CS 166 Project Report

Group Information

Group 73

Name: Shan Santhakumar
NetID: ssant096
Name: Chris Schlenker
NetID: cschl005

Implementation Description

The Project implementation divides the different functionalities into individual functions ad provided. The provided initial menu navigation is expanded on via a 10th admin option and submenu navigation for each appropriate option. The executeQueryAndPrintResults function is modified to better align the column headers and values, and frequently used queries for prompts are generalized into a helper function for modular implementation.

Helper Functions

SelectByDistance

This helper function lets the user select among a list of options sorted by distance from an origin. The function is passed the column name of the data (Store ID, for example), and a query result with values (id, latitude, longitude) and the origin (latitude, longitude) and an optional **display** count limit (does not restrict valid selections). This is used anywhere where a selection is sorted by distance (mostly store selection, but also warehouse selection, whose origin is the store making a product supply request). This implements its own **Pair** class - a very simple and common implementation. You may find it in the source code above this block, but is not worthy of its own screenshot.

```
//data is a list of lists, whose first dimension is each entry, and second dimension is size 3, the ID, latitude, longitude
//latitude and longitude is the refernece for measurement
public static int SelectByDistance(String idName, List<List<String>> data, double latitude, double longitude, int limit){
   List<Pair<Integer, Double>> locations = new ArrayList<Pair<Integer, Double>>();
   List<Integer> ids = new ArrayList<Integer>(); //a little redundant but much more efficient to search for valid selection
   for (int i = 0; i < data.size(); i++)
      locations.add(new Pair<Integer, Double>(Integer.parseInt(data.get(i).get(0)), calculateDistance(latitude, longitude, Double.parseDouble(data.get(i).get(1)), Double.parseDouble(data.get(i).get(2)))));
   Collections.sort(locations, Comparator.comparing(p -> p.getRight()));
   System.out.println(String.format("%-" + (idName.length() + 1) + "s%-10s", idName, "Distance"));
   for (int i = 0; i < locations.size(); i++){
      ids.add(locations.get(i).getLeft());
      if (limit == -1 || i < limit)
         System.out.println(String.format("%-" + (idName.length() + 1) + "d%-10.2f", locations.get(i).getLeft(), locations.get(i).getRight()));
   }

   int sel = readChoice();
   while (!ids.contains(sel)){
      System.out.println("Invalid selection. Please enter an id among the listed options.");
      sel = readChoice();
   }
   return sel;
}

public static int SelectByDistance(String idName, List<List<String>> data, double latitude, double longitude) { return SelectByDistance(idName, data, latitude, longitude, -1); }
```

Menu Navigation

First is the provided login page

Each user is either a customer, manager or admin. Creating a new user defaults to a customer, though an admin can later change their type.

After login, the user is prompted with the Main menu

```
MAIN MENU
---------
1. Create user
2. Log in
9. < EXIT
```

```
MAIN MENU
---------
1. View Stores within 30 miles
2. View Product List
3. Place a Order
4. View 5 recent orders
5. Update Product
6. View 5 recent Product Updates Info
7. View 5 Popular Items
8. View 5 Popular Customers
9. Place Product Supply Request to Warehouse
10. Admin options
.........................
20. Log out
Please make your choice:
```

Note that the data from the following prompts is not consistent with the data we began with, as it contains several updates from test runs.

We aimed for some extra credit via a good user intefact, hence those prior helper functions, and you can see our results in each menu option response.

1. **View Stores within 30 miles:** This menu option acts the same for all user types, displaying all stores within 30 “miles.” It uses the provided calculateDistance function, and displays the results sorted by this distance. This uses the **very wrong assumption** that the distance function returns in untis of miles

```
Store ID  Distance  Manager ID     Date Established
1         12.42mi   25             1953-03-13
12        29.08mi   10             1958-06-13
```

   This function collects the data of all stores and calculates the distance of each using the provided function. It then discards any entires with distance > 30 and sorts the remaining entries by distance and displays the above data.

```java
public static void viewStores(Amazon esql) {
try{
    String id = curr_user_id;
    String query = String.format("SELECT S.*, U.latitude, U.longitude FROM Store S, Users U WHERE U.userID = '%s'", id);
    List<List<String>> data = esql.executeQueryAndReturnResult(query);
    //storeID,latitude,longitude,managerID,dateEstablished, userLatitude, userLongitude
    //we have all stores, now show within 30mi
    double limit = 30;
    //make the last index
    for (int i = 0; i < data.size(); i++){
       double dist = calculateDistance(Double.parseDouble(data.get(i).get(5)), Double.parseDouble(data.get(i).get(6)), Double.parseDouble(data.get(i).get(1)), Double.parseDouble(data.get(i).get(2)));
       data.get(i).add(String.format("%.2f", dist));
    }
    Collections.sort(data, Comparator.comparing(p -> Double.parseDouble(p.get(7))));
    System.out.println(String.format("%-10s%-10s%-15s%-15s", "Store ID", "Distance", "Manager ID", "Date Established"));
    for (int i = 0; i < data.size(); i++){
       if (Double.parseDouble(data.get(i).get(7)) > limit) break; //stop checking, since we're sorted if this one is too far away, all future ones are too
       System.out.println(String.format("%-10s%-10s%-15s%-15s", data.get(i).get(0), data.get(i).get(7) + "mi", data.get(i).get(3), data.get(i).get(4)));
    }

 }catch(Exception e){
    System.err.println (e.getMessage ());
 }
}
```

2. **View Product List:** This menu option acts the same for all user types, displaying the nearest 10 stores (using the aforementioned SelectByDistance helper function), though any valid store ID may be chosen.

```
Please select a store (any valid ID may be entered, but these are the nearest 10 stores)
Store ID Distance
1        12.42
12       29.08
18       32.61
2        35.92
16       44.34
17       51.00
7        52.74
5        58.65
20       59.95
6        73.22
Please make your choice:
```

After selecting a store, all products with that storeId are displayed

```
Please make your choice: 1
productname             numberofunits   priceperunit
7up                     39              3
Brisk                   37              3
Donuts                  80              7
Egg                     83              3
Hot and Sour Soup       78              5
Ice Cream               39              6
Lemonade                41              8
Orange Juice            42              6
Pepsi                   23              4
Pudding                 23              3
```

via this function

```
public static void viewProducts(Amazon esql) {
try{
    System.out.println("Please select a store (any valid ID may be entered, but these are the nearest 10 stores)");
    String myId = esql.getUserId();
    List<List<String>> stores = esql.executeQueryAndReturnResult("SELECT storeId, latitude, longitude FROM STORE");
    List<List<String>> myloc = esql.executeQueryAndReturnResult(String.format("SELECT latitude, longitude FROM USERS where userId = %s", myId));
    int storeId = SelectByDistance("Store ID", stores, Double.parseDouble(myloc.get(0).get(0)), Double.parseDouble(myloc.get(0).get(1)), 10);
    String query = String.format("SELECT productName, numberOfUnits, pricePerUnit FROM Product P WHERE P.storeID = '%s' ORDER BY productName", storeId);
    esql.executeQueryAndPrintResult(query);
  }catch(Exception e){
    System.err.println (e.getMessage ());
  }
}
```

3. **Place Order:** This menu option acts the same for all user types, prompting to select a store (via the SelectByDistance helper function).

```
Please make your choice: 3
Please select a store (any valid ID may be entered, but these are the nearest 10 stores)
Store ID Distance
1        12.42
12       29.08
18       32.61
2        35.92
16       44.34
17       51.00
7        52.74
5        58.65
20       59.95
6        73.22
```

After selecting a store, they are provided a list of available products (sorted alphabetically) as well as stock count and price.They are prompted to select a product by name and order amount.

```
Please make your choice: 1
productname              numberofunits   priceperunit
7up                      39              3
Brisk                    37              3
Donuts                   80              7
Egg                      83              3
Hot and Sour Soup        78              5
Ice Cream                39              6
Lemonade                 41              8
Orange Juice             42              6
Pepsi                    23              4
Pudding                  23              3
        Enter a product name: 7up
        Enter number of units: 9
Congratulations, you purchased 9 units of '7up' from store 1
```

If there are a sufficient number of products to purchase, this updates the orders table (using the appropriate sequence) and the products table. If not, they are informed of the insufficient unit count and returned to the menu.

```
public static void placeOrder(Amazon esql) {
   try{
      System.out.println("Please select a store (any valid ID may be entered, but these are the nearest 10 stores)");
      String myId = esql.getUserId();
      List<List<String>> stores = esql.executeQueryAndReturnResult("SELECT storeId, latitude, longitude FROM STORE");
      List<List<String>> myloc = esql.executeQueryAndReturnResult(String.format("SELECT latitude, longitude FROM USERS where userId = %s", myId));
      int storeId = SelectByDistance("Store ID", stores, Double.parseDouble(myloc.get(0).get(0)), Double.parseDouble(myloc.get(0).get(1)), 10);

      esql.executeQueryAndPrintResult(String.format("SELECT productName, numberOfUnits, pricePerUnit FROM PRODUCT WHERE storeId = %s ORDER BY productName", storeId));

      System.out.print("\tEnter a product name: ");
      String prod_name = in.readLine();
      while (esql.executeQuery(String.format("SELECT productName FROM PRODUCT WHERE storeId = %d AND productName = '%s'", storeId, prod_name)) == 0){
         System.out.print("This store does not have this product. Please enter a valid product name: ");
         prod_name = in.readLine();
      }

      System.out.print("\tEnter number of units: ");
      int nUnits = Integer.parseInt(in.readLine());

      String query = String.format("SELECT * FROM Product P WHERE P.storeID = %d AND P.productName = '%s' AND P.numberOfUnits >= %d", storeId, prod_name, nUnits);
      int rows = esql.executeQuery(query);

      if(rows > 0){
         esql.executeUpdate(String.format("INSERT INTO ORDERS (orderNumber, customerID, storeID, productName, unitsOrdered, orderTime) VALUES (nextval('orders_orderNumber_seq'), %s, %s, '%s', %s, TIMESTAMP '%s')", myId, storeId, prod_name, nUnits, new java.sql.
         esql.executeUpdate(String.format("UPDATE Product SET numberOfUnits = numberOfUnits - %d WHERE Product.storeID = %d AND Product.productName = '%s'", nUnits, storeId, prod_name));

         System.out.println(String.format("Congratulations, you purchased %d units of '%s' from store %d", nUnits, prod_name, storeId));
      } else {
         System.out.println("Not enough units available.");
      }
   }catch(Exception e){
      System.err.println (e.getMessage ());
   }
}
```

4. **View 5 recent orders**: For customers, this menu option displays their 5 recent orders

```
Please make your choice: 4
storeid productname          unitsordered    ordertime
5       7up                  6               2024-03-17 17:19:23
10      Hot and Sour Soup    45              2016-09-10 19:58:00
15      7up                  7               2016-09-10 18:27:00
2       Pepsi                7               2016-09-10 17:45:00
3       Pudding              40              2016-09-10 15:02:00
```

Managers, however, are prompted to choose among their orders, or store orders.

```
Please make your choice: 4
Would you like to see your personal orders, or order information for your store(s)?
1. See my Orders
2. See my Store(s) Orders
Please make your choice:
```

Viewing their own orders works just like a customer. Viewing their store orders prompts them to choose among thier managed stores (auto selects for 1 managed store and cancels for 0).

```
Please make your choice: 4
Would you like to see your personal orders, or order information for your store(s)?
1. See my Orders
2. See my Store(s) Orders
Please make your choice: 2
Store ID Distance
12       21.94
7        57.05
3        86.62
13       90.69
Please make your choice: 12
ordernumber     name          productname     unitsordered    ordertime
494             Brandt        Brisk           12              2016-09-10 19:53:00
487             Sid_Stamm     Pudding         46              2016-09-10 19:47:00
452             Admin         Pepsi           43              2016-09-10 19:16:00
407             Ellis         Pudding         24              2016-09-10 18:44:00
385             Eryn          Orange Juice    24              2016-09-10 18:30:00
```

```java
public static void viewRecentOrders(Amazon esql) {
  try {
    String c_id = esql.getUserId();
    String qC = String.format("SELECT * FROM USERS U WHERE U.type = 'customer' AND U.userID = '%s'", c_id);
    String qM = String.format("SELECT * FROM USERS U WHERE U.type = 'manager' AND U.userID = '%s'", c_id);
    if (esql.executeQuery(qC) > 0) { //customer access
      esql.executeQueryAndPrintResult(String.format("SELECT storeID, productName, unitsOrdered, orderTime FROM ORDERS O WHERE O.customerID = %s ORDER BY orderTime DESC LIMIT 5", c_id));
    } else if (esql.executeQuery(qM) > 0) { //manager access
      System.out.println("Would you like to see your personal orders, or order information for your store(s)?");
      System.out.println("1. See my Orders");
      System.out.println("2. See my Store(s) Orders");
      int sel = readChoice();
      switch(sel){
        case 1:
          esql.executeQueryAndPrintResult(String.format("SELECT storeID, productName, unitsOrdered, orderTime FROM ORDERS O WHERE O.customerID = %s ORDER BY orderTime DESC LIMIT 5", c_id));
          break;
        case 2:
        //get the storeId of this manager
        List<List<String>> storesManaged = esql.executeQueryAndReturnResult(String.format("SELECT storeID, latitude, longitude FROM STORE WHERE managerID = %s", c_id));
        List<List<String>> myloc = esql.executeQueryAndReturnResult(String.format("SELECT latitude, longitude FROM USERS WHERE userId = %s", c_id));
        //for each store managed
        if (storesManaged.size() > 1){ //choose a store
          sel = SelectByDistance("Store ID", storesManaged, Double.parseDouble(myloc.get(0).get(0)), Double.parseDouble(myloc.get(0).get(1)));
          esql.executeQueryAndPrintResult(String.format("SELECT O.orderNumber, U.name, O.productName, O.unitsOrdered, O.orderTime FROM ORDERS O, USERS U WHERE O.storeID = %s AND O.customerID = U.userID ORDER BY orderTime DESC LIMIT 5", sel));
        }
        else if (storesManaged.size() == 1){ //1 store
          //print out the orderId, customerName, storeID, productName, date for each order
          esql.executeQueryAndPrintResult(String.format("SELECT O.orderNumber, U.name, O.productName, O.unitsOrdered, O.orderTime FROM ORDERS O, USERS U WHERE O.storeID = %s AND O.customerID = U.userID ORDER BY orderTime DESC LIMIT 5", storesManaged.get(0).g
        } else
          System.out.println("You are not the manager of any store. No orders to display.");
        break;
      }
    }
  } catch (Exception e) {
    System.err.println(e.getMessage());
  }
}
```

5. **Update Product:** This menu option, and all subsequent options (except 10) are only accessible to managers. Customers and admins may attempt the selection, but are immediately returned to the main menu.
   Managers are prompted to choose among their managed stores (if they have multiple).
   They are then provided a list of products at that store (as well as units and price) and prompted to choose one by name, then update the units and price.

```
Please make your choice: 5
Please select a store among those you manage:
Store ID Distance
12       21.94
7        57.05
3        86.62
13       90.69
Please make your choice: 12
productname             numberofunits   priceperunit
7up                     31              3
Brisk                   48              3
Donuts                  39              7
Egg                     38              3
Hot and Sour Soup       46              5
Ice Cream               50              6
Lemonade                36              8
Orange Juice            41              6
Pepsi                   32              4
Pudding                 33              3
        Enter the name of a product to modify: 7up
        Enter updated number of units: 34
        Enter updated price per unit: 4
7up at store 12 now has 34 units priced at 4 each
```

   Those responses update the table like so (via the 2nd option)

```
productname             numberofunits   priceperunit
7up                     34              4
Brisk                   48              3
Donuts                  39              7
Egg                     38              3
Hot and Sour Soup       46              5
Ice Cream               50              6
Lemonade                36              8
Orange Juice            41              6
Pepsi                   32              4
Pudding                 33              3
```

   As well as the products update table using the appropriate sequence for updatenumber (see next entry results).

```java
public static void updateProduct(Amazon esql) {
try{
      String myId = esql.getUserId();
      int rows = esql.executeQuery(String.format("SELECT * FROM USERS U WHERE U.type = 'manager' AND U.userID = '%s'", myId));
      if(rows > 0){
         System.out.println("Please select a store among those you manage:");
         List<List<String>> stores = esql.executeQueryAndReturnResult(String.format("SELECT storeId, latitude, longitude FROM STORE WHERE managerId = %s", myId));
         if (stores.size() > 1){
            List<List<String>> myloc = esql.executeQueryAndReturnResult(String.format("SELECT latitude, longitude FROM USERS where userId = %s", myId));
            int storeId = SelectByDistance("Store ID", stores, Double.parseDouble(myloc.get(0).get(0)), Double.parseDouble(myloc.get(0).get(1)));

            updateProduct(esql, myId, storeId);
         } else if (stores.size() == 1)
            updateProduct(esql, myId, Integer.parseInt(stores.get(0).get(0)));
         else
            System.out.println("You do not manage any stores!");

      } else{
         System.out.println("Access denied: Manager access only!");
      }

   }catch(Exception e){
      System.err.println (e.getMessage ());
   }
}
```

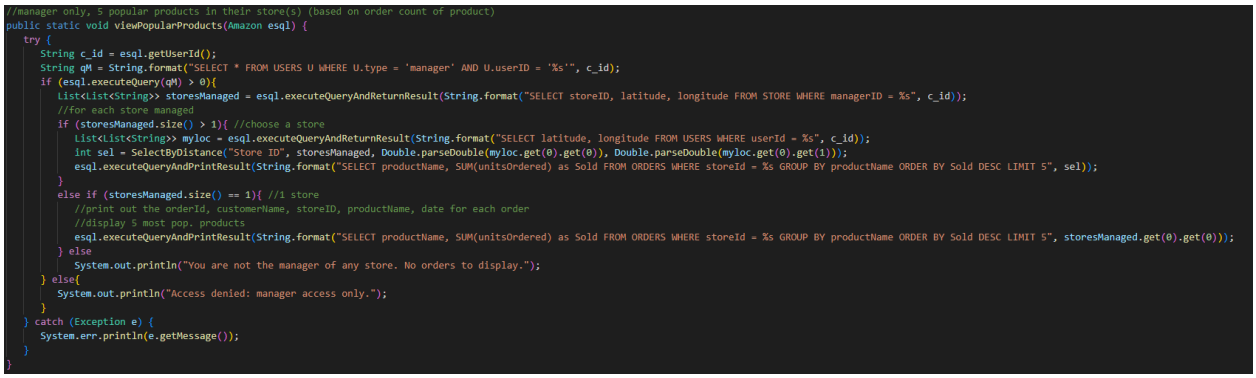
6. **View 5 recent Product Updates Info**: This option prompts to select a store among those managed by the user, and displays the 5 most recent updates at that store.

```
Please make your choice: 6
Please select a store among those you manage:
Store ID Distance
12       21.94
7        57.05
3        86.62
13       90.69
Please make your choice: 12
Last 5 updates at this store:
updatenumber    managerid       storeid productname     updatedon
65              10              12      7up             2024-03-17 18:05:26
64              10              12      Pudding         2024-03-17 17:30:06
63              10              12      7up             2024-03-17 16:43:45
61              10              12      7up             2024-03-17 16:37:20
45              10              12      Orange Juice    2016-09-10 13:44:00
```

```java
public static void viewRecentUpdates(Amazon esql) {
 try{
      String c_id = esql.getUserId();
      String q1 = String.format("SELECT * FROM USERS U WHERE U.type = 'manager' AND U.userID = '%s'", c_id);
      int rows = esql.executeQuery(q1);
      if(rows > 0){
         System.out.println("Please select a store among those you manage:");
         List<List<String>> stores = esql.executeQueryAndReturnResult(String.format("SELECT storeId, latitude, longitude FROM STORE WHERE managerId = %s", c_id));
         List<List<String>> myloc = esql.executeQueryAndReturnResult(String.format("SELECT latitude, longitude FROM USERS where userId = %s", c_id));
         int storeId = SelectByDistance("Store ID", stores, Double.parseDouble(myloc.get(0).get(0)), Double.parseDouble(myloc.get(0).get(1)));
         System.out.println("Last 5 updates at this store: ");
         String query = String.format("SELECT * FROM (SELECT * FROM ProductUpdates PU WHERE PU.managerID = '%s' AND PU.storeId = %d ORDER BY updatedOn DESC LIMIT 5) AS recent_updates ORDER BY updatedOn DESC", c_id, storeId);
         esql.executeQueryAndPrintResult(query);
      } else {
            System.out.println("Access denied: manager access only.");
      }
   }catch(Exception e){
      System.err.println (e.getMessage ());
   }
}
```

7. **View 5 Popular Items**: This option again prompts to select a store among those you manage, and displays the 5 most popular items by sell count.

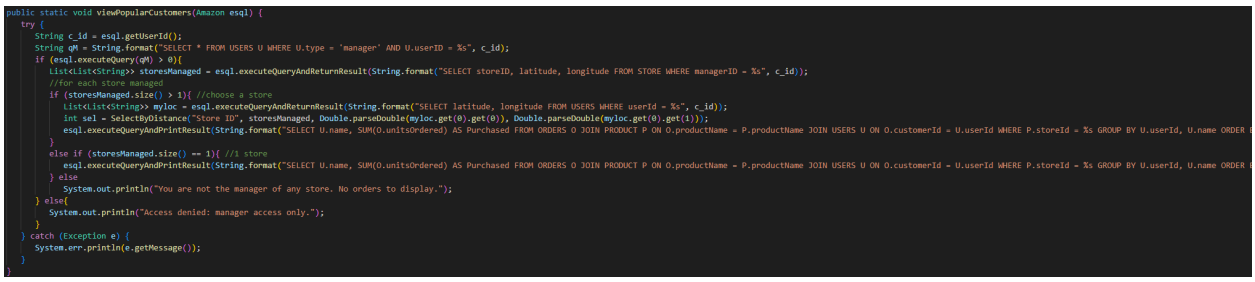
```
Please make your choice: 7
Store ID Distance
12       21.94
7        57.05
3        86.62
13       90.69
Please make your choice: 12
productname     sold
Orange Juice    84
Brisk           79
Egg             75
Pudding         75
Pepsi           50
```



```
//manager only, 5 popular products in their store(s) (based on order count of product)
public static void viewPopularProducts(Amazon esql) {
   try {
      String c_id = esql.getUserId();
      String qM = String.format("SELECT * FROM USERS U WHERE U.type = 'manager' AND U.userID = '%s'", c_id);
      if (esql.executeQuery(qM) > 0){
         List<List<String>> storesManaged = esql.executeQueryAndReturnResult(String.format("SELECT storeID, latitude, longitude FROM STORE WHERE managerID = %s", c_id));
         //for each store managed
         if (storesManaged.size() > 1){ //choose a store
            List<List<String>> myloc = esql.executeQueryAndReturnResult(String.format("SELECT latitude, longitude FROM USERS WHERE userId = %s", c_id));
            int sel = SelectByDistance("Store ID", storesManaged, Double.parseDouble(myloc.get(0).get(0)), Double.parseDouble(myloc.get(0).get(1)));
            esql.executeQueryAndPrintResult(String.format("SELECT productName, SUM(unitsOrdered) as Sold FROM ORDERS WHERE storeId = %s GROUP BY productName ORDER BY Sold DESC LIMIT 5", sel));
         }
         else if (storesManaged.size() == 1){ //1 store
            //print out the orderId, customerName, storeID, productName, date for each order
            //display 5 most pop. products
            esql.executeQueryAndPrintResult(String.format("SELECT productName, SUM(unitsOrdered) as Sold FROM ORDERS WHERE storeId = %s GROUP BY productName ORDER BY Sold DESC LIMIT 5", storesManaged.get(0).get(0)));
         } else
            System.out.println("You are not the manager of any store. No orders to display.");
      } else{
         System.out.println("Access denied: manager access only.");
      }
   } catch (Exception e) {
      System.err.println(e.getMessage());
   }
}
```

8. **View 5 Popular Customers**: Similar to option 7, but displays the 5 most popular customers by total units purchased.

```
Please make your choice: 8
Store ID Distance
12       21.94
7        57.05
3        86.62
13       90.69
Please make your choice: 12
name                purchased
Paige               575
Dedric              289
Lester              286
Janelle.Schneider   274
Kobe                267
```



```
public static void viewPopularCustomers(Amazon esql) {
   try {
      String c_id = esql.getUserId();
      String qM = String.format("SELECT * FROM USERS U WHERE U.type = 'manager' AND U.userID = %s", c_id);
      if (esql.executeQuery(qM) > 0){
         List<List<String>> storesManaged = esql.executeQueryAndReturnResult(String.format("SELECT storeID, latitude, longitude FROM STORE WHERE managerID = %s", c_id));
         //for each store managed
         if (storesManaged.size() > 1){ //choose a store
            List<List<String>> myloc = esql.executeQueryAndReturnResult(String.format("SELECT latitude, longitude FROM USERS WHERE userId = %s", c_id));
            int sel = SelectByDistance("Store ID", storesManaged, Double.parseDouble(myloc.get(0).get(0)), Double.parseDouble(myloc.get(0).get(1)));
            esql.executeQueryAndPrintResult(String.format("SELECT U.name, SUM(O.unitsOrdered) AS Purchased FROM ORDERS O JOIN PRODUCT P ON O.productName = P.productName JOIN USERS U ON O.customerId = U.userId WHERE P.storeId = %s GROUP BY U.userId, U.name ORDER
         }
         else if (storesManaged.size() == 1){ //1 store
            esql.executeQueryAndPrintResult(String.format("SELECT U.name, SUM(O.unitsOrdered) AS Purchased FROM ORDERS O JOIN PRODUCT P ON O.productName = P.productName JOIN USERS U ON O.customerId = U.userId WHERE P.storeId = %s GROUP BY U.userId, U.name ORDER
         } else
            System.out.println("You are not the manager of any store. No orders to display.");
      } else{
         System.out.println("Access denied: manager access only.");
      }
   } catch (Exception e) {
      System.err.println(e.getMessage());
   }
}
```

9. **Place Product Supply Request to Warehouse**: This option prompts the manager to select a store they manage. Then to choose a product to order (first listed for view, sorted by units on hand - ascending to easier see what is short on supply). Then to choose a warehouse (sorted by distance from the selected store) and finally a number to order. This updates the product count in the store and the productSupplyRequests table using the appropriate sequence.

```
Please make your choice: 9
Select a store among those you manage
Store ID Distance
12       21.94
7        57.05
3        86.62
13       90.69
Please make your choice: 12
Please select a product to order:
 1. Pudding                     : 33 Units on hand.
 2. 7up                         : 34 Units on hand.
 3. Lemonade                    : 36 Units on hand.
 4. Egg                         : 38 Units on hand.
 5. Donuts                      : 39 Units on hand.
 6. Orange Juice                : 41 Units on hand.
 7. Pepsi                       : 42 Units on hand.
 8. Hot and Sour Soup           : 46 Units on hand.
 9. Brisk                       : 48 Units on hand.
10. Ice Cream                   : 50 Units on hand.
Please make your choice: 1
Choose a warehouse by ID
Warehouse ID | Distance estimate from your store
Warehouse ID Distance
5            11.70
1            21.94
4            41.16
2            54.70
3            85.49
Please make your choice: 5
Warehouse ID: 5
How many units would you like?
Please make your choice: 10
Warehouse 5 delivered 10 units of Pudding to store 12!
```

```java
//helper function, called once we determine a storeId
private static void placeProductSupplyRequests(Amazon esql, int mgrId, int storeId, double latitude, double longitude){
   try{
      //productName (display sorted by lowest quantity), number of units needed, warehouseId (sort by distance)
      List<List<String>> products = esql.executeQueryAndReturnResult(String.format("SELECT productName, numberOfUnits FROM PRODUCT WHERE storeId = %s ORDER BY numberOfUnits ASC", storeId));
      String pName;
      int nUnits;
      int whId;
      System.out.println("Please select a product to order:");
      for (int i = 0; i < products.size(); i++){
         System.out.println(String.format("%2d. %s: %s Units on hand.", i+1, products.get(i).get(0), products.get(i).get(1)));
      }
      int sel = readChoice();
      while (sel < 1 || sel > products.size()){
         System.out.println("Invalid selection. Please enter a number in range.");
         sel = readChoice();
      }
      pName = products.get(sel-1).get(0);

      System.out.println("Choose a warehouse by ID");
      System.out.println("Warehouse ID | Distance estimate from your store");
      //warehouseID,area,latitude,longitude
      List<List<String>> whFull = esql.executeQueryAndReturnResult(String.format("SELECT wareHouseID,latitude,longitude FROM WAREHOUSE"));
      //id, dist
      whId = SelectByDistance("Warehouse ID", whFull, latitude, longitude);
      System.out.println(String.format("Warehouse ID: %d", whId));

      System.out.println("How many units would you like?");
      nUnits = readChoice();

      //requestNumber,managerID,warehouseID,storeID,productName,unitsRequested
      //UPDATE the Product table
      esql.executeUpdate(String.format("UPDATE PRODUCT SET numberOfUnits = numberOfUnits + %d WHERE PRODUCT.storeId = %s AND PRODUCT.productName = '%s'", nUnits, storeId, pName));
      //UPDATE the product request table
      //use sequence productsupplyrequests_requestNumber_seq

      esql.executeUpdate(String.format("INSERT INTO PRODUCTSUPPLYREQUESTS (requestNumber,managerID,warehouseID,storeID,productName,unitsRequested) VALUES (nextval('productsupplyrequests_requestNumber_seq'), %s, %s, %s, '%s', %s)", mgrId, whId, storeId, pName, nUnits));

      System.out.println(String.format("Warehouse %d delivered %d units of %s to store %d!", whId, nUnits, pName.trim(), storeId));
   } catch (Exception e){
      System.err.println(e.getMessage());
   }
}

//manager only - input storeID, productName, number of units needed, warehouseID of supplier. assume all locations have enough
public static void placeProductSupplyRequests(Amazon esql) {
   try {
      String c_id = esql.getUserId();
      String qM = String.format("SELECT * FROM USERS U WHERE U.type = 'manager' AND U.userID = %s", c_id);
      if (esql.executeQuery(qM) > 0){
         List<List<String>> storesManaged = esql.executeQueryAndReturnResult(String.format("SELECT storeID, latitude, longitude FROM STORE WHERE managerID = %s", c_id));
         List<List<String>> myloc = esql.executeQueryAndReturnResult(String.format("SELECT latitude, longitude FROM USERS WHERE userId = %s", c_id));
         //for each store managed
         if (storesManaged.size() > 1){ //choose a store
            System.out.println("Select a store among those you manage");
            int storeId = SelectByDistance("Store ID", storesManaged, Double.parseDouble(myloc.get(0).get(0)), Double.parseDouble(myloc.get(0).get(1)));
            int index;
            for (index = 0; index < storesManaged.size(); index++)
               if(storeId == Integer.parseInt(storesManaged.get(index).get(0)) break;

            placeProductSupplyRequests(esql, Integer.parseInt(c_id), storeId, Double.parseDouble(storesManaged.get(index).get(1)), Double.parseDouble(storesManaged.get(index).get(2)));
         }
         else if (storesManaged.size() == 1){ //1 store
            placeProductSupplyRequests(esql, Integer.parseInt(c_id), Integer.parseInt(storesManaged.get(0).get(0)), Double.parseDouble(storesManaged.get(0).get(1)), Double.parseDouble(storesManaged.get(0).get(2)));
         } else
            System.out.println("You are not the manager of any store. No orders to display.");
      } else{
         System.out.println("Access denied: manager access only.");
      }
   } catch (Exception e) {
      System.err.println(e.getMessage());
   }
}
```

10. **Admin options:** This menu option is only available to user of the type ‘admin’ and prompts the user to select to change user or product information. You can then either view or update the specified selection

```
Please make your choice: 10
1. View and update user information
2. View and update product information
Enter your choice: 1
        Enter 1 to view users or 2 to update users: 2
        Enter user id to update: 100
        Enter updated name of user: SQLLover
        Enter updated password of user: xyz
        Enter updated latitude of user: 56.4
        Enter updated longitude of user: 67.3
        Enter updated type of user: admin
```

```java
//ADMIN MENU SELECTION
public static void viewAndUpdateProducts(Amazon esql) {
    try {
        System.out.print("\tEnter 1 to view products or 2 to update products: ");
        String input = in.readLine();
        if("1".equals(input)){
                System.out.print("\tEnter product name to view: ");
                String input_name = in.readLine();
                String q1 = String.format("SELECT * FROM Product P WHERE P.productName = '%s'", input_name);
                esql.executeQueryAndPrintResult(q1);
                esql.executeUpdate(q1);
        } else if ("2".equals(input)){
                System.out.print("\tEnter store id of product to update: ");
                String input_id = in.readLine();
      System.out.print("\tEnter product name of product to update: ");
                String input_name2 = in.readLine();
                System.out.print("\tEnter updated store id: ");
                String new_id = in.readLine();
                System.out.print("\tEnter updated product name: ");
                String new_name = in.readLine();
                System.out.print("\tEnter updated number of units: ");
                String new_units = in.readLine();
                System.out.print("\tEnter updated price per unit: ");
                String new_price = in.readLine();
                String q2 = String.format("UPDATE Product SET storeID = '%s', productName = '%s', numberOfUnits = '%s', pricePerUnit = '%s' WHERE Product.storeID = '%s' AND Product.productName = '%s'", new_id, new_name, new_units, new_price, input_id, input_name2);
                esql.executeUpdate(q2);
                List<List<String>> mgrId = esql.executeQueryAndReturnResult(String.format("SELECT managerID FROM STORE WHERE storeId = %s", new_id));
                esql.executeUpdate(String.format("INSERT INTO PRODUCTUPDATES (updateNumber, managerID, storeID, productName, updatedOn) VALUES (nextval('productupdates_updateNumber_seq'), %s, %s, '%s', TIMESTAMP '%s')", mgrId.get(0).get(0), new_id, new_name, new ja
        } else {
                System.out.println("Invalid choice.");
        }
    } catch (Exception e) {
        System.err.println(e.getMessage());
    }
}
//ADMIN MENU SELECTION
public static void viewAndUpdateUsers(Amazon esql) {
    try {
        System.out.print("\tEnter 1 to view users or 2 to update users: ");
    String input = in.readLine();
    if("1".equals(input)){
        System.out.print("\tEnter user id to view: ");
        String input_id = in.readLine();
        String q1 = String.format("SELECT * FROM USERS U WHERE U.userID = '%s'", input_id);
        esql.executeQueryAndPrintResult(q1);
        esql.executeUpdate(q1);
    } else if ("2".equals(input)){
        System.out.print("\tEnter user id to update: ");
                String input_id2 = in.readLine();
        System.out.print("\tEnter updated name of user: ");
                String new_name = in.readLine();
        System.out.print("\tEnter updated password of user: ");
                String new_password = in.readLine();
        System.out.print("\tEnter updated latitude of user: ");
                String new_lat = in.readLine();
        System.out.print("\tEnter updated longitude of user: ");
                String new_long = in.readLine();
        System.out.print("\tEnter updated type of user: ");
                String new_type = in.readLine().toLowerCase();
                String q2 = String.format("UPDATE Users SET name = '%s', password = '%s', latitude = '%s', longitude = '%s', type = '%s' WHERE Users.userID = '%s'", new_name, new_password, new_lat, new_long, new_type, input_id2);
                esql.executeUpdate(q2);
    } else {
        System.out.println("Invalid choice.");
    }
     } catch (Exception e) {
        System.err.println(e.getMessage());
     }
}
```

Indexes

We have also included the following indexes for extra credit. Each section is explained and justified by the same number indicated in the screenshot comments

1. The composite index on name, password will speed up user authentication
2. The foreign key ID indexes are likely the biggest optimization here, with the frequency of joins and comparisons of these values.
3. Nearly every function we've added has a check for the calling user type, so this index will slightly improve every single set of queries.
4. About 80% of our queries that reference a storeId sort by productName for a cleaner display and user experience. So, this query will speed up each of those, quite frequent, queries. Furthermore, product inventory is frequently updated by purchases and supply requests. This composite index will further speed up each of these queries.
5. In several of these functions, orders are retrieved by customerId and sorted by orderTime. This composite index will optimize each of these queries.
6. Recent updates are frequently queried by managerId and storeId (to match the provided updates with their corresponding manager and store) and sorted by updateOn. This index will optimize these queries.
7. Similar to the composite index 4, order information is retrieved by storeId, since each prompt first requests a specified store. These aggregated queries on orders by storeIds calculate sales, and this index will drastically improve performance on these queries

.

```sql
DROP INDEX IF EXISTS idx_users_name_password;
DROP INDEX IF EXISTS idx_users_userid;
DROP INDEX IF EXISTS idx_store_storeid;
DROP INDEX IF EXISTS idx_store_managerid;
DROP INDEX IF EXISTS idx_orders_customerid;
DROP INDEX IF EXISTS idx_users_type_userid;
DROP INDEX IF EXISTS idx_product_storeid_productname;
DROP INDEX IF EXISTS idx_orders_customerid_ordertime;
DROP INDEX IF EXISTS idx_productupdates_managerid_storeid_updatedon;
DROP INDEX IF EXISTS idx_orders_storeid_productname;

--1. Composite index for user authentication
CREATE INDEX idx_users_name_password
ON USERS (name, password);

--2. Indexes on Foreign Key Columns
CREATE INDEX idx_users_userid
ON USERS (userID);

CREATE INDEX idx_store_storeid
ON STORE (storeId);

CREATE INDEX idx_store_managerid
ON STORE (managerID);

CREATE INDEX idx_orders_customerid
ON ORDERS (customerID);

--3. Composite index for user type and ID searches
CREATE INDEX idx_users_type_userid
ON USERS (type, userID);

--4. Composite index for product searches by store and product name ordering and for inventory management
CREATE INDEX idx_product_storeid_productname
ON PRODUCT (storeId, productName);

--5. Composite index for orders retrieval and ordering by time
CREATE INDEX idx_orders_customerid_ordertime
ON ORDERS (customerID, orderTime DESC);

--6. Composite index for recent updates in product updates
CREATE INDEX idx_productupdates_managerid_storeid_updatedon
ON PRODUCTUPDATES (managerID, storeId, updatedOn DESC);

--7. Composite index for aggregate sales on orders
CREATE INDEX idx_orders_storeid_productname
ON ORDERS (storeId, productName);
```

Triggers were intentionally not implemented, as we did not believe them necessary, and a trigger on product updates would interfere with purchases, as we did not want those to update the productUpdates table. Since we would only then be implementing 2 of 3 appropriate triggers, we decided they would not improve the storefront database experience.

Problems/Findings

Problems we encountered include implementing the functions for the administrator and figuring out how to get the user id of each user. We were able to solve the administrator issue by creating a 10th option to select specifically for administrators at the main menu and implementing a function for that option as well. We were able to solve the user id issue by greeting a global variable that would get the user id from the log in function.

We were unhappy with the misalignment of the executeQueryAndPrintResults function, but with a bit of modification, we are happy with the new, cleaner alignment and overall menu navigation.

Contributions

Shan - implemented functionality for browse stores, browse products, order products, admin, and partially implemented update product information

Chris - implemented functionality for browse orders list, manager, popular product and customer, put supply request, and partially implemented update product information and improved menu navigation and wrote indexes.