Project Report: Online Shopping Database Management System

Omer Seyfeddin Koc [1]

[1]Northeastern University
360 Huntington Ave, Boston, MA 02115 USA

Abstract

This project aims to develop an Online Shopping Database Management System that is both intuitive and efficient, addressing the complexity of customer interactions, orders, and inventory in the e-commerce space. The system is structured with a robust backend using MySQL and Python, supporting essential features such as user data management, product tracking, inventory control, and customer reviews. For administrators, it includes analytical tools for monitoring and managing seller performance and sales data. Simultaneously, it provides a hassle-free shopping experience for customers through a streamlined interface for making purchases and updating personal information, with the ultimate goal of delivering a seamless, user-friendly experience for all stakeholders.

Introduction

This database system underpins an online shopping platform, designed to manage intricate relationships between customers, orders, products, and transactional operations. Through tables that encapsulate customer profiles, order histories, and product catalogs, it facilitates a user-friendly shopping journey. The system integrates functionalities for product categorization, inventory management, customer feedback through reviews, and real-time order tracking. It leverages MySQL for structured data storage and Python for application logic, employing libraries like Tkinter for GUI interaction and PyMySQL for database connectivity. Advanced database constructs, such as stored procedures and triggers, automate routine tasks like calculating loyalty points and managing discounts. By sourcing realistic data sets for simulation, the system ensures a robust testing environment. The dual-application interface supports distinct administrative and customer use cases, aiming for operational efficiency and enhanced customer engagement.

The Admin DBMS application is strategically designed to streamline data management and enhance decision-making through customized analysis buttons for each section, enabling administrators to perform complex queries with ease. Specialized functionalities allow admins to quickly retrieve information, such as identifying poorly rated sellers or visually analyzing the geographical distribution of addresses by state. Moreover, dedicated sections for inserting, updating, and deleting records empower admins with efficient data manipulation tools, ensuring that the management of customer data, product listings, and transactional records is both user-friendly and effective. The system's architecture, with its analytical capabilities and simplified data operations, aligns with the objective to provide a seamless and insightful administrative experience, improving overall service quality.

The customer application within the Online Shopping Database Management System is elegantly designed for simplicity, offering a straightforward shopping experience while also allowing customers to update their existing personal data such as name, password, and address with ease. This ensures a user-friendly interface that simplifies the shopping process and maintains up-to-date customer information for a personalized and efficient service.

Database Design

ERD of the Database

The ERD (Entity-Relationship Diagram) for the Online Shopping Database is as follows:

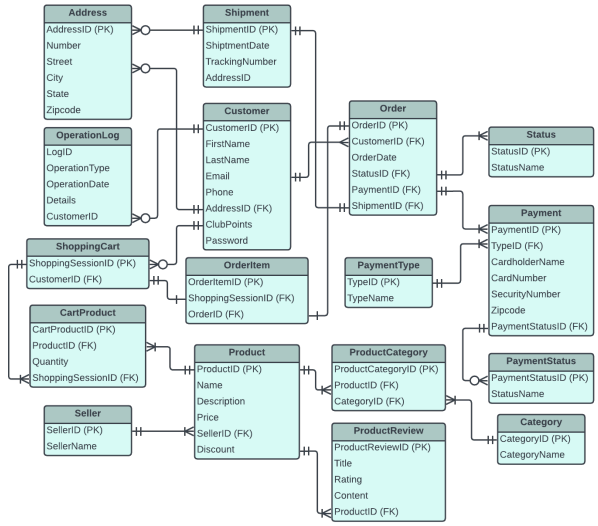

Figure 1: ERD of the database.

The ERD presents the underlying structure of the database system that supports an online shopping platform. It is composed of various tables, each with a specific purpose and interconnected to form a cohesive database.

- **Customer Table:** At the core of the system is the "Customer" table, which stores essential customer information, including first name, last name, email, phone number, and a reference to their address through the "AddressID" field. This table acts as a central point for tracking customer interactions and orders.
- **Address Table:** The "Address" table is linked to the "Customer" table via the "AddressID" field, establishing a relationship between customers and their physical addresses. This ensures accurate delivery and contact information for orders.
- **Order Table:** The "Order" table records information about customer orders. It includes data such as the order date, the customer responsible for the order (via "CustomerID"), the order's status ("StatusID"), the payment details ("PaymentID"), and the shipping information ("ShipmentID"). This table facilitates order tracking and management.
- **Product Table:** Detailed information about the products available for purchase is stored in the "Product" table. It includes fields for product name, description, price, and references the seller via "SellerID." This allows customers to browse and select products.
- **Category Table:** The "Category" table is used to store various product categories. Each category is uniquely identified by a "CategoryID" and is associated with a descriptive name stored in the "CategoryName" field.
- **ProductCategory Table:** The "ProductCategory" table serves as a bridge between products and categories. It allows each product to be associated with one or more categories, enabling product categorization. The table includes a "ProductCategoryID" as a unique identifier, along with "ProductID" and "CategoryID" fields, both of which are foreign keys referencing the respective IDs in the "Product" and "Category" tables.
- **ProductReview Table:** Customer feedback and reviews are captured in the "ProductReview" table. This includes the review title, rating, content, and references to the reviewed product via "ProductID." Product reviews provide valuable insights for other customers.
- **Seller Table:** The "Seller" table stores information about the sellers offering products in the online store. It includes the seller's name and serves as a reference for products they provide.
- **ShoppingCart and CartProduct Tables:** These tables are crucial for managing customer shopping sessions. The "ShoppingCart" table maintains information about each shopping session, while the "CartProduct" table links products to specific shopping carts, tracking quantities and aiding in the checkout process.
- **Payment and PaymentStatus Tables:** The "Payment" table records payment details, including payment type ("TypeID"), cardholder name, card number, security number, and zipcode. The "PaymentStatus" table manages different payment statuses, ensuring transparent transaction records.
- **Shipment Table:** Shipping information is stored in the "Shipment" table, including shipment date, tracking number, and a reference to the delivery address via "AddressID."
- **Status Table:** The "Status" table maintains various statuses that can be assigned to orders. It helps in tracking the progress of orders and managing their status.
- **PaymentType Table:** Different types of payment methods are recorded in the "PaymentType" table, ensuring flexibility in payment options for customers.
- **ShoppingCart Table:** The "ShoppingCart" table is used to manage shopping sessions. It employs an auto-incremented "ShoppingSessionID" field as a unique identifier for each shopping session and associates them with customers through the "CustomerID" field, which references customer data in the "Customer" table.
- **OperationLog Table:**The "OperationLog" table is designed to capture and log various database operations. It includes an auto-incremented "LogID" for unique identification, "OperationType" to record the type of operation (e.g., Insert, Delete, Update), "OperationDate" to store the timestamp of the operation, "Details" for additional information about the operation, and "CustomerID" to link the log entry to a specific customer from the "Customer" table.

All these tables have been created in MySQL along with their relevant SQL queries, and their relationships have been established.

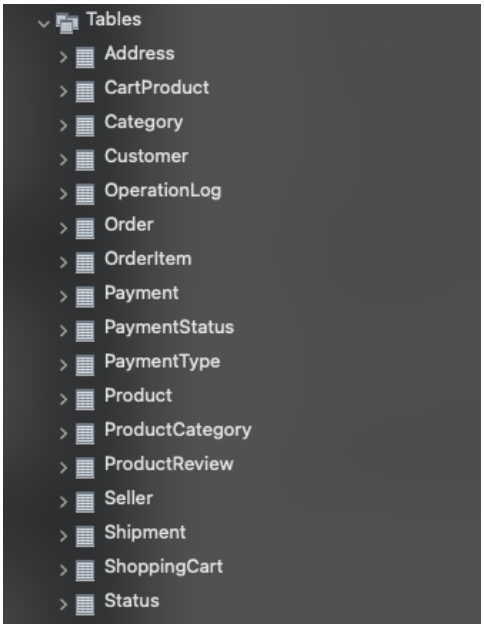

Figure 2: Tables in MySQL Workbench.

Key Design Decisions

In designing the Online Shopping Database, several key decisions were made to ensure efficient and effective data management, as well as to support the objectives of the system. Here are the key design decisions explained:

1. **Normalization:** Normalization procedures were followed to minimize data redundancy and improve data integrity. This decision resulted in a well-organized schema that reduces the chances of data anomalies.
2. **Table Structure:** Each table in the database was carefully designed to represent specific entities within the online shopping system. For example, the "Customer" table stores customer information, while the "Product" table stores product details. This modular approach enhances data organization and retrieval.
3. **Foreign Keys:** The use of foreign keys establishes relationships between tables. For instance, the "Customer" table's "AddressID" field references the "Address" table, linking customer profiles to their addresses. This design decision ensures data consistency and supports referential integrity.
4. **Data Types:** The choice of appropriate data types for each field is crucial for optimizing storage and query performance. For example, using "DECIMAL(10, 2)" for the "Price" field in the "Product" table ensures accurate representation of prices with two decimal places.
5. **Stored Procedures:** In addition to maintaining a well-structured database, I've integrated a total of 19 stored procedures to further enhance database functionality and operational efficiency. These procedures address various aspects of database management and enable the simplification of complex and frequently used operations.

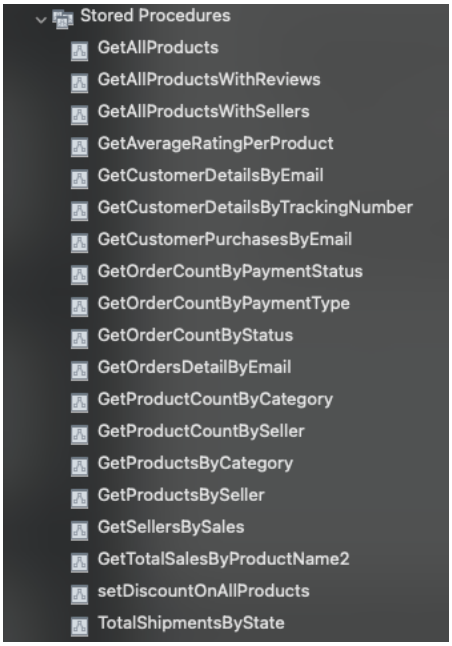

Figure 3: All Stored Procedure in MySQL Workbench.

For instance, I've developed procedures like "GetAllProductsWithReviews" to provide a comprehensive overview of products along with their associated reviews. This allows admin to make informed decisions based on product ratings and feedback. Another example is the "GetOrderCountByPaymentStatus" procedure, which assists in tracking and categorizing orders based on their payment status. This functionality aids in better understanding transaction trends and improving order management. Furthermore, I've implemented procedures such as "setDiscountOnAllProducts" to facilitate the application of discounts to multiple products simultaneously. This feature streamlines pricing adjustments and promotional activities.

These stored procedures not only simplify database operations but also contribute to a more user-friendly and efficient system. They play a crucial role in ensuring that data is easily accessible and that various tasks, from order tracking to product management, can be executed with ease.

6. **Triggers:** In addition to stored procedures, triggers have been strategically incorporated into the database design. A total of three triggers have been implemented to automate specific actions, further enhancing the system's efficiency and ensuring data consistency. One notable trigger is the "calculate_club_points" trigger, which is applied to the CartProduct table. This trigger plays a pivotal role in customer engagement by automatically awarding customers with a 5% club points bonus with every purchase. This automation not only encourages customer loyalty but also reduces the need for manual point calculations, streamlining the rewards process.

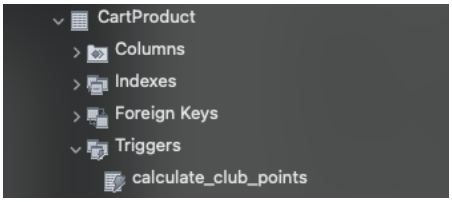

Figure 4: Club Points Trigger in MySQL Workbench.

Additionally, two triggers have been specifically designed for customers to interact with the "OperationLog" table. These triggers enable customers to log their actions and changes within the system. This proactive approach empowers customers to maintain a comprehensive record of their interactions, contributing to transparency and accountability.

The incorporation of triggers in the database design is a testament to the system's adaptability and its commitment to automating routine tasks, ultimately enhancing both customer and administrative experiences.

7. **Views:** The database architecture has been augmented with a series of Views to streamline data retrieval for reporting and analysis. The Views are specifically designed to present data in a format that's ready for con-

sumption by the front-end application or for administrative overview. The key Views incorporated are:

- cancelledorders: Provides a snapshot of all orders that have been cancelled, allowing for quick retrieval of such records without the need for complex queries.
- deliveredorders: Aggregates all successfully delivered orders, enabling customer service to swiftly access records of completed transactions.
- out_for_deliveryorders: Captures all orders currently out for delivery, offering real-time insights into the logistics process.
- processingorders: Lists all orders that are in the processing stage, ensuring that the operations team can monitor and manage ongoing order fulfilment activities effectively.
- shippedorders: Showcases orders that have been shipped, facilitating tracking and management of the dispatch process.

These Views are pivotal in providing segmented data perspectives, enhancing the efficiency of order management and supporting the strategic decision-making process.

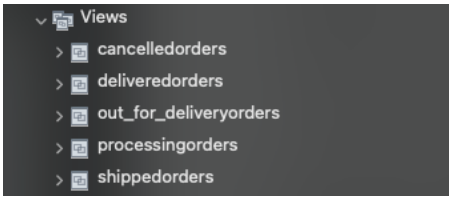

Figure 5: Views in MySQL Workbench.

8. **Functions:** The database also utilizes a set of Functions to encapsulate complex business logic, ensuring accuracy and consistency of data manipulation. The key Functions include:
   - GetCategoryCountForProduct: This Function counts the number of categories associated with each product, which is essential for managing product classifications and facilitating category-based search operations within the platform.
   - GetReviewCountForProduct: It calculates the total number of reviews for a given product. This Function is critical for gauging product popularity and customer feedback, directly influencing the purchasing decisions of potential customers.
   - resetDiscountsOnAllProducts: This Function is utilized to reset discounts across all products. It plays a crucial role during promotional periods or when adjusting pricing strategies, ensuring that all product discounts are uniformly and accurately updated across the platform.

   Incorporating these Functions into the database system centralizes the business logic within the database, minimizes redundant code, and significantly reduces the complexity of SQL queries required by the application. This approach not only promotes data integrity but also enhances the performance and scalability of the database system.

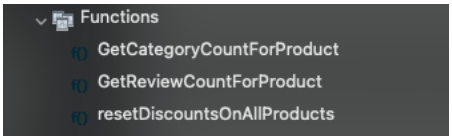

Figure 6: Functions in MySQL Workbench.

9. **User-Friendly Interface:** The inclusion of a user-friendly interface using Python and libraries like Tkinter and PyMySQL enhances the overall user experience. The interface simplifies data entry and retrieval, making it accessible to both administrators and customers.
10. **Historical Data Logging:** The introduction of the "OperationLog" table serves as a historical log to track changes made to customer data. This decision provides administrators with a record of modifications and supports data auditing.

LogID	OperationType	OperationDate	Details	CustomerID
1	Insert	2023-12-04 20:59:44	Inserted a new record with ID 59	59
2	Update	2023-12-04 21:02:08	Updated a record with ID 59	59
3	Update	2023-12-04 23:15:28	Updated a record with ID 1	1
4	Update	2023-12-04 23:15:28	Updated a record with ID 2	2
5	Update	2023-12-04 23:15:28	Updated a record with ID 3	3
6	Update	2023-12-04 23:15:28	Updated a record with ID 4	4
7	Update	2023-12-04 23:15:28	Updated a record with ID 5	5
8	Update	2023-12-04 23:15:28	Updated a record with ID 6	6
9	Update	2023-12-04 23:15:28	Updated a record with ID 7	7
10	Update	2023-12-04 23:15:28	Updated a record with ID 8	8
11	Update	2023-12-04 23:15:28	Updated a record with ID 9	9
12	Update	2023-12-04 23:15:28	Updated a record with ID 10	10

Figure 7: OperationLog Table Example in MySQL Workbench.

11. **Realistic Data Sets:** To ensure a robust testing environment, realistic data sets were sourced from external sources such as generatedata.com and helium10.com. This choice enables thorough testing of the system's functionality.

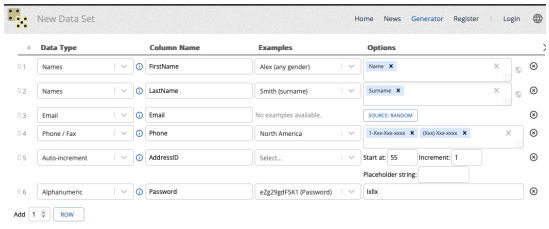

Figure 8: generatedata.com

In conclusion, these design decisions were made with a focus on data integrity, efficiency, user-friendliness, and scalability to create a robust Online Shopping Database Management System that meets the objectives of providing a seamless shopping experience and efficient data management.

Normalization

In designing my database, I took several steps to ensure data integrity and optimize its structure. One of the key normalization procedures I implemented was the creation of the ”Address” table. This table is responsible for storing address information, and I used a unique ”AddressID” as the primary key to prevent duplicate data and ensure efficient data retrieval. To associate customers with their addresses, I established a relationship between the ”Customer” table and the ”Address” table using the ”AddressID” field. This way, each customer’s address details are stored in a separate table, avoiding data redundancy.

I also created reference tables like ”PaymentStatus” and ”Status” to define specific status types for payments and general purposes. This approach ensures consistency in status definitions throughout the database and simplifies data management. For product-related information, I designed the ”Product” table, which stores details about products available in the online store. To link products with their respective sellers, I added a ”SellerID” field, establishing a connection between the ”Product” and ”Seller” tables.

To enable effective product categorization, I introduced the ”Category” and ”ProductCategory” tables. The latter allows me to assign multiple categories to each product, enhancing the classification of products. Product reviews are stored in the ”ProductReview” table, with each review linked to a specific product. In terms of managing shopping carts and their contents, I created the ”ShoppingCart” and ”CartProduct” tables. These tables are linked through the ”CustomerID” field, allowing me to create individual shopping carts for each customer.

The ”PaymentType” table was introduced to define payment types and link them to payment transactions. Finally, I developed the ”Shipment” and ”Order” tables to manage order and shipment information, with references to customer and payment details. By implementing these normalization procedures, I aimed to ensure that my database remains well-organized, minimizes data redundancy, and supports efficient data retrieval and management.

Data Collection

In populating the database, I utilized external data from two different sources to ensure a diverse and comprehensive dataset:

1. Generatedata.com: To simulate a variety of customer profiles, I used the generatedata.com platform, which provides customizable data generation capabilities. This allowed me to create a substantial amount of sample customer data, including names, addresses, emails, and phone numbers.
2. Helium10.com: For product-related information, I sourced data from Helium10, a well-known e-commerce tool. Helium10 offers product listings, including names, descriptions, prices, and seller information, which were incorporated into the database.

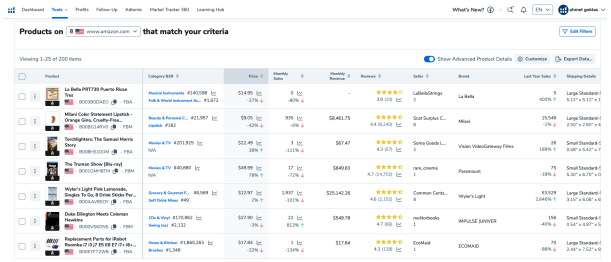

Figure 9: Helium10.com

Before integrating the external data into the database, I performed data cleaning and formatting procedures to ensure consistency and data integrity. These procedures involved standardizing data formats, removing duplicate entries, and validating data entries for accuracy.

Application

The application consists of two parts, each designed for different users. The first part is tailored for administrators, while the second part is intended for customers.

Admin Application

As the developer of this database management system for an online shopping platform, I’ve designed it with a variety of features to facilitate efficient administrative tasks. Here’s a breakdown of what my application offers and how it interacts with the database:

- **User Authentication:** An admin login feature is implemented to ensure that only authorized users can access the system. This involves verifying usernames and passwords against predefined credentials.

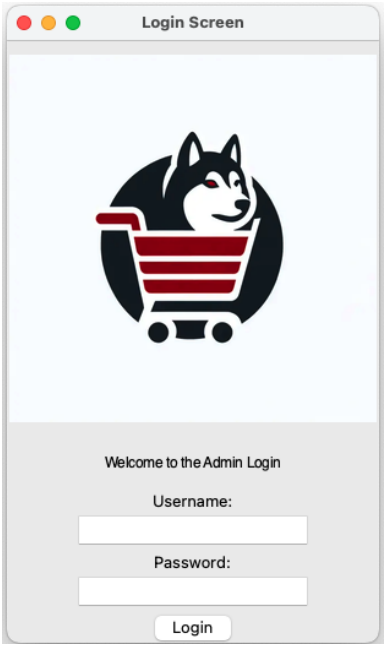

Figure 10: Admin Login Page

- **Database Interaction:** The application connects to a MySQL database using the `pymysql` library. This setup facilitates robust data storage and retrieval. SQL queries and stored procedures are executed for various operations, ranging from fetching data to updating and deleting records.
- **Data Visualization and Reporting:** Capabilities for displaying query results in an easily understandable format are integrated, along with an option to export data to Excel for more detailed analysis. For visual representation, `matplotlib` is used to plot data, such as showing the distribution of addresses across different states.
- **Tabbed Interface:** To maintain organization and user-friendliness, functionalities are divided across different tabs. Each tab is dedicated to a specific aspect like Tables, Customers, Orders, Products, Reviews, and Sellers, each with tailored functionalities.
- **Dynamic Data Retrieval:** The application is designed to fetch and display data based on user inputs, such as state names or email addresses. It can display lists of products, customers, and orders, with functionalities to update or delete records as needed.
- **GUI-Based Interaction:** A graphical user interface is developed using `tkinter`, enhancing the user experience. The interface includes interactive elements like buttons, entry fields, labels, combo boxes, and list boxes for data input and command execution.
- **Custom Queries and Procedures:** Users can run custom SQL queries directly from the application. Additionally, stored procedures are used for more complex database operations, streamlining the process and enhancing efficiency.

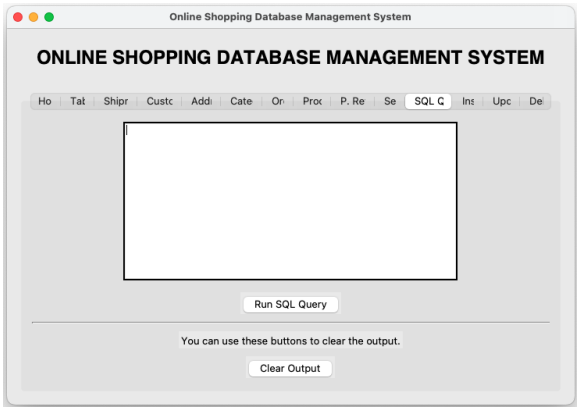

Figure 11: SQL Query Tab for Admin DBMS

- **Error Handling and Notifications:** The application is equipped to display error messages for issues related to database connections or query execution. It also provides success notifications upon the successful completion of database operations.

In terms of data storage and retrieval:

- The application's backbone is a MySQL database, indicative of a structured, relational data approach.
- Data retrieval and manipulation are handled through structured SQL queries and stored procedures, ensuring efficient and secure database interactions.
- I use pandas to manage and display data in DataFrame format, which is particularly useful for data analysis and manipulation.
- For functionalities like viewing address distributions or analyzing product categories, the application queries corresponding tables and performs necessary aggregations or joins.
- The integration of matplotlib for plotting implies that the application can transform query results into visual formats, providing intuitive insights into the data.

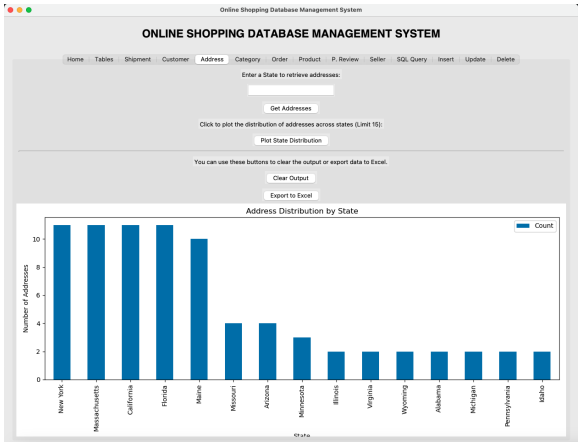

Figure 12: State Distribution Plot

In summary, my application is a comprehensive tool for managing various administrative aspects of an online shopping database, offering a blend of user-friendliness and robust database management capabilities.

Customer Application

As the architect of this customer portal for an online shopping platform, I've designed a system that is both comprehensive and intuitive. Here's an in-depth look at its primary features and the way it interacts with the underlying database:

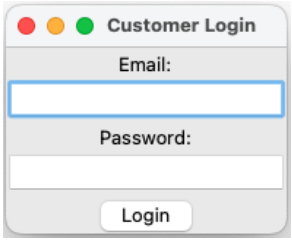

Figure 13: Customer Login Page

- **User Authentication:** I've crafted a login mechanism that authenticates customers using their email and password. This system verifies the provided credentials

against the records in the Customer table within our database.

- **Customer Information Management:** The portal prominently displays customer information such as names, emails, and phone numbers on a user-specific tab. I've also included the capability for customers to update their personal information, which is then synchronized with the database.

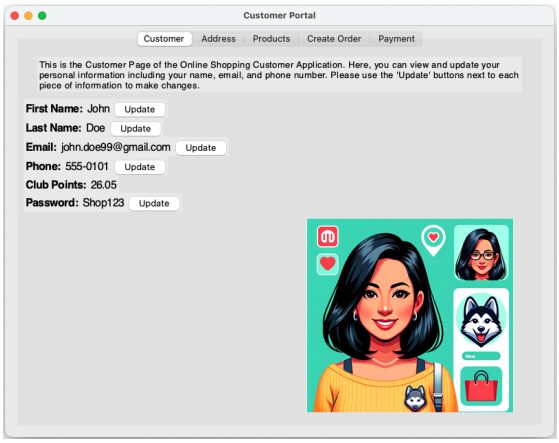

Figure 14: Mainpage in Customer App

- **Address Management:** My application permits customers to review and modify their address details. Any alterations are promptly reflected in the Address table in the database, ensuring data consistency.
- **Shopping Session Management:** Customers are empowered to initiate new shopping sessions. Commencing a session automatically generates a record in the ShoppingCart table.
- **Product Browsing and Cart Management:** The application showcases an array of products for customers to peruse. When customers add items to their cart, the CartProduct table is updated accordingly, reflecting their current selections.

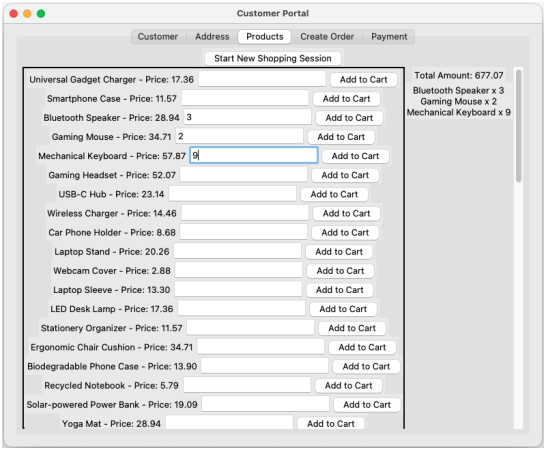

Figure 15: Products Tab in Customer App

- **Order Creation and Management:** I've enabled customers to convert their shopping carts into orders. This process involves creating a new entry in the Order table, which is then associated with the customer's active shopping session.
- **Payment Processing:** The system allows customers to enter their payment information for order processing. The provided payment details are securely stored in the Payment table.

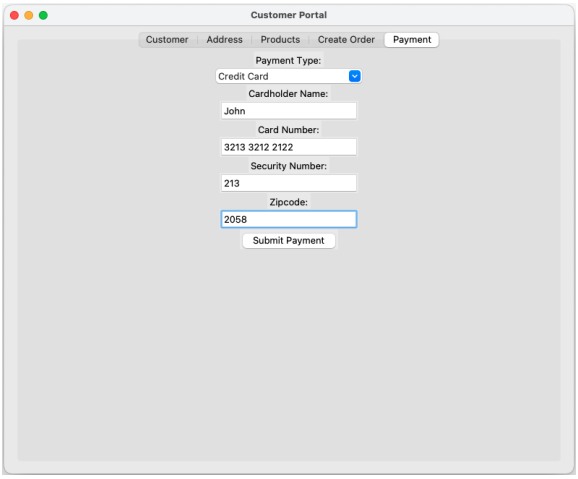

Figure 16: Payment Tab in Customer App

- **Shipment Creation:** Upon the creation of an order, a corresponding shipment record is generated in the Shipment table. This record is linked to the customer's address, streamlining the delivery process.
- **Data Storage and Retrieval:**
  - I've integrated the pymysql library to establish a robust connection to a MySQL database, which serves as the foundation for data storage and retrieval.
  - My application executes a variety of SQL queries for different tasks, ranging from data fetching to record updates and insertions.
  - The Customer table is central to user authentication, as it validates customer logins. Updates to customer profiles and addresses are transacted within the Customer and Address tables, respectively.
  - The ShoppingCart table keeps track of each customer's shopping session.
  - Product displays are populated using data from the Product table.
  - The CartProduct table is dynamic, reflecting the items each customer has chosen to add to their cart.
  - New orders are recorded in the Order table, with a link established to the respective customer's shopping session.
  - Payment information is meticulously stored within the Payment table, while shipment records are maintained in the Shipment table.

In essence, my application offers a robust customer interface for an online shopping experience, underpinned by a strong backend system that manages data storage and retrieval with efficiency and reliability.

Conclusions / Future Directions

Through the development of this Online Shopping DBMS, I've gained invaluable insights and experience in several key areas. First, the importance of robust database design and integration cannot be overstressed. Efficient SQL queries and a well-structured database are critical for performance and scalability. Second, crafting an intuitive and responsive user interface with tkinter has taught me a great deal about the nuances of user-centric design and its impact on application success. Lastly, utilizing functions, views, and triggers to automate processes within the database has been enlightening. It not only saves time but also reduces errors.

With more time, I would like to have elevated the project to a visually rich, real-time integrated feature within a website. My aspiration was to enhance the interactive experience, but as a Data Science student with nascent skills in software engineering, I chose to leverage Python—a language that aligns more closely with my current expertise.

For future DS5110 students, I recommend starting with tables of a smaller size. However, be aware that as the project progresses, the introduction of new attributes and consequent tables will inevitably lead to an expansion of the overall project scope. Also, given the extensive amount of code required to both establish the database and design the program, it's crucial to maintain thorough documentation throughout the development process.