CS 166 Project Report

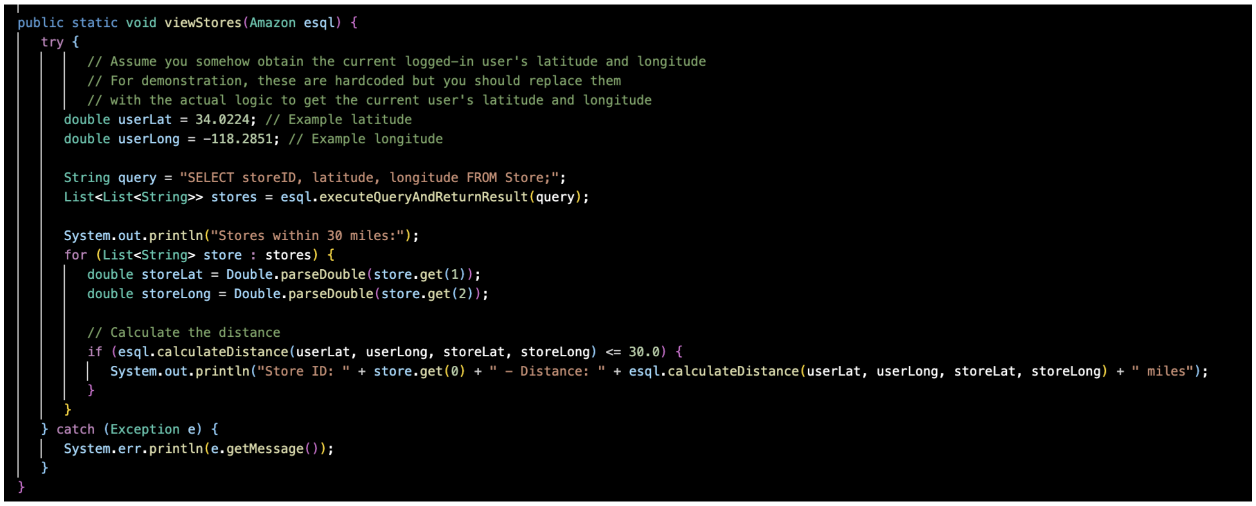
Group Information

Group number: 70
Bryan Pham 682360941
Justin Ly 862240567

Implementation Description

The program simulates an Amazon storefront managing system, utilizing the interactions between the data and user interfaces. The functionalities are based on the roles of the customers and the roles of the administrators. The implementation operations include viewing products, placing orders, viewing recent orders, updating product information, management through authoritative uses, and other interactions. Each function calls and executes SQL queries that interact with the database and display the information based on it.

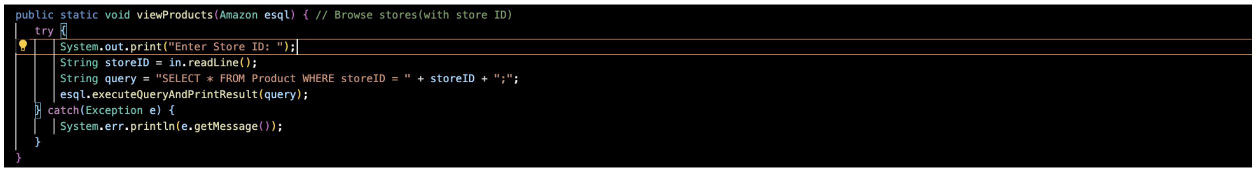
ViewStores:



```java
public static void viewStores(Amazon esql) {
    try {
        // Assume you somehow obtain the current logged-in user's latitude and longitude
        // For demonstration, these are hardcoded but you should replace them
        // with the actual logic to get the current user's latitude and longitude
        double userLat = 34.0224; // Example latitude
        double userLong = -118.2851; // Example longitude

        String query = "SELECT storeID, latitude, longitude FROM Store;";
        List<List<String>> stores = esql.executeQueryAndReturnResult(query);

        System.out.println("Stores within 30 miles:");
        for (List<String> store : stores) {
            double storeLat = Double.parseDouble(store.get(1));
            double storeLong = Double.parseDouble(store.get(2));

            // Calculate the distance
            if (esql.calculateDistance(userLat, userLong, storeLat, storeLong) <= 30.0) {
                System.out.println("Store ID: " + store.get(0) + " - Distance: " + esql.calculateDistance(userLat, userLong, storeLat, storeLong) + " miles");
            }
        }
    } catch (Exception e) {
        System.err.println(e.getMessage());
    }
}
```

Using the already created distance, we calculated whether or not a store is in a 30-mile radius of the user. Specifically utilizing the latitude and longitude to filter out what stores are within the area for the best user experience.

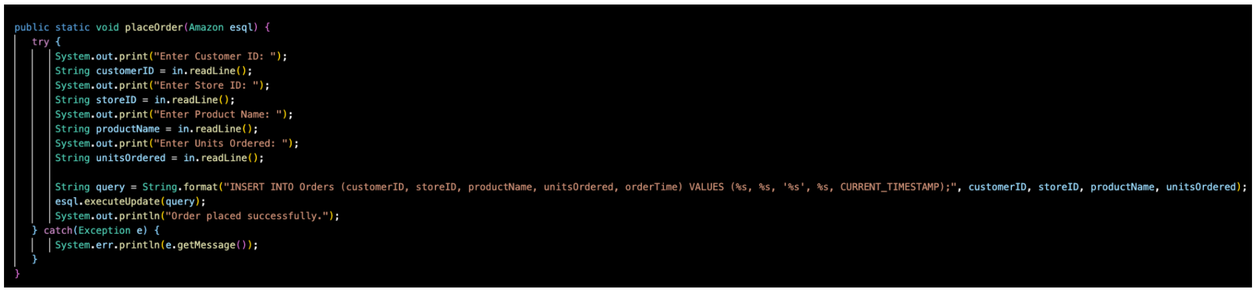
ViewProducts:



```
public static void viewProducts(Amazon esql) { // Browse stores(with store ID)
    try {
        System.out.print("Enter Store ID: ");
        String storeID = in.readLine();
        String query = "SELECT * FROM Product WHERE storeID = " + storeID + ";";
        esql.executeQueryAndPrintResult(query);
    } catch(Exception e) {
        System.err.println(e.getMessage());
    }
}
```

The function asks the user for storeID and the query retrieves all the products from that storeID. The query is then printed out based on the products.

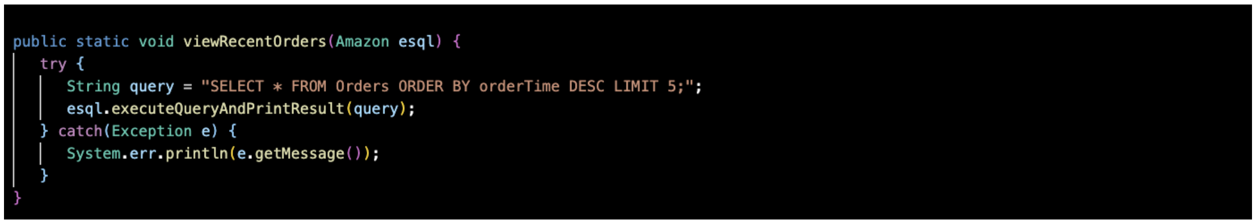
PlaceOrders:



```
public static void placeOrder(Amazon esql) {
    try {
        System.out.print("Enter Customer ID: ");
        String customerID = in.readLine();
        System.out.print("Enter Store ID: ");
        String storeID = in.readLine();
        System.out.print("Enter Product Name: ");
        String productName = in.readLine();
        System.out.print("Enter Units Ordered: ");
        String unitsOrdered = in.readLine();

        String query = String.format("INSERT INTO Orders (customerID, storeID, productName, unitsOrdered, orderTime) VALUES (%s, %s, '%s', %s, CURRENT_TIMESTAMP);", customerID, storeID, productName, unitsOrdered);
        esql.executeUpdate(query);
        System.out.println("Order placed successfully.");
    } catch(Exception e) {
        System.err.println(e.getMessage());
    }
}
```

The function asks the user for customerID, storeID, product, and Units ordered, and updates the inventory based on the information. This is then inserted into the Orders table to update the new information.

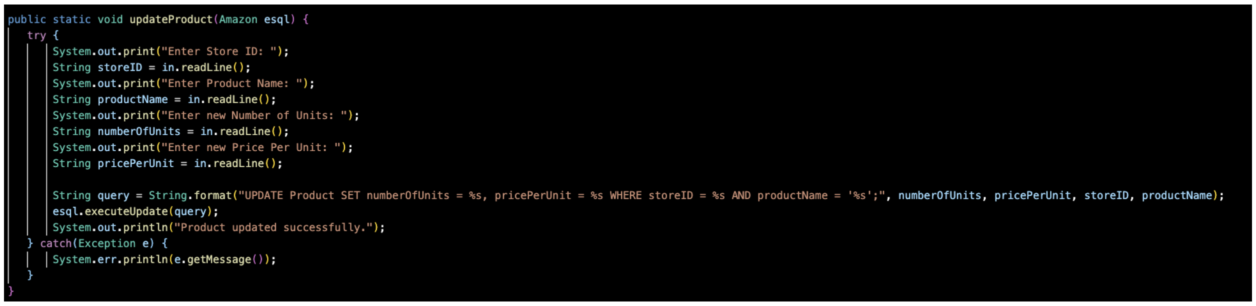
ViewRecentOrders:



```
public static void viewRecentOrders(Amazon esql) {
    try {
        String query = "SELECT * FROM Orders ORDER BY orderTime DESC LIMIT 5;";
        esql.executeQueryAndPrintResult(query);
    } catch(Exception e) {
        System.err.println(e.getMessage());
    }
}
```

Executes a query to retrieve the last five orders that have been placed, arranged in decreasing order by order time. Users may quickly view their most recent transactions using this.

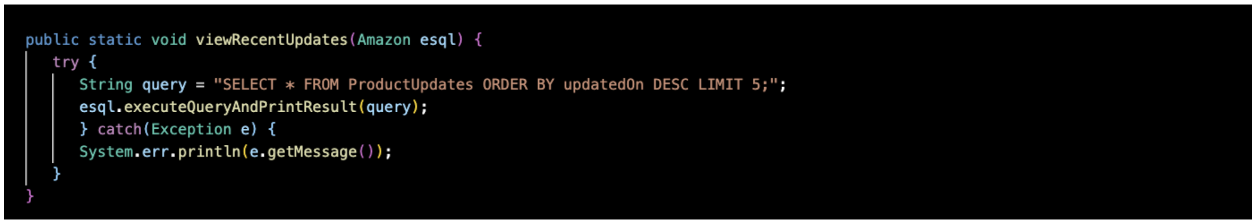
UpdateProduct:



```
public static void updateProduct(Amazon esql) {
    try {
        System.out.print("Enter Store ID: ");
        String storeID = in.readLine();
        System.out.print("Enter Product Name: ");
        String productName = in.readLine();
        System.out.print("Enter new Number of Units: ");
        String numberOfUnits = in.readLine();
        System.out.print("Enter new Price Per Unit: ");
        String pricePerUnit = in.readLine();

        String query = String.format("UPDATE Product SET numberOfUnits = %s, pricePerUnit = %s WHERE storeID = %s AND productName = '%s';", numberOfUnits, pricePerUnit, storeID, productName);
        esql.executeUpdate(query);
        System.out.println("Product updated successfully.");
    } catch(Exception e) {
        System.err.println(e.getMessage());
    }
}
```

Updates the relevant product record in the database after gathering new product information from the user, such as the quantity and cost per unit.

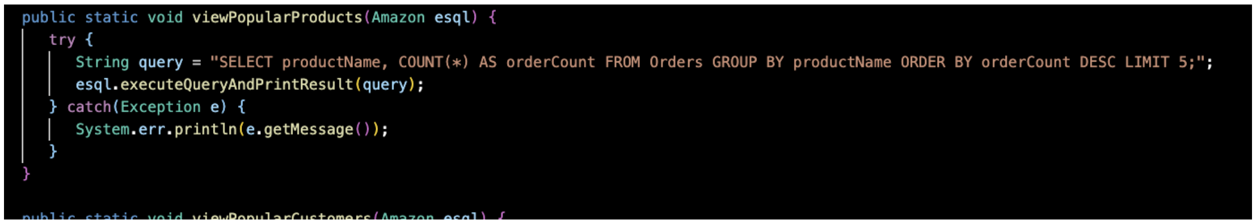
ViewRecentUpdates:



```
public static void viewRecentUpdates(Amazon esql) {
    try {
        String query = "SELECT * FROM ProductUpdates ORDER BY updatedOn DESC LIMIT 5;";
        esql.executeQueryAndPrintResult(query);
        } catch(Exception e) {
        System.err.println(e.getMessage());
    }
}
```

This method gives managers access to the most recent modifications to product information by retrieving the last five product updates from the `ProductUpdates` database.

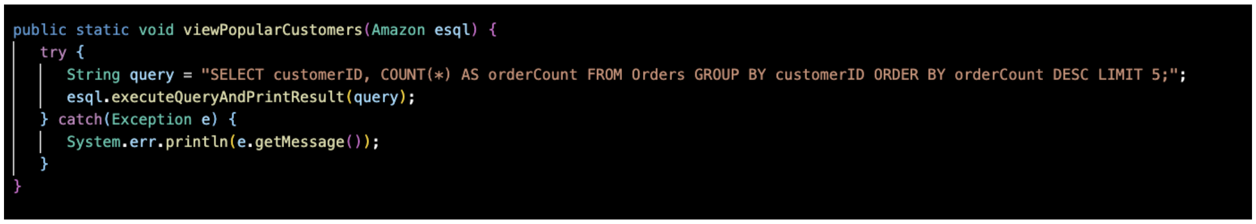
ViewPopularProducts:



```
public static void viewPopularProducts(Amazon esql) {
    try {
        String query = "SELECT productName, COUNT(*) AS orderCount FROM Orders GROUP BY productName ORDER BY orderCount DESC LIMIT 5;";
        esql.executeQueryAndPrintResult(query);
    } catch(Exception e) {
        System.err.println(e.getMessage());
    }
}

public static void viewPopularCustomers(Amazon esql) {
```

Aims to provide the top 5 most ordered goods in terms of quantity. Order data is aggregated from the orders table, grouped by product name, and then sorted in decreasing order by order count.

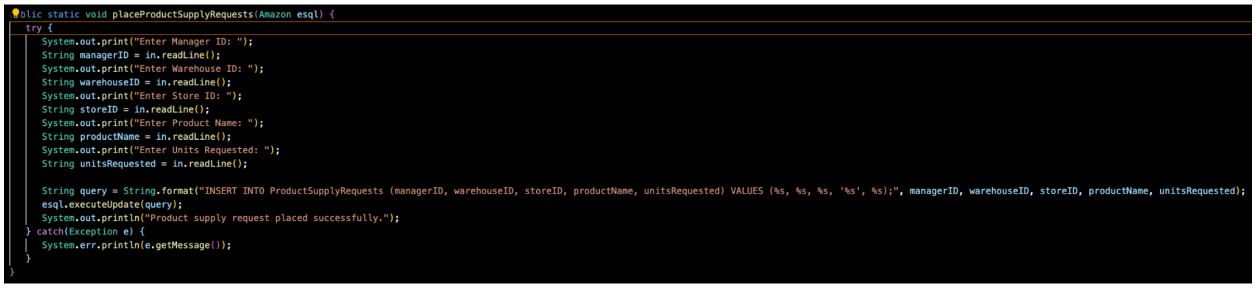
ViewPopularCustomers:



```
public static void viewPopularCustomers(Amazon esql) {
    try {
        String query = "SELECT customerID, COUNT(*) AS orderCount FROM Orders GROUP BY customerID ORDER BY orderCount DESC LIMIT 5;";
        esql.executeQueryAndPrintResult(query);
    } catch(Exception e) {
        System.err.println(e.getMessage());
    }
}
```

Provide insights into consumer involvement and loyalty by identifying the top 5 customers who have placed the most orders. Using customer IDs as a grouping, this function aggregates data from the `Orders` database.

PlaceProductsSupplyRequests:



```
blic static void placeProductSupplyRequests(Amazon esql) {
    try {
        System.out.print("Enter Manager ID: ");
        String managerID = in.readLine();
        System.out.print("Enter Warehouse ID: ");
        String warehouseID = in.readLine();
        System.out.print("Enter Store ID: ");
        String storeID = in.readLine();
        System.out.print("Enter Product Name: ");
        String productName = in.readLine();
        System.out.print("Enter Units Requested: ");
        String unitsRequested = in.readLine();

        String query = String.format("INSERT INTO ProductSupplyRequests (managerID, warehouseID, storeID, productName, unitsRequested) VALUES (%s, %s, %s, '%s', %s);", managerID, warehouseID, storeID, productName, unitsRequested);
        esql.executeUpdate(query);
        System.out.println("Product supply request placed successfully.");
    } catch(Exception e) {
        System.err.println(e.getMessage());
    }
}
```

Enables managers to create a new record to the `ProductSupplyRequests` table to request more stock for goods. The manager's request for additional units from a particular warehouse is reflected in the inquiry.

Problems/Findings

One of the main problems that we encountered was creating a function for finding a store within 30 miles based on the given locations. Even though we were given the calculateDistance formula/helper function, we ran into some difficulties when creating the function. The logic made sense but our output wasn’t what we wanted. The issue was with our conditional statements and how we printed them out. However, we were still able to figure out why it wasn't working properly, due to global variables.

Contributions

Basically, dividing the work in half, Bryan worked on the viewStores, viewProducts, PlaceORder, ViewRecentORders, and updatedProduct functions. Justin worked on viewRecentUpdates,viewPopularProducts,viewPopularCustomers, and place ProductSupplyRequest. Whenever we needed help or assistance with anything we both worked on it together. Essentially worked on the project 50/50 and worked on the report together.