Godfrey Lozada, Rajeev Thundiyil
Professor Tsotras
Database Management CS166
2 December 2022

Project Report

Log in:

1. **Create User:** If the user wants to create a new account for themselves, they must have a unique username to go along with it. A query is used to make sure there are no matching usernames, and the function also now checks to make sure the new password is greater than 3 characters long.
2. **Log In:** When logging in, the user is now given a message if the username/password combination they have put in does not match any pairing in the database.

Menu:

1. **Menu Layout:** Functions specific to Manager accounts are marked with “[M]”, and Admin accounts are marked with “[A]”. If a lower-level user tries to access a higher-level function, an admin/manager check will be implemented and will deny the user access.
2. **viewStores(Retail esql, String user)**
    a. View stores within 30 units of the user after checking the distances between the user’s longitude and latitude and the stores’ longitude and latitude
3. **viewProduct(Retail esql , String user)**
    a. User can input a storeID in order to check its products. It views the products only if the store is within 30 units of the user.
4. **placeOrder(Retail esql, String user)**
    a. The user can see stores within 30 units of them, and can select a store to order from. They then can view the products that are available in each store
    b. The user orders the product they want, along with the number of units they want
    c. Function adds the order into the order table
5. **viewRecentOrders(Retail esql, String user)**
    a. This function takes in the userID of the user logged in, and proceeds to run a query in which it gets the order information of the 5 recent orders
        i. We manage to get the recent orders by arranging the orderNumber in descending order
6. **updateProduct(Retail esql, String manager)**
    a. A check is made to make sure that the user logged in is a manager
    b. Manager can see the stores they manage, and can select a store they want to update. It will then show the store’s products
    c. They can select a product and change the number of units and price per unit
7. **viewRecentUpdates(Retail esql, String manager)**
    a. Check to see if user is a manager
    b. Queries run by getting the 5 recent manager updates by putting the updateNumber into descending order in order to get the most recent one
8. **viewPopularProducts(Retail esql, String manager)**

    a. Popular products are shown by searching for the products that have the most units ordered

9. **viewPopularCustomer(Retail esql, String manager)**
    a. Popular customer are shown by searching the amount of orders they do before selecting the ones who have done the most.
10. **placeProductSupplyRequests(Retail esql, String manager)**
    a. The manager chooses the store, item, warehouse, and number of units in order to set up a delivery from a warehouse to a store. ProductSupplyRequests table is updated accordingly
11. **viewUsers(Retail esql, String admin)**
    a. The admin can insert a full name or part of a name and the application will return all users that have a matching substring.
12. **updateUserInfo(Retail esql, String admin)**
    a. The admin specifies the user they want to modify, and is then given an option to change their name, password, and location. Queries are then ran to search for the corresponding row in Users and updated the information.
13. **peekManagerData(Retail esql, String admin)**
    a. Lists all managers for all stores. Uses a query that returns storeID, store name, User ID, and Username.
14. **checkIfManager(Retail esql, String manager)**
    a. Helper function to check if a user is a manager.
15. **checkIfAdmin(Retail esql, String admin)**
    a. Helper function to check if a user is an Admin.

Triggers:

1. **Product_supply_table:** Before every new insert on the ProjectSupplyRequests table, this trigger inserts the next serial number into the "requestsNumber" column. Additionally, it runs an update on the Product table, and adds the "unitsRequested" value from ProductSupplyRequests to the "numberOfUnits" value of the respective row in product, matching the storeID and productName.
2. **Product_update_table:** This trigger ensures that for every row that is inserted to the "ProductUpdates" table, the next serial number will be added to "updateNumber" and the "CURRENT_TIMESTAMP" is added to "updatedOn".
3. **Product_order_table:** Similar to "Product_Supply_table", it populated the "orderNumber" column with the next serial number, and the "orderTime" with the correct timestamp before every insert. It also updates the Product table, changing the numberOfUnits and subtracting the units ordered.

Problems along the way: Along the way the only main issues we encountered were trying to get the right queries done with proper syntax. We also had issues with running multiple triggers concurrently. Often, one trigger would trigger another one accidentally, specifically triggers 1 and 3, since they both updated the Product table on initial implementation. To resolve this, we set triggers to run only when their respective tables were modified, and not the product table itself. We also noticed the console would get crowded with text after just a few queries, so a flush was used after every function.