CS 166 Project Report

Group Information

Group Number 24
Paimon Goulart (pgoul002)
Ritish Chilkepalli (rchil008)

Implementation Description

The criteria for Phase 3 of CS 166 focuses on creating a useful and intuitive database application for a simulation of the Amazon Store. The project involves building a Java-based user interface and a SQL database structure. Students must improve a Java template to connect to a PostgreSQL database and run SQL queries, as well as load supplied data records into their database. The application allows functions like user registration, login/logout, product browsing and ordering, and product information updating in order to serve non-SQL users, notably customers and managers of the Amazon Store.

Create User:

```
public static void CreateUser(Amazon esql){
   try{
      System.out.print("\tEnter name: ");
      String name = in.readLine();
      System.out.print("\tEnter password: ");
      String password = in.readLine();
      System.out.print("\tEnter latitude: ");
      String latitude = in.readLine();       //enter lat value between [0.0, 100.0]

      if (latitude.matches("^-?[0-9]*\\.?[0-9]+$") && Double.parseDouble(latitude) >= 0.0 && Double.parseDouble(latitude) <= 100.0){
             System.out.print("\tEnter longitude: ");  //enter long value between [0.0, 100.0]
             String longitude = in.readLine();

             if(longitude.matches("^-?[0-9]*\\.?[0-9]+$") && Double.parseDouble(longitude) >= 0.0 && Double.parseDouble(longitude) <= 100.0){

                    String type="Customer";

                    String query = String.format("INSERT INTO USERS (name, password, latitude, longitude, type) VALUES ('%s','%s', %s, %s,'%s')", name, password, latitude, longitude, type);
                    esql.executeUpdate(query);
                    System.out.println ("User successfully created!");
             }
            else{
                    System.out.println("Error: Longitude must be a number between 0.0 and 100.0.");
             }
      }
      else{
             System.out.println("Error: Latitude must be a number between 0.0 and 100.0.");
      }
   }catch(Exception e){
      System.err.println (e.getMessage ());
   }
}//end CreateUser
```

Here, we stuck mostly with the sample code given where a user is prompted to enter a name, password, latitude and longitude, then is put through an Insert Query with SQL. We decided to add input checking for latitude and longitude so that the user gets a concise simplified error message that anyone is able to understand, checking if latitude and longitude are the correct inputs.

Login:

```
...
public static String LogIn(Amazon esql){
   try{
      System.out.print("\tEnter name: ");
      String name = in.readLine();
      System.out.print("\tEnter password: ");
      String password = in.readLine();

      String query = String.format("SELECT * FROM USERS WHERE name = '%s' AND password = '%s'", name, password);
      String IdQuery = String.format("SELECT userID FROM USERS WHERE name = '%s' AND password = '%s'", name, password);
      List<List<String>> userIdResults = esql.executeQueryAndReturnResult(IdQuery);
      if (!userIdResults.isEmpty()){
         userID = userIdResults.get(0).get(0);
      }

      int userNum = esql.executeQuery(query);
      if (userNum > 0){
            return name;
      }
      else{
             System.out.println("Invalid login, user does not exist!");
      }

      return null;
   }catch(Exception e){
      System.err.println (e.getMessage ());
      return null;
   }
}//end
```

Here we stuck with the sample code that was originally given to us. The only change we made was to set a global userID variable to the userID given from login so that we are able to use them for the rest of our queries.

View Stores:

```
public static void viewStores(Amazon esql) {
      try{
       String StoreQuery = String.format("SELECT storeID, latitude, longitude FROM Store");
       String userQuery = String.format("SELECT latitude, longitude FROM Users WHERE userID = '%s'", userID);
       List<List<String>> storeResults = esql.executeQueryAndReturnResult(StoreQuery);
       List<List<String>> userResults = esql.executeQueryAndReturnResult(userQuery);
       double tempDistance;
       int storesize = storeResults.size();
       for(int i = 0; i < storesize; i++){
              tempDistance = esql.calculateDistance(Double.parseDouble(userResults.get(0).get(0)), Double.parseDouble(userResults.get(0).get(1)), Double.parseDouble((storeResults.get(i).get(1))), Doubl
e.parseDouble((storeResults.get(i).get(2))));
              if(tempDistance <= 30.0){
                     System.out.println("Store ID: " + (storeResults.get(i).get(0)) + " Distance: " + tempDistance);
              }
       }
      }
      catch(Exception e){
            System.err.println(e.getMessage());
      }

}
public static void viewProducts(Amazon esql) {
        try{
              System.out.print("\tEnter storeID: ");
              String tempstoreID = in.readLine();
              if(!tempstoreID.matches("^-?[0-9]*\\.?[0-9]+$")){
                     tempstoreID = "0";
              }
              String ItemQuery = String.format("SELECT productName, numberOfUnits, pricePerUnit FROM Product p, Store s WHERE s.storeID = '%s' AND s.storeID = p.storeID", tempstoreID);
              List<List<String>> productResults = esql.executeQueryAndReturnResult(ItemQuery);
              int productsize = productResults.size();
              if(productsize == 0){
                     System.out.println("Store ID does not exist");
              }
              else{
                     for(int i = 0; i < productsize; i++){
                            System.out.println("Product Name: " + productResults.get(i).get(0) + " Number of Units: " + productResults.get(i).get(1) + " Price per Unit: " + productResults.get(i).ge
t(2));
                     }
              }

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

In view stores, we first use two SQL queries, one to retrieve the latitude and longitude from every store in the database, Store Query, and one to retrieve the latitude and longitude of the current user, userQuery. After this, we use a for loop in order to check the retrieved latitudes and longitudes of every store, and compare it to that of the user using calculate distance, we then output the stores that have a distance of less than or equal to 30.0 of the users location.

View Products:

```
    public static void viewProducts(Amazon esql) {
            try{
                    System.out.print("\tEnter storeID: ");
                    String tempstoreID = in.readLine();
                    if(!tempstoreID.matches("^-?[0-9]*\\.?[0-9]+$")){
                            tempstoreID = "0";
                    }
                    String ItemQuery = String.format("SELECT productName, numberOfUnits, pricePerUnit FROM Product p, Store s WHERE s.storeID = '%s' AND s.storeID = p.storeID", tempstoreID);
                    List<List<String>> productResults = esql.executeQueryAndReturnResult(ItemQuery);
                    int productsize = productResults.size();
                    if(productsize == 0){
                            System.out.println("Store ID does not exist");
                    }
                    else{
                            for(int i = 0; i < productsize; i++){
                                    System.out.println("Product Name: " + productResults.get(i).get(0) + " Number of Units: " + productResults.get(i).get(1) + " Price per Unit: " + productResults.get(i).ge
t(2));
                            }
                    }

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

In view products, we prompt the user to enter a store ID, if that store ID is not a number of some sort, it is automatically set to 0 so that the store ID does not exist error is prompted. For this, we use one Query, ItemQuery, to select all of the items from the input store, and then display them using a for loop.

Place Order:

```
    public static void placeOrder(Amazon esql) {
            try{
                    System.out.print("\tEnter storeID: ");
                    String tempstoreID = in.readLine();
                    System.out.print("\tEnter Product Name: ");
                    String tempProductName = in.readLine();
                    System.out.print("\tEnter Number of Units Needed: ");
                    String tempUnits = in.readLine();

                    if(!tempstoreID.matches("^-?[0-9]*\\.?[0-9]+$")){
                            tempstoreID = "0";
                    }
                    String StoreLocationQuery = String.format("SELECT latitude, longitude FROM Store WHERE storeID = '%s'", tempstoreID);
                    String userLocationQuery = String.format("SELECT latitude, longitude FROM Users WHERE userID = '%s'", userID);

                    String stockQuery = String.format("SELECT numberOfUnits FROM Product p, Store s WHERE p.storeID = s.storeID AND productName = '%s' AND s.storeID = '%s'", tempProductName, tempstoreID);

                    String updateItemQuery = String.format("UPDATE PRODUCT SET numberOfUnits = numberOfUnits - %s WHERE storeID = '%s' AND productName = '%s'", tempUnits, tempstoreID, tempProductName);

                    DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
                    LocalDateTime now = LocalDateTime.now();
                    String date = dtf.format(now);

                    String OrderNumber = String.format("SELECT orderNumber FROM Orders ORDER BY orderNumber DESC LIMIT 1;");
                    List<List<String>> mostRecentOrderNumber = esql.executeQueryAndReturnResult(OrderNumber);
                    int thisOrder = Integer.parseInt(mostRecentOrderNumber.get(0).get(0)) + 1;
                    String addOrder = String.format("INSERT INTO ORDERS (orderNumber, customerID, storeID, ProductName, unitsOrdered, orderTime) VALUES ('%s','%s', '%s', '%s','%s','%s')", thisOrder, userID
, tempstoreID, tempProductName, tempUnits, date);

                    List<List<String>> StoreLocation = esql.executeQueryAndReturnResult(StoreLocationQuery);
                    List<List<String>> userLocation = esql.executeQueryAndReturnResult(userLocationQuery);
                    List<List<String>> itemStock = esql.executeQueryAndReturnResult(stockQuery);

                     if(itemStock.size() == 0){
                            System.out.println("Item does not exist at this location");
                    }
                    if(itemStock.size() > 0){
                         int tempstock = Integer.parseInt(itemStock.get(0).get(0));
                         double tempDistance;

                         tempDistance = esql.calculateDistance(Double.parseDouble(userLocation.get(0).get(0)), Double.parseDouble(userLocation.get(0).get(1)), Double.parseDouble((StoreLocation.get(0).get(0
))), Double.parseDouble((StoreLocation.get(0).get(1))));


                        if(tempDistance <= 30){
                                System.out.println("Valid Store");
                                if(tempstock >= Integer.parseInt(tempUnits)){
                                   esql.executeUpdate(addOrder);
                                   esql.executeUpdate(updateItemQuery);
                                   System.out.println("Order successfully placed!");

                                }
                                else{
                                        System.out.println("Invalid Order, not enough units available");
                                }

                        }

                    else{
                            System.out.println("Invalid store, store location too far");
                    }
                    }
            }
            catch(Exception e){
                    System.err.println(e.getMessage());
            }
    }
```

In place order, we first prompt the user to enter a valid storeID, product name, and number of units. We input check all of these values in order to make sure they are correct, such as checking if the input matches the number of units available.

- The first query, store location query, retrieves the latitude and longitude of the input store so that we can check if the store is valid and close enough to the user.

- The second query, user location query is used to get the user location needed to make this comparison.
- After this, we have a stock query, which retrieves the units available from that item, this is then used to check if the number of units fits the amount of units left of that item.
- Then we have an update item query which updates the item in the product database to match the new amount of units after the order has been placed.
- We also have an add order query which adds the order to the orders table after an order is placed.

View Recent Orders:

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

        try{
             String checkManager = String.format("SELECT managerID FROM Store s WHERE managerID = '%s'", userID);
             String CustomerOrders = String.format("SELECT storeID, productName, unitsOrdered, orderTime FROM Orders WHERE customerID = '%s' ORDER BY orderNumber DESC LIMIT 5;", userID);
             String managerCustomerOrders = String.format("SELECT o.orderNumber, u.name, o.storeID, o.productName, o.orderTime FROM Orders o, Users u, Store s WHERE s.storeID = o.storeID AND o.customer
ID = u.userID AND s.managerID = '%s'", userID);
             List<List<String>> managerExists = esql.executeQueryAndReturnResult(checkManager);

             if(managerExists.size() > 0){
                    System.out.println("Orders throughout manager's stores");
                    esql.executeQueryAndPrintResult(managerCustomerOrders);
                    System.out.println("Manager's Orders: ");
             }
             esql.executeQueryAndPrintResult(CustomerOrders);
        }

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

In view recent orders:

- We have a check manager query which is used to check if the user is a manager first.
- Then we have a customer orders query which is used to retrieve the 5 most recent orders that were placed by that customer.
- After this we have a manager customer orders, which is used to display all the orders placed by customers at the store that a manager manages. This is able to both display the customers 5 most recent orders, as well as all the orders from a store a manager manages if a manager were to login and use that option.

Update Product:

```
public static void updateProduct(Amazon esql) {
        try{
                String checkManager = String.format("SELECT managerID FROM Store s WHERE managerID = '%s'", userID);
                List<List<String>> managerExists = esql.executeQueryAndReturnResult(checkManager);

                if(managerExists.size() > 0){
                        System.out.println("Manager Valid");
                        System.out.print("\tEnter storeID for product update: ");
                        String tempstoreID = in.readLine();
                        if(!tempstoreID.matches("^-?[0-9]*\\.?[0-9]+$")){
                                tempstoreID = "0";
                        }
                        String validStore = String.format("SELECT storeID FROM Store WHERE managerID = %s AND storeID = %s", userID, tempstoreID);

                        List<List<String>> validStores = esql.executeQueryAndReturnResult(validStore);

                        if(validStores.size() > 0){
                                System.out.println(validStores.get(0).get(0));
                                System.out.println("Valid Store Selected");

                                System.out.print("\tEnter product name: ");
                                String tempProductName = in.readLine();
                                System.out.print("\tEnter new amount of units: ");
                                String units = in.readLine();
                                System.out.print("\tEnter new price per unit: ");
                                String price = in.readLine();

                                String stockQuery = String.format("SELECT numberOfUnits FROM Product p, Store s WHERE p.storeID = s.storeID AND productName = '%s' AND s.storeID = '%s'", tempProductName,
 tempstoreID);
                                List<List<String>> itemStock = esql.executeQueryAndReturnResult(stockQuery);

                                if(itemStock.size() == 0){
                                        System.out.println("Item does not exist at this location");
                                }

                                else if(Integer.parseInt(units) >= 0 && Double.parseDouble(price) >= 0){
                                        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
                                        LocalDateTime now = LocalDateTime.now();
                                        String date = dtf.format(now);
                                        System.out.println("Valid input");
                                        String updateItemQuery = String.format("UPDATE PRODUCT SET numberOfUnits = %s, pricePerUnit = %s WHERE storeID = '%s' AND productName = '%s'", units, price, temps
toreID, tempProductName);
```

```
                                        String updateNumber = String.format("SELECT updateNumber FROM ProductUpdates ORDER BY updateNumber DESC LIMIT 1;");
                                        List<List<String>> mostRecentUpdateNumber = esql.executeQueryAndReturnResult(updateNumber);
                                        int thisUpdate = Integer.parseInt(mostRecentUpdateNumber.get(0).get(0)) + 1;
                                        System.out.println(thisUpdate);
                                        String addProductUpdate = String.format ("INSERT INTO ProductUpdates (updateNumber, managerID, storeID, productName, updatedOn) VALUES ('%s','%s', '%s', '%s','%s'
", thisUpdate, userID, tempstoreID, tempProductName, date);
                                        esql.executeUpdate(updateItemQuery);
                                        esql.executeUpdate(addProductUpdate);

                                }
                                else{
                                        System.out.println("Invalid Input");
                                }

                        }
                        else{
                                System.out.println("Store selection invalid");
                        }
                }


                else{
                        System.out.println("Invalid, this option only available to managers!");
                }
        }

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

In the Update Item query, we have several queries to implement functionality.

- First we have check manager which checks if the current user is a manager or not as this option is only available to managers.
- After this, we have the manager input a store which is checked using the validStore query, which retrieves all the stores that the manager has access to.
- Then we have the manager input the name of the product, the amount of units they want to change it to, and the new price per unit.
- We then use stock query again to check if that item exists in that store or not.
- After this we check if the input is valid, seeing if the amount of units and price are greater than or equal to 0.
- We also have an update number query, which is used to retrieve the most recent update number, to be used in our update item query to accurately display the update number.
- Then we use Update Item Query to update the item based on the inputs provided by the user.

View Recent Updates:

```
    public static void viewRecentUpdates(Amazon esql) {
            try{
                    String checkManager = String.format("SELECT managerID FROM Store s WHERE managerID = '%s'", userID);
                    List<List<String>> managerExists = esql.executeQueryAndReturnResult(checkManager);
                    if(managerExists.size() > 0){
                            String Updates = String.format("SELECT *  FROM ProductUpdates ORDER BY updateNumber DESC LIMIT 5;");
                            esql.executeQueryAndPrintResult(Updates);

                    }
                    else{
                            System.out.println("Invalid, this option only available to managers!");
                    }
            }

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

In view recent updates:

- We have a check manager query to check if the current user is a manager or not as this option is only available to managers.
- We then have a Updates query, which retrieved the 5 most recent updates from ProductUpdates.

View Popular Products:

```
public static void viewPopularProducts(Amazon esql) {
        try{
                String checkManager = String.format("SELECT managerID FROM Store s WHERE managerID = '%s'", userID);
                List<List<String>> managerExists = esql.executeQueryAndReturnResult(checkManager);
                if(managerExists.size() > 0){

                    System.out.print("\tEnter storeID for popular products: ");
                    String tempstoreID = in.readLine();
                    if(!tempstoreID.matches("^-?[0-9]*\\.?[0-9]+$")){
                            tempstoreID = "0";
                    }
                    String validStore = String.format("SELECT storeID FROM Store WHERE managerID = %s AND storeID = %s", userID, tempstoreID);

                    List<List<String>> validStores = esql.executeQueryAndReturnResult(validStore);

                    if(validStores.size() > 0){
                            String popularItems = String.format("Select productName, storeID, numberOfUnits FROM Product WHERE storeID = %s ORDER BY numberOfUnits ASC LIMIT 5;", tempstoreID);
                            esql.executeQueryAndPrintResult(popularItems);

                    }

                    else{
                            System.out.println("Store selection invalid");
                    }
                }
                else{
                      System.out.println("Invalid, this option only available to managers!");
                }
        }

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

}
```

In view popular products:

- We first have a check manager query which checks if a current user is a manager or not as this option is only available to managers.
- We then have a valid store query which checks if the store imputed by the manager is one of the stores that they manage, as well as if the store exists or not.
- After this, we have a popular items query, which retrieves the items that have the lowest amount of stock, indicating that they are popular and ordered a lot. This query then returns the 5 items with the lowest stock in that table based on the store number.

View Popular Customers:

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

        try{
                String checkManager = String.format("SELECT managerID FROM Store s WHERE managerID = '%s'", userID);
                List<List<String>> managerExists = esql.executeQueryAndReturnResult(checkManager);
                if(managerExists.size() > 0){

                    System.out.print("\tEnter storeID for popular customers: ");
                    String tempstoreID = in.readLine();
                    if(!tempstoreID.matches("^-?[0-9]*\\.?[0-9]+$")){
                            tempstoreID = "0";
                    }
                    String validStore = String.format("SELECT storeID FROM Store WHERE managerID = %s AND storeID = %s", userID, tempstoreID);

                    List<List<String>> validStores = esql.executeQueryAndReturnResult(validStore);

                    if(validStores.size() > 0){
                            String popularCustomers = String.format("SELECT u.name, COUNT(o.customerID) FROM Orders o, Users u WHERE o.customerID = u.userID AND storeID = %s GROUP BY(u.name) ORDER BY
COUNT(o.customerID) DESC LIMIT 5;", tempstoreID);
                            esql.executeQueryAndPrintResult(popularCustomers);

                    }

                    else{
                            System.out.println("Store selection invalid");
                    }
                }
                else{
                      System.out.println("Invalid, this option only available to managers!");
                }
        }

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


}
```

In view popular customers:

- We first have a check manager query which checks if the current user is a manager as this option is only available to managers.

- We then have a valid store query which checks if the store entered is a store that that manager manages.
- Then we have a popular customer query which retrieves the 5 highest counted customers within that store. The query first counts the names of each customer that appears in the store's orders, then it orders this by the count, and returns the 5 highest counted customers.

Place Product Supply Request:

```
    public static void placeProductSupplyRequests(Amazon esql) {
            try{
                    String checkManager = String.format("SELECT managerID FROM Store s WHERE managerID = '%s'", userID);
                    List<List<String>> managerExists = esql.executeQueryAndReturnResult(checkManager);
[
                   if(managerExists.size() > 0){
                           System.out.print("\tEnter storeID: ");
                           String tempstoreID = in.readLine();
                           if(!tempstoreID.matches("^-?[0-9]*\\.?[0-9]+$")){

                                   tempstoreID = "0";
                           }

                           String validStore = String.format("SELECT storeID FROM Store WHERE managerID = %s AND storeID = %s", userID, tempstoreID);

                           List<List<String>> validStores = esql.executeQueryAndReturnResult(validStore);

                           if(validStores.size() > 0){
                                 System.out.print("\tEnter Product Name: ");
                                 String tempProductName = in.readLine();
                                 String stockQuery = String.format("SELECT numberOfUnits FROM Product p, Store s WHERE p.storeID = s.storeID AND productName = '%s' AND s.storeID = '%s'", tempProductName, t
empstoreID);
                                 List<List<String>> itemStock = esql.executeQueryAndReturnResult(stockQuery);

                                 if(itemStock.size() > 0){

                                        System.out.print("\tEnter Number of Units Needed: ");
                                        String tempUnits = in.readLine();
                                        if(Integer.parseInt(tempUnits) > 0){
                                                System.out.print("\tEnter warehouseID: ");
                                                String tempWarehouse = in.readLine();

                                                String validWarehouse = String.format("SELECT WarehouseID FROM Warehouse WHERE WarehouseID = %s", tempWarehouse);

                                                List<List<String>> validWarehouses = esql.executeQueryAndReturnResult(validWarehouse);

                                                if(validWarehouses.size() > 0){
                                                        String requestNumber = String.format("SELECT requestNumber FROM ProductSupplyRequests ORDER BY requestNumber DESC LIMIT 1;");
                                                        List<List<String>> mostRecentRequestNumber = esql.executeQueryAndReturnResult(requestNumber);
                                                        int thisRequest = Integer.parseInt(mostRecentRequestNumber.get(0).get(0)) + 1;
                                                        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
                                                        LocalDateTime now = LocalDateTime.now();
                                                        String date = dtf.format(now);

                                                        String addSupplyRequests = String.format ("INSERT INTO ProductSupplyRequests (requestNumber, managerID, warehouseID, storeID, productName, unitsRequ
ested) VALUES ('%s', '%s','%s', '%s', '%s','%s')", thisRequest, userID, tempWarehouse, tempstoreID, tempProductName, tempUnits);

                                                        String updateNumber = String.format("SELECT updateNumber FROM ProductUpdates ORDER BY updateNumber DESC LIMIT 1;");
                                                        List<List<String>> mostRecentUpdateNumber = esql.executeQueryAndReturnResult(updateNumber);
                                                        int thisUpdate = Integer.parseInt(mostRecentUpdateNumber.get(0).get(0)) + 1;

                                                        String addProductUpdate = String.format ("INSERT INTO ProductUpdates (updateNumber, managerID, storeID, productName, updatedOn) VALUES ('%s','%s', '
%s', '%s','%s')", thisUpdate, userID, tempstoreID, tempProductName, date);

                                                        String updateItemQuery = String.format("UPDATE PRODUCT SET numberOfUnits = numberOfUnits + %s WHERE storeID = '%s' AND productName = '%s'", tempUnit
s, tempstoreID, tempProductName);

                                                        esql.executeUpdate(addSupplyRequests);
                                                        esql.executeUpdate(addProductUpdate);
                                                        esql.executeUpdate(updateItemQuery);



                                                }
                                                else{
                                                        System.out.println("Invalid warehouse selected");
                                                }

                                        }
                                        else{
                                                System.out.println("Unit amount must be greater than 0");
                                        }
                                 }
                                 else{
                                        System.out.println("Item does not exist at this location");
                                 }
                           }
                           else{
                                 System.out.println("Store selection invalid");
                           }
                   }
                   else{
                           System.out.println("Invalid, this option only available to managers!");
                   }


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

    }
```

In place product supply request:

- First we have check manager which checks if the current user is a manager or not as this option is only available to managers.
- After this, we have the manager input a store which is checked using the validStore query, which retrieves all the stores that the manager has access to.

- After this, we have an item stock query which is able to retrieve the counts of items in the store. Indicating if that item exists in the store or not.
- After this, we have valid Warehouse, which checks if the warehouse imputed by the manager is a warehouse that exists.
- Then we have a request number query which is used retrieve the most recent request number, in order to accurately add the request to the table.
- After this we have the add supply request query which is used to add the supply request to the supply request table.
- We then have an update number table, which is used to get the most recent update number from product updates in order to be inserted later to the product update table as an accurate update number.
- We then have add product update, which adds the product update to the product update table after the request has been made.
- We then have an update item query, which is used to then update the product in the product table after the request has been made.

View Admin Options:

```
public static void viewAdminOptions(Amazon esql) {
        try{
            String checkAdmin = String.format("SELECT userID FROM Users s WHERE type = 'admin' AND userID = '%s'", userID);
            List<List<String>> AdminResults = esql.executeQueryAndReturnResult(checkAdmin);

            if(AdminResults.size() > 0){
                    System.out.println("Hello Admin, what would you like to do?");
                    System.out.println("1. View All User Information");
                    System.out.println("2. View All Product Information");
                    System.out.println("3. Change User Information");
                    System.out.println("4. Change Product Information");
                    System.out.println("5. Exit");
                    System.out.print("Please make your choice: ");
                    String choice = in.readLine();

                    if(Integer.parseInt(choice) == 1){
                            String showUsers = String.format("SELECT * FROM Users");
                            esql.executeQueryAndPrintResult(showUsers);
                    }
                    else if(Integer.parseInt(choice) == 2){
                            String showItems = String.format("SELECT * FROM Product");
                            esql.executeQueryAndPrintResult(showItems);
                    }
                    else if (Integer.parseInt(choice) == 3){
                            System.out.print("Enter the User ID for user you would like to change: ");
                            String tempuserID = in.readLine();

                            String checkUser = String.format("SELECT name FROM Users s WHERE userID = '%s'",tempuserID);
                            List<List<String>> userResult = esql.executeQueryAndReturnResult(checkUser);
                            if(!userResult.isEmpty()){
                                    System.out.println("Would you like to change user name, password, latitude, longitude, or all?");
                                    System.out.println("1. Change User Name");
                                    System.out.println("2. Change Password");
                                    System.out.println("3. Change Latitude");
                                    System.out.println("4. Change Longitude");
                                    System.out.println("5. Change All");
                                    System.out.println("6. Exit");
                                    System.out.print("Please make you choice: ");
                                    String userChoice = in.readLine();

                                    if(Integer.parseInt(userChoice) == 1){
                                            System.out.print("Enter the new User Name you would like to use: ");
                                            String newUserName = in.readLine();
                                            String updateNameQuery = String.format("UPDATE Users SET name = '%s' WHERE userID = '%s'", newUserName, tempuserID);
                                            esql.executeUpdate(updateNameQuery);
                                            System.out.println("Change successful!");
                                    }
                                    else if(Integer.parseInt(userChoice) == 2){
                                            System.out.print("Enter the new Password you would like to use: ");
                                            String newPassword = in.readLine();
                                            String updateNameQuery = String.format("UPDATE Users SET password = '%s' WHERE userID = '%s'", newPassword, tempuserID);
                                            esql.executeUpdate(updateNameQuery);
                                            System.out.println("Change successful!");

                                    }
                                    else if(Integer.parseInt(userChoice) == 3){
                                            System.out.print("Enter the new latitude you would like to use: ");
                                            String newLatitude = in.readLine();

                                            if(newLatitude.matches("^-?[0-9]*\\.?[0-9]+$") && Double.parseDouble(newLatitude) >= 0.0 && Double.parseDouble(newLatitude) <= 100.0){
                                                                                                                                                                    859,3-24      83%
                                                                                                                                                                    1,1           Top
```

```
                                        String updateLatitudeQuery = String.format("UPDATE Users SET latitude = '%s' WHERE userID = '%s'", newLatitude, tempuserID);
                                        esql.executeUpdate(updateLatitudeQuery);
                                        System.out.println("Change successful!");
                                }
                                else{
                                        System.out.println("Error: Latitude must be a number between 0.0 and 100.0.");
                                }
                        }
                        else if(Integer.parseInt(userChoice) == 4){
                                System.out.print("Enter the new longitude you would like to use: ");
                                String newLongitude = in.readLine();

                                if(newLongitude.matches("^-?[0-9]*\\.?[0-9]+$") && Double.parseDouble(newLongitude) >= 0.0 && Double.parseDouble(newLongitude) <= 100.0){
                                        String updateLongitudeQuery = String.format("UPDATE Users SET longitude = '%s' WHERE userID = '%s'", newLongitude, tempuserID);
                                        esql.executeUpdate(updateLongitudeQuery);
                                        System.out.println("Change successful!");
                                }
                                else{
                                        System.out.println("Error: Longitude must be a number between 0.0 and 100.0.");
                                }
                        }
                        else if(Integer.parseInt(userChoice) == 5){
                                System.out.print("Enter the new User Name you would like to use: ");
                                String newUserName = in.readLine();

                                System.out.print("Enter the new Password you would like to use: ");
                                String newPassword = in.readLine();

                                System.out.print("Enter the new latitude you would like to use: ");
                                String newLatitude = in.readLine();

                                if(newLatitude.matches("^-?[0-9]*\\.?[0-9]+$") && Double.parseDouble(newLatitude) >= 0.0 && Double.parseDouble(newLatitude) <= 100.0){
                                        System.out.print("Enter the new longitude you would like to use: ");
                                        String newLongitude = in.readLine();
                                        if(newLongitude.matches("^-?[0-9]*\\.?[0-9]+$") && Double.parseDouble(newLongitude) >= 0.0 && Double.parseDouble(newLongitude) <= 100.0){
                                                String updateEverythingQuery = String.format("UPDATE Users SET name = '%s', password = '%s', latitude = '%s', longitude = '%s'  WHERE userID
 = '%s'", newUserName, newPassword, newLatitude, newLongitude, tempuserID);
                                                esql.executeUpdate(updateEverythingQuery);
                                                System.out.println("Change successful!");
                                        }
                                        else{
                                                System.out.println("Error: Longitude must be a number between 0.0 and 100.0.");
                                        }
                                }
                                else{
                                        System.out.println("Error: Latitude must be a number between 0.0 and 100.0.");
                                }
                        }
                        else if(Integer.parseInt(userChoice) == 6){
                                System.out.println("Goodbye!");

                        }
                        else{
                                System.out.println("Invalid Choice");
                        }

                }
                else{
                        System.out.println("Invalid userID, user does not exist");
                                                                                                                                                                912,3-24      89%

                }
        }
        else if (Integer.parseInt(choice) == 4){
                System.out.print("Enter the Store ID for product you would like to change: ");
                String tempstoreID = in.readLine();
                if(!tempstoreID.matches("^-?[0-9]*\\.?[0-9]+$")){
                        tempstoreID = "0";
                }
                System.out.print("Enter the product you would like to change: ");
                String tempproductName = in.readLine();
                String checkStore = String.format("SELECT storeID, productName FROM Product s WHERE storeID = '%s' AND productName = '%s'",tempstoreID, tempproductName);
                List<List<String>> storeResult = esql.executeQueryAndReturnResult(checkStore);

                if(!storeResult.isEmpty()){
                        System.out.println("Valid store and product combination");
                        System.out.println("Would you like to change number of units, price per unit, or all?");
                        System.out.println("1. Change Number of Units");
                        System.out.println("2. Change Price Per Unit");
                        System.out.println("3. Change All");
                        System.out.println("4. Exit");
                        System.out.print("Please make you choice: ");

                        String userChoice = in.readLine();

                        if(Integer.parseInt(userChoice) == 1){
                                System.out.print("Enter the new number of units: ");
                                String newUnits = in.readLine();
                                if(newUnits.matches("^-?[0-9]*\\.?[0-9]+$") && Integer.parseInt(newUnits) >= 0){
                                        String updateProductNameQuery = String.format("UPDATE Product SET numberOfUnits = '%s' WHERE storeID = '%s' AND productName = '%s'", newUnits, temps
toreID, tempproductName);
                                        esql.executeUpdate(updateProductNameQuery);
                                        System.out.println("Change successful!");
                                }
                                else{
                                        System.out.println("Error, new number of units must be a number greater than or equal to 0");
                                }
                        }

                        else if(Integer.parseInt(userChoice) == 2){
                                System.out.print("Enter the new price per unit: ");
                                String newPrice = in.readLine();
                                if(newPrice.matches("^-?[0-9]*\\.?[0-9]+$") && Integer.parseInt(newPrice) >= 0){
                                        String updateProductPriceQuery = String.format("UPDATE Product SET pricePerUnit = '%s' WHERE storeID = '%s' AND productName = '%s'", newPrice, temps
toreID, tempproductName);
                                        esql.executeUpdate(updateProductPriceQuery);
                                        System.out.println("Change successful!");
                                }
                                else{
                                        System.out.println("Error, new price of units must be a number greater than or equal to 0");
                                }

                        }

                   if(Integer.parseInt(userChoice) == 3){
                                System.out.print("Enter the new number of units: ");
                                String newUnits = in.readLine();
                                if(newUnits.matches("^-?[0-9]*\\.?[0-9]+$") && Integer.parseInt(newUnits) >= 0){
                                        System.out.print("Enter the new price per unit: ");
                                        String newPrice = in.readLine();
                                        if(newPrice.matches("^-?[0-9]*\\.?[0-9]+$") && Integer.parseInt(newPrice) >= 0){
                                                                                                                                                                975,24        95%
```

```
                                                String updateProductPriceQuery = String.format("UPDATE Product SET numberOfUnits = '%s', pricePerUnit = '%s' WHERE storeID = '%s' AND produc
tName = '%s'", newUnits, newPrice, tempstoreID, tempproductName);
                                                esql.executeUpdate(updateProductPriceQuery);
                                                System.out.println("Change successful!");
                                        }
                                                else{
                                                System.out.println("Error, new price of units must be a number greater than or equal to 0");
                                                }
                                        }
                                        else{
                                                System.out.println("Error, new number of units must be a number greater than or equal to 0");
                                        }
                                }
                                else if(Integer.parseInt(userChoice) == 4){
                                        System.out.println("Goodbye!");
                                }
                                else{
                                        System.out.println("Invalid choice");
                                }


                        }

                        else{
                                System.out.println("Invalid storeID or product name, that combination of storeID and product name does not exist!");
                        }

                }
                else if (Integer.parseInt(choice) == 5){
                        System.out.println("Goodbye!");

                }

                else{
                        System.out.println("Invalid choice");
                }

            }
            else{
                System.out.println("Invalid, this option is only availiable to Administrators!");
            }
         }
         catch(Exception e){
               System.err.println(e.getMessage());
         }
   }

}//end Amazon
```

In View Admin Options, we display the options that an admin has, the ability to update any user in the table, as well as the ability to update any product in the table. We decided to make it so an Admin cannot change a UserID, Product Name, or StoreID in the tables as this will mess up the primary key, and would give errors if attempted to be done. We decided to leave this functionality out in order to keep a well-maintained database with no duplicate errors, but we allowed admins to change everything else.

- We reused a lot of the queries as shown above to implement the functionality here. However, some new queries include:
- Check Admin, this is used to check if the current user is an admin or not, giving them access to make changes in this table.
- Show users, which shows all the users in the users table.
- Show product, which shows all the products in the product table.
- Then different variations of update user in order to update specific fields in user like name, password, etc.
- Then different variations of update product in order to update units available and price per unit.

Extra Credit:
One of the extra credit fields we decided to work on was having a robust error checker in our UI, in order to give users errors that are more understandable to an everyday person rather than someone who needs to be proficient in coding. As shown above, we implemented this through a combination of queries, and several input checks as well.

Here are some examples of error checks done through the program with the inputs and outputs given:

Example of input only being available to manager:

```
Please make your choice: 6
Invalid, this option only available to managers!
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. View Admin Options
.......................
20. Log out
Please make your choice:
```

Example of product name and storeID being imputed as random strings:

```
Please make your choice: 4
Enter the Store ID for product you would like to change: jkdfnasdfs
Enter the product you would like to change: fasasada
Invalid storeID or product name, that combination of storeID and product name does not exist!
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. View Admin Options
.......................
20. Log out
Please make your choice:
```

Another example of storeID being imputed as a random string:

```
Please make your choice: 7
        Enter storeID for popular products: asndaodn
Store selection invalid
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. View Admin Options
.........................
20. Log out
Please make your choice:
```

Indexes:
Another field that we decided to implement as extra credit were indexes in order to speed up and optimize our queries. Here is what our Indexes.sql looks like:

```
DROP INDEX IF EXISTS idx_users_name;
DROP INDEX IF EXISTS idx_users_type;
DROP INDEX IF EXISTS idx_store_managerID;
DROP INDEX IF EXISTS idx_product_numberOfUnits;
DROP INDEX IF EXISTS idx_orders_customerID;
DROP INDEX IF EXISTS idx_orders_storeID;
DROP INDEX IF EXISTS idx_orders_productName;
DROP INDEX IF EXISTS idx_orders_orderTime;
DROP INDEX IF EXISTS idx_supplyrequests_managerID;
DROP INDEX IF EXISTS idx_supplyrequests_warehouseID;
DROP INDEX IF EXISTS idx_supplyrequests_storeID;
DROP INDEX IF EXISTS idx_productupdates_managerID;

CREATE INDEX idx_users_name ON Users USING BTREE (name);
CREATE INDEX idx_users_type ON Users USING BTREE (type);
CREATE INDEX idx_store_managerID ON Store USING BTREE (managerID);
CREATE INDEX idx_product_numberOfUnits ON Product USING BTREE (numberOfUnits);
CREATE INDEX idx_orders_customerID ON Orders USING BTREE (customerID);
CREATE INDEX idx_orders_storeID ON Orders USING BTREE (storeID);
CREATE INDEX idx_orders_productName ON Orders USING BTREE (productName);
CREATE INDEX idx_orders_orderTime ON Orders USING BTREE (orderTime);
CREATE INDEX idx_supplyrequests_managerID ON ProductSupplyRequests USING BTREE (managerID);
CREATE INDEX idx_supplyrequests_warehouseID ON ProductSupplyRequests USING BTREE (warehouseID);
CREATE INDEX idx_supplyrequests_storeID ON ProductSupplyRequests USING BTREE (storeID);
CREATE INDEX idx_productupdates_managerID ON ProductUpdates USING BTREE (managerID);
```

Here is what each index aimed at doing:

idx_users_name: Enhances the speed of login operations by quickly locating user records based on the name.

idx_users_type: Facilitates efficient querying of users based on their type (e.g., distinguishing between customers, managers, and admins).

idx_store_managerID: Speeds up retrieval of stores managed by a specific user, important for functionalities related to managers.

idx_product_numberOfUnits: Optimizes product-related queries that involve the number of available units, such as checking stock levels.

idx_orders_customerID, idx_orders_storeID, idx_orders_productName: These indexes improve the performance of queries to find orders by customer ID, store ID, and product name, respectively, which is used for order management and reporting.

idx_orders_orderTime: Enables efficient retrieval of recent orders by indexing the order time, supporting functionalities like browsing the last 5 recent orders.

idx_supplyrequests_managerID, idx_supplyrequests_warehouseID,

idx_supplyrequests_storeID: These indexes are essential for quickly processing and managing product supply requests, linking managers, warehouses, and stores.

idx_productupdates_managerID: Aids in the fast retrieval of product update records for managers, which is key for managing and reviewing product updates.

Problems/Findings
Implementation from SQL queries to Java. We had a hard time getting used to java as we were both relatively new to it. Implementing the SQL queries with correct syntax at first was difficult however we quickly got used to it. Having Indexes to work as well as Drop Indexes was difficult as we never had to create indexes for a database this large, so it was difficult finding indexes that work, as well as determining what to index and where. Some queries were complex and took longer time, queries like place order, and admin were particularly difficult, as they were very complicated functions, and took a unison of queries working together in order to implement properly.

Contributions
Ritish - Implemented indexes such as view stores and view popular users. Contributed to helping with the SQL queries and also helped translate into Java implementation
Paimon - Place orders, View recent orders, Update products, Place product supplier requests, and view admin option. Contributed to the SQL queries as well as debugging.