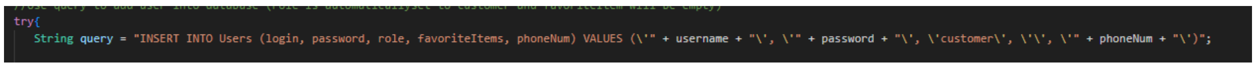
Pizza Delivery Database

Group Information

David Kim
Angel Hernandez

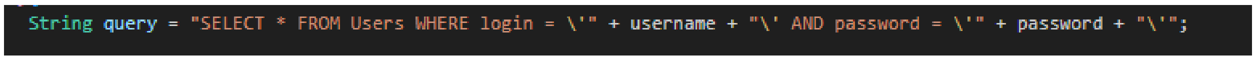
Implementation Description

We implemented the pizza store system using a terminal text interface using Java and PostgreSQL to query the database for the functions. The primary purpose of the program is to provide the user with options to create an account and place orders at this pizza store. There are other features the user has access to to facilitate and support this process (viewing menu, lookup stores, etc.). Some options are limited based on the user's role, which is recognized based on the user's login and role information. There are 3 roles: customer, manager, and driver. All 3 roles have access to the same options, except for a few.



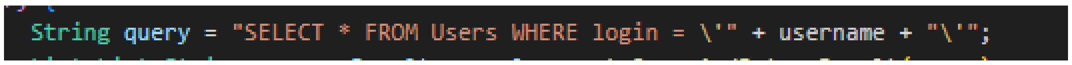
```
try{
    String query = "INSERT INTO Users (login, password, role, favoriteItems, phoneNum) VALUES (\'" + username + "\', \'" + password + "\', \'customer\', \'\', \'" + phoneNum + "\')";
```

This query creates a new user by inserting a new user into the Users table. The user must provide a login, password, and phone number and all other information are filled with preset information.



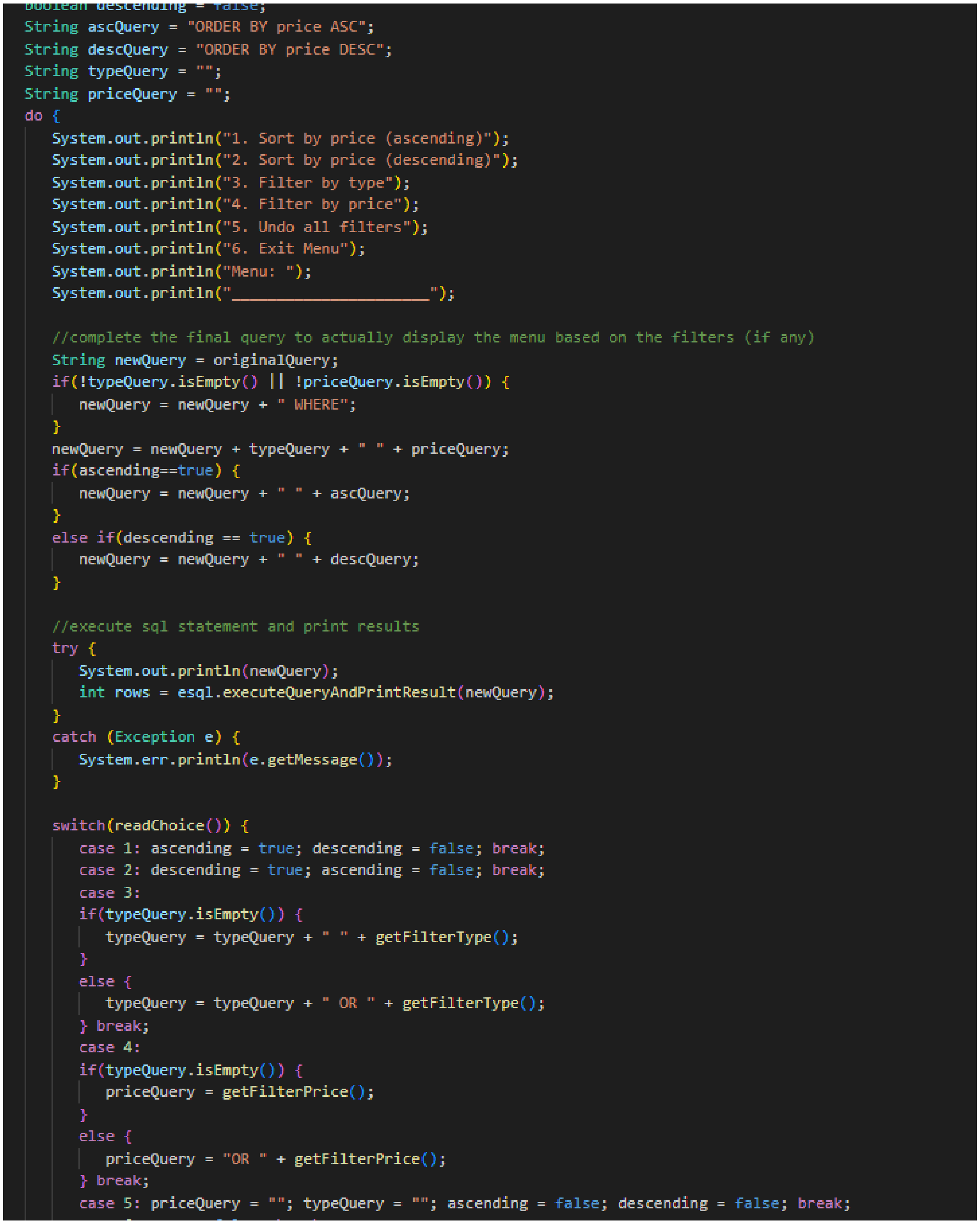
```
String query = "SELECT * FROM Users WHERE login = \'" + username + "\' AND password = \'" + password + "\'";
```

This query returns the information of the user based on the user's login and password. This query is used to confirm whether the login information is correct during login. If the query returns something, then login is successful, but if it returns empty then login fails.

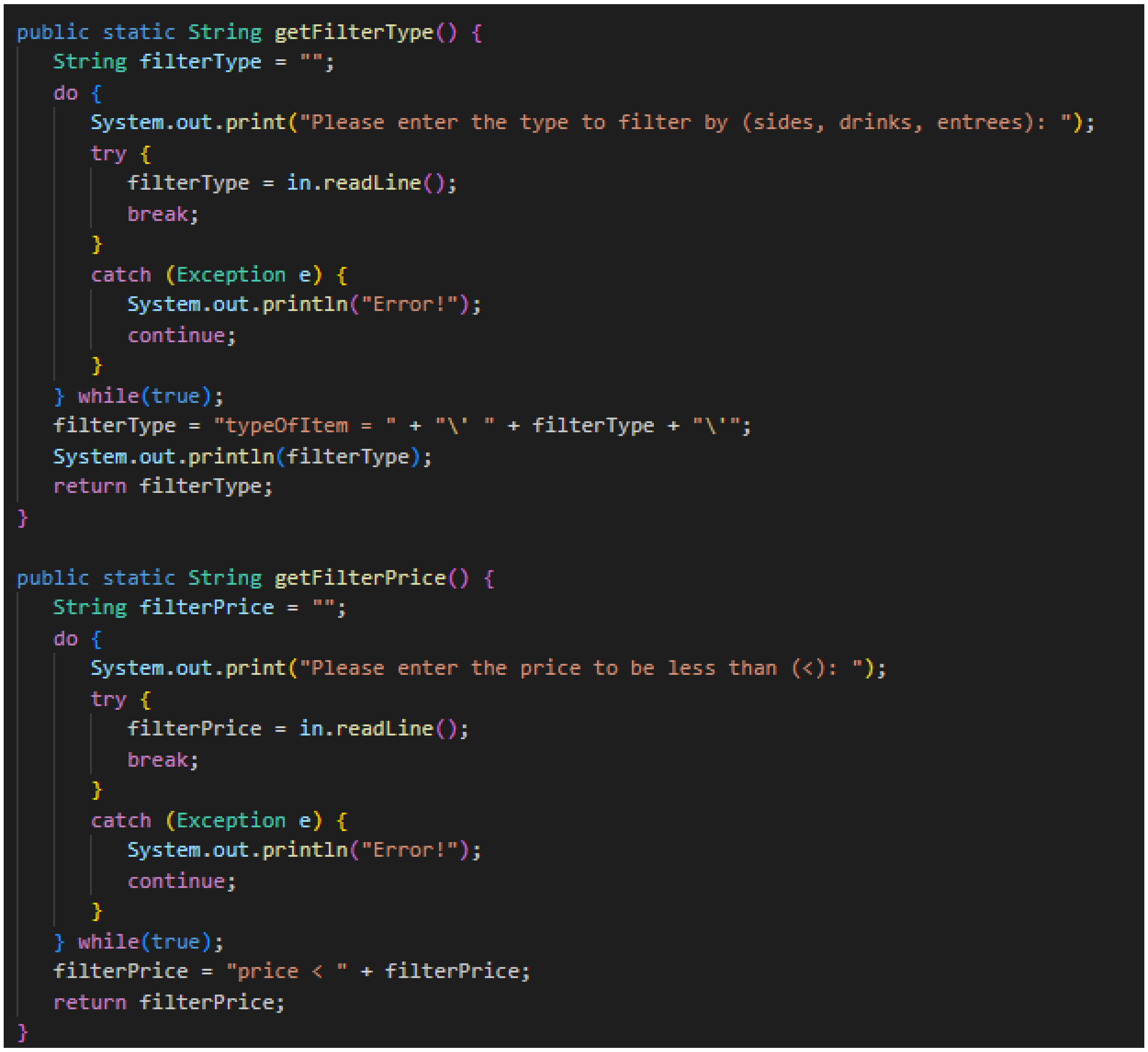


```
String query = "SELECT * FROM Users WHERE login = \'" + username + "\'";
```

Similar to the previous query, this query returns all information of the user matching the login. However, it is different in that it is used to display an already logged in user's information.

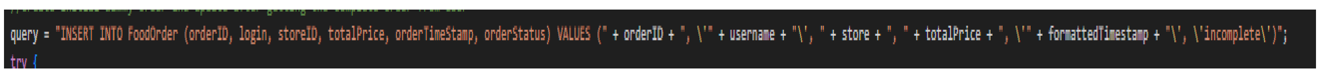


```
boolean descending = false;
String ascQuery = "ORDER BY price ASC";
String descQuery = "ORDER BY price DESC";
String typeQuery = "";
String priceQuery = "";
do {
    System.out.println("1. Sort by price (ascending)");
    System.out.println("2. Sort by price (descending)");
    System.out.println("3. Filter by type");
    System.out.println("4. Filter by price");
    System.out.println("5. Undo all filters");
    System.out.println("6. Exit Menu");
    System.out.println("Menu: ");
    System.out.println("______________________");

    //complete the final query to actually display the menu based on the filters (if any)
    String newQuery = originalQuery;
    if(!typeQuery.isEmpty() || !priceQuery.isEmpty()) {
        newQuery = newQuery + " WHERE";
    }
    newQuery = newQuery + typeQuery + " " + priceQuery;
    if(ascending==true) {
        newQuery = newQuery + " " + ascQuery;
    }
    else if(descending == true) {
        newQuery = newQuery + " " + descQuery;
    }

    //execute sql statement and print results
    try {
        System.out.println(newQuery);
        int rows = esql.executeQueryAndPrintResult(newQuery);
    }
    catch (Exception e) {
        System.err.println(e.getMessage());
    }

    switch(readChoice()) {
        case 1: ascending = true; descending = false; break;
        case 2: descending = true; ascending = false; break;
        case 3:
        if(typeQuery.isEmpty()) {
            typeQuery = typeQuery + " " + getFilterType();
        }
        else {
            typeQuery = typeQuery + " OR " + getFilterType();
        } break;
        case 4:
        if(typeQuery.isEmpty()) {
            priceQuery = getFilterPrice();
        }
        else {
            priceQuery = "OR " + getFilterPrice();
        } break;
        case 5: priceQuery = ""; typeQuery = ""; ascending = false; descending = false; break;
```

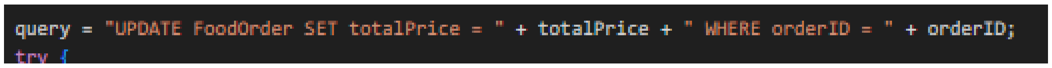


```java
public static String getFilterType() {
   String filterType = "";
   do {
      System.out.print("Please enter the type to filter by (sides, drinks, entrees): ");
      try {
         filterType = in.readLine();
         break;
      }
      catch (Exception e) {
         System.out.println("Error!");
         continue;
      }
   } while(true);
   filterType = "typeOfItem = " + "\' " + filterType + "\'";
   System.out.println(filterType);
   return filterType;
}

public static String getFilterPrice() {
   String filterPrice = "";
   do {
      System.out.print("Please enter the price to be less than (<): ");
      try {
         filterPrice = in.readLine();
         break;
      }
      catch (Exception e) {
         System.out.println("Error!");
         continue;
      }
   } while(true);
   filterPrice = "price < " + filterPrice;
   return filterPrice;
}
```

String newQuery is the actual query used to retrieve the menu items' information based on the filters applied. As you can see, the query is composed of multiple parts and constructed based on the filters applied by the user which are (sorting, price range, type).

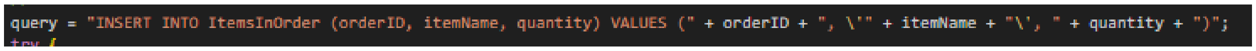


```java
query = "INSERT INTO FoodOrder (orderID, login, storeID, totalPrice, orderTimeStamp, orderStatus) VALUES (" + orderID + ", \'" + username + "\', " + store + ", " + totalPrice + ", \'" + formattedTimestamp + "\', \'incomplete\')";
try {
```



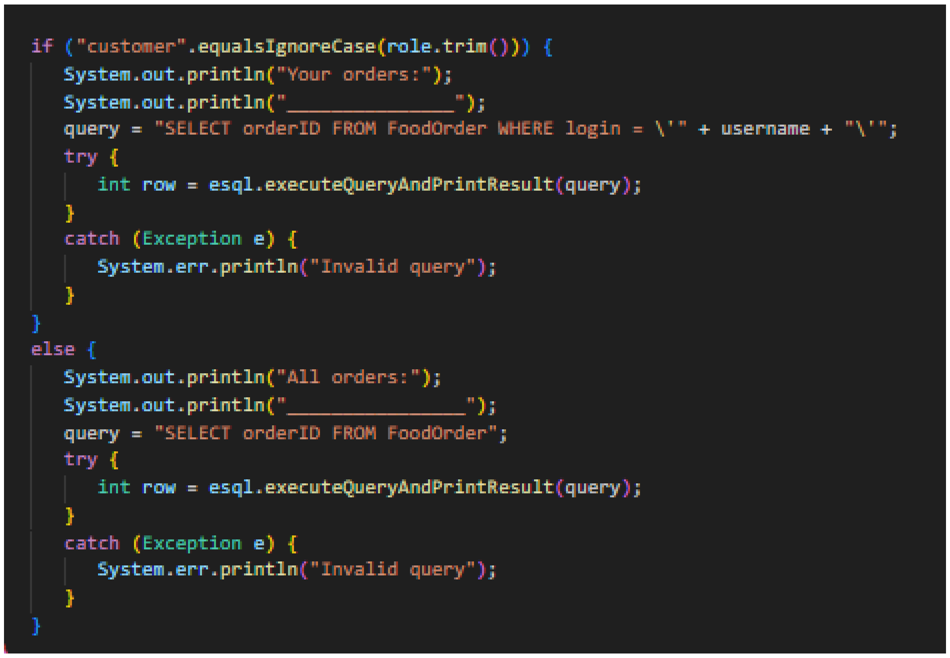
```java
query = "UPDATE FoodOrder SET totalPrice = " + totalPrice + " WHERE orderID = " + orderID;
try {
```

This query implements placing an order. This query inserts a new order into FoodOrder with a generated unique orderID and all the necessary information, including storeID, totalPrice, and the current time. The totalPrice is initially set to 0 at the beginning of the order and later updated once the user finishes selecting all the items for the order.



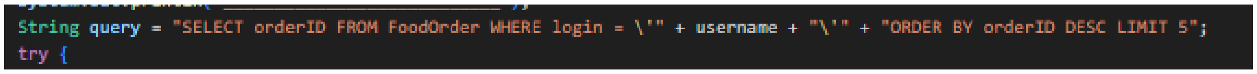
```
query = "INSERT INTO ItemsInOrder (orderID, itemName, quantity) VALUES (" + orderID + ", \'" + itemName + "\', " + quantity + ")";
```

And this query inserts each item in an order into the ItemsInOrder table. This query is implemented after the query inserting into FoodOrder due to the foreign key dependencies with orderID.



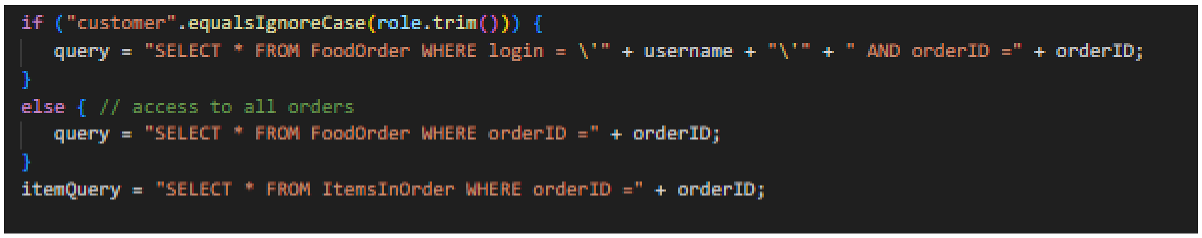
```
if ("customer".equalsIgnoreCase(role.trim())) {
    System.out.println("Your orders:");
    System.out.println("_______________");
    query = "SELECT orderID FROM FoodOrder WHERE login = \'" + username + "\'";
    try {
        int row = esql.executeQueryAndPrintResult(query);
    }
    catch (Exception e) {
        System.err.println("Invalid query");
    }
}
else {
    System.out.println("All orders:");
    System.out.println("_______________");
    query = "SELECT orderID FROM FoodOrder";
    try {
        int row = esql.executeQueryAndPrintResult(query);
    }
    catch (Exception e) {
        System.err.println("Invalid query");
    }
}
```

These queries retrieve the orderID from the FoodOrder table. Based on the user's role a different query is executed. A normal customer can only see their own orders, while drivers and managers can see all orders placed.



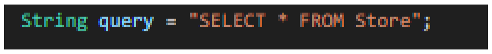
```
String query = "SELECT orderID FROM FoodOrder WHERE login = \'" + username + "\'" + "ORDER BY orderID DESC LIMIT 5";
try {
```

This query is similar to the customer's query for retrieving the orderID from the table. However, it is limited to the 5 most recent orders, and is used for all user's regardless of role to show the 5 most recent orders placed by their account.



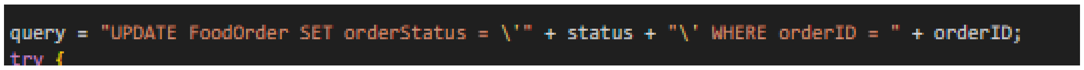
```
if ("customer".equalsIgnoreCase(role.trim())) {
    query = "SELECT * FROM FoodOrder WHERE login = \'" + username + "\'" + " AND orderID =" + orderID;
}
else { // access to all orders
    query = "SELECT * FROM FoodOrder WHERE orderID =" + orderID;
}
itemQuery = "SELECT * FROM ItemsInOrder WHERE orderID =" + orderID;
```

These queries are to retrieve more detailed information about a specific order. The query to be executed is based on the user's role, a customer can see only their own order, while managers and drivers can see all. itemQuery is used to retrieve information on each item in the order.



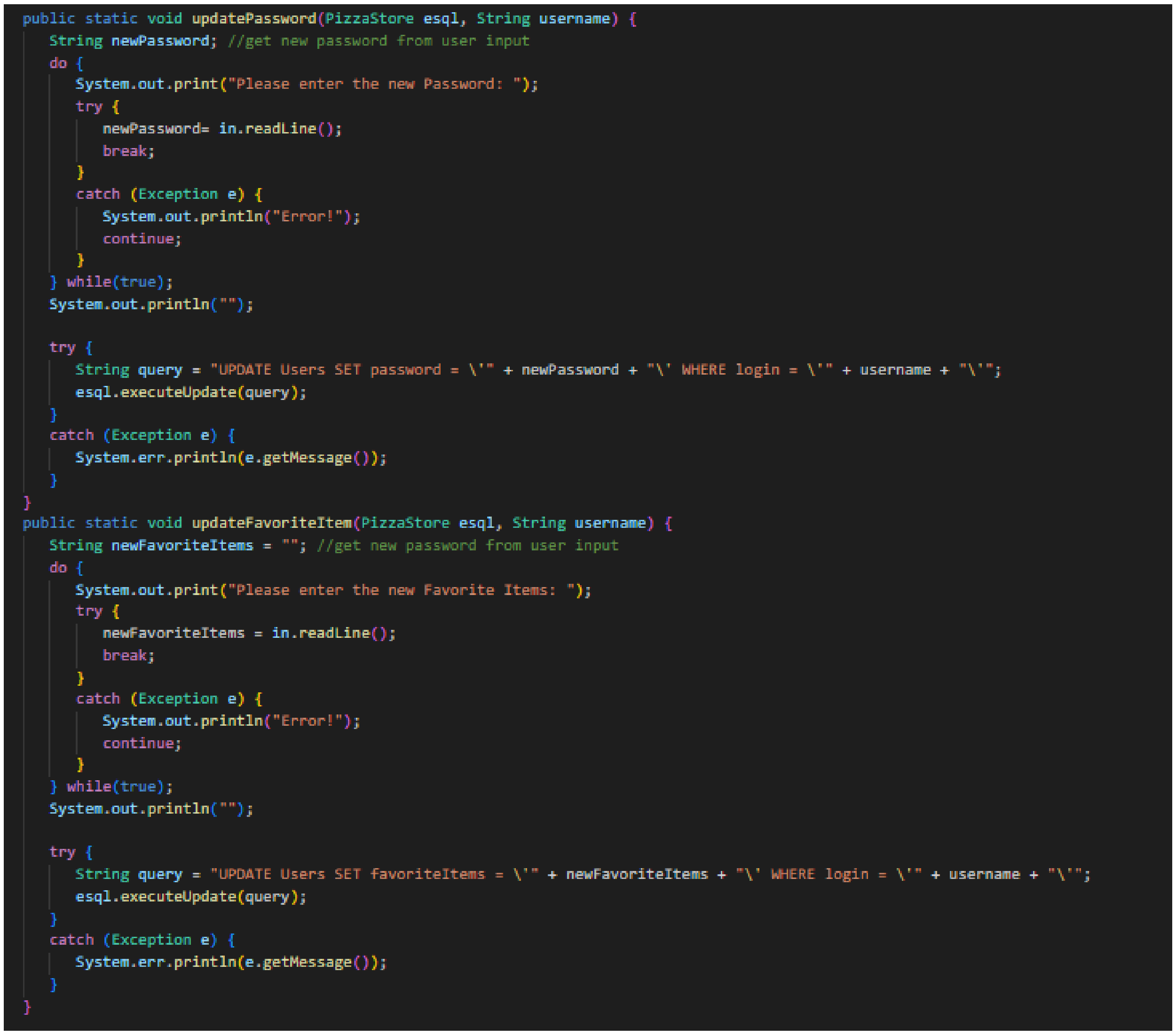
```
String query = "SELECT * FROM Store";
```

This query retrieves information about all stores

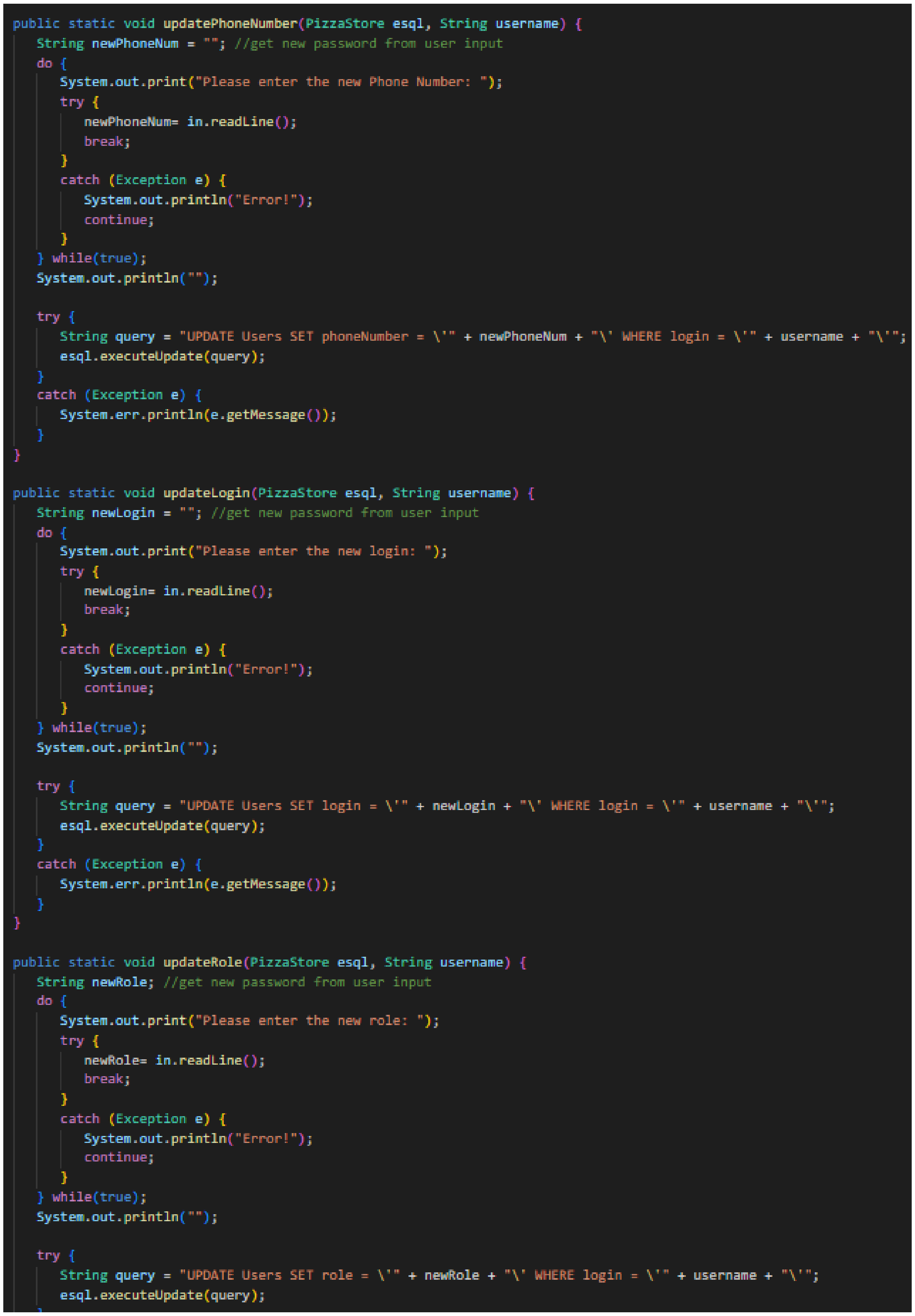


```
query = "UPDATE FoodOrder SET orderStatus = \'" + status + "\' WHERE orderID = " + orderID;
try {
```

This query is used to update an order’s status from incomplete to complete for vice versa. This query can only be executed by managers and drivers.

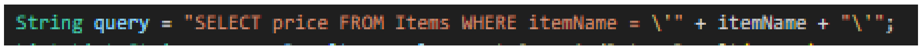


```java
public static void updatePassword(PizzaStore esql, String username) {
    String newPassword; //get new password from user input
    do {
        System.out.print("Please enter the new Password: ");
        try {
            newPassword= in.readLine();
            break;
        }
        catch (Exception e) {
            System.out.println("Error!");
            continue;
        }
    } while(true);
    System.out.println("");

    try {
        String query = "UPDATE Users SET password = \'" + newPassword + "\' WHERE login = \'" + username + "\'";
        esql.executeUpdate(query);
    }
    catch (Exception e) {
        System.err.println(e.getMessage());
    }
}
public static void updateFavoriteItem(PizzaStore esql, String username) {
    String newFavoriteItems = ""; //get new password from user input
    do {
        System.out.print("Please enter the new Favorite Items: ");
        try {
            newFavoriteItems = in.readLine();
            break;
        }
        catch (Exception e) {
            System.out.println("Error!");
            continue;
        }
    } while(true);
    System.out.println("");

    try {
        String query = "UPDATE Users SET favoriteItems = \'" + newFavoriteItems + "\' WHERE login = \'" + username + "\'";
        esql.executeUpdate(query);
    }
    catch (Exception e) {
        System.err.println(e.getMessage());
    }
}
```



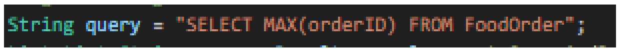
```java
public static void updatePhoneNumber(PizzaStore esql, String username) {
   String newPhoneNum = ""; //get new password from user input
   do {
      System.out.print("Please enter the new Phone Number: ");
      try {
         newPhoneNum= in.readLine();
         break;
      }
      catch (Exception e) {
         System.out.println("Error!");
         continue;
      }
   } while(true);
   System.out.println("");

   try {
      String query = "UPDATE Users SET phoneNumber = \'" + newPhoneNum + "\' WHERE login = \'" + username + "\'";
      esql.executeUpdate(query);
   }
   catch (Exception e) {
      System.err.println(e.getMessage());
   }
}

public static void updateLogin(PizzaStore esql, String username) {
   String newLogin = ""; //get new password from user input
   do {
      System.out.print("Please enter the new login: ");
      try {
         newLogin= in.readLine();
         break;
      }
      catch (Exception e) {
         System.out.println("Error!");
         continue;
      }
   } while(true);
   System.out.println("");

   try {
      String query = "UPDATE Users SET login = \'" + newLogin + "\' WHERE login = \'" + username + "\'";
      esql.executeUpdate(query);
   }
   catch (Exception e) {
      System.err.println(e.getMessage());
   }
}

public static void updateRole(PizzaStore esql, String username) {
   String newRole; //get new password from user input
   do {
      System.out.print("Please enter the new role: ");
      try {
         newRole= in.readLine();
         break;
      }
      catch (Exception e) {
         System.out.println("Error!");
         continue;
      }
   } while(true);
   System.out.println("");

   try {
      String query = "UPDATE Users SET role = \'" + newRole + "\' WHERE login = \'" + username + "\'";
      esql.executeUpdate(query);
```

These are functions that use the same queries to update a user's information. These functions are used by two functions: updateUser and updateProfile. The first is reserved for managers who can edit any user's information, while the second is meant for all users editing their own profile, but cannot change their own login and role.



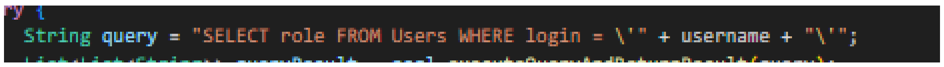
```
String query = "SELECT price FROM Items WHERE itemName = \'" + itemName + "\'";
```

This query retrieves the price of an item based on the itemName provided by the user and is used to calculate the final totalPrice by summing the product of price and quantity.



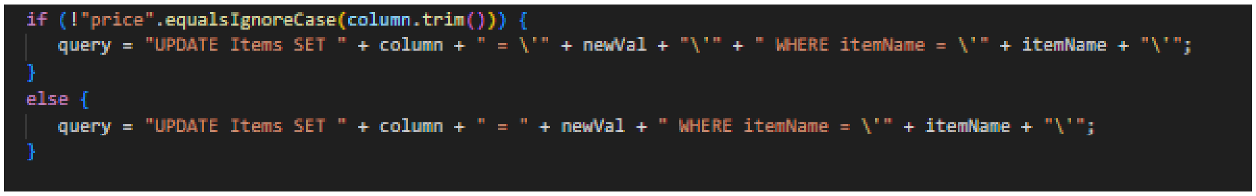
```
String query = "SELECT MAX(orderID) FROM FoodOrder";
```

This query retrieves the largest orderID amongst all orders. This is used to generate a new orderID, which will be the current largest orderID + 1.



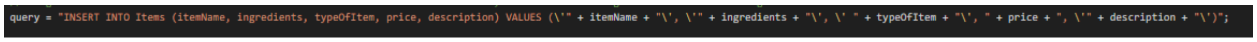
```
ry {
  String query = "SELECT role FROM Users WHERE login = \'" + username + "\'";
```

This query retrieves the role of a user based on their login. This query is implemented in a helper function which returns a user's role and is used to verify their permissions for role based access.



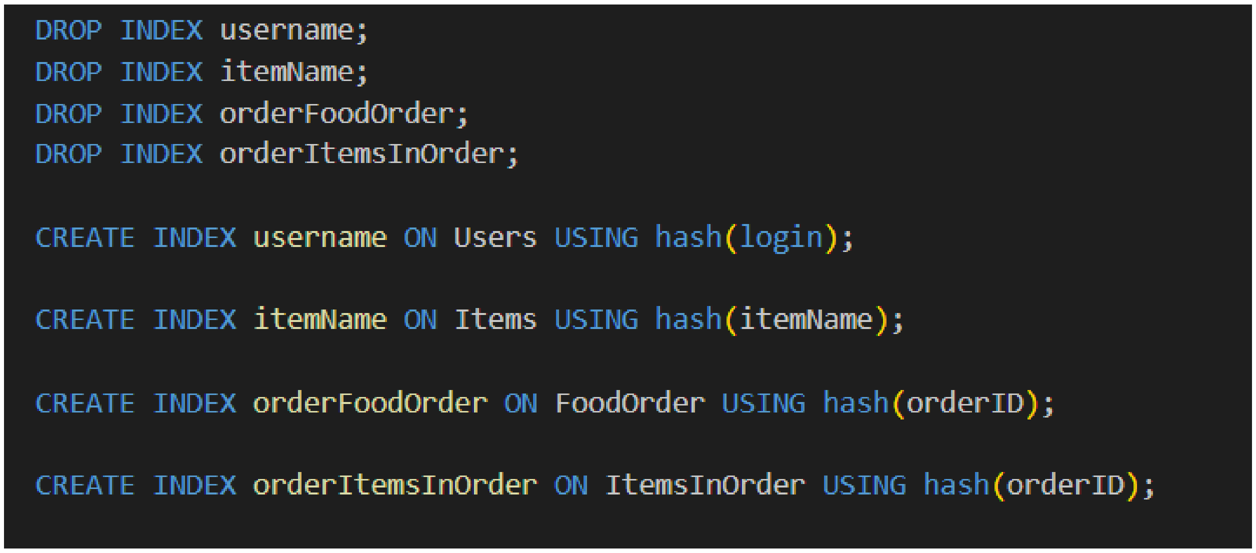
```
if (!"price".equalsIgnoreCase(column.trim())) {
    query = "UPDATE Items SET " + column + " = \'" + newVal + "\'" + " WHERE itemName = \'" + itemName + "\'";
}
else {
    query = "UPDATE Items SET " + column + " = " + newVal + " WHERE itemName = \'" + itemName + "\'";
}
```

This query updates an item from the Items table based on the information the user provides to update a single column of an entry.



```
query = "INSERT INTO Items (itemName, ingredients, typeOfItem, price, description) VALUES (\'" + itemName + "\', \'" + ingredients + "\', \' " + typeOfItem + "\', " + price + ", \'" + description + "\')";
```

This query is used to insert a new item into the Items table, with all the item's information provided by the user.



```sql
DROP INDEX username;
DROP INDEX itemName;
DROP INDEX orderFoodOrder;
DROP INDEX orderItemsInOrder;

CREATE INDEX username ON Users USING hash(login);

CREATE INDEX itemName ON Items USING hash(itemName);

CREATE INDEX orderFoodOrder ON FoodOrder USING hash(orderID);

CREATE INDEX orderItemsInOrder ON ItemsInOrder USING hash(orderID);
```

These queries create indexes for the tables in the database. These columns were chosen to have indexes due to their frequent lookups in the queries and hash indexing was used due to these columns being used in equality searches.

Problems/Findings

In implementing this database system, a problem we encouncted early on was properly accessing the correct index from the database. This proved to be challenging due to the different syntax of the SQL database and that of java. Additionally, once properly accessing the database indices, properly formatting the queries results proved to be both difficult but rewarding once understood.
One issue with how the queries are implemented in the code is that it is vulnerable to sql injection attacks. Special characters can break a query and this could cause some serious issues. There are also some inconsistencies with the loaded data which affect formatting, such as an item of the menu which should be ‘entree’ is stored at ‘ entree’..

Contributions

<u>Angel Hernandez</u>
My main contribution in this project involved solving the issue relating to the proper retrieval of data from the database. This involved having to test the proper way to access the table's entries. Additionally, was able to format the queries output and overall user interface for a better user experience.

<u>David Kim</u>
My contribution to this project was the implementation of the menu options and the necessary functions, as well as the queries required for the functions. I also took part in

testing and fixing any bugs/errors in the code. Finally, I created indexes to optimize the queries based on frequently searched tables and columns.