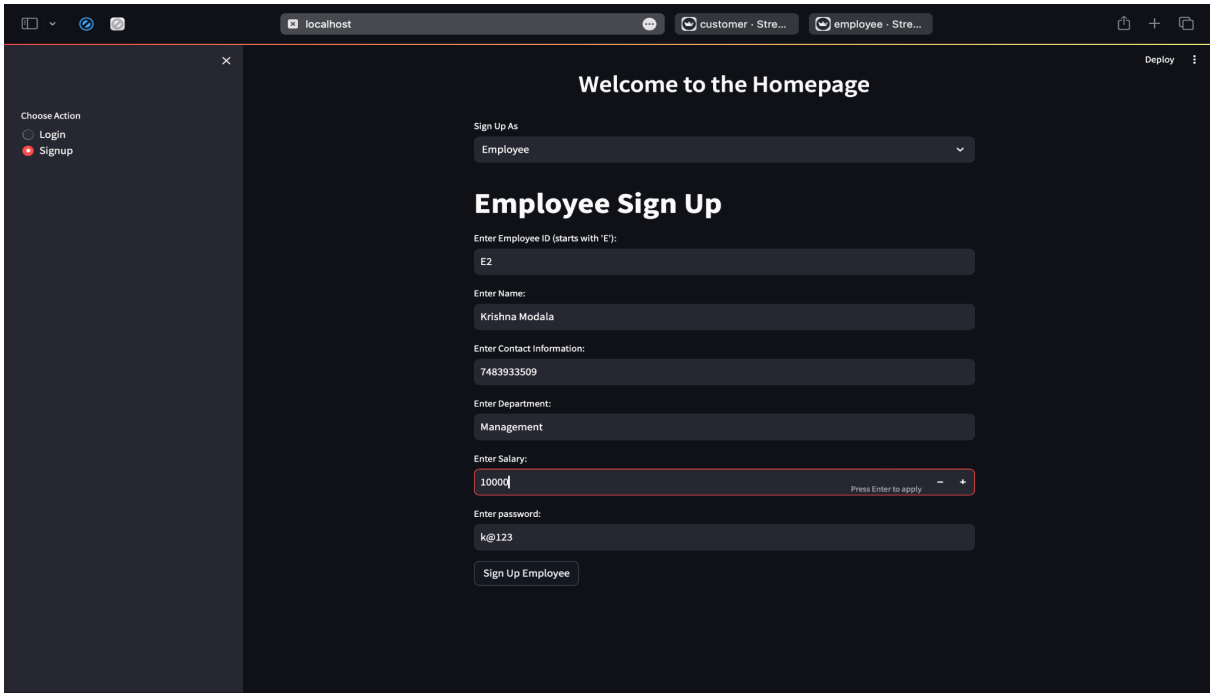
Signing up as customer:

When registering as a customer, the customer ID should commence with the letter 'C'.



Signing up as Employee:

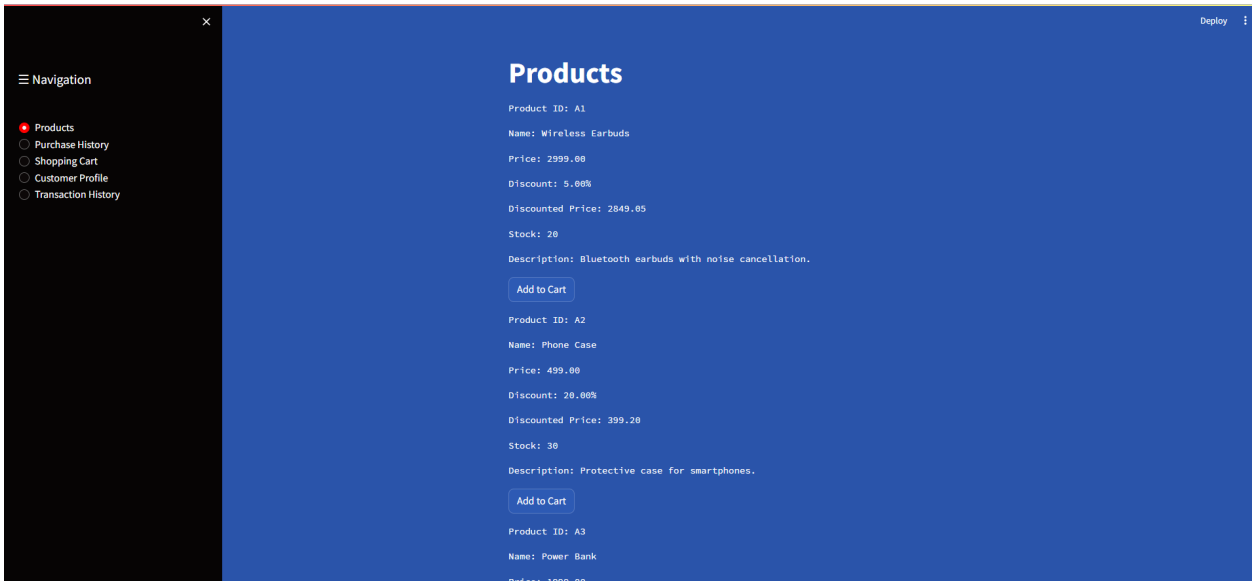
When enrolling as an employee, the employee ID should begin with the letter 'E'. There are three departments: Sales, Finance, and Management, each with distinct features.



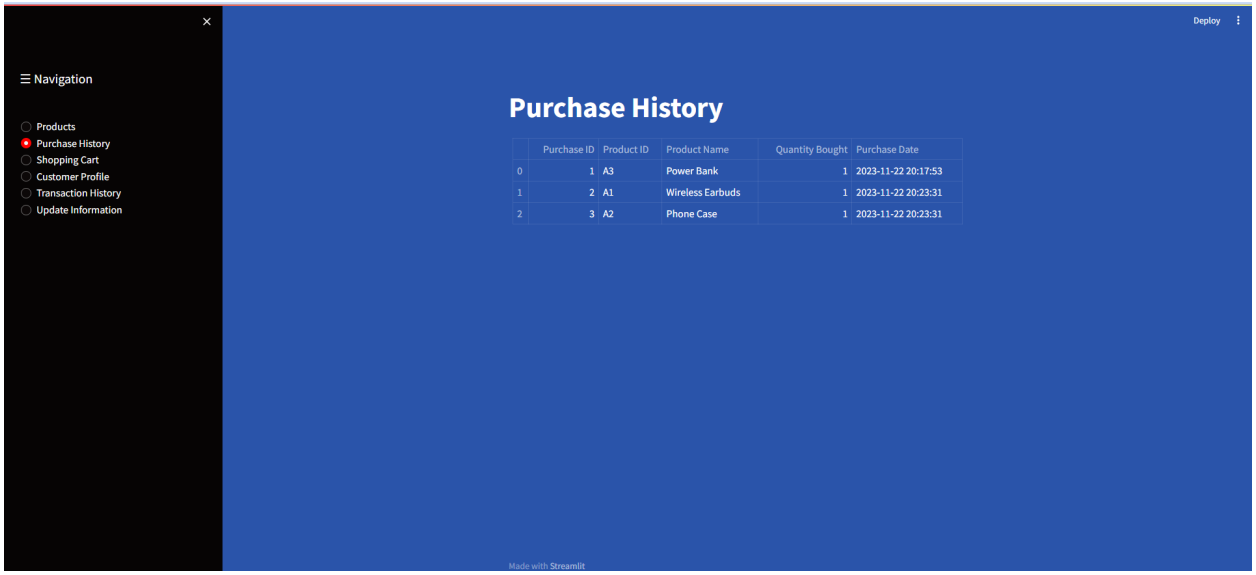
CUSTOMER Window:

The customer interface comprises products, purchase history, shopping cart, customer profile, and transaction history. Details of each feature are outlined below.

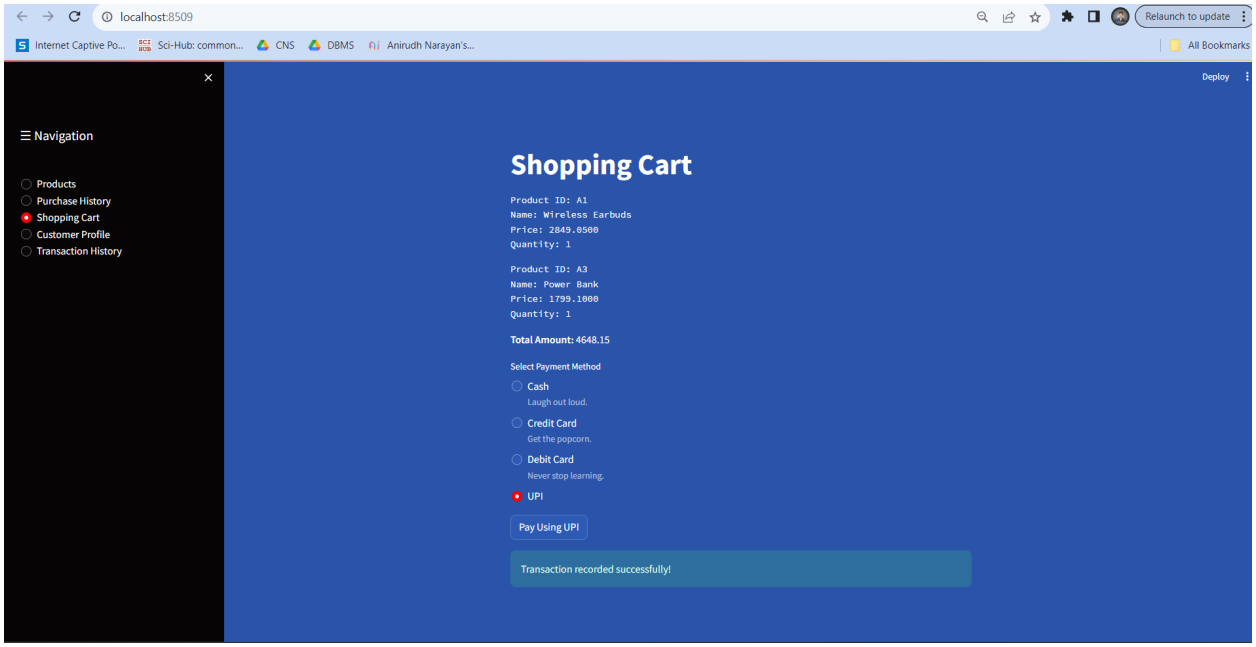
Products Tab: This tab showcases all available products along with their respective details. Customers have the option to add their preferred products to the shopping cart.



Purchase History Tab: This section displays the complete purchase history of the customer.

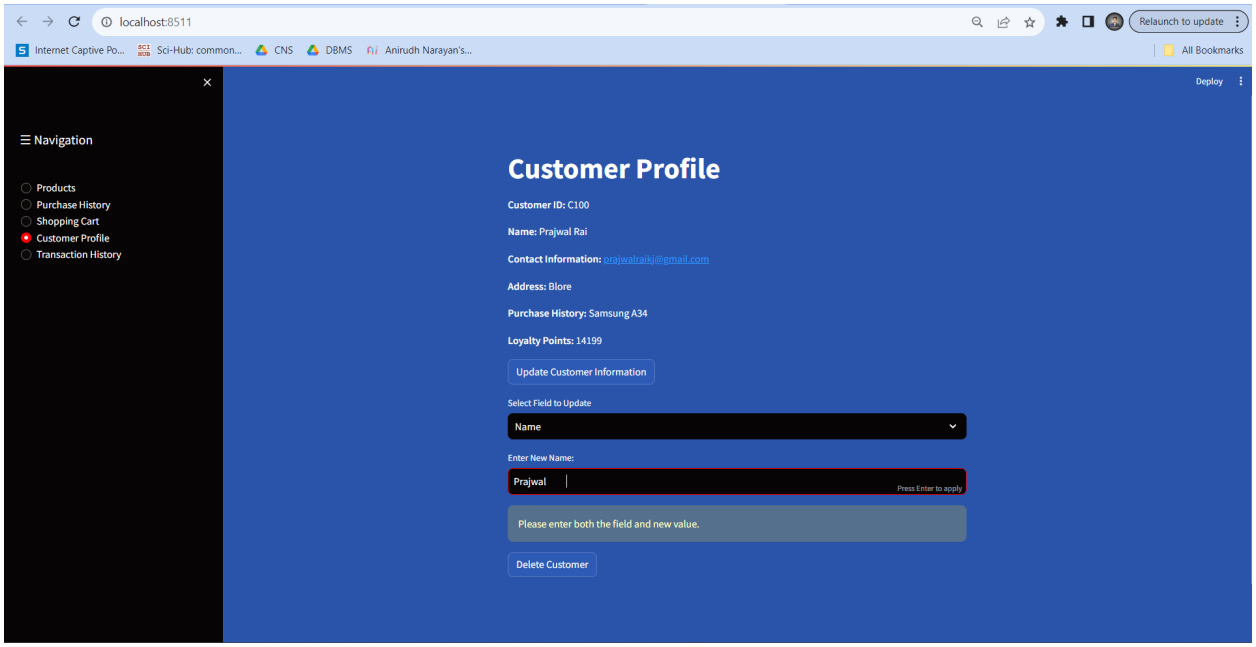


<u>Shopping Cart:</u> This section allows customers to view all the products added in the Products Tab. Various payment methods are accepted, and the total amount is displayed for checkout.

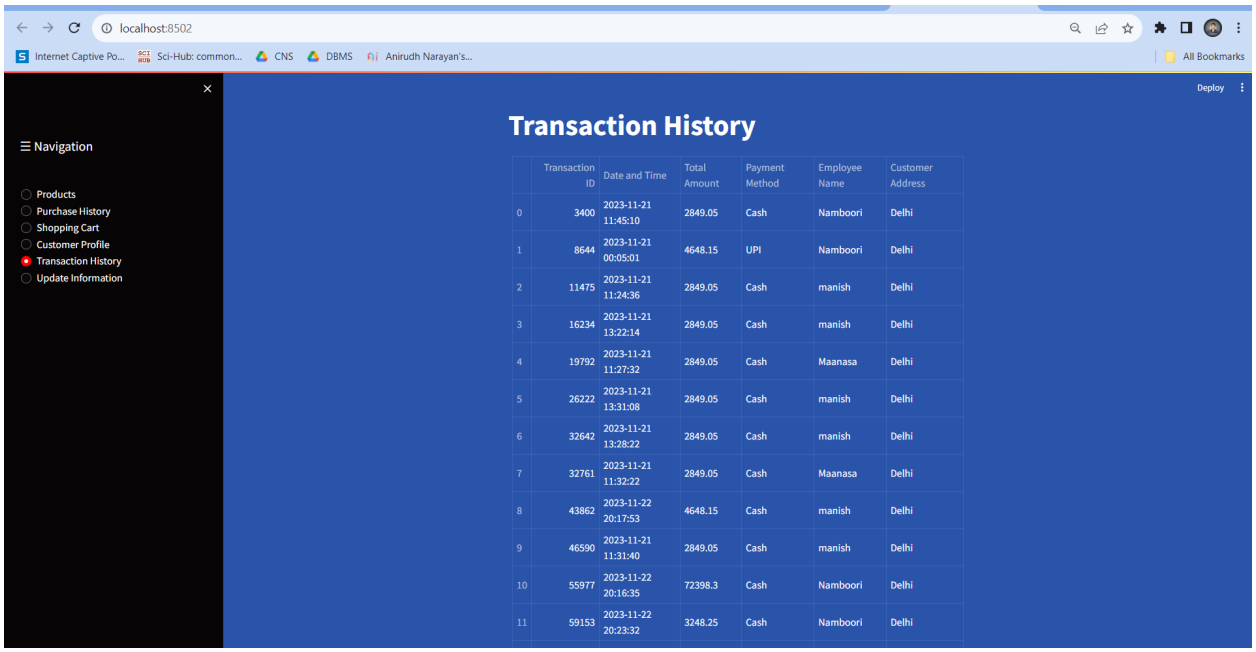


<u>Customer Profile:</u> This tab provides comprehensive details about the customer, including their name, ID, and contact information.

Customers can update or delete their information as demonstrated.



<u>Transaction History:</u> This tab displays a record of all transactions initiated by the customer.

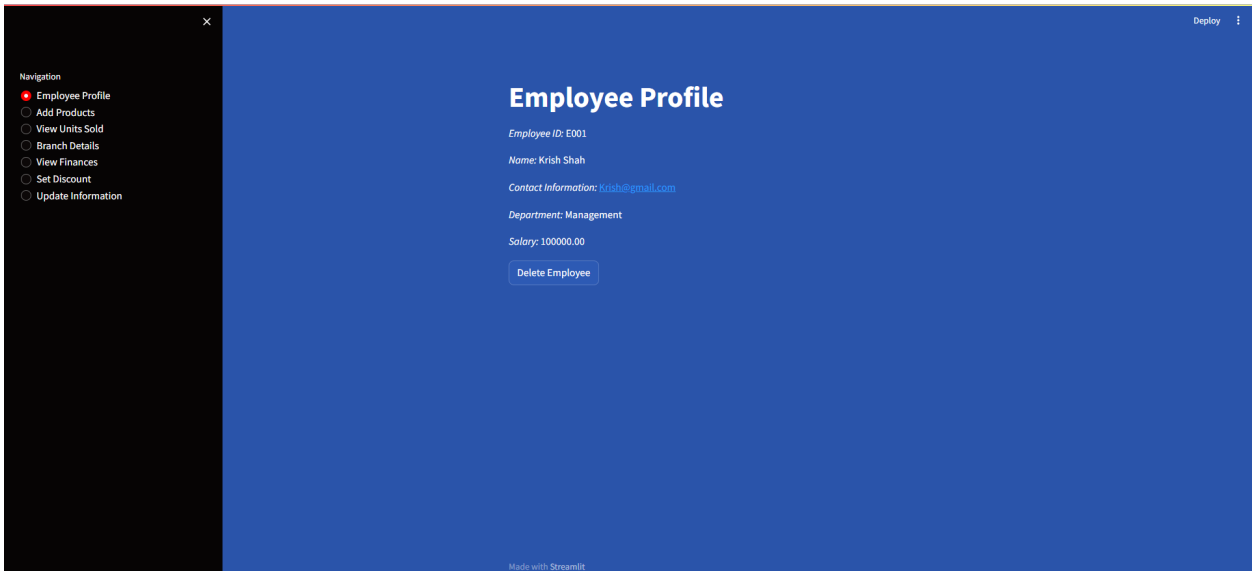


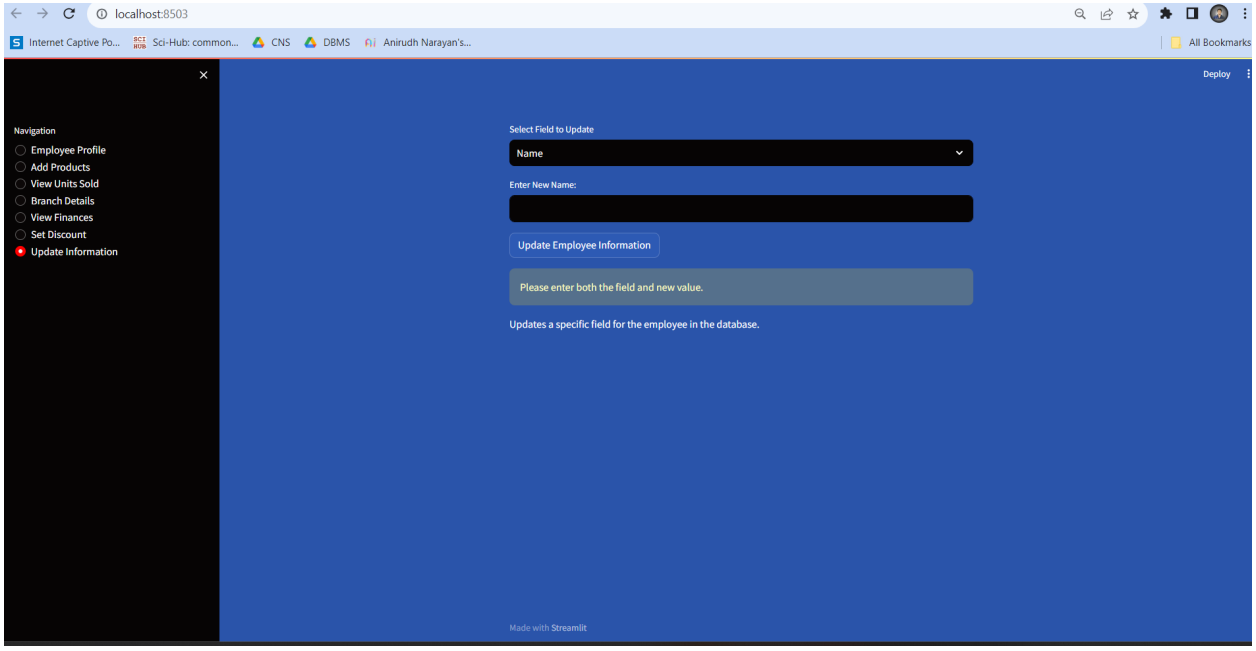
	Transaction ID	Date and Time	Total Amount	Payment Method	Employee Name	Customer Address
0	3400	2023-11-21 11:45:10	2849.05	Cash	Namboori	Delhi
1	8644	2023-11-21 00:05:01	4648.15	UPI	Namboori	Delhi
2	11475	2023-11-21 11:24:36	2849.05	Cash	manish	Delhi
3	16234	2023-11-21 13:22:14	2849.05	Cash	manish	Delhi
4	19792	2023-11-21 11:27:32	2849.05	Cash	Maanasa	Delhi
5	26222	2023-11-21 13:31:08	2849.05	Cash	manish	Delhi
6	32642	2023-11-21 13:28:22	2849.05	Cash	manish	Delhi
7	32761	2023-11-21 11:32:22	2849.05	Cash	Maanasa	Delhi
8	43862	2023-11-22 20:17:53	4648.15	Cash	manish	Delhi
9	46590	2023-11-21 11:31:40	2849.05	Cash	manish	Delhi
10	55977	2023-11-22 20:16:35	72398.3	Cash	Namboori	Delhi
11	59153	2023-11-22 20:23:32	3248.25	Cash	Namboori	Delhi

EMPLOYEE Window

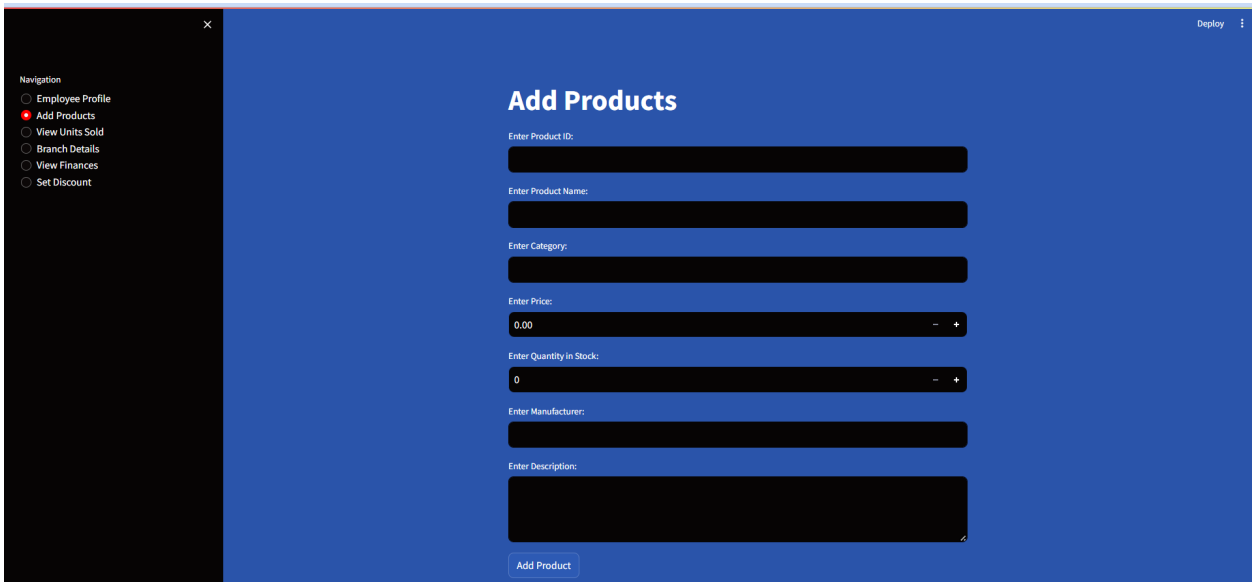
This tab includes features such as employee profile, a section for adding products, viewing the number of products sold, branch details, financial insights, and managing discounts. Access to specific tabs is restricted and granted based on the department in which the employee works.

Employee Profile: This section displays comprehensive details about the employee, and the fields of employee information can be updated as needed.

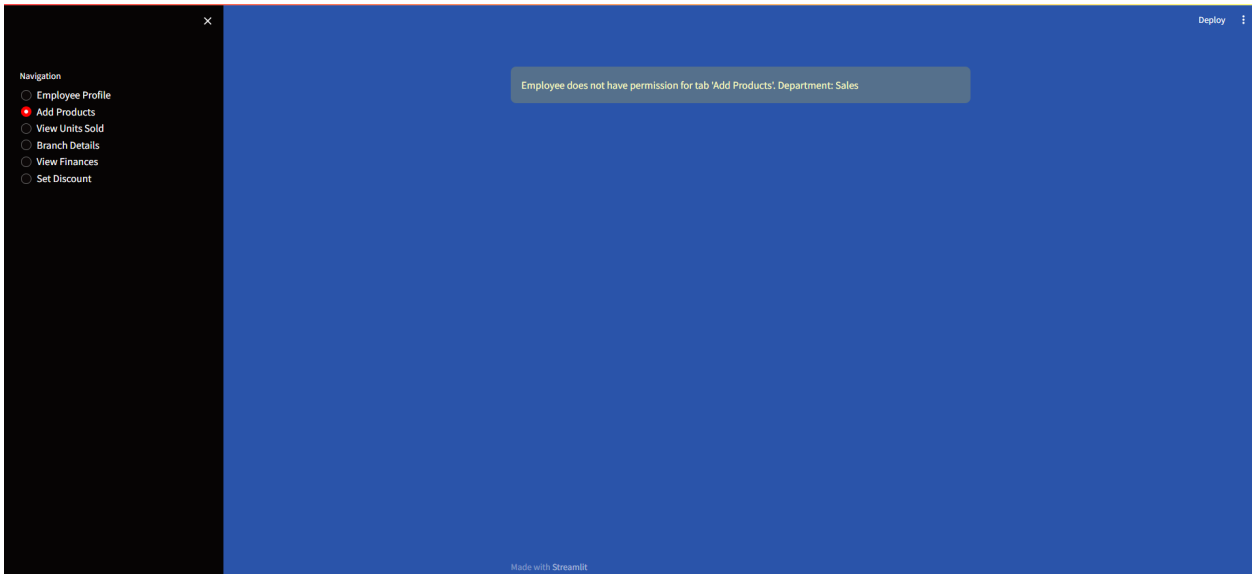




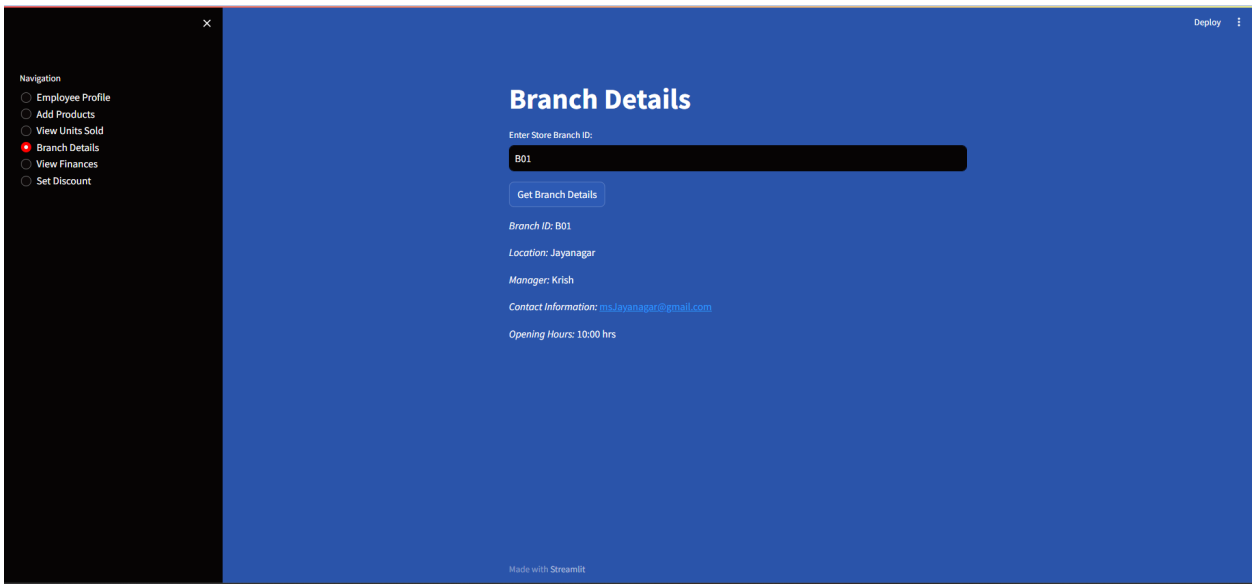
<u>Add Products:</u> This tab allows the addition of new products, and access to this feature is restricted to the management department.



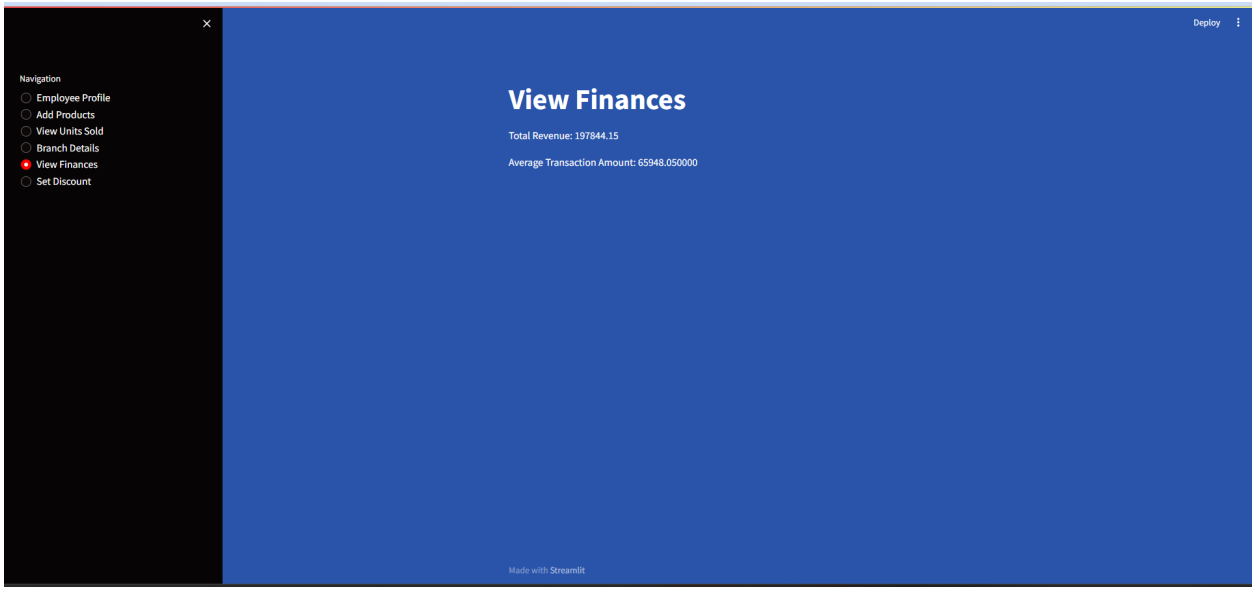
When attempting to add products from other departments, the action is not accepted, as illustrated below.



<u>Branch Details:</u> This feature retrieves specific information about a particular branch, including its location, manager, and contact details.



<u>View Finances:</u> This tab displays the total revenue generated by a specific employee in the sales department.



Procedures/Functions/Triggers and their Code snippets for invoking them:

Procedure:

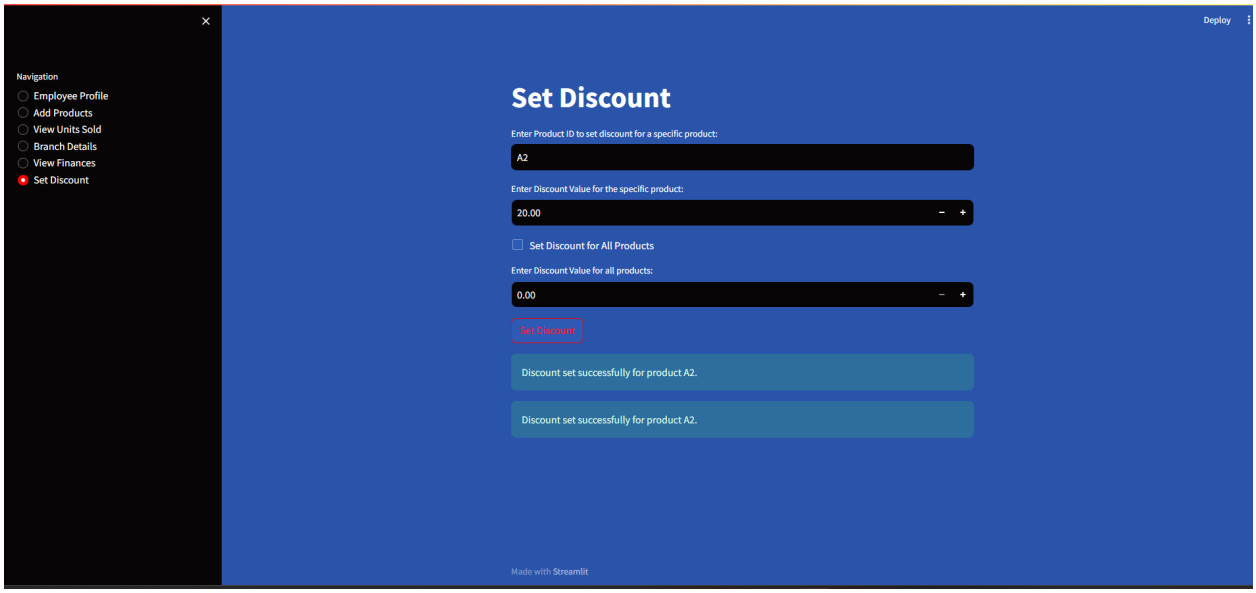
The "set_discount" procedure is designed to update the discount value for a specific product in the "product" table of your database

Code:

```
DELIMITER //
CREATE PROCEDURE set_discount(IN product_id VARCHAR(10), IN
discount_value DECIMAL(10,2))
BEGIN
    UPDATE product SET Discount = discount_value WHERE ProductID
= product_id;
END;
DELIMITER ;
```

Before setting the discount for product A2:

ProductID	Name	Category	Price	QuantityInStock	Manufacturer	Description	Discount
A1	Wireless Earbuds	Accessories	2999.00	20	Sony	Bluetooth earbuds with noise cancellation.	5.00
A2	Phone Case	Accessories	499.00	30	Spigen	Protective case for smartphones.	10.00
A3	Power Bank	Accessories	1999.00	25	Anker	Portable charger with high capacity.	10.00
P1	Samsung Galaxy S21	Phone	59999.00	10	Samsung	Flagship smartphone with advanced features.	10.00
P2	iPhone 13	Phone	79999.00	8	Apple	Latest iPhone model with powerful performance.	10.00
P3	OnePlus 9	Phone	49999.00	15	OnePlus	Affordable flagship killer with high-end specs.	10.00

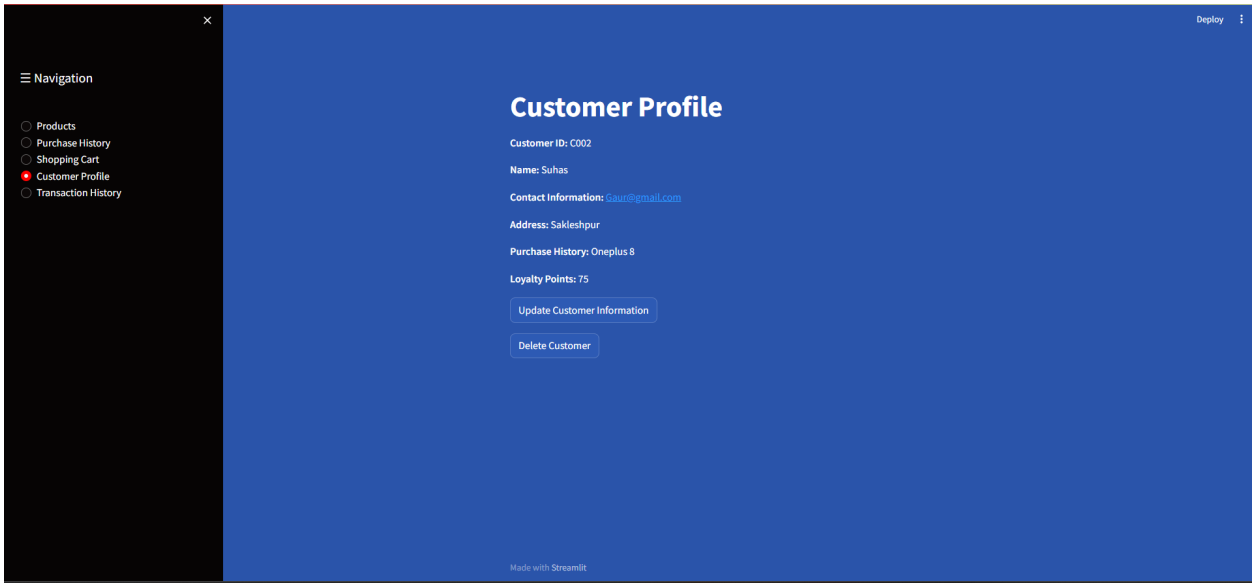



```
mysql> select * from product;
+----------+-------------------+-------------+----------+-----------------+--------------+-----------------------------------------------+----------+
| ProductID | Name              | Category    | Price    | QuantityInStock | Manufacturer | Description                                   | Discount |
+----------+-------------------+-------------+----------+-----------------+--------------+-----------------------------------------------+----------+
| A1       | Wireless Earbuds  | Accessories |  2999.00 |              20 | Sony         | Bluetooth earbuds with noise cancellation.    |     5.00 |
| A2       | Phone Case        | Accessories |   499.00 |              30 | Spigen       | Protective case for smartphones.              |    20.00 |
| A3       | Power Bank        | Accessories |  1999.00 |              25 | Anker        | Portable charger with high capacity.          |    10.00 |
| P1       | Samsung Galaxy S21 | Phone       | 59999.00 |              10 | Samsung      | Flagship smartphone with advanced features.   |    10.00 |
| P2       | iPhone 13         | Phone       | 79999.00 |               8 | Apple        | Latest iPhone model with powerful performance. |    10.00 |
| P3       | OnePlus 9         | Phone       | 49999.00 |              15 | OnePlus      | Affordable flagship killer with high-end specs. |    10.00 |
+----------+-------------------+-------------+----------+-----------------+--------------+-----------------------------------------------+----------+
6 rows in set (0.00 sec)
```

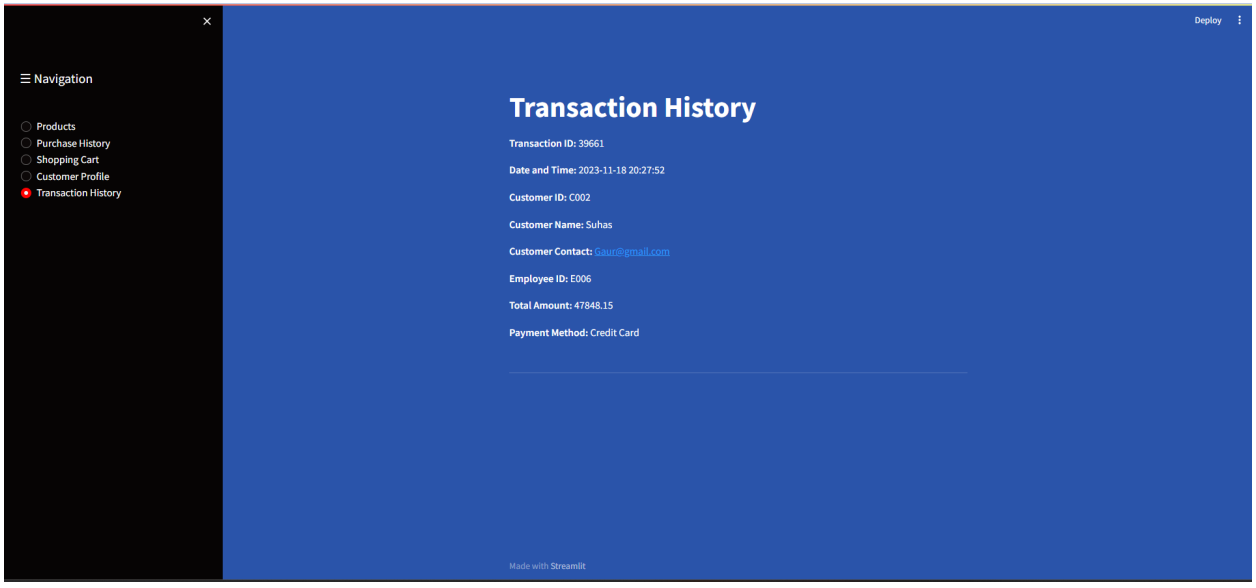
Trigger:

Code:

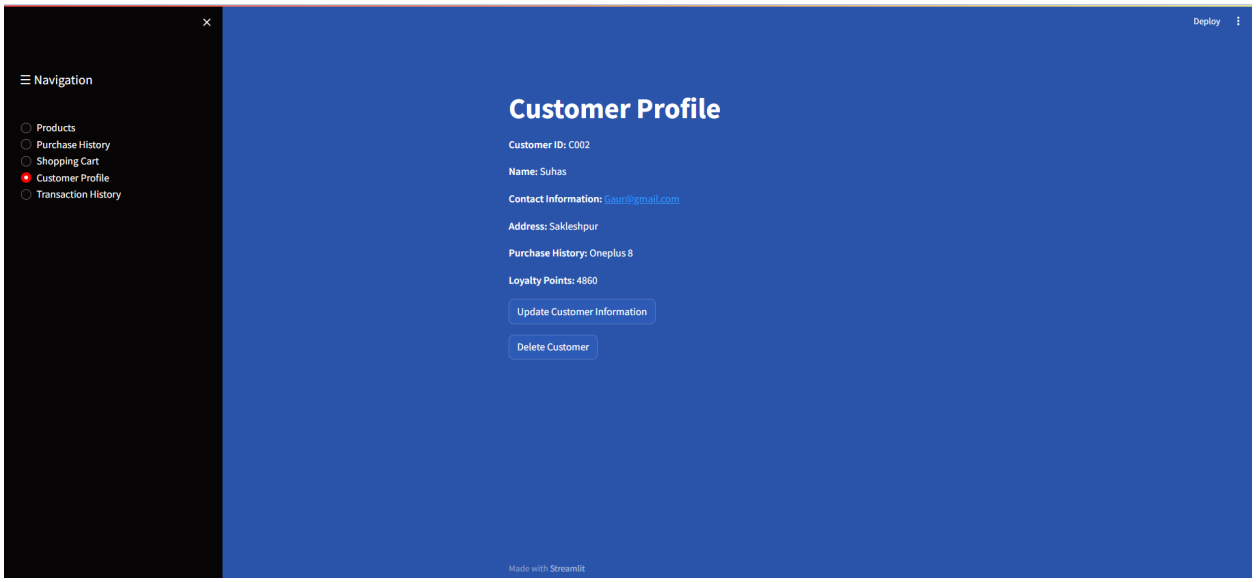
```
DELIMITER //
CREATE TRIGGER newTransaction
AFTER INSERT ON transaction
FOR EACH ROW
BEGIN
  UPDATE customer
  SET LoyaltyPoints = LoyaltyPoints + NEW.TotalAmount * 0.1
  WHERE customerid = NEW.CustomerID;
END;
//
DELIMITER ;
```



Initially this customer has 75 loyalty points



After making the following transaction, the loyalty points have been increased as seen in the screenshot attached below:


Deploy
Navigation
Products
Purchase History
Shopping Cart
Customer Profile
Transaction History
Customer Profile
Customer ID: C002
Name: Suhas
Contact Information: Gaur@gmail.com
Address: Sakleshpur
Purchase History: Oneplus 8
Loyalty Points: 4860
Update Customer Information
Delete Customer
Made with Streamlit

Nested query, aggregate query and join query

Nested Query:

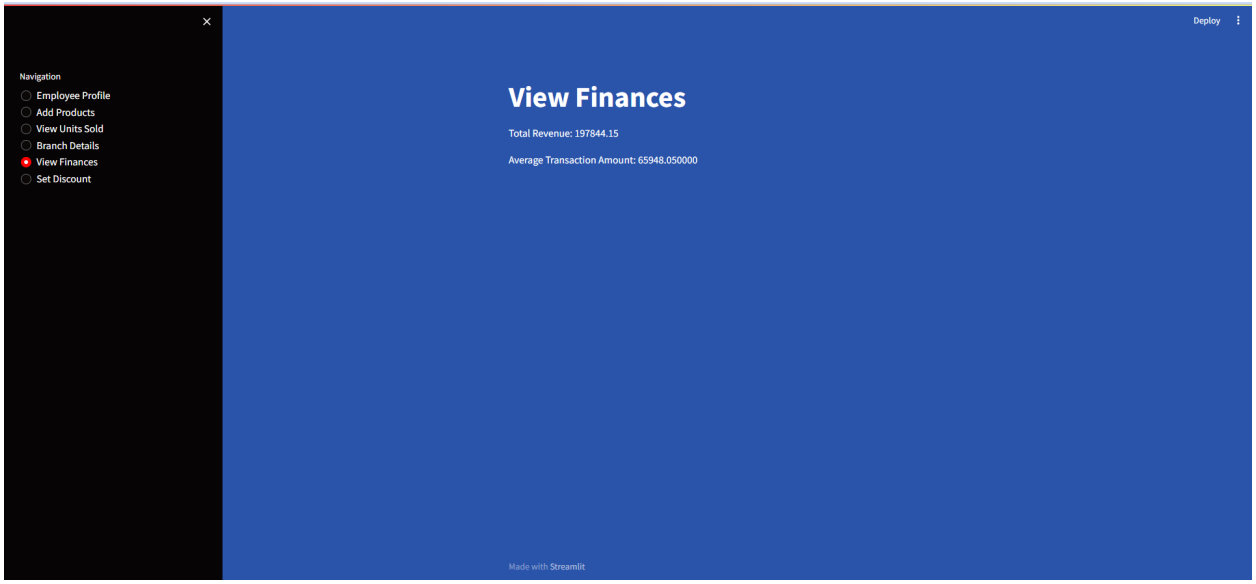
The nested query in the "View Finances" section is designed to calculate the total revenue based on the transactions associated with customers handled by the current employee.

```
query_finances = f"""
            SELECT SUM(TotalAmount)
            FROM Transaction
            WHERE customerid IN (
                SELECT customerid
                FROM Customer
                WHERE employeeid = '{employee_id}'
            )
```

Aggregate Query:

```
query_average_transaction = f"""
        SELECT AVG(TotalAmount)
        FROM Transaction
        WHERE customerid IN (SELECT customerid FROM Customer
WHERE employeeid = '{employee_id}')
```

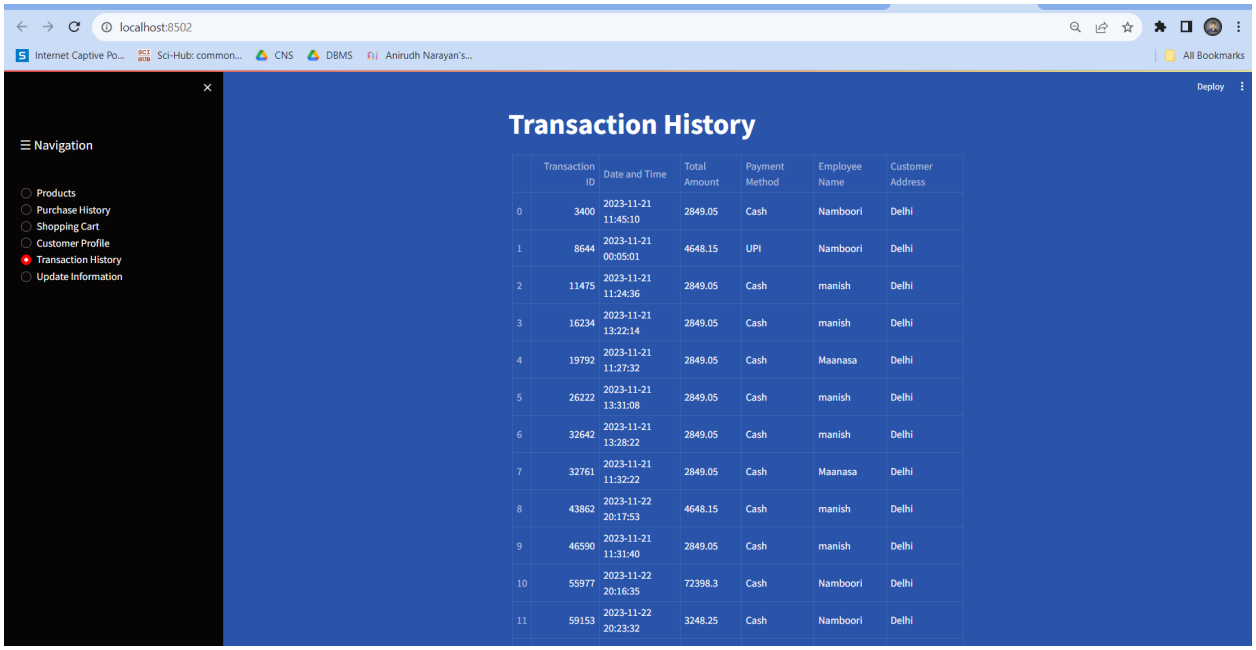
Aggregate query that calculates the average transaction amount for the transactions associated with the current employee's customers.



Using nested queries and the aggregate function, the store will be able to display the total revenue and average transaction amount earned by the employee with the ID E005.

Join: This query retrieves transaction history for a specific customer identified by the provided user_id

```
query_transaction_history = f"""
            SELECT t.TransactionID, t.DateAndTime,
t.TotalAmount, t.PaymentMethod, e.Name AS EmployeeName,
c.Address
            FROM Transaction t
            JOIN Employee e ON t.EmployeeID = e.EmployeeID
            JOIN Customer c ON t.CustomerID = c.CustomerID
            WHERE t.CustomerID = '{user_id}'
        """
```



Transaction History

	Transaction ID	Date and Time	Total Amount	Payment Method	Employee Name	Customer Address
0	3400	2023-11-21 11:45:10	2849.05	Cash	Namboori	Delhi
1	8644	2023-11-21 00:05:01	4648.15	UPI	Namboori	Delhi
2	11475	2023-11-21 11:24:36	2849.05	Cash	manish	Delhi
3	16234	2023-11-21 13:22:14	2849.05	Cash	manish	Delhi
4	19792	2023-11-21 11:27:32	2849.05	Cash	Maanasa	Delhi
5	26222	2023-11-21 13:31:08	2849.05	Cash	manish	Delhi
6	32642	2023-11-21 13:28:22	2849.05	Cash	manish	Delhi
7	32761	2023-11-21 11:32:22	2849.05	Cash	Maanasa	Delhi
8	43862	2023-11-22 20:17:53	4648.15	Cash	manish	Delhi
9	46590	2023-11-21 11:31:40	2849.05	Cash	manish	Delhi
10	55977	2023-11-22 20:16:35	72398.3	Cash	Namboori	Delhi
11	59153	2023-11-22 20:23:32	3248.25	Cash	Namboori	Delhi

List of Functionalities

app.py

1. id_exists(user_id, user_type)

```
  def id_exists(user_id, user_type):
      query = f"SELECT * FROM {user_type} WHERE {user_type}ID =
'{user_id}'"
      cursor = db.cursor()
      cursor.execute(query)
      result = cursor.fetchone()
      return result is not None
```

2. signup_customer()

```
   def signup_customer():
      query = f"INSERT INTO Customer (CustomerID, Name,
ContactInformation, Address, PurchaseHistory, LoyaltyPoints,
Password) VALUES ('{customer_id}', '{name}', '{contact_info}',
'{address}', '{purchase_history}', {loyalty_points},
'{password}')"
      execute_query(query, "Customer")
```

3. signup_employee()

```
   - Handles the employee sign-up process.
   def signup_employee()
   query = f"INSERT INTO Employee (EmployeeID, Name,
ContactInformation, Department, Salary, Password) VALUES
('{employee_id}', '{name}', '{contact_info}', '{department}',
```

```
{salary}, '{password}')"
        execute_query(query, "Employee")
```

4. execute_query(query, user_type)

```
def execute_query(query, user_type):
        cursor = db.cursor()
        cursor.execute(query)
        db.commit()
        st.success(f"{user_type} signed up successfully!")
```

5. login()

- Handles the user login process.

```
 def login():

        if user_id.startswith("C"):
            query = f"SELECT * FROM Customer WHERE CustomerID =
'{user_id}' AND Password = '{password}'"
```

```
            signup_function = signup_customer
        elif user_id.startswith("E"):
            query = f"SELECT * FROM Employee WHERE EmployeeID =
'{user_id}' AND Password = '{password}'"
            signup_function = signup_employee
        else:
            st.error("Invalid User ID format. Please use 'C' for
Customer or 'E' for Employee.")
            return
```

```
        cursor.execute(query)
        result = cursor.fetchone()
```

These functions collectively handle user sign-up, login, and database interactions within a Streamlit application for a Mobile Store Management System.

Customer.py

1. Retrieve User Information from Database

```
    - Retrieves user information from the database using the user
ID.
    user_id = os.environ.get("USER_ID")  # Change this to USER_ID

    db = mysql.connector.connect(
        host="localhost",
        user="root",
        password="@Ronaldo04",
        database="dbms_project"
)
```

```
    cursor = db.cursor()
    query_customer = f"SELECT * FROM Customer WHERE CustomerID =
'{user_id}'"
    cursor.execute(query_customer)
    customer_info = cursor.fetchone()
```

2. Define Custom CSS Styles

```
- Defines custom CSS styles for Streamlit app.
custom_css = """
<style>
.burger-menu {
    font-size: 20px;
}

.sidebar-content {
    margin-left: 20px;
}
</style>
st.markdown(custom_css, unsafe_allow_html=True)
```

3. Sidebar Navigation with Burger Menu

```
- Implements sidebar navigation with a burger menu.
  st.sidebar.markdown('<p class="burger-menu">&#x2630;
Navigation</p>', unsafe_allow_html=True)
  selected_tab = st.sidebar.radio("", ["Products", "Purchase
History", "Shopping Cart", "Customer Profile", "Transaction
History"], key='1')
```

4. Display Products

```
- Displays product details in a Streamlit app.
  if selected_tab == "Products":
```

5. Display Purchase History

```
- Displays purchase history details for the user.
  elif selected_tab == "Purchase History":
```

6. Display Customer Profile

```
- Displays customer information and allows updating.
  elif selected_tab == "Customer Profile":
```

7. Display Transaction History

```
- Displays transaction history for the user.
  elif selected_tab == "Transaction History":
```

8. Display Shopping Cart

```
  elif selected_tab == "Shopping Cart":
```

9. Pay Now Button

```
- Handles the payment process and records the transaction.
  pay_now = st.button(f"Pay Using {payment_method}")
  if pay_now:
```

10. Database Connection and Cursor Initialization

```
- Establishes a connection to the MySQL database and initializes
```

```
the cursor.
    db = mysql.connector.connect(
        host="localhost",
        user="root",
        password="@Ronaldo04",
        database="dbms_project"
    )
    cursor = db.cursor()
```

These functions collectively handle user interactions, database queries, and display content based on the selected tab in a Streamlit application for a Mobile Store Management System.

employee.py

Here are the functions used in the provided `employee.py` code along with code snippets and explanations:

1. Set Discount for a Specific Product

```
  - Sets a discount for a specific product using the stored
procedure `set_discount`.
   def set_discount(product_id, discount_value):
       try:
           cursor.callproc('set_discount', (product_id,
discount_value))
           db.commit()
           st.success(f"Discount set successfully for product
{product_id}.")
       except mysql.connector.Error as err:
           st.error(f"Error: {err}")
```

2. Set Discount for All Products

```
- Sets a discount for all products using the stored procedure
`set_discount_for_all`.
   def set_discount_for_all_products(discount_value):
       try:
           cursor.callproc('set_discount_for_all',
(discount_value,))
           db.commit()
           st.success("Discount set successfully for all
products.")
       except mysql.connector.Error as err:
           st.error(f"Error: {err}")
```

3. Update Employee Information

```
 - Updates a specific field for the employee in the database.
   def update_employee_info(employee_id, field, new_value):
       query_update_employee = f"UPDATE Employee SET {field} =
'{new_value}' WHERE EmployeeID = '{employee_id}'"
       cursor.execute(query_update_employee)
       db.commit()
       st.success(f"Employee {field} updated successfully!")
```

4. Delete Employee

```
   - Deletes the employee from the database.
   def delete_employee(employee_id):
       query_delete_employee = f"DELETE FROM Employee WHERE
EmployeeID = '{employee_id}'"
       cursor.execute(query_delete_employee)
       db.commit()
       st.success("Employee deleted successfully!")
```

5. View Units Sold

```
 - Displays information related to units sold.
   if selected_tab == "View Units Sold":
       if employee_info and (employee_info[3].strip() == "Sales"
or employee_info[3].strip() == "Management"):
           st.title("View Units Sold")
           # Debugging statement
           st.write("Employee has permission to access View
Units Sold tab.")
       else:
           # Debugging statement
           st.warning(f"Employee does not have permission for
```

```
tab '{selected_tab}'. Department: {employee_info[3].strip()}")
```

6. Branch Details

```
 - Displays details of a store branch.

   if selected_tab == "Branch Details":
       st.title("Branch Details")
       # ... (code for fetching and displaying branch details)
```

7. View Finances

```
   - Displays financial information such as total revenue and
average transaction amount.
   if selected_tab == "View Finances":
       if employee_info and (employee_info[3].strip() ==
"Management" or employee_info[3].strip() == "Finance" or
employee_info[3].strip() == "Sales"):
           st.title("View Finances")
           # ... (code for fetching and displaying financial
information)
       else:
           st.warning(f"Employee does not have permission for
tab '{selected_tab}'. Department: {employee_info[3].strip()}")
```

8. Set Discount

```
   - Allows employees in the Management department to set
discounts for specific products or all products.
   if selected_tab == "Set Discount":
       if employee_info and employee_info[3].strip() ==
"Management":
           # ... (code for setting discounts)
```

```
        else:
            st.warning("You do not have permission to set the
discount. Only Management department can set the discount.")
```

These functions collectively handle employee interactions, database queries, and display content based on the selected tab in a Streamlit application for a Mobile Store Management System.

Github Link:

https://github.com/prajwalraikj/Mobile-Store-Management-System.git