Phase 3 Report

Special compiling requirement: Absolute paths are used for load_data.sql, change if necessary.

Carlos Alberto Velazquez Medina

viewStores

For this implementation, I added code to create_tables.sql to define a function called calculate_distance which takes the latitude and longitude. Afterward, I used a query that returns the storeID, store name, and distance of those stores which are within 30 miles of a specific user's location.

viewProducts

To implement this function, I used a query that returns the productName, numberOfUnits, and pricePerUnit for a store with a specific ID.

placeOrder

To implement this function, I begin by receiving input for Store ID, Product Name, and Number of Units. After this, I use a query that returns the latitude, and longitude of stores with a store. Then, I use a query that returns the latitude, longitude of the current user. I access the store's latitude and longitude and also the user's latitude and longitude. I calculate the distance between the store and user and check if the distance is less than 30. If it is, I insert the order into the Orders table. If not, I tell the current user that the store isn't within 30 miles

viewRecentOrders

a query that returns storeID, storeName, productName, unitsOrdered, and orderTime of the last 5 orders of a customer at a store

updateProduct

I use a query that returns the type of a user and I access the user's type and store it into userType String. Afterward, I check that the current user is a manager. If not, then I tell the user that they're not a manager. If so, I ask for the StoreID, Product Name of the product they want to update and ask what the new number of units and price per unit will be. I use a query to return a store's attributes that are run by a particular manager. I check if anything is returned. If not, I tell the user that they're not the manager of that store. If so, then the manager runs that specific store so now I check if the product exists in the store. If not, I tell the user/manager that the selected product doesn't exist in the store. If so, I update the product table to include the product's updated information and insert the update into the product update.

viewRecentUpdates

I use a query that returns the type of the user. If the user is a manager, then I run a query that returns the updateNumber, managerID, storeID, productName, and updateOn of the last 5 updates to a product. If not, then I tell the user that they're not a manager.

Isiah Montalvo

viewPopularProducts

This is a function that when the user is a manager or admin, it executes a query that selects the 5 most popular products at a managers store. It does this by using the sum aggregate function to sum the total amount of units ordered for products in the Orders table from only the stores

that the user is a manager of. The records are then returned in descending sorted order to give the 5 products with the most product ordered.

viewPopularCustomers

This is a function that when the user is a manager or admin, it executes a query that selects the 5 most popular users at any of the given stores the user manages. This is done by using the count aggregate function to determine the total amount of orders each individual customer places at a given store that the user manages. The table is then sorted in descending order by the total counts and displays the top 5 customers who placed the most orders at a given store. The information displayed includes the customers id, name, and type as well as the total amount of orders they have placed.

placeProductSupplyRequests

This is a function that when the user is a manager or admin, it allows the user to place a product supply request. This is done by first taking the user input for the product name, the units requested, and the warehouse id. These variables along with the users manager id is formatted into a query that inserts the record of these variables into the ProductSupplyRequests table and then displays this inserted record to the user. This is done by sorting the table in descending order based on the serialized request number and returning only the first record which is the inserted record. Then, the number of units of the ordered product is updated within the product table ensuring that only the inventory of the store that the user manages is updated. This product update is then also inserted into the ProductUpdates table to be later displayed in the viewRecentUpdates function.

viewRecentOrders

Updated this function to utilize and execute different queries based on whether the user is a customer or a manager. If the user is a customer, their recently placed orders are displayed to them. If the user is a manager, then they are able to display all distinct customer records for every order placed at a store that they are the manager of.

Initialized Admin Privileges

Updated every function that allows manager-specific queries to also allow for admin privileges. This is done by treating the admin essentially as the manager of every store within the database by allowing the admin to use the managerID of any manager to access their stores and update their records freely. An if statement is used to determine if the user is an admin, and if they are, all manager ids are displayed to them which are used to choose store locations to update.

Error-handling and User-Friendly GUI

Updated most functions with error handling specific to the newly implemented admin privileges. This was done by updating previous error messages to account for admins and utilizing a while loop to ensure that the user can only enter valid inputs, otherwise an error message is displayed and they are asked to try a valid input. Also updated functions that require user input with a list of valid options to choose from when valid options are limited. More detailed and explicit instructions are given for certain functions and when input is invalid, detailed error messages are given to choose valid input.