CS 166 Project Report

Group Information
Group 4
Alexander Tran, atran388
Bryan Duan, bduan004

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

PizzaStore.java
This file is the completed version of the PizzaStore.java template file that was provided. In this file, the basic operations required of a pizza store user interface and its database association were completed including creating a new account, food orders, and the store. Within each option choice the user is prompted to provide information relevant to the option chosen. For example, when creating a user, the user will be prompted to enter in their login, password, and phone number. Formatting errors and outlier cases are taken into account and will result in an error message popup, upon which the user will be exited back to the main menu containing all of the choice options that they may choose. Some options are limited to certain user roles only. These roles will take the user's role into consideration before allowing access to the option. If say a customer role was trying to access option 11 update user, which should only be functional for managers, our program will display an error message stating that the user's role does not have access to this functionality and the main menu will reappear for the user to choose another option.

CreateUser function:
```
String query = String.format(
            "INSERT INTO Users (login, password, role, favoriteItems,
phoneNum) VALUES ('%s', '%s', 'customer', NULL, '%s');",
            login, password, phone);
         esql.executeUpdate(query); // create user in db
         System.out.println("User created successfully!");
```
Upon choosing the option to create a user, the program runs the CreateUser function which prompts the user for their account's login, password, and phone number. The query then takes this information into account and stores their account's login, password, and phone number into the database. Role is automatically defaulted to customer and favoriteItems is initiated to NULL which can be altered with the updateProfile function.

LogIn function:

```
String query = String.format(
            "SELECT * FROM Users WHERE login = '%s' AND password = '%s';",
            login, password);
         int userCount = esql.executeQuery(query);
```

The user is prompted to provide their login and password to gain access to their account. This information is used in this query and the number of tuples containing the specified login and password are returned into userCount, which is to be used later to determine if there was a successful login or an unsuccessful login.

viewProfile function:

```
String query = String.format(
            "SELECT favoriteItems, phoneNum FROM Users WHERE login = 
'%s';",
            login);
         List<List<String>> result = 
esql.executeQueryAndReturnResult(query);
```

The user's login information is stored within the function parameter and as such this query uses this valid login to search the database for their favoriteItems and phoneNum. This information is needed so that the viewProfile function can display this information which is uniquely inputted by each user for their own account profile.

updateProfile:

```
String updateFavoriteItemQuery = String.format(
                  "UPDATE Users SET favoriteItems = '%s' WHERE login = 
'%s';",
                  newFavoriteItem, login);
               esql.executeUpdate(updateFavoriteItemQuery);
```

```
String updatePhoneQuery = String.format(
                  "UPDATE Users SET phoneNum = '%s' WHERE login = '%s';",
                  newPhone, login);
               esql.executeUpdate(updatePhoneQuery);
```

```
String updatePasswordQuery = String.format(
                  "UPDATE Users SET password = '%s' WHERE login = '%s';",
                  newPassword, login);
               esql.executeUpdate(updatePasswordQuery);
```

These queries are used to update the user's information (specifically favoriteItems, phoneNum, and password) whose login is given by the variable login (as login is the primary key of the Users table, each update would only update the one account profile).

viewMenu function:

```
String itemMenu = String.format(
                  "SELECT * FROM Items;");
               List<List<String>> result =
esql.executeQueryAndReturnResult(itemMenu);
```

This query contains all the tuples and information stored in the Items table. It is used to display the full menu without any sort of sorting or filtering applied.

```
String typeQuery = String.format(
                  "SELECT * FROM Items WHERE typeOfItem IN ('%s');",
                  String.join("', '", types));
               List<List<String>> typeResult =
esql.executeQueryAndReturnResult(typeQuery);
```

This query contains all of the tuples and information of the Items table in which the item's typeOfItem is the one specified by the user.

```
String priceQuery = String.format(
                  "SELECT * FROM Items WHERE price <= %d ORDER BY price
DESC;"
                  , maxPrice);
               List<List<String>> priceResult =
esql.executeQueryAndReturnResult(priceQuery);
```

This query contains all information about each item whose price is lower than the specified maxPrice and is ordered from highest to lowest pricing.

```
String priceQueryTwo = String.format(
                  "SELECT * FROM Items WHERE price <= %d ORDER BY price
ASC;"
                  , maxPriceTwo);
               List<List<String>> priceResultTwo =
esql.executeQueryAndReturnResult(priceQueryTwo);
```

This query does the same as above except it orders the tuples by lowest to highest price.

```
String bothQuery = String.format(
                    "SELECT * FROM Items WHERE typeOfItem IN ('%s') AND
price <= %d ORDER BY price DESC;",
                    String.join("', '", types), maxPrice);
                List<List<String>> bothResult =
esql.executeQueryAndReturnResult(bothQuery);
```

This query selects all tuples and all of their information in the Items table returned only if their typeOfItem (specified by the user. multiple types can be included in which case all types added to the query will be returned) and price is as indicated. Items are sorted by price in descending order from highest to lowest.

```
String bothQueryTwo = String.format(
                "SELECT * FROM Items WHERE typeOfItem IN ('%s') AND price
<= %d ORDER BY price ASC;",
                String.join("', '", types), maxPrice);
            List<List<String>> bothResultTwo =
esql.executeQueryAndReturnResult(bothQueryTwo);
```

Similar to the last query but ordered from lowest to highest price.

placeOrder function:

```
String storeQuery = "SELECT storeID, address, city, state FROM Store WHERE
isOpen = true;";
         List<List<String>> stores =
esql.executeQueryAndReturnResult(storeQuery);
```

Returns all but the isOpen information on the stores in which are open.

```
String itemQuery = String.format(
                "SELECT price FROM Items WHERE itemName LIKE '%s';",
                itemName);
            List<List<String>> itemResults =
esql.executeQueryAndReturnResult(itemQuery);
```

Finds all the Items where itemName is the specified name in the variable itemName and returns their price. ItemName is read from user input after prompting and this query is used to not only check if such an item exists in the database, but also to add its price to the total calculated later in the function.

```
String lastOrderQuery = "SELECT MAX(orderID) FROM FoodOrder;";
               List<List<String>> lastOrderResult =
esql.executeQueryAndReturnResult(lastOrderQuery);
```

Used to find the max orderID from the FoodOrder table, to be used to calculate the new order's orderID which is default to 10000 should there be no other items, or to be the highest orderID so far + 1.

```
String insertOrderQuery = String.format(
                  "INSERT INTO FoodOrder (orderID, login, storeID,
totalPrice, orderTimestamp, orderStatus) VALUES (%d, '%s', '%s', %d, '%s',
'%s');",
                  orderID, login, selectedStoreID, totalOrderPrice,
orderTimestamp, orderStatus);
               esql.executeUpdate(insertOrderQuery);
```

Used to insert the values prompted by the user or calculated earlier in the function into FoodOrder to officiate the order.

```
String insertItemsInOrderQuery = String.format(
                     "INSERT INTO ItemsInOrder (orderID, itemName,
quantity) VALUES (%d, '%s', %d);",
                     orderID, itemNameInOrder, quantityInOrder);
                  esql.executeUpdate(insertItemsInOrderQuery);
```

Same as before but only some of the attributes orderID, itemName, and quantity are stored into the ItemsInOrder table instead. Every placed order needs to be stored into both tables and so this query takes care of that end for the ItemsInOrder table.

viewAllOrders function:

```
String roleQuery = String.format(
            "SELECT role FROM Users WHERE login = '%s';",
            login);
         List<List<String>> roleResult =
esql.executeQueryAndReturnResult(roleQuery);
```

Used to gather the role information of the user given their unique login. Used to determine the function's accessibility given their role and different prompts/functionalities are provided based on their role.

```
orderQuery = String.format(
               "SELECT orderID, storeID, totalPrice, orderTimestamp,
orderStatus FROM FoodOrder WHERE login = '%s';",
               login);
```
Stores orderID, storeID, totalPrice, orderTimestamp, orderStatus from the FoodOrder table where the login is the login stored in the login variable. Used to get the information of the orders placed under the user's account.

```
String query = String.format(
               "SELECT login FROM Users WHERE login = '%s';",
               viewLogin);
            int userCount = esql.executeQuery(query);
```
Used to check if a specific login exists in the User table by filtering with the value stored in viewLogin.

```
orderQuery = String.format(
               "SELECT orderID, storeID, totalPrice, orderTimestamp,
orderStatus FROM FoodOrder WHERE login = '%s';",
               viewLogin);
```
Stores orderID, storeId, totalPrice, orderTimestamp, orderStatus from the FoodOrder table where login is stored in the viewLogin variable instead of Login. Also used to get information of the orders placed under the user's account.

viewRecentOrders function:
```
String roleQuery = String.format(
            "SELECT role FROM Users WHERE login = '%s';",
            login);
         List<List<String>> roleResult =
esql.executeQueryAndReturnResult(roleQuery);
```
Retrieves the role of a user from the Users table where login matches the value of the login variable. The purpose of this is to identify the kind of access the user has and grant them access to what's available.

```
orderQuery = String.format(
               "SELECT orderID, storeID, totalPrice, orderTimestamp,
orderStatus FROM FoodOrder WHERE login = '%s' ORDER BY orderTimestamp DESC
LIMIT 5;",
               login);
```

This query retrieves the orderID, storeID, totalPrice, orderTimestamp, orderStatus from Foodorder of the 5 most recent orders from a specific user. This allows the user to quickly track the status of recent orders instead of viewing the whole history.

```
String query = String.format(
               "SELECT login FROM Users WHERE login = '%s';",
               viewLogin);
            int userCount = esql.executeQuery(query);
```

Checks whether a specific login exists within the Users table.

```
orderQuery = String.format(
               "SELECT orderID, storeID, totalPrice, orderTimestamp,
orderStatus FROM FoodOrder WHERE login = '%s' ORDER BY orderTimestamp DESC
LIMIT 5;",
               viewLogin);
```

Retrieves the 5 most recent orders from a specific user int the Food order table. Retrieves the orderID, storeID, totalPrice, orderTimestamp, and orderstatus from FoodOrder where login is stored within viewLogin.

viewOrderInfo function:

```
String roleQuery = String.format(
            "SELECT role FROM Users WHERE login = '%s';",
            login);
         List<List<String>> roleResult =
esql.executeQueryAndReturnResult(roleQuery);
```

Retrieves the role of a specific user from the Users table. roleResult which is a list of list of strings holds the roles from the query.

```
orderQuery = String.format(
               "SELECT orderTimestamp, totalPrice, orderStatus FROM
FoodOrder WHERE login = '%s' AND orderID = '%s';",
               login, orderID);

            queryNum = esql.executeQuery(orderQuery);
```

Query computes the orderTimestamp, totalPrice, orderStatus information of the order in which belong to the user under account login - login and whose orderID is orderID.

```
orderQuery = String.format(
               "SELECT orderTimestamp, totalPrice, orderStatus FROM
FoodOrder WHERE orderID = '%s';",
               orderID);
```

```
            queryNum = esql.executeQuery(orderQuery);
```
Same query as above but this is written under the case in which the user's role is manager or driver.

```
String itemsQuery = String.format(
            "SELECT itemName, quantity FROM ItemsInOrder WHERE orderID = 
'%s';",
            orderID);
         List<List<String>> itemsResults = 
esql.executeQueryAndReturnResult(itemsQuery);
```
Gets the itemName and quantity attributes of orders in the ItemsInOrder table whose orderID is orderID.

viewStores function:
```
String storeQuery = "SELECT storeID, address, city, state, isOpen, 
reviewScore FROM Store;";
         List<List<String>> storeResults = 
esql.executeQueryAndReturnResult(storeQuery);
```
Gets the storeID, address, city, state, isOpen, reviewScore attributes of all stores in the Store table.

updateOrderStatus function:
```
String roleQuery = String.format(
            "SELECT role FROM Users WHERE login = '%s';",
            login);
         List<List<String>> roleResult = 
esql.executeQueryAndReturnResult(roleQuery);
```
Gets the role attribute of the user whose login is login.

```
String orderQuery = String.format(
            "SELECT orderStatus FROM FoodOrder WHERE orderID = '%s';",
            orderID);
```

```
        List<List<String>> orderResult =
esql.executeQueryAndReturnResult(orderQuery);
```
Gets the orderStatus of the order in the FoodOrder table whose orderID is the orderID stored in orderID.

```
String changeQuery = String.format(
            "UPDATE FoodOrder SET orderStatus = '%s' WHERE orderID =
'%s';",
            orderStatus, orderID);
        esql.executeUpdate(changeQuery);
```
Updates the orderStatus attribute of the order in FoodOrder where the orderID is the value stored in orderID.

updateMenu function:
```
String roleQuery = String.format(
            "SELECT role FROM Users WHERE login = '%s';",
            login);
        List<List<String>> roleResult =
esql.executeQueryAndReturnResult(roleQuery);
```
Finds the role of the user with login stored in login.

```
String nameQuery = String.format(
                  "SELECT COUNT(*) FROM Items WHERE itemName = '%s';",
                  itemName);
               List<List<String>> nameResult =
esql.executeQueryAndReturnResult(nameQuery);
```
Finds if there is an item with itemName with value stored in itemName in the database already.

```
String createQuery = String.format(
                  "INSERT INTO Items (itemName, ingredients, typeOfItem,
price, description) VALUES ('%s', '%s', '%s', %d, '%s');",
                  itemName, ingredients, typeOfItem, price, description);
               esql.executeUpdate(createQuery);
```

Inserts the itemName, ingredients, typeOfItem, price, and description attributes with the given values stored in itemName, ingredients, typeOfItem, price, description.

```
String updateNewNameQuery = String.format(
                  "UPDATE Items SET itemName = '%s' WHERE itemName =
'%s';",
                  newName, updateName);
               esql.executeUpdate(updateNewNameQuery);
```

Updates the itemName to the value stored in newName of the item with itemName stored in updateName.

```
String updateIngredientsItemQuery = String.format(
                  "UPDATE Items SET ingredients = '%s' WHERE itemName =
'%s';",
                  newIngredients, updateIngredientsName);
               esql.executeUpdate(updateIngredientsItemQuery);
```

Updates the ingredients attribute to the value stored in newIngredients of the item with itemName stored in updateIngredientsName.

```
String updateItemTypeQuery = String.format(
                  "UPDATE Items SET typeOfItem = '%s' WHERE itemName =
'%s';",
                  newType, updateTypeName);
               esql.executeUpdate(updateItemTypeQuery);
```

Updates the typeOfItem attribute to the value stored in newType for the item with itemName stored in updateTypeName.

```
String updatePriceQuery = String.format(
                  "SELECT * FROM Items WHERE itemName = '%s';",
                  updatePriceName);
               List<List<String>> updatePriceResult =
esql.executeQueryAndReturnResult(updatePriceQuery);
```

Grabs all information of the item with the itemName stored in updatePriceName.

```
String updateItemPriceQuery = String.format(
                    "UPDATE Items SET price = %d WHERE itemName = '%s';",
                    Double.parseDouble(newPrice), updatePriceName);
                esql.executeUpdate(updateItemPriceQuery);
```
Updates the price to newPrice on the item with itemName.

```
String updateItemDescriptionQuery = String.format(
                    "UPDATE Items SET description = '%s' WHERE itemName =
'%s';",
                    newDescription, updateDescriptionName);
                esql.executeUpdate(updateItemDescriptionQuery);
```
Updates the description of items with the itemName stored in updateDescriptionName.

^^ The following queries are repeated multiple times across different cases but the purpose and reasoning behind them are generally the same.

updateUser function:

```
String roleQuery = String.format(
            "SELECT role FROM Users WHERE login = '%s';",
            login);
         List<List<String>> roleResult =
esql.executeQueryAndReturnResult(roleQuery);
```
Contains the role attribute of users with login stored in variable login. Used to determine if the updateUser functionality is accessible to this user based on their role.

```
String checkQuery = String.format(
            "SELECT * FROM Users WHERE login = '%s';",
            mLogin);
         int userCount = esql.executeQuery(checkQuery);
```
Used to determine if there is a user with the login stored in the variable mLogin. Needed to determine if the login stored in mLogin is not in the database already and if it is not, the login in mLogin can be used as a new login.

```
String mQuery = String.format(
                 "SELECT login, role FROM Users WHERE login = '%s';",
                 mLogin);
              List<List<String>> mResult =
esql.executeQueryAndReturnResult(mQuery);
```
Query is used to contain the login and role attributes of users whose login is the same as that stored in mLogin (only one user as login must be unique).

```
String checkLoginQuery = String.format(
                        "SELECT COUNT(*) FROM Users WHERE login = '%s';",
                        newLogin);
                     List<List<String>> checkResult =
esql.executeQueryAndReturnResult(checkLoginQuery);
```
Query is used to calculate whether there are users with the login stored in the variable newLogin. This is used to determine if the login in newLogin is a valid new login as logins must be unique in the database.

```
String updateLogin = String.format(
                        "UPDATE Users SET login = '%s' WHERE login = '%s';",
                        newLogin, mLogin);
                     esql.executeUpdate(updateLogin);
```
Updates the login of the user with login stored in variable mLogin (old login), into the new login stored in variable newLogin.

```
String updateRole = String.format(
                        "UPDATE Users SET role '%s' WHERE login = '%s';",
                        newRole, mLogin);
                     esql.executeUpdate(updateRole);
```
Updates the user's role into the new desired role which was given by the user's input.

Extra credit

Most of our functions utilize the login information in the Users table and so we utilize a hash index on this attribute for this table.

```
DROP INDEX IF EXISTS loginRole;

CREATE INDEX loginRole
ON Users
USING BTREE
(login);
```

Problems/Findings
There was a lot of learning java that went into doing this project which we have not spent too much time on. That made it take a lot longer than expected and caused frustrations with error logics that we have not encountered before given that we come from a background of C++. However once we got the hang of the language we found it to be very straightforward and easy to apply the logic and reasoning we wanted into our code.

Contributions
Both project members helped contribute to the reasoning behind the functionalities of the project and helped each other with syntax errors and such. Work on the doc was split into a do what you can when you can basis and allowed for it to be completed in a timely manner without much hassle.