CS 166 Project Report

Group Information

Group #

Alex Tom - atom015
William Dang - wdang008

Implementation Description

Include high-level description of your implementation (1 paragraph max)

Our implementation allows users to place and track food orders in a pizza store database. It includes features like placing orders, viewing order history, and managing user roles. The system interacts with a PostgreSQL database, using SQL queries for data retrieval and updates. User roles (e.g., customer, manager, driver) determine access permissions. Error handling ensures robustness, and input validation prevents invalid entries. The design emphasizes modularity and efficiency, enabling smooth order processing and data management.

Include screenshots and/or code snippets for each query. In addition, explain how you implemented each query and what purpose it fulfills. (no more than 3-4 sentences per query)

```
String createUserQry = ("INSERT INTO USERS (login, password, role, favoriteItems,
phoneNum) VALUES ('" + username + "', '" + password + "', 'Customer', '', '" +
phoneNum + "');");
```

- Inserts new user values into the USERS table. The user is asked to input strings before this query is executed.

```
String query = "SELECT * FROM Users WHERE login = '" + username + "' AND password = '"
+ password + "';";
```

- Retrieves the credentials of a user and compares them to the login information. A table is created from the select query and if it contains 2 values, then user is successfully logged in

```
String userProfile = String.format("SELECT * FROM USERS WHERE login = '%s';",
username);
```

- Retrieves all information from the user table when a user decides to view their profile. The get command is used to print out the values for the user to view.

```
updateQry = String.format("UPDATE USERS SET favoriteItems = '%s' WHERE login = '%s';",
userchoice, username);
```

- Updates the user's favoriteItem. The update profile function takes input from the user and updates their favorite item in the User table. This function uses similar queries to update the user password, phone number, and role (only manager can access).

```
1.viewMenuQry = "SELECT * FROM ITEMS";
```

```
2.viewMenuQry = String.format("SELECT * FROM ITEMS WHERE typeOfItem = ' %s'",
userChoice);
3.viewMenuQry = "SELECT * FROM ITEMS WHERE price <= " + price;
```

- The viewMenu function has 3 selections. First query allows users to view the full menu. The second query takes input from the user and compares it to “typeOfItem” in the table and displaying it afterwards. The third query displays all items under a certain price after taking input from the user (and ordering it by descending or ascending).

```
locateQry = String.format("SELECT * FROM STORE WHERE storeID = %d", storeID);
locateQry = String.format("SELECT * FROM ITEMS WHERE itemName = '%s';", userinput);
String orderQry = String.format("INSERT INTO FOODORDER (login, storeID, totalPrice,
orderTimestamp, orderStatus) " + "VALUES('%s', '%d', '%.2f', NOW(), 'incomplete')
RETURNING orderID;", username, storeID, totalPrice);
tring insertQry = String.format(" INSERT INTO ITEMSINORDER (orderID, itemName,
quantity) VALUES ('%d', '%s', '%d');", orderID, items.get(i), quant.get(i));
```

- From the place order function. First takes input from the user to determine what store they want to order from and creates a table for that. Then takes input to determine the item they want to order and creates a table for that. orderQry inserts a creates a new FoodOrder and insertQry creates a new itemsInOrder

```
String query = "SELECT * FROM FoodOrder WHERE login ='" + targetUser + "';";
```

- Creates a table of all the orderID’s from the customer or another user (manager access). Used a for loop and a print statement to display the table contents.

```
String query = "SELECT orderID, orderTimestamp, orderStatus FROM FoodOrder WHERE login
= '" + targetUser + "' ORDER BY orderTimestamp DESC LIMIT 5;";
```

- Creates a table of the 5 most recent orderID’s from the customer or another user (manager access). Used a for loop and a print statement to display the table contents.

```
String orderQuery = "SELECT orderTimestamp, totalPrice, orderStatus, login " +
                              "FROM FoodOrder WHERE orderID = '" + orderID + "';";
String itemsQuery = "SELECT itemName, quantity FROM ItemsInOrder WHERE orderID = '" +
orderID + "';";
```

- From viewOrderInfo. orderQuery creates a table of all the information about a specific orderID given through input by the user. “.get” functions and a print statement is used to display the contents. itemsQuery is used to find the item name and quantity in the specific orderID

```
String viewStoresQry = "SELECT * FROM STORE;";
```

- viewStoresQry creates a table of all the stores. “.get” functions are used to display each and every value from the STORE table.

```
String query = "UPDATE FoodOrder SET orderStatus ='" + orderStatus + "' WHERE orderID
='" + orderID + "';";
```

- Update query to update the order status. User inputs the new orderStatus and inputs the desired order to be changed.

```
1.upQry = String.format("UPDATE ITEMS SET itemName = '%s' WHERE itemName = '%s'",
userChoice, I);
2.upQry = String.format("INSERT INTO ITEMS (itemName, ingredients, typeOfItem, price,
description) VALUES ('%s', '%s', ' %s', '%f', '%s');", iN, In, tI, pR, dE);
```

- The first upQry updates the itemName to a new itemName from user input. Similar queries are used to update ingredients, typeOfItem, etc. The second upQry allows the user (manager) to add a new item

```
String roleQuery = "SELECT role FROM users WHERE username = '" + username + "';";
StringBuilder queryBuilder = new StringBuilder("UPDATE Users SET ");
queryBuilder.append("role = '").append(newRole).append("', ");
```

- From updateUser function. roleQuery is used to find the role of the current user (if not manager states that user does not have permission). queryBuilder basically constructs dynamic update queries that allows the manager to update the information of a user (role, favoriteItem, etc).

If you did any extra credit, provide screenshots and/or code snippets. Explain how you implemented the extra credit. (triggers/stored procedures, performance tuning, etc)

Implementation of trigger:

```
1   DROP SEQUENCE IF EXISTS orderID_Seq;
2
3   DROP TRIGGER IF EXISTS orderID_trigger ON FoodOrder;
4
5   DROP FUNCTION IF EXISTS generateOrderID();
6
7   CREATE SEQUENCE orderID_Seq START 10004; -- Previous Order ends on 10003
8
9   CREATE OR REPLACE FUNCTION generateOrderID() RETURNS TRIGGER AS $$
10  BEGIN
11    NEW.orderID := nextval('orderID_Seq');
12    RETURN NEW;
13  END;
14  $$ LANGUAGE plpgsql;
15
16  CREATE TRIGGER orderID_trigger
17  BEFORE INSERT ON FoodOrder
18  FOR EACH ROW
19  EXECUTE PROCEDURE generateOrderID();
```

- This trigger is used to create unique order IDs. Whenever a new row is inserted into FoodOrder, the orderID_trigger executes generateOrderID(), which assigns the next value from orderID_Seq to the orderID column. This eliminates the need for manually specifying orderID, ensuring uniqueness and sequential numbering

Implementation of indexes:

```
DROP INDEX IF EXISTS itemsOrder;
DROP INDEX IF EXISTS foodTime;

CREATE INDEX itemsOrder ON ItemsInOrder(orderID);
CREATE INDEX foodTime ON FoodOrder(login, orderTimestamp DESC);
```

The first index speeds up the select query inside the viewOrderInfo function. The second index speeds up the select queries in viewOrderInfo, viewAllOrders, and viewRecentOrders.

Problems/Findings

Include problems/findings you encountered while working on the project (1-2 paragraphs max)

Some problems we had encountered were first figuring out what functions did what and what functions returned what. Looking over the given code was one of the harder steps before being able to actually apply the functions. Another thing was figuring out how to create a unique ID, we eventually learned about triggers and that was very helpful in being able to provide the unique ID's for the orders.

Contributions

Include descriptions of what each member worked on (1 paragraph max)

William - Implemented Trigger File and Worked on a couple of the functions like UpdateProfile, ViewMenu, placeOrder, part of ViewAllOrders and ViewRecentOrders and UpdateMenu.

Alex - Created indexes, worked on updateUser, updateOrderStatus, viewOrderInfo, viewRecentOrders, and viewAllOrders