CS 411 – Project Task 1 (PT1) Stage 4

Title	*Description*
Team Name	SelectStar
Project Title	Data-Driven Forecasting and Visualization of Fresh Food Demand in a Retail Store
Team Members	Mihir Shah (mihirss3@illinois.edu) Mohit Badve (mbadve2@illinois.edu) Prajakta Pikale (ppikale2@illinois.edu) Riya Tendulkar (rtend@illinois.edu)

Contents

1. Changes in application as compared to the original proposal.

(Please list out changes in the directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).)

The direction of the final project aligns closely with what was proposed in the initial report. We have successfully implemented the key features outlined in our proposal, which include:

- **Login and Password Recovery**: We developed a secure login page that allows analysts to access the system using their email addresses and passwords. Given that the dashboard displays sensitive information, we have restricted account creation to administrators only. This ensures that only authorized personnel can create new accounts, thereby enhancing security. For role-based access control, administrators have full permissions to add, modify, or delete data in the database, while analysts are granted read-only access to view dashboards and data. This separation of roles helps maintain data integrity and protects sensitive information from unauthorized access.
- **Current Inventory Analysis**: The dashboard prominently features a primary chart that visualizes the available products in the database alongside their respective quantities. This graphical representation allows analysts to quickly assess inventory levels at a glance. Additionally, we have implemented a search functionality based on the "Category" attribute, enabling users to filter and display products belonging to specific categories. This feature enhances usability by allowing analysts to focus on relevant data without sifting through unnecessary information.
- **Inventory Expiry Tracking**: To effectively manage perishable items, we introduced an expiry tracking system that provides alerts based on inventory status. The system categorizes items into four distinct statuses: "In Stock," "Soon to be Expiring," "Out of Stock," and "Soon to be Out of Stock." This proactive approach allows analysts to monitor inventory closely and take timely action to prevent stockouts or waste due to expired products. By keeping track of expiry dates, we aim to optimize inventory management and reduce losses associated with perishable goods.
- **Demand Forecasting and Future Stock Requirements**: One of the most significant enhancements is our implementation of a demand forecasting feature using Machine Learning algorithms. By analyzing historical sales data, we trained a model that predicts the quantity of products required for the following day. This predictive capability enables analysts to make informed decisions about inventory replenishment, ensuring that stock levels align with anticipated demand. By accurately forecasting future needs, we aim to improve operational efficiency and reduce instances of overstocking or stockouts.

  Overall, these features reflect our commitment to fulfilling the objectives set forth in our original proposal while enhancing functionality and usability. The implementation not only meets our initial goals but also adds significant value by improving user experience and operational efficiency within the application.

2. Evaluation of Application Usefulness: Achievements and Shortcomings

(Discuss what you think your application achieved or failed to achieve regarding its usefulness.)

In evaluating the usefulness of our application, we can identify several significant achievements as well as some shortcomings that warrant attention.

Achievements:

1. **Enhanced Inventory Management**: One of the primary goals of our application was to improve inventory management processes. We successfully implemented features that allow users to monitor stock levels in real-time, track expiry dates, and receive alerts for low stock situations. This functionality enables businesses to make informed decisions about restocking and minimizes the risk of stockouts or wastage due to expired products.
2. **Predictive Analytics**: The integration of a Machine Learning algorithm for demand forecasting represents a major achievement. By utilizing historical data to predict future product needs, we have equipped users with valuable insights that can enhance planning and resource allocation. This predictive capability helps businesses optimize their inventory levels and respond proactively to changing market demands.
3. **User-Friendly Interface**: The development of a user-friendly interface for both administrators and analysts has significantly improved the overall user experience. The intuitive design allows users to navigate the application easily, perform CRUD operations, and access critical data without extensive training or technical expertise.
4. **Role-Based Access Control**: Implementing role-based access control has enhanced security and data integrity within the application. By restricting permissions based on user roles, we ensure that sensitive information is only accessible to authorized personnel, thereby reducing the risk of data breaches or unauthorized modifications.
5. **Automation of Processes**: The use of triggers and stored procedures has automated various backend processes, such as updating inventory levels and refreshing data tables. This automation reduces manual intervention, minimizes errors, and improves overall operational efficiency.

Shortcomings:

1. **Limited Algorithm Comparison**: While we successfully implemented a Linear Regression model for demand forecasting, we did not explore or compare other algorithms that might yield better accuracy or performance. This limitation means that our predictive capabilities may not be fully optimized, potentially impacting the reliability of our forecasts.
2. **Limited Data**: Due to the majority of the data being a mock-up data, the charts and predictions that are being displayed on the dashboard are not accurate and hence are not giving a clear picture of the intended functionality. We were unable to find any dataset which correctly meets the requirements of all our tables and database schema. We can continue to monitor if any such dataset becomes available and use that for our project.

Overall, our application has achieved several key objectives that enhance inventory management and provide valuable insights through predictive analytics. However, there are notable shortcomings that need to be addressed to further improve its usefulness. By focusing on algorithm optimization, real-time data integration, scalability, user feedback

mechanisms, and mobile accessibility, we can enhance the application's effectiveness and better meet the needs of its users in future iterations.

3. Data Schema and Source Modifications

(Discuss if you changed the schema or source of the data for your application)

In the course of developing our application, we implemented two significant changes to the data schema to enhance functionality and improve performance:

- **Creation of a Wide Table**: To effectively train our Machine Learning algorithm and accurately predict forecast values, it was essential to have access to all relevant data features. Given the complexity of our database, which contains numerous tables, joining them multiple times for each prediction would have resulted in considerable processing delays. To address this challenge, we opted to de-normalize the data by creating a new wide table named Stock_Smart_Wide. This table consolidates columns from all relevant tables within the database into a single structure. By doing so, we streamline the data retrieval process, allowing for faster model training and more efficient predictions without the overhead of complex joins.
- **Introduction of an Alerts Table**: To enhance inventory management and provide timely notifications regarding stock levels, we added an "Alerts" table. This table is designed to track critical inventory statuses, such as products that are running low, nearing expiration, or currently out of stock. We implemented triggers that automatically log alerts whenever a product's status changes—whether it is about to run out or has expired. This proactive approach enables analysts to respond swiftly to inventory issues, thereby minimizing potential disruptions in supply and ensuring that perishable items are managed effectively.

Aside from these two additions, we have maintained the existing schema for all other tables in the database as previously submitted in our design proposal. In addition to utilizing data from existing sources, we also incorporated mock-up data to ensure the completeness and robustness of our dataset. This mock data allows us to test our application thoroughly and validate its functionality under various scenarios. Overall, these modifications not only optimize our data processing capabilities but also enhance the application's ability to provide timely insights and alerts regarding inventory management.

4. Evolution of the ER Diagram and Table Implementations: Design Changes and Rationale

(*Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?*)

In the development of our application, we made a modifications to the Entity-Relationship (ER) diagram and the table implementations to better meet our project requirements. These changes reflect our evolving understanding of the data needs and the functionality of the application. We introduced a new entity called Alerts to enhance our inventory management capabilities. This entity includes the following attributes:

a. AlertId (Primary Key): A unique identifier for each alert.
b. AlertDateTime: The timestamp indicating when the alert was generated.
c. ProductId (Foreign Key): A reference to the product associated with the alert.
d. ProductName: The name of the product that triggered the alert.
e. AlertType: The type of alert (e.g., low stock, nearing expiration).

f. Quantity: The quantity of the product related to the alert.

The purpose of this Alerts table is to store critical information about inventory items that are either running low or approaching their expiration dates. Each time an update occurs in the database that affects inventory levels, triggers are employed to automatically log relevant information into this table if any product's quantity falls below a predefined threshold.

The Alerts table is designed to have a relationship with the Product table, represented as "Product Has Alerts." This relationship allows us to efficiently track alerts associated with specific products, facilitating better inventory oversight and management.
Aside from these additions, we have maintained the existing schema for all other tables as proposed in our initial design stage. This consistency ensures that we leverage our original design while enhancing functionality through new elements.
We have also implemented various constraints within our schema to enforce data integrity and ensure that relationships between entities are properly maintained. These constraints help prevent issues such as duplicate entries or invalid foreign key references.
Differences Between Original and Final Design
The primary difference between our original design and the final implementation lies in the addition of the Alerts entity. Initially, our design did not account for automated tracking of inventory alerts, which has now become a crucial feature for proactive inventory management. The decision to add an Alerts table stems from our need for a more responsive inventory system that can notify users about critical stock levels and expiration dates. By having a dedicated entity for alerts, we can streamline operations and improve decision-making processes regarding inventory management. Overall, these design changes enhance our application's ability to manage inventory effectively while maintaining clarity in data relationships. The inclusion of the Alerts entity provides significant value by ensuring that users are promptly informed about important inventory statuses, ultimately leading to better resource management and operational efficiency.

5. Functionality Adjustments: Additions and Removals Explained

(Discuss what functionalities you added or removed. Why?)

In the course of developing our application, we made several important adjustments to the functionalities offered. These changes include both additions and removals, each driven by specific needs and priorities.

Additions: We introduced new web pages specifically designed for administrators to perform CRUD (Create, Read, Update, Delete) operations on the backend data. This enhancement allows administrators to efficiently manage and update data directly through a user-friendly interface rather than relying on backend scripts or database management tools. By enabling these operations via web pages, we improve usability and streamline the process of maintaining accurate and up-to-date information within the application. This functionality is crucial for ensuring that the data reflects current inventory levels, product details, and other relevant information, thereby supporting effective decision-making.

Removals: We made the decision to remove the functionality for downloading information as a PDF report. This choice was based on the realization that implementing

this feature would redirect our focus from enhancing the database components of the application to developing additional Python functionalities. Given our goal of strengthening the backend database capabilities, we prioritized efforts on integrating features such as Stored Procedures and Triggers, which are essential for maintaining data integrity and automating processes within the database.

The removal of the PDF download feature allows us to concentrate on optimizing our database structure and improving overall application performance. While this specific functionality was eliminated, all other existing features have been retained and remain fully operational. This ensures that users still have access to essential functionalities while we enhance the underlying architecture of the application.

In summary, our adjustments reflect a strategic approach to enhancing our application's capabilities. By adding web-based CRUD functionalities for administrators, we improve data management efficiency. Simultaneously, by removing less critical features like PDF downloads, we can focus on building a robust backend that supports our application's long-term goals. These changes ultimately contribute to a more effective and user-friendly system that meets the needs of both administrators and analysts.

6. Integration of Advanced Database Programs: Enhancing Application Performance

(Explain how you think your advanced database programs complement your application.)

To optimize the performance and functionality of our application, we have integrated several advanced database programs that significantly enhance its capabilities. These enhancements are crucial for ensuring efficient data management, maintaining data integrity, and automating processes. Below are the key components we have implemented:

1. Transactions

We have utilized database transactions to guarantee that all operations related to a single action are completed successfully before committing any changes to the database. For instance, when a new order is placed, multiple entries must be recorded in the Orders table, and the Inventory table must be updated accordingly. By leveraging transactions, we ensure that:

- Data is first inserted into the Orders table.
- The Inventory table is subsequently updated to reflect the new stock levels.
- The inventory, order relation is inserted into the "Orders_Contains_Inventories" table.
- The entire process is committed only after all operations for that order are successfully executed.

This approach minimizes the risk of data inconsistency and ensures that the database remains in a reliable state, even in cases of failure or error during the transaction process.

2. Constraints

To maintain data integrity and enforce business rules within our database schema, we have implemented various constraints at both the attribute and tuple levels:

Attribute Level Constraints:

- Foreign Key and Primary Key constraints ensure proper relationships between tables.
- Email and password formats are validated to adhere to established rules.
- Product quantities and unit prices cannot be negative, preventing invalid entries.

Tuple Level Constraints:

In the Inventory table, we enforce a constraint that ensures the expiry date of a product is always later than its manufacture date. This physical rule helps maintain logical consistency within our data.

These constraints help prevent erroneous data entry and ensure that all records adhere to predefined rules, thus enhancing overall data quality.

3. Triggers

Triggers have been implemented to automate specific functionalities within our database. This automation reduces manual intervention and enhances operational efficiency. Key use cases for triggers in our project include:

Creation of the Wide Table: To support daily forecasting needs, we refresh the wide table each day. A trigger checks if the current date differs from the last update date at midnight. If so, it automatically refreshes the wide table for use in predictions.

Updating Alerts/Inventory Table: Whenever new data is added or updated in the Orders table, triggers automatically update the Inventory table to adjust product quantities accordingly. Additionally, if any product's quantity falls below a certain threshold, the Alerts table is updated without requiring manual checks.

This automation ensures that our application remains responsive to changes in inventory levels and can proactively alert users about critical stock statuses.

4. Stored Procedures

Stored procedures have been utilized to encapsulate complex operations within the database. By defining procedures for frequently executed tasks, we improve performance by reducing network traffic between the application and database server. Stored procedures allow us to execute multiple SQL statements as a single call, optimizing resource usage and enhancing efficiency. We have utilized stored procedure to add orders in the database. The procedure is implemented such that at first the order is added in the Order table and the rest is executed via the transaction.

The integration of these advanced database programs—transactions, constraints, triggers, and stored procedures—significantly enhances our application's performance and reliability. By ensuring data integrity through constraints and automating processes with triggers and stored procedures, we create a robust system capable of handling complex operations efficiently. These enhancements not only improve user experience but also lay a solid foundation for future scalability and functionality within our application.

7. Technical Challenges Faced: Insights and Lessons Learned for Future Teams

(Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.)

- A significant technical challenge we faced was implementing dynamic cross-chart filtering to ensure the search bar could seamlessly filter data across two distinct charts—bar and doughnut—while maintaining responsiveness. Each chart used different datasets, requiring a unified filtering logic to reconcile varying structures such as product names and quantities. Initially, this led to performance issues, particularly with real-time search suggestions and frequent re-renders during typing. We overcame this by introducing a debounce mechanism to delay search processing, precomputing unique product names for efficient suggestion generation, and isolating chart configurations to minimize unnecessary re-renders. These optimizations ensured a responsive and scalable filtering system while delivering a smooth user experience.

- Initially, our team performed cost analysis for complex join queries during the design phase, using projected data volumes. However, as the application transitioned to testing and production, the database grew significantly due to the insertion of real-world and test data. This growth altered the query performance equations, resulting in higher computational costs and degraded query efficiency. We needed a method to re-evaluate and optimize query costs dynamically as data volumes increased, without halting production or compromising testing timelines.
- One significant technical challenge our team faced was the excessive time taken to execute SQL queries that joined multiple tables while preparing data for our machine learning algorithms. As the size of our dataset grew, these complex joins severely impacted our model training and evaluation speed. To overcome this issue, we decided to implement a denormalization strategy by creating a wide table that consolidated essential data from various sources into a single flat structure. This approach significantly reduced query execution time and improved overall performance, allowing us to iterate more quickly on our models. This experience taught us the importance of balancing normalization and denormalization based on performance needs, as well as the value of continuous monitoring and documentation for future maintenance.
- One challenge we faced was implementing backend APIs to manage inventory data and ensure it synced correctly with the front end in real-time. The APIs needed to handle tasks like adding new products, updating quantities, and removing expired items, all while maintaining data accuracy and quick response times. Initially, we encountered issues with inconsistent data syncing, where changes in the database weren't reflected correctly on the front end due to inefficient query structures and missing validation checks. Debugging these issues revealed that some updates were failing silently due to conflicts in database constraints or API timeouts. To solve this, we optimized the database queries by indexing frequently accessed columns and restructuring the logic to minimize redundant operations. We also added validation layers to ensure that only accurate and complete data could be processed by the API. Additionally, we implemented detailed error logging to quickly identify and resolve issues during data updates. To improve communication between the backend and front end, we ensured the APIs sent structured and consistent JSON responses, making it easier for the front end to handle updates dynamically. This experience helped us understand the importance of building efficient, reliable backend systems and maintaining clear, consistent data flow between components.

8. Comparative Analysis: Changes from Original Proposal to Final Application

(Are there other things that changed comparing the final application with the original proposal?)

In comparing our final application to the original proposal, several noteworthy changes and enhancements emerged that reflect our evolving understanding of the project requirements and user needs.

1. **Enhanced Database Structure**: One of the most significant changes was the addition of two new tables: the **Stock_Smart_Wide** table and the **Alerts** table. The wide table was created to consolidate data from multiple sources, facilitating more efficient machine learning training and prediction processes. The Alerts table was introduced to track critical inventory statuses, such as low stock and impending expirations, which were not explicitly detailed in our original proposal.

2. **Refined User Interface (UI)**: While the original proposal outlined basic UI functionalities, our final application features a more sophisticated and user-friendly interface. Through collaborative brainstorming sessions, we developed a design that enhances user experience, making it easier for administrators to perform CRUD operations and for analysts to access critical data.

3. **Role-Based Access Control**: The implementation of role-based access control was further refined in the final application. Initially, our proposal mentioned basic user permissions; however, we established a clear distinction between administrator and analyst roles. This ensures that only authorized personnel can modify data while allowing analysts to view dashboards and reports securely.

4. **Automated Processes**: The final application incorporates advanced database concepts such as triggers and stored procedures more extensively than initially proposed. These features automate various tasks, such as updating inventory levels based on order entries and refreshing the wide table daily for accurate forecasting, thus improving operational efficiency.

5. **Focus on Backend Optimization**: While our original proposal emphasized both frontend and backend development, we ultimately decided to prioritize backend optimizations using advanced database techniques. This shift allowed us to enhance data integrity, performance, and automation capabilities significantly.

9. Future Enhancements: Opportunities for Improvement Beyond the Interface

(Describe future work that you think, other than the interface, that the application can improve on)

We believe that although we have achieved our goals proposed in the initial stages, there are a few other aspects which can be enhanced in the future –

1. *Enhancement of the creative component – We have leveraged Machine Learning Algorithm (Linear Regression) to predict the quantity of the product that is required for the following day. However, we feel that we need to compare the accuracy and performance of various algorithms to decide which algorithm would give the most accurate results.*
2. *Sending Email Notifications – We could add an additional feature where the application could directly send notifications to the user.*
3. *Addition of Multiple Regions – We could scale up the application and database such that it is catering to multiple regions and stores and not a single store.*

10. Team Dynamics and Task Allocation: Reflections on Collaboration and Management

(Describe the final division of labor and how well you managed teamwork.)

At the outset of our project, our entire team came together to brainstorm the key features we wanted to implement, as well as strategies to enhance database performance through advanced database concepts. This collaborative effort allowed us to align our vision and identify the functionalities necessary for our application.

Following our initial discussions, we focused on designing the user interface (UI) and determining how to effectively implement the desired functionalities within that framework. To ensure clarity and accountability, we compiled a comprehensive list of tasks along with their expected timelines.

During our brainstorming sessions, we focused on how to efficiently implement the database by leveraging the advanced concepts we had learned. We discussed various

strategies for optimizing performance, such as using transactions to ensure data integrity, applying constraints to maintain data quality, and implementing triggers to automate processes. By collaboratively exploring these advanced database techniques, we identified specific areas where they could enhance our application's functionality and responsiveness.

The final division of labor was structured as follows: we split the team into two groups. One group concentrated on frontend development, which involved creating an intuitive interface and integrating it with the data retrieved from the backend. The other group focused on backend development, which included designing the database architecture and ensuring seamless data retrieval to supply the frontend with necessary information.

To maintain effective communication and track progress, we held daily meetings. During these calls, we reviewed the status of tasks from the previous day and established goals for the upcoming meeting. This regular check-in not only kept everyone accountable but also fostered a sense of teamwork and collaboration.

Additionally, we implemented a systematic approach to merging code branches during our calls. By doing so, we minimized potential conflicts and ensured that all components were integrated smoothly. This collaborative merging process was vital in maintaining coherence between the frontend and backend developments.