CS 166 Project Report

Group Information
Group 3
Shaan Pakala (NetID: spaka002)
Ainaz Estiri (NetID: aesti002)

High Level Implementation Description:

This implementation of the Amazon stores uses the terminal as its user interface. The user navigates a menu to login or create an account, and once the user logins there is a menu for options the user can select. Some of these options include viewing stores, placing an order, updating product, viewing store customer information, editing users, and other functions. Some functions are only available for managers accounts, and some only for administrator accounts. If the option selected is to display something, the results will appear directly above the menu, and then the user can select another option. A lot of the functions to display information will have further options to select an attribute such as storeID for example. Also the user can delete their account, upon which the user's order information will also be deleted (using a trigger) to respect the user's privacy. Admin accounts can also delete any store, which will then delete the store's products, orders, ProductSupplyRequests, and ProductUpdates (using a trigger). Admin accounts can also delete any warehouse, which will then delete all the warehouse's ProductSupplyRequests.

Functions:

Create Account

This function allows the user to create a new customer account. This makes sure the username is not already taken to ensure they are unique.

Log In

This function requires the user to enter the username and password to log into that account. If the username does not exist, the user is taken back to the menu. If the username does exist and then the password is wrong, the user is also taken back to the menu. Otherwise the user is prompted to select one of the following menu options.

Menu Options:

99) Delete Account

The user is allowed to delete their account. Managers, however, cannot perform this function because they have to manage their store. Upon deletion, the user's entire order information will be deleted in order to respect the user's privacy.

1) View stores within 30 miles

```
storeid longitude       latitude
1       25.580180       94.771430
12      14.130330       81.018290
```

```
String query = "SELECT Store.storeID, Store.longitude, Store.latitude "+
               "FROM Store "+
               "WHERE SQRT(POWER((Store.longitude - " + esql.current_userLong +
                  "), 2) + POWER((Store.latitude - " + esql.current_userLat + "), 2)) < 30";
```

This is the query to compute the stores within 30 miles of the current user. The current user information is just stored in variables of the Amazon class so executing this query will retrieve those stores.

2) View Product List

productname	numberofunits	priceperunit
7up	28	3
Pepsi	32	4
Lemonade	26	8
Brisk	48	3
Orange Juice	41	6
Donuts	39	7
Pudding	24	3
Ice Cream	50	6
Hot and Sour Soup	46	5
Egg	38	3

```
String query = "SELECT Product.productName, Product.numberOfUnits, Product.pricePerUnit "+
               "FROM Product "+
               "WHERE Product.storeID = ";
System.out.print("\tEnter Store ID: ");
String storeID = in.readLine();

// check if this is an actual store
if (!valid_storeID(esql, storeID)){
   System.out.println("\n----------------------\n| Invalid Store ID! |\n\n----------------------\n");
   return;}

query += storeID;
```

Here the user can view product information of the store, where the user inputs the storeID. If this store doesn't exist or the storeID is not an integer, "valid_storeID" will return false and take the user back to the menu. Otherwise the results will display.

3) Place an Order

```
public static void placeOrder(Amazon esql) {
   try{
      System.out.print("\tEnter storeID: ");
      String storeID = in.readLine();

      // check if this is an actual store
      if (!valid_storeID(esql, storeID)){
         System.out.println("\n----------------------\n| Invalid Store ID! |\n\n----------------------\n");
         return;}

      String storeLat = esql.executeQueryAndReturnResult(
                  "SELECT Store.latitude "+
                  "FROM Store "+
                  "WHERE Store.storeID = " + storeID
      ).get(0).get(0);

      String storeLong = esql.executeQueryAndReturnResult(
                  "SELECT Store.longitude "+
                  "FROM Store "+
                  "WHERE Store.storeID = " + storeID
      ).get(0).get(0);

      double sLat = Double.parseDouble(storeLat);
      double sLong = Double.parseDouble(storeLong);

      double uLat = Double.parseDouble(esql.current_userLat);
      double uLong = Double.parseDouble(esql.current_userLong);

      double distance = esql.calculateDistance(sLat, sLong, uLat, uLong);

      if (distance > 30) {
         System.out.println("Sorry, you have to be within 30 miles of the store. \nSelect Option 1 to see stores within 30 miles.\n\n");
         return;
      }

      System.out.print("\tEnter product name: ");
      String productName = in.readLine();

      // check if this is an actual product in the store
      if (!valid_product(esql, storeID, productName)){
         System.out.println("\n----------------------\n| Invalid Product Name! |\n\n----------------------\n");
         return;}
```

Here is the beginning of the place order function. This shows the user being prompted for the storeID to order from. This function first makes sure the storeID is valid. Then it makes sure the user is within 30 miles of the store, then if they are satisfied it asks the

user for the product name. It makes sure this is a valid product in this specified store.

```
System.out.print("\tEnter number of units: ");
String numUnits = in.readLine();

if (!valid_number(numUnits, true)){
   System.out.println("\n----------------------\n| Invalid Number! |\n\n----------------------\n");
   return;}

// check if there is enough units available
if (!valid_num_units(esql, numUnits, storeID, productName)){
   System.out.println("\n----------------------\n| Sorry, not enough units available. |\n\n----------------------\n");
   return;}

String price = esql.executeQueryAndReturnResult(
            "SELECT (Product.pricePerUnit * " + numUnits + ") " +
            "FROM Product " +
            "WHERE Product.storeID = " + storeID + " AND Product.productName = \'" + productName + "\'"
).get(0).get(0);

System.out.print("\nThis costs $" + price + ". \nEnter yes to confirm: ");
String response_y = in.readLine();

if (! (response_y.equals("yes") || response_y.equals("Yes") || response_y.equals("y") || response_y.equals("Y"))){return;}

String query = "INSERT INTO Orders (customerID, storeID, productName, unitsOrdered, orderTime) " +
            "VALUES (" + esql.current_userID + ", " + storeID +
            ", \'" + productName + "\', " + numUnits + ", CURRENT_TIMESTAMP)";

esql.executeUpdate(query);
```

Here the user is prompted to enter the number of units to order, and the function makes sure there are enough units available in store. Then the price of the order is displayed (calculated using Product.pricePerUnit * numUnits), and the user is asked to confirm. If they are satisfied, then the query to insert the order is executed. The product information will be updated as well using a trigger.

4) View 5 Recent Orders

```
ordernumber     productname     unitsordered    storeid ordertime
501     7up                             10      12      2024-03-17 12:24:34.895716
452     Pepsi                           43      12      2016-09-10 19:16:00
409     Hot and Sour Soup               20      13      2016-09-10 18:47:00
269     Hot and Sour Soup               5       16      2016-09-10 16:53:00
5       Orange Juice                    20      4       2016-09-10 13:02:00
```

```
String query = "SELECT Orders.orderNumber, Orders.productName, Orders.unitsOrdered, Orders.storeId, Orders.orderTime " +
            "FROM Orders "+
            "WHERE Orders.customerID = " + esql.current_userID + " " +
            "ORDER BY Orders.orderTime DESC " +
            "LIMIT 5";

int rowCount = esql.executeQueryAndPrintResult(query);
System.out.println ("total row(s): " + rowCount);
```

Here is the query to simply display Order information for all the 5 recent orders the current user has made.

5) View 10 Favorite Products

```
productname     total_units_ordered
Pepsi                           43
Hot and Sour Soup               25
Orange Juice                    20
7up                             10
```

```
System.out.print("\tEnter StoreID (enter \'-\' for all): ");
String storeID = in.readLine();

String query = "SELECT productName, SUM(Orders.unitsOrdered) as Total_Units_Ordered";

if (!storeID.equals("-")){
   // check if this is an actual store
   if (!valid_storeID(esql, storeID)){
      System.out.println("\n----------------------\n| Invalid Store ID! |\n\n----------------------\n");
      return;}
   query+= ", storeID ";
   }
else{query+= " ";}

query+= "FROM Orders "+
        "WHERE customerID = " + esql.current_userID + " ";

if (!storeID.equals("-")){query+= "AND storeID = " + storeID + " ";}

query+= "GROUP BY productName";

if (!storeID.equals("-")){query+= ", storeID ";}
else{query+=" ";}

query+= "ORDER BY Total_Units_Ordered DESC " +
        "LIMIT 10";

int rowCount = esql.executeQueryAndPrintResult(query);
System.out.println ("total row(s): " + rowCount);
```

This function asks the user for the storeID (user can select all by entering “-”) to display their most ordered products from these stores. It will return at most 10 products in descending order of total units ordered.

6) View Store Information

```
storeid longitude       latitude        dateestablished manager_name
1       25.580180       94.771430       1953-03-13      Luz
2       3.659240        86.247250       1955-04-03      Karianne
3       14.251070       9.255290        1938-12-30      Paige
4       59.023060       1.174670        1954-04-17      Karianne
5       53.958550       42.096240       1984-09-07      Aubrey
6       64.677520       30.408910       1971-03-23      Rozella
7       87.339080       81.234030       1950-05-05      Paige
8       49.595340       7.189840        1971-04-07      Rozella
9       75.928090       10.683980       1968-12-02      Clotilde
10      17.303860       9.746990        1908-09-11      Rozella
11      10.882990       17.009880       1998-05-19      Clotilde
12      14.130330       81.018290       1958-06-13      Paige
13      28.841020       3.444470        1926-12-05      Paige
14      22.354760       21.924100       1966-10-17      Aubrey
15      88.319440       30.798250       1933-08-12      Aubrey
16      0.303490        74.187430       1975-11-24      Aubrey
17      1.163500        62.558630       1957-12-13      Karianne
18      62.756950       77.795670       1927-02-21      Rozella
19      58.764400       17.992480       1956-11-24      Luz
20      53.623250       40.649810       1970-05-10      Luz
```

```
String query = "SELECT Store.storeID, Store.longitude, Store.latitude, Store.dateEstablished, Users.name as manager_name ";

query+= "FROM Store, Users "+
        "WHERE Store.managerID = Users.userID";

if (!storeID.equals("-")){query+= " AND Store.storeID = " + storeID;}
```

First the function asks the user for the storeID to view the information of (or all of the stores as pictured). This function makes sure the storeID is valid (not pictured since it is the same as previous functions). Then the function computes the above query to fetch information about the store, including location, date established, and manager_name.

Manager only functions:

7) View Stores you Manage

storeid	dateestablished	number_of_products	number_of_orders	total_order_income
12	1958-06-13	10	18	32382
13	1926-12-05	10	20	19420
7	1950-05-05	10	23	57431
3	1938-12-30	10	30	88950

```
public static void managerViewStores(Amazon esql){
   // only managers can perform this function
   if (!esql.current_userType.equals("manager")){
      System.out.println("Sorry, only managers can perform this function.\n\n");
      return;
   }
   try{

      String query = "SELECT Store.storeID, Store.dateEstablished, "+
                     "COUNT(DISTINCT Product.productName) as number_of_products, COUNT(DISTINCT Orders.orderNumber) as number_of_orders, "+
                     "SUM(Product.numberOfUnits * Product.pricePerUnit) as total_order_income " +
                     "FROM Product, Orders, Store " +
                     "WHERE Store.managerID = " + esql.current_userID + " " +
                     "AND Product.storeID = Store.storeID AND Orders.storeID = Store.storeID " +
                     "GROUP BY Store.storeID, Store.dateEstablished " +
                     "ORDER BY number_of_orders";


      int rowCount = esql.executeQueryAndPrintResult(query);
      System.out.println ("total row(s): " + rowCount);
   }catch(Exception e){
      System.err.println (e.getMessage());
   }

}
```

This function displays store information for all the stores the current user manages. First it makes sure the user is a manager (regular customers and admins cannot use this function). Then it displays storeID, date established, number_of_products, number_of_orders, and total_order_income. The last three attributes are computed in the query.

8) Update Product

This is a manager only function. For the update product function, the user is prompted to enter a storeID and product name to update. These two are checked to make sure they both exist. Then the number of units and price per unit can be updated by the manager. These numbers are checked to make sure they are actually numbers and they are not negative. Then the product information is updated accordingly.

9) View 5 Recent Product Updates

updatenumber	managerid	storeid	productname	updatedon
51	10	12	7up	2024-03-17 12:41:46.254892
46	10	13	Donuts	2016-09-10 13:45:00
45	10	12	Orange Juice	2016-09-10 13:44:00
35	10	13	Orange Juice	2016-09-10 13:34:00
34	10	12	Brisk	2016-09-10 13:33:00

This function retrieves the 5 most recent product updates that the current user performed. This will not let the user perform this function if they are not a manager.

10) View 5 popular items

productname	count
Pudding	3
Orange Juice	3
Brisk	3
7up	2
Egg	2

This function displays the 5 most popular items (products with the most number of orders) at a store that the user inputs. This function will only work for managers and it will only work if the manager is the actual manager of the inputted store. This also makes sure the storeID is valid (like in previous functions).

11) View 5 popular customers

customerid	name	count
1	Admin	2
94	Nicholaus_Frami	1
48	Bernard.Lubowitz	1
84	Darren.Boyer	1
56	Emelia.Runte	1

This function displays the 5 most popular customers (customers with the most number of orders) at a store that the user inputs. This function will only work for managers and it will only work if the manager is the actual manager of the inputted store. This also makes sure the storeID is valid (like in previous functions).

12) Place Product Supply Request

This is a manager only function. The user is prompted to input the storeID, warehouseID, productName, and number of units requested. The user has to be verified as the store's actual manager, and the storeID, warehouseID, and productName have to be verified that they exist. Then the number of units has to be a valid non negative number as well. Then the product supply request is performed by inserting it into the table as well as updating the corresponding product information using a trigger.

13) View Supply Requests

requestnumber	managerid	warehouseid	storeid	productname	unitsrequested
7	10	2	7	Ice Cream	62
3	10	3	3	Brisk	30

This function again only works for managers. It asks for the storeID (only works if this is the store's manager) and the number of results to display. It displays the results in descending order according to the request number.

14) View Store Order Information

ordernumber	customerid	storeid	productname	unitsordered	ordertime
52	75	12	Pepsi	7	2016-09-10 13:49:00
56	62	12	Donuts	9	2016-09-10 13:52:00
150	94	12	Egg	47	2016-09-10 15:10:00
214	93	12	Brisk	19	2016-09-10 16:11:00
228	56	12	Ice Cream	33	2016-09-10 16:21:00
284	10	12	Brisk	48	2016-09-10 17:10:00
315	84	12	Orange Juice	19	2016-09-10 17:40:00
346	37	12	Donuts	39	2016-09-10 17:58:00
351	48	12	7up	47	2016-09-10 18:03:00
355	41	12	Orange Juice	41	2016-09-10 18:06:00
370	15	12	Egg	28	2016-09-10 18:19:00
377	3	12	Pudding	5	2016-09-10 18:23:00
385	65	12	Orange Juice	24	2016-09-10 18:30:00
407	13	12	Pudding	24	2016-09-10 18:44:00
452	1	12	Pepsi	43	2016-09-10 19:16:00
487	6	12	Pudding	46	2016-09-10 19:47:00
494	28	12	Brisk	12	2016-09-10 19:53:00

This function again only works for managers. It asks for the storeID (or all stores) to display order information about. It only works if the current user is the store's actual manager. The output is shown.

15) View Store Customers

userid	name	type	number_of_orders
75	Favian.Kohler	customer	7
3	Bob	customer	3
65	Eryn	customer	3
35	Camron_Howe	customer	3
4	Marshall.Johns	customer	3
64	Tyshawn	customer	3
45	Christelle.Hirthe	customer	2

This function is only for managers. It asks for the storeID (or all stores) and makes sure it's valid. Then it displays the customers in descending order according to the total number of orders at this store.

16) View Top Spenders

userid	name	type	total_spending
75	Favian.Kohler	customer	782
65	Eryn	customer	508
4	Marshall.Johns	customer	496
35	Camron_Howe	customer	432
85	Marguerite	customer	372
64	Tyshawn	customer	366
79	Broderick_Pfannerstill	customer	350

This function is only for managers. It asks for the storeID (or all stores) and makes sure it's valid. Then it displays the customers in descending order according to the total money spent at this store. This is simply computed using SUM(Product.pricePerUnit * Orders.numUnits).

Admin only functions:

17) View Users

name	password	userid	type	latitude	longitude				
Jerome.Klocko					xyz	30	customer	65.145530	7.996910
Newell					xyz	31	customer	12.162220	14.567590
Audra					xyz	32	customer	27.735670	25.599180
Marjolaine.Hamill					xyz	33	customer	81.839200	95.105370
Fermin.Runolfsson					xyz	34	customer	72.764540	23.799860
Camron_Howe					xyz	35	customer	15.894780	81.425530
Hal_Mills					xyz	36	customer	19.542550	53.126710
Janelle.Schneider					xyz	37	customer	68.191480	39.983680

This function will only work for administrator users (not managers or regular customers). It asks for a range of user IDs to display (first and last userID inclusive). Then it displays all user information for this range of users.

18) Edit Users

This function only works for admin users. It asks for a userID to edit. It then displays the current user information. Then it asks for which attributes to edit (or all attributes), and the updated values for these attributes. It makes sure the username is not already taken, and that the location numbers are valid.

19) View Products

```
Enter storeID (enter '-' for all): -
Order by?
0. No Order
1. Number of Units Ascending
2. Number of Units Descending
3. Price Per Unit Ascending
4. Price Per Unit Descending
Enter a number: 4
How many results? (enter '-' for all):

storeid productname     numberofunits   priceperunit
8       Donuts                          100     7
5       Lemonade                        100     8
6       Hot and Sour Soup               98      5
9       Hot and Sour Soup               97      5
3       Egg                             96      3
5       Ice Cream                       95      6
4       Orange Juice                    94      6
```

First this asks for the storeID to view products of (or all stores). Then it asks how to arrange the products to display, and how many results to display. Then it displays the product information (storeID, productName, numberofunits, priceperunit).

20) Edit Products

```
Enter a storeID (enter '-' for all): -
Enter a product name (enter '-' for all): 7up
Set new number of units (enter '-' for no change): 100
Set new price per unit (enter '-' for no change): -
```

This is an admin only function. It prompts the user for the storeID (or all stores), product name, and new number of units and price per unit. It then changes the product information for that store (or across all stores).

21) View Everything

This is an admin only function. It allows the user to view any relation in the database. It then will allow the user to enter a range to display (according to the primary key except for products). For products it allows the user to select a range using storeID, price per unit, or number of units.

22) Delete Store

This is an admin only function. It prompts the user to enter a storeID to delete. It verifies this is a valid number and storeID. If this is satisfied, the store is deleted. A trigger function will also delete the store's products, orders, productUpdates, and productSupplyRequests.

23) Delete Warehouse

This is an admin only function. It prompts the user to enter a WarehouseID to delete. It verifies this is a valid number and WarehouseID. If this is satisfied, the warehouse is deleted. A trigger function will also delete the warehouse's productSupplyRequests.

Extra Credit:

Some indexes were implemented to increase efficiency of some queries. Most relations have indexes on the primary key attribute, which is definitely helpful for the 22) View Everything function where it selects by ranges according to the primary key. It is also helpful in a lot of other functions where it needs to display query results.

Triggers were also implemented in this project. This includes a trigger for when the user wants to delete their account. Upon deletion of the account, all of the order information for the user will be deleted to respect the user's privacy. There is also a trigger function for when a store gets deleted. Upon deletion of a store, the store's products, orders, productUpdates, and productSupplyRequests get deleted as well. A trigger function also deletes a warehouse's productSupplyRequests when the warehouse gets deleted. Triggers are also used to update the product information when orders and product supply requests are placed.

Finally there are various ways to ensure that a "bad input" gives a user friendly response. For example storeID's and productName's are verified to exist before any query is performed. The userID's storeID's are made sure to be integers before any query is performed as well. Also a username cannot be chosen if it already exists in the database to ensure unique usernames.

Some extra functions (that were not required) were also implemented. These include the delete account, delete store, delete warehouse, view top spenders, and view everything functions.

Problems/Findings

Some initial problems were making sure the user was not a manager or admin for some of these functions. This was solved by just making an Amazon class variable to store the current user information so it could be used to verify identity in various functions.

Another initial problem was making sure the user was within 30 miles of the stores they are ordering from, and displaying only the stores within 30 miles. This was also solved with the previous method of storing the current user information in Amazon class variables. Storing the longitude and latitude allowed for easy calculation and verification.

Another problem was some of the triggers but it was because of some primary key constraints. This was fixed by just deleting the corresponding records in other relations as well.

Contributions

Shaan: Manager & Administrator functions. Trigger functions. General user interface (menu, functionality, etc.). Helped write the report

Ainaz: Customer & Manager functions. Performance tuning. General user interface (menu, functionality, etc.). Helped write the report.