ShopSync

Carmen Sofia Florez Juajibioy
Systems Engineer - Database Designer
ShopSync
Bogotá D.C, Colombia
Email: sofiflorez@gmail.com

***Abstract*—This paper presents a comprehensive outline of the requirements, design, and database modeling for a Shop Management Application aimed at improving operational efficiency and decision-making for store owners. Employing relational databases, the project encapsulates the development of a versatile platform tailored to meet the dynamic needs of retail management. The document details the conceptual framework, emphasizing the application's user-centric design, stakeholder analysis, business model, and user stories to define the scope and functionalities required.**

I. Introduction

The project targets store owners, managers, employees, and indirectly, customers, by facilitating an integrated management solution. It offers functionalities such as inventory management, order processing, sales tracking, and customer data analysis. A subscription-based business model underpins the application, offering various service tiers to accommodate diverse business sizes and needs.

Furthermore, the paper delves into the database design and modeling aspect, crucial for supporting the application's data handling requirements. It discusses the rationale behind using relational databases, the data structure, and the entity-relationship model that forms the backbone of the application. This section aims to provide insights into the technical considerations for ensuring data integrity, security, and scalability.

II. Development

A. Stakeholders

In the development of the Shop Management Application, identifying and understanding the roles of various stakeholders is paramount:

1) **As a store owner,** I want to add new products to my inventory to keep my offering updated.
2) **As a store manager,** I want to view a summary of daily sales to assess the business performance.
3) **As a store owner,** I seek to generate monthly sales reports to analyze performance and plan financially.
4) **As a store owner,** I wish to manage customer orders, including confirmation, packaging, and shipping of products.
5) **As a store owner,** I want to access a detailed transaction history by customer to understand their purchase preferences.

B. Business Model

The Shop Management Application is an all-encompassing solution designed to enable store owners to efficiently manage their businesses. This application provides essential tools for inventory management, order processing, and tracking sales and customers. Characterized by its user-friendly interface and robust functionalities, this platform aims to enhance operational efficiency and strengthen decision-making for businesses of various sizes. The business model of the Shop Management Application is predicated on a subscription-based concept, analogous to subscribing to a monthly or annual service. Store owners are required to pay a recurring fee to access the comprehensive suite of features and tools provided by the application. Depending on the specific requirements and size of their business, they have the option to select from a range of subscription plans, each offering varying levels of functionality.

C. Tools

To develop the Shop Management Application as outlined, several key tools and technologies will be essential to ensure the project's success. These tools are categorized based on the project phases: requirement analysis, design, development, and deployment. Requirement Analysis and Design

1) **UML (Unified Modeling Language) Tools:** This tool helps to create diagrams that represent the system's design, including use case diagrams for stakeholder actions and class diagrams for database modeling. Examples include Lucidchart and Visual Paradigm.
2) **Wireframing and Prototyping Tools:** To design the application's user interface and experience, considering the user-centric design approach. Tools like Adobe XD, Sketch, or Figma can facilitate this process by allowing the creation of interactive prototypes.
3) **Integrated Development Environment (IDE):** A robust IDE such as Visual Studio Code, IntelliJ IDEA, or Eclipse will support the coding process, offering debugging tools and version control integration.
4) **Relational Database Management System (RDBMS):** Given the application's reliance on relational databases, an RDBMS like PostgreSQL, MySQL, or Oracle Database will be critical for storing and managing structured data efficiently.
5) **Framework for Backend Development:** A server-side framework such as Django (Python), Ruby on Rails

(Ruby), or Spring Boot (Java) will help in rapidly developing secure, scalable back-end services.

6) **Frontend Development Libraries/Frameworks:** React.js, Angular, or Vue.js could be employed to create a responsive, dynamic user interface that enhances user experience.

III. Conceptual Model

1) : User Stories To fully capture and address the functional requirements of the Shop Management Application from varied perspectives, several user stories have been developed. These narratives serve as a foundation for designing and implementing the features that will meet the specific needs of different users within the system. By exploring the desires and challenges faced by store owners and managers, these user stories aim to ensure that the application not only enhances operational efficiency but also provides insightful data to support strategic decision-making. Below are the user stories that illustrate the essential functionalities sought after by our primary stakeholders.

1) As a store owner, I want to add new products to my inventory so that my offering stays updated.
2) As a store owner, I want to view a summary of daily sales so that I can assess the business performance.
3) As a store owner, I want to generate monthly sales reports so that I can analyze performance and plan financially.
4) As a store owner, I want to manage customer orders, including confirmation, packaging, and shipping of products so that I can ensure a smooth and efficient order fulfillment process.
5) As a store owner, I want to access a detailed transaction history by customer so that I can understand their purchase preferences and tailor my offerings to better meet their needs.

Entities Definition

1) User
   - user_id: Primary key, uniquely identifies each user.
   - username: Unique username for login.
   - password: Hashed password for security.
   - role: Defines the user's role (e.g., owner, manager, employee).
   - email: User's email address.
   - created_at: Timestamp of account creation.
2) Product
   - product_id: Primary key, uniquely identifies each product.
   - name: Name of the product.
   - description: Description of the product.
   - price: Price of the product.
   - quantity_in_stock: Current stock level of the product.
   - category_id: Foreign key, references the category of the product.
3) Category
   - category_id: Primary key, uniquely identifies each category.
   - name: Name of the category.
   - description: Description of the category.
4) Order
   - order_id: Primary key, uniquely identifies each order.
   - user_id: Foreign key, references the customer who placed the order.
   - total_amount: Total amount of the order.
   - status: Status of the order (e.g., confirmed, shipped, delivered).
   - created_at: Timestamp of order creation.
5) OrderItem
   - order_item_id: Primary key, uniquely identifies each order item.
   - order_id: Foreign key, references the order.
   - product_id: Foreign key, references the product.
   - quantity: Quantity of the product ordered.
   - price: Price of the product at the time of ordering.
6) Customer
   - customer_id: Primary key, uniquely identifies each customer.
   - user_id: Foreign key, references the user account of the customer.
   - first_name: First name of the customer.
   - last_name: Last name of the customer.
   - address: Address of the customer.
   - phone_number: Phone number of the customer.
7) SalesReport
   - report_id: Primary key, uniquely identifies each sales report.
   - user_id: Foreign key, references the user who generated the report.
   - period: Time period of the report (e.g., monthly, yearly).
   - total_sales: Total sales amount for the period.
   - total_orders: Total number of orders for the period.
   - created_at: Timestamp of report creation.
8) Transaction
   - transaction_id: Primary key, uniquely identifies each transaction.
   - order_id: Foreign key, references the order associated with the transaction.
   - payment_method: Method of payment (e.g., credit card, cash).
   - amount: Amount of the transaction.
   - status: Status of the transaction (e.g., successful, failed).
   - created_at: Timestamp of transaction creation.

IV. Relationship definition

- **User - Order:** A user can place multiple orders, but each order belongs to a single user. This is a one-to-many (1:N)

	E1	E2	E3	E4	E5	E6	E7	E8
E1	/////			X		X	X	
E2		/////	X		X			
E3		X	/////					
E4	X			/////	X			X
E5		X		X	/////			
E6	X					/////		
E7	X						/////	
E8				X				/////

relationship between the User entity and Order.

- **Order - Order Item:** An order can contain multiple items, but each order item belongs to a single order. This is a one-to-many (1:N) relationship between the Order entity and Order Item.
- **Product - Order Item:** A product can be part of multiple order items, but each order item belongs to a single product. This is a one-to-many (1:N) relationship between the Product entity and Order Item.
- **Product - Category:** A product belongs to a category, but a category can include multiple products. This is a one-to-many (1:N) relationship between the Product entity and Category.
- **User - Customer:** A user can be a customer, but each customer is associated with a single user. This is a one-to-one (1:1) relationship between the User entity and Customer.
- **User - Sales Report:** A user can generate multiple sales reports, but each sales report is generated by a single user. This is a one-to-many (1:N) relationship between the User entity and Sales Report.
- **Order - Transaction:** An order can generate a transaction, but each transaction is associated with a single order. This is a one-to-one (1:1) relationship between the Order entity and Transaction.

V. Relational Algebra for ShopSync

1) List all products in a specific category:

$$\pi_{\text{name, price, quantity\_in\_stock}}(\text{Product} \bowtie_{\text{Product.category\_id = Category.category\_id}} \sigma_{\text{name = 'Electronics'}}(\text{Category}))$$

2) Find all orders placed by a specific user:

$$\pi_{\text{order\_id, total\_amount, status}}(\text{Order} \bowtie_{\text{Order.user\_id = User.user\_id}} \sigma_{\text{username = 'john\_doe'}}(\text{User}))$$

3) Get the total sales amount for a specific period:

$$\gamma_{\text{period, SUM(total\_sales)}}(\sigma_{\text{period = 'June 2024'}}(\text{SalesReport}))$$

4) List all products with low stock:

$$\sigma_{\text{quantity\_in\_stock ¡ 10}}(\text{Product})$$

5) Find the average price of products in each category:

$$\gamma_{\text{category\_id, AVG(price)}}(\text{Product}) \bowtie_{\text{Product.category\_id = Category.category\_id}} \text{Category}$$

6) Get the details of the most expensive product:

$$\pi_{\text{name, description, price}}(\sigma_{\text{price = MAX(price)}}(\text{Product}))$$

7) List all customers who have placed more than 5 orders:

$$\gamma_{\text{user\_id, COUNT(order\_id)}}(\text{Order}) \bowtie_{\text{Order.user\_id = Customer.user\_id}} \sigma_{\text{count\_order\_id ¿ 5}}(\text{Customer})$$

8) Find all orders that include a specific product:

$$\pi_{\text{order\_id, user\_id, total\_amount}}(\text{Order} \bowtie_{\text{Order.order\_id = OrderItem.order\_id}} \sigma_{\text{product\_id = '123'}}(\text{OrderItem}))$$

9) Calculate the total revenue generated from a specific product:

$$\gamma_{\text{product\_id,SUM(price} \times \text{quantity)}}(\sigma_{\text{product\_id = '123'}}(\text{OrderItem}))$$

10) List all transactions that were unsuccessful:

$$\sigma_{\text{status = 'Failed'}}(\text{Transaction})$$

1) **Find all orders made by a specific user:** This query retrieves the identifiers of the orders, the total amounts, and the statuses of all the orders made by the user with the username "john_doe". A join is performed between the `Order` and `User` tables using the `user_id` as the matching criterion, and then the results are filtered to include only those orders made by "john_doe".
2) **Get the total sales amount for a specific period:** This query calculates the total sum of sales made during the period "June 2024". The records from the `SalesReport` table that correspond to that period are selected and the total sales amounts are summed.
3) **List all products with low stock:** This query identifies all products whose quantity in stock is less than 10 units. The records from the `Product` table are filtered to include only those with a quantity in stock below the specified threshold.
4) **Find the average price of products in each category:** This query calculates the average price of products in each category. Products are grouped by `category_id` and the average of the prices within each group is calculated.
5) **Get the details of the most expensive product:** This query selects the name, description, and price of the product with the maximum price in the entire `Product` table. The `MAX` function is used to determine the highest price, and then the results are filtered to include only the product with that price.

6) **List all customers who have made more than 5 orders:** This query identifies the customers who have made more than 5 orders. Orders are grouped by `user_id` and the number of orders for each customer is counted. Then, the results are filtered to include only those customers with more than 5 orders.
7) **Find all orders that include a specific product:** This query retrieves the identifiers of the orders, the identifiers of the users, and the total amounts of all orders that include the product with `product_id` equal to '123'. A join is performed between the `Order` and `OrderItem` tables, and then the results are filtered to include only those orders that contain the specified product.
8) **Calculate the total revenue generated by a specific product:** This query calculates the total sum of the revenue generated by the product with `product_id` equal to '123'. Order items are filtered to include only those that correspond to the specified product, and then the products of the prices and quantities are summed to obtain the total revenue.
9) **List all transactions that were not successful:** This query identifies all transactions whose status is "Failed". The records from the `Transaction` table are filtered to include only those with a failure status.

VI. Data Sourcers and DB testing

A. Option:

In terms of data search and practicality, it is proposed to use Scripts and the Faker library in python to fill the database.