CS 166 Project Report

Group Information

Kyle Lin (klin112) and Vansh Nagpal (vnagp002)

Implementation Description

Include high-level description of your implementation (1 paragraph max)

We created a Java app with a command-line-interface for interacting with a pizza store. Customers, drivers, and store managers can use our app. Customers can search for food and place orders. Drivers and managers can also update orders. Managers can also update users.

Include screenshots and/or code snippets for each query. In addition, explain how you implemented each query and what purpose it fulfills. (no more than **3-4 sentences** per query)

viewMenu

This shows the items. The user can optionally filter items by item type and by price. They can also choose whether to sort by ascending price or descending price.

```
String itemType = askUserAndTrim("Enter the type of items you want to see
(e.g., \"entree\").\nIf you want to see all items (regardless of type),
enter \"all\": ");

        System.out.println();

        System.out.print("Enter the maximum price of items you want to
see.\nIf you want to see all items (regardless of price), enter \"0\": ");

        double priceLimit = readFloatChoice();

        System.out.print("\n\nWhat order do you want to see the items
in?\n1. Ascending price order\n2. Descending price order\n\nPlease type a
number: ");

        int priceOrder = readChoiceWithin(1, 2);

        String query = "SELECT * FROM Items I WHERE ";

          if (!itemType.equals("all")) {
             // We use a LIKE to deal with potential leading spaces.
             query += "I.typeOfItem LIKE '%" + itemType + "'";
          } else {
             query += "1=1";
          }
```

```
            query += " AND ";

            if (priceLimit > 0) {
               query += "I.price <= " + priceLimit;
            } else {
               query += "1=1";
            }

            query += " ORDER BY I.price ";

            if (priceOrder == 1) {
               query += "ASC";
            } else {
               query += "DESC";
            }

            query += ";";
```

placeOrder

The customer chooses a store. They choose the store using the store ID, since the address is too long to type. Then they enter one or more items, and the desired quantity. When they have no more items to order, they type “done,” and the order is placed.

```
print("Enter the ID of the store you want to order from.\n");
        int storeID = readChoice();

        String query = "SELECT S.storeID FROM Store S WHERE S.storeID = "
+ storeID + ";";
        List<List<String>> storeResult =
esql.executeQueryAndReturnResult(query);

        if (storeResult.size() == 0) {
           print("Store not found.\n");
           return;
        }

        List<String> itemNames = new ArrayList<String>();
        List<Integer> quantities = new ArrayList<Integer>();
        double totalPrice = 0.0;

        while (true) {
```

```
            String itemName = askUserAndTrim("Enter the name of the item
you want to order.\nType \"done\" if you are finished adding items: ");

            if (itemName.equals("done")) {
               break;
            }

            query = "SELECT I.price FROM Items I WHERE I.itemName = '" +
itemName + "';";
            List<List<String>> itemResult =
esql.executeQueryAndReturnResult(query);
            if (itemResult.size() == 0) {
               print("Item not found.\n");
               continue;
            }

            double price = parseDoubleOr(itemResult.get(0).get(0), 0.0);

            print("\nEnter desired quantity: ");

            int quantity = readChoice();

            itemNames.add(itemName);

            quantities.add(quantity);

            totalPrice += quantity * price;

            print("\n");
         }

         // Get max existing orderID.
         query = "SELECT MAX(F.orderID) FROM FoodOrder F;";
         List<List<String>> maxOrderIDResult =
esql.executeQueryAndReturnResult(query);
         int randomFallbackOrderID = (int) (Math.random() * 1000000000);
```

```
        int orderID = parseIntOr(maxOrderIDResult.get(0).get(0),
randomFallbackOrderID) + 1;

        String login = esql.getLogin();

        query = "INSERT INTO FoodOrder (orderID, login, storeID,
totalPrice, orderTimestamp, orderStatus) VALUES (" + orderID + ", '" +
login + "', " + storeID + ", " + totalPrice + ", NOW(), 'incomplete');";
         esql.executeUpdate(query);

         for (int i = 0; i < itemNames.size(); ++i)
         {
           String itemName = itemNames.get(i);
           int quantity = quantities.get(i);

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

           esql.executeUpdate(query);
         }

         print("\n\nOrder successfully placed!\n\n");
```

viewAllOrders

This query's behavior depends on the user's role. If they are a customer, this lists all their orders. If they are a driver or manager, they get to choose any user (including themselves), and then this lists that user's orders.

```
String customerLogin = askUserAndTrim("Enter the login name of the
customer whose orders you want to see. Type \"all\" to see the orders of
all customers.\nLogin name: ");

        String query = "SELECT F.orderID, F.login, F.storeID,
F.totalPrice, F.orderTimestamp, F.orderStatus FROM FoodOrder F";

        if (!customerLogin.equals("all")) {
```

```
            query += " WHERE F.login = '" + customerLogin + "'";
        }

        query += ";";

        List<List<String>> result =
esql.executeQueryAndReturnResult(query);

        print("\n\nORDER IDS:\n\n");

        for (int i = 0; i < result.size(); ++i) {
            List<String> row = result.get(i);

            String orderID = row.get(0);
            String login = row.get(1);
            String storeID = row.get(2);
            String totalPrice = row.get(3);
            String orderTimestamp = row.get(4);
            String orderStatus = row.get(5);

            print(orderID + "\n");
        }
```

The above code is for the user-is-driver-or-manager case. The user-is-customer case is similar, except customerLogin is hard-coded to the user's login.

viewRecentOrders

This is like "viewAllOrders" except it only lists the 5 most recent orders. The code is almost identical, so I won't include it again (it would be redundant). The key difference is we add "`... ORDER BY F.orderTimestamp DESC LIMIT 5;`" to the end of the query.

viewOrderInfo

This displays the info for a given order ID. A customer can only view their own order. Drivers and managers can view anyone's order.

```
String login = esql.getLogin();
        // Get role
        String query = "SELECT U.role FROM Users U WHERE U.login = '" +
login + "';";
        List<List<String>> roleResult =
esql.executeQueryAndReturnResult(query);
        String role = roleResult.get(0).get(0).trim();
        boolean canSeeOtherUsersOrders = role.equals("manager") ||
role.equals("driver");

        print("Enter the order ID you want to see.");
        int targetOrderID = readChoice();

        query = "SELECT F.orderID, F.login, F.storeID, F.totalPrice,
F.orderTimestamp, F.orderStatus FROM FoodOrder F WHERE F.orderID = '" +
targetOrderID + "';";

        List<List<String>> result =
esql.executeQueryAndReturnResult(query);

        if (result.size() == 0) {
           print("Order not found.\n");
           return;
        }

        // This should iterate at most once.
        for (int i = 0; i < result.size(); ++i) {
           List<String> row = result.get(i);

           String orderID = row.get(0);
           String orderLogin = row.get(1);
           String storeID = row.get(2);
           String totalPrice = row.get(3);
           String orderTimestamp = row.get(4);
           String orderStatus = row.get(5);
```

```
            if (!(canSeeOtherUsersOrders || orderLogin.equals(login))) {
               print("This order does not belong to you.\n");
               return;
            }

            print("\nORDER INFO\n");
            print("Order ID: " + orderID + "\n");
            print("Customer login: " + orderLogin + "\n");
            print("Store ID: " + storeID + "\n");
            print("Total price: " + totalPrice + "\n");
            print("Order timestamp: " + orderTimestamp + "\n");
            print("Order status: " + orderStatus + "\n\n");
        }
```

viewStores

This lists the stores. For the user's convenience, the stores are ordered by their store ID.

```
String query = "SELECT S.storeID, S.address, S.city, S.state, S.isOpen,
S.reviewScore FROM Store S ORDER BY S.storeID ASC;";
        List<List<String>> result =
esql.executeQueryAndReturnResult(query);

        print("\n\nSTORES:\n\n");

        for (int i = 0; i < result.size(); ++i) {
            List<String> row = result.get(i);

            String storeID = row.get(0);
            String address = row.get(1);
            String city = row.get(2);
            String state = row.get(3);
            String isOpen = row.get(4);
            String reviewScore = row.get(5);
```

```
            print("Store ID: " + storeID + "\n");
            print("    Address: " + address + "\n");
            print("    City: " + city + "\n");
            print("    State: " + state + "\n");
            print("    Is open: " + isOpen + "\n");
            print("    Review score: " + reviewScore + "\n\n");
        }
```

viewProfile

This query selects information from the “Users” relation and prints out the user’s favorite item and phone number.

```
public static void viewProfile(PizzaStore esql) {
       try {
          String currentUser = esql.getLogin();
          String query = "SELECT U.favoriteItems, U.phoneNum FROM Users U WHERE
U.login = '" + currentUser + "';";

          List<List<String>> result = esql.executeQueryAndReturnResult(query);

          System.out.println("\nProfile details:");
          for (List<String> row : result) {
             System.out.println("\tFavorite Items: " + row.get(0));
             System.out.println("\tPhone Number: " + row.get(1));
             System.out.print("\n");
          }
       }
       catch (Exception e) {
          System.err.println(e.getMessage());
       }
    }
```

updateProfile

This query allows users to update their profile attributes: password, phone number, and favorite items. The user is asked for their current password before updating the password or phone number for security purposes.

```
public static void updateProfile(PizzaStore esql) {
```

```
      try {
         System.out.println("\nOPTIONS: ");
         System.out.println("1. Update password");
         System.out.println("2. Update phone number");
         System.out.println("3. Update favorite item");
         System.out.println("4. Exit");

         switch (readChoice()) {
            case 1:
               updatePassword(esql);
               break;
            case 2:
               updatePhoneNumber(esql);
               break;
            case 3:
               updateFavoriteItems(esql);
               break;
            case 4:
               break;
         }
      }
      catch (Exception e) {
         System.err.println(e.getMessage());
      }
   }

public static void updatePassword(PizzaStore esql) {
      try {
         System.out.print("Enter your current password: ");
         String password = in.readLine();
         while (password == null || password.trim().isEmpty()) {
            System.out.println("\nPassword cannot be empty. Please try again.");
            password = in.readLine();
         }

         String login = esql.getLogin();
         String query = "SELECT U.login FROM Users U WHERE U.login = '" + login
+ "' AND U.password = '" + password + "';";
         List<List<String>> result = esql.executeQueryAndReturnResult(query);
```

```
         if (result.size() < 1) {
            System.out.println("\nIncorrect password. Please try again.\n");
            return;
         }

         System.out.print("Enter your new password: ");
         password = in.readLine();
         while (password == null || password.trim().isEmpty()) {
            System.out.println("\nPassword cannot be empty. Please try again.");
            password = in.readLine();
         }

         query = "UPDATE Users SET password = '" + password + "' WHERE login =
'" + login + "';";

         esql.executeUpdate(query);

         System.out.println("Password successfully updated!");
      }
      catch (Exception e) {
         System.err.println(e.getMessage());
      }
   }

public static void updatePhoneNumber(PizzaStore esql) {
      try {
         System.out.print("Enter your password: ");
         String password = in.readLine();
         while (password == null || password.trim().isEmpty()) {
            System.out.println("\nPassword cannot be empty. Please try again.");
            password = in.readLine();
         }

         String login = esql.getLogin();
         String query = "SELECT U.login FROM Users U WHERE U.login = '" + login
+ "' AND U.password = '" + password + "';";
         List<List<String>> result = esql.executeQueryAndReturnResult(query);
```

```
         if (result.size() < 1) {
            System.out.println("\nIncorrect password. Please try again.\n");
            return;
         }

         System.out.print("Enter your new phone number: ");
         String phoneNum = in.readLine();
         while (phoneNum == null || phoneNum.trim().isEmpty() ||
!phoneNum.matches("\\d{10}")) {
            System.out.println("\nPhone number cannot be empty and must be 10
consecutive digits (Ex: 1234567890). Please try again.");
            phoneNum = in.readLine();
         }

         query = "UPDATE Users SET phoneNum = '" + phoneNum + "' WHERE login =
'" + login + "';";

         esql.executeUpdate(query);

         System.out.println("Phone number successfully updated!");
      }
      catch (Exception e) {
         System.err.println(e.getMessage());
      }
   }
public static void updateFavoriteItems(PizzaStore esql) {
      try {
         System.out.print("Enter your new favorite item: ");
         String item = in.readLine();
         while (item == null || item.trim().isEmpty()) {
            System.out.println("\nLogin cannot be empty. Please try again.");
            item = in.readLine();
         }

         String query = "SELECT I.itemName FROM Items I WHERE I.itemName = '" +
item + "';";
         int rowCount = esql.executeQuery(query);

         if (rowCount < 1) {
```

```
            System.out.println("\nItem not found. Please enter a valid item from
the menu.\n");
            return;
         }

         query = "UPDATE Users SET favoriteItems = '" + item + "' WHERE login =
'" + esql.getLogin() + "';";
         esql.executeUpdate(query);

         System.out.println("Favorite item updated!");
      }
      catch (Exception e) {
         System.err.println(e.getMessage());
      }
   }
```

updateOrderStatus

The user's role is checked using the private login variable upon login. If the user is a manager or driver, they are able to update the order status to either complete or incomplete. If the user is a customer, they are unable to update the order status.

```
public static void updateOrderStatus(PizzaStore esql) {
     try {
        String login = esql.getLogin();

        String query = "SELECT U.role FROM Users U WHERE U.login = '" + login
"';";

        List<List<String>> result = esql.executeQueryAndReturnResult(query);

        String role = result.get(0).get(0);

        if (!role.trim().equals("manager") || !role.trim().equals("driver")) {
           System.out.println("Insufficient privileges. Please ask a manager or
driver for assistance with your order status.");
           return;
        }
```

```
         System.out.print("Enter order ID to update: ");
         Integer orderID = Integer.parseInt(in.readLine());

         System.out.println("Is the order complete?");
         System.out.println("Options:");
         System.out.println("1. Yes");
         System.out.println("2. No");

         String query = "";

         switch (readChoice()) {
            case 1:
               query = "UPDATE FoodOrder SET orderStatus = 'completed' WHERE
orderID = " + orderID + ";";
               break;
            case 2:
               query = "UPDATE FoodOrder SET orderStatus = 'incomplete' WHERE
orderID = " + orderID + ";";
               break;
            default:
               System.out.println("Invalid choice. Order status not updated.");
               return;
         }

         esql.executeUpdate(query);

         System.out.println("Order '" + orderID + "' was updated!");
      }
      catch (Exception e) {
         System.err.println(e.getMessage());
      }
   }
```

updateMenu

The user's role is checked before making changes to the menu items. The user is asked for the item's information to ensure that existing items can not be readded (avoiding duplication). Additionally, input checking is done to ensure that non-null fields are entered correctly.

```java
public static void updateMenu(PizzaStore esql) {
      try {
         String login = esql.getLogin();

         String query = "SELECT U.role FROM Users U WHERE U.login = '" + login +
"';";
         List<List<String>> result = esql.executeQueryAndReturnResult(query);

         String role = result.get(0).get(0);

         if (!role.trim().equals("manager")) {
            System.out.println("\nInsufficient privileges.");
            return;
         }

         System.out.println("\nOPTIONS: ");
         System.out.println("1. Update item");
         System.out.println("2. Add item");
         System.out.println("3. Exit");

         switch (readChoice()) {
            case 1:
               updateMenuItem(esql);
               break;
            case 2:
               addMenuItem(esql);
               break;
            case 3:
               break;
         }
      }
      catch (Exception e) {
         System.err.println(e.getMessage());
      }
   }

public static void updateMenuItem(PizzaStore esql) {
      try {
         System.out.print("Enter item to update: ");
```

```
            String item = in.readLine();
            while (item == null || item.trim().isEmpty()) {
               System.out.println("\nItem cannot be empty. Please try again.");
               item = in.readLine();
            }

            String query = "SELECT I.itemName FROM Items I WHERE I.itemName = '" +
item + "';";
            int rowCount = esql.executeQuery(query);

            if (rowCount < 1) {
               System.out.println("Item not found. Please try again.");
               return;
            }

            System.out.print("Enter field to update: ");
            String field = in.readLine();
            while (field == null || field.trim().isEmpty()) {
               System.out.println("\nField cannot be empty. Please try again.");
               field = in.readLine();
            }

            System.out.println("Enter updated information: ");
            String info = in.readLine();
            while (info == null || info.trim().isEmpty()) {
               System.out.println("\nInformation cannot be empty. Please try
again.");
               info = in.readLine();
            }

            query = "UPDATE Item SET " + field + " = " + info + " WHERE itemName =
'" + item + "';";

            esql.executeUpdate(query);

            System.out.println("Menu item updated!");
         }
         catch (Exception e) {
            System.err.println(e.getMessage());
```

```
      }
   }

   public static void addMenuItem(PizzaStore esql) {
      try {
         System.out.print("Enter item to add: ");
         String item = in.readLine();
         while (item == null || item.trim().isEmpty()) {
            System.out.println("\nItem cannot be empty. Please try again.");
            item = in.readLine();
         }

         String query = "SELECT I.itemName FROM Items I WHERE I.itemName = '" +
item + "';";
         int rowCount = esql.executeQuery(query);

         if (rowCount >= 1) {
            System.out.println("Item already exists. Please try again.");
            return;
         }

         System.out.print("Enter item ingredients: ");
         String ingredients = in.readLine();
         while (ingredients == null || ingredients.trim().isEmpty()) {
            System.out.println("\nIngredients cannot be empty. Please try
again.");
            ingredients = in.readLine();
         }

         System.out.print("Enter type of item: ");
         String type = in.readLine();
         while (type == null || type.trim().isEmpty()) {
            System.out.println("\nType of item cannot be empty. Please try
again.");
            type = in.readLine();
         }

         System.out.println("Enter item price: ");
         Double price = Double.parseDouble(in.readLine());
```

```
         while (price < 0) {
            System.out.println("\nPrice must be a positive value. Please try
again.");
            price = Double.parseDouble(in.readLine());
         }

         System.out.print("Enter description: ");
         String description = in.readLine();

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

         esql.executeUpdate(query);

         System.out.println("Menu item added!");
      }
      catch (Exception e) {
         System.err.println(e.getMessage());
      }
   }
```

updateUser

Only a manager is able to update a user's login and role. A manager can not update their own login or role and must ask another manager to do so. Checks are performed to ensure that the user being updated exists.

```
public static void updateUser(PizzaStore esql) {
      try {
         String login = esql.getLogin();

         String query = "SELECT U.role FROM Users U WHERE U.login = '" + login +
"';";
         List<List<String>> result = esql.executeQueryAndReturnResult(query);

         String role = result.get(0).get(0);

         if (!role.trim().equals("manager")) {
            System.out.println("\nInsufficient privileges. Please contact a
manager to update login/role.");
```

```
            return;
         }

         System.out.println("\nOPTIONS: ");
         System.out.println("1. Update user's login");
         System.out.println("2. Update user's role");
         System.out.println("3. Exit");

         switch (readChoice()) {
            case 1:
               updateUserLogin(esql);
               break;
            case 2:
               updateUserRole(esql);
               break;
            case 3:
               break;
         }
      }
      catch (Exception e) {
         System.err.println(e.getMessage());
      }
   }

public static void updateUserLogin(PizzaStore esql) {
      try {
         System.out.print("Enter user's login: ");
         String login = in.readLine();
         while (login == null || login.trim().isEmpty()) {
            System.out.println("\nLogin cannot be empty. Please try again.");
            login = in.readLine();
         }

         if (login.trim().equals(esql.getLogin())) {
            System.out.println("\nYou are unable to update your own login.
Please ask another manager.");
            return;
         }
```

```
         String query = "SELECT U.login FROM Users U WHERE U.login = '" + login
+ "';";
         int rowCount = esql.executeQuery(query);

         if (rowCount < 1) {
            System.out.println("\nUser was not found. Please enter a valid
login.\n");
            return;
         }

         System.out.print("Enter user's new login");
         String newLogin = in.readLine();
         while (newLogin == null || newLogin.trim().isEmpty()) {
            System.out.println("\nNew login cannot be empty. Please try
again.");
            newLogin = in.readLine();
         }

         query = "UPDATE Users SET login = '" + newLogin + "' WHERE login = '" +
login + "';";
         esql.executeUpdate(query);

         System.out.println("User login updated!");
      }
      catch (Exception e) {
         System.err.println(e.getMessage());
      }
   }
public static void updateUserRole(PizzaStore esql) {
      try {
         System.out.print("Enter user's login: ");
         String login = in.readLine();
         while (login == null || login.trim().isEmpty()) {
            System.out.println("\nLogin cannot be empty. Please try again.");
            login = in.readLine();
         }

         if (login.trim().equals(esql.getLogin())) {
```

```
            System.out.println("\nYou are unable to update your own role. Please
ask another manager.");
            return;
         }

         String query = "SELECT U.login FROM Users U WHERE U.login = '" + login
+ "';";
         int rowCount = esql.executeQuery(query);

         if (rowCount < 1) {
            System.out.println("\nLogin was not found. Please enter a valid
login.\n");
            return;
         }

         System.out.print("Enter user's new role: ");
         String role = in.readLine();
         while (login == null || login.trim().isEmpty()) {
            System.out.println("\nRole cannot be empty. Please try again.");
            login = in.readLine();
         }

         query = "UPDATE Users SET role = '" + role + "' WHERE login = '" +
login + "';";
         esql.executeUpdate(query);

         System.out.println("User role updated!");
      }
      catch (Exception e) {
         System.err.println(e.getMessage());
      }
   }
```

If you did any extra credit, provide screenshots and/or code snippets. Explain how you implemented the extra credit. (triggers/stored procedures, performance tuning, etc)

1. We added input checking to ensure validity of the data and security measures by asking for the user's password when making changes to their profile. Kyle also improved the formatting for the output of the queries, making the interface more user friendly.

```
Please make your choice: 3
Enter the type of items you want to see (e.g., "entree").
If you want to see all items (regardless of type), enter "all": all

Enter the maximum price of items you want to see.
If you want to see all items (regardless of price), enter "0":
Please make your choice: 0


What order do you want to see the items in?
1. Ascending price order
2. Descending price order

Please type a number: Please make your choice: 1
Item: Sprite
    Type of item:  drinks
    Price: 1.99
    Description:  delicious!
    Ingredients:  High Fructose Corn Syrup, Lemon, Lime

Item: Water Bottle
    Type of item:  drinks
    Price: 1.99
    Description:  a classic!
    Ingredients:  Hydrogen, Oxygen
```

```java
   public static void updatePassword(PizzaStore esql) {
      try {
         System.out.print("Enter your current password: ");
         String password = in.readLine();
         while (password == null || password.trim().isEmpty()) {
            System.out.println("\nPassword cannot be empty. Please try again.");
            password = in.readLine();
         }

         String login = esql.getLogin();
         String query = "SELECT U.login FROM Users U WHERE U.login = '" + login
+ "' AND U.password = '" + password + "';";
```

```
            List<List<String>> result = esql.executeQueryAndReturnResult(query);

            if (result.size() < 1) {
               System.out.println("\nIncorrect password. Please try again.\n");
               return;
            }
```

2. We implemented a trigger to automatically increment the orderID as a new order is created in the FoodOrder relation.

```
DROP SEQUENCE IF EXISTS orderID_seq;
DROP TRIGGER IF EXISTS orderID_trigger ON FoodOrder;

CREATE SEQUENCE orderID_seq START WITH 10000;

CREATE OR REPLACE FUNCTION increment_orderID()
RETURNS TRIGGER AS
$BODY$
BEGIN
    NEW.orderID := nextval('orderID_seq');
    RETURN NEW;
END;
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER orderID_trigger
BEFORE INSERT ON FoodOrder
FOR EACH ROW
EXECUTE PROCEDURE increment_orderID();
```

3. We implemented performance tuning by creating indexes on our data. For example, we created a login-and-password index on the user table because there are many rows and users log in frequently (presumably once per session). We created an item-type index because users filter items by their type (e.g., “entree”). We did the same for item price, except we used a B-tree, since this is a range query (e.g., “price < 8.99”).

```
DROP INDEX IF EXISTS USER_LOGIN_PASSWORD_INDEX;
DROP INDEX IF EXISTS ITEM_TYPE_INDEX;
```

```
DROP INDEX IF EXISTS ITEM_PRICE_INDEX;

-- Speeds up the login process.
CREATE INDEX USER_LOGIN_PASSWORD_INDEX
ON Users
(login, password);

-- Speeds up evaluation when the customer filter by item type.
CREATE INDEX ITEM_TYPE_INDEX
ON Items
(typeOfItem);

-- Speeds up evaluation when the customer filters by price range.
CREATE INDEX ITEM_PRICE_INDEX
ON Items
-- This is a range query, so we use a B-tree.
USING BTREE
(price);
```

Problems/Findings

Include problems/findings you encountered while working on the project (**1-2 paragraphs max**)

Some of the data was in an unexpected format. For example, the typeOfItem sometimes had a value of " entree" (with a leading space) instead of the expected "entree." To fix this, we used SQL's "LIKE" syntax to perform "loose matching."

Contributions

Include descriptions of what each member worked on (**1 paragraph max**)

Vansh worked mainly on handling updates to the databases and creating a trigger to automate data insertion. Kyle worked mainly on adding indexes to optimize queries and wirting queries to fetch, filter, and display data from the database using user input.

Contributions	
Vansh	**Kyle**

project report	project report
triggers	indexes
viewProfile	viewMenu
updateProfile	placeOrder
updateOrderStatus	viewAllOrders
updateMenu	viewRecentOrders
updateUser	viewOrderInfo
	viewStores