DB Final Report - Group Seventeen

Team Members:

- Jeffrey Pan
- Erik Earl
- Eduardo Bautista
- Vishnu Polkampally
- Elwin He

Name of database backup file (should include .bak extension):
restaurantDB.bak
Name of Virtual Machine: csdswinlab017
Password for the username: csds341group17
Instructions for running code on VM: See the user manual (included below in this file) for the description on how to run the application.
Name of zip file that contains Java code: DeliverablesGroup17.zip
Date of TA Meeting: Dec 7th, 1:30 PM

Project Description:

We will analyze the data in a burger chain. This will include looking into their finances, customer information, menu items, suppliers and more. The objective of this project is to obtain a deeper understanding of the restaurant's operations. This database schema is catering towards a restaurant franchise. Restaurants could use this information to keep track of suppliers, customers, and equipment. Furthermore, this information could be used to check the flow of the supply chain. Restaurant franchises could use this database to gain an edge over other restaurants, understand their customers better and their profitability.

Restaurant Database E-R Diagram:

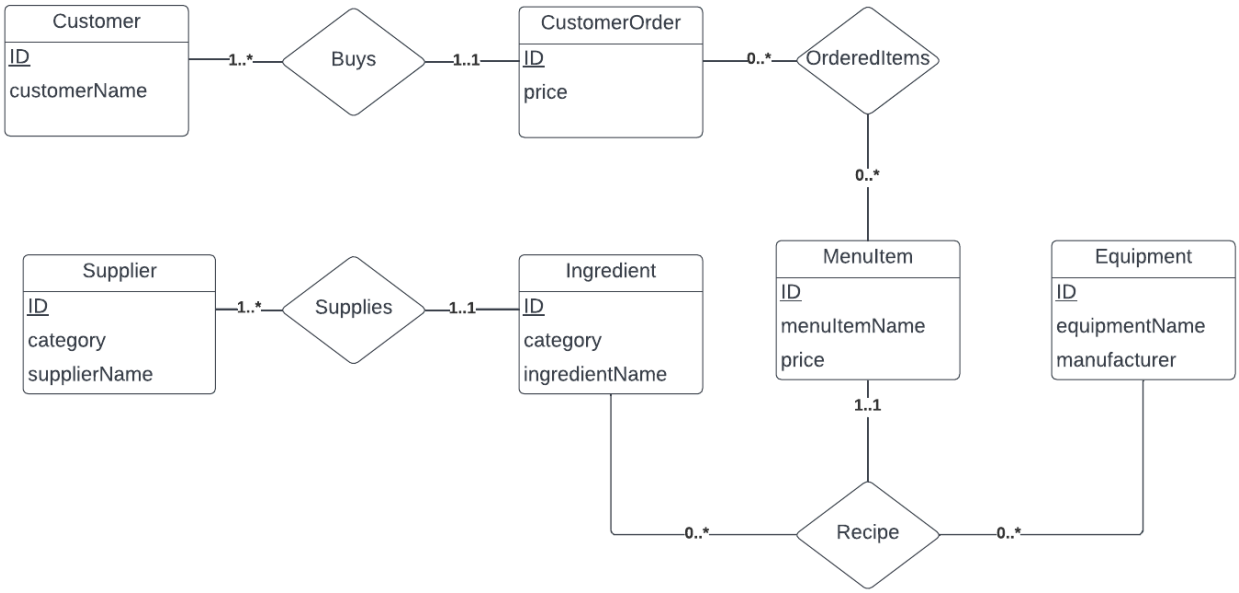

Table Descriptions:

Customer: Stores ID and name of the customer to be later on used by the database to match them with their order.

MenuItem: Stores menu item ID, the name of the menu item, and price. This table is employed to create recipes. Additionally, the customer is able to make orders based on the existing menu items.

Ingredient: Stores the ID of the ingredient, category of the ingredient, and the name of the ingredient. Ingredients are used to create new menu items and keep track of inventory.

Equipment: Equipment includes all the necessary tools and machines needed to make the recipes. Every time a menu item is created or updated, equipment is created or changed depending on the requirements of the recipe.

Recipe: Stores menuItemID, with its respective equipmentID and ingredientID. They reference MenuItem (ID), Equipment (ID), and Ingredient (ID), respectively.

Supplier: Stores the supplier's ID, the category of the ingredient it supplies, and the name of the supplier. This relation is used to keep track of the currently active suppliers.

CustomerOrder: Stores ID of the order and price. This information is used to select all orders over and under or equal input price.

Supplies: Stores the ingredientID with its respective supplierID. The former references Ingredient (ID) and the latter references Supplier (ID)

Buys: Stores customerID with its respective orderID. The former references Customer (ID) and the latter references CustomerOrder (ID).

OrderedItems: Stores orderID with its respective menuItemID. The former references CustomerOrder (ID) and the latter references MenuItem (ID).

The order in which the tables need to be created are Customer, CustomerOrder, Ingredient, Supplier, Equipment, and MenuItem first, as they do not have any foreign key constraints and do not reference other tables. Following these six tables, Recipe, OrderedItems, Buys, and Supplies can be created after which they reference the first six tables created and have foreign key constraints.

SQL Queries/Use Cases:

Use case 1: MakeOrder- Erik

- Function: Allows the user to input a new restaurant order
- Input: The existing customer name that will be ordering
- Input: The menu item that the customer is ordering

```
insert into CustomerOrder (price) values (
    (SELECT price from MenuItem where @item_name = menuItemName));
insert into Buys (customerID, orderID) values (
    (SELECT ID from Customer where @customer_name = customerName),
    (SELECT max(ID) from CustomerOrder));
insert into OrderedItem (orderID, menuItemID) values (
    (SELECT max(ID) from CustomerOrder),
    (SELECT ID from MenuItem where @item_name = menuItemName));
```

Use case 2: MakeNewMenuItem- Erik

- Function: Allows the user to create a new menu item for the restaurant menu
- Input: A menu item name for the new menu item
- Input: A float for the price of the new menu item
- Input: A ingredient that the new menu item needs
- Input: A equipment that the new menu item needs

- Ingredients and equipments are inserted as pairs where the input ingredient and equipment correspond to one another (ie potato corresponds with fryer)
- Because only one ingredient and one equipment can be added with the use case, in order to have multiple ingredients/equipments corresponding with a single menu item, MakeNewMenuItem must be called multiple times, each time for each ingredient/equipment

```
insert into MenuItem (menuItemName, price) values (@menu_item_name, @price);
insert into Recipe (ingredientID, menuItemID, equipmentID) values(
   (SELECT ID from Ingredient where ingredientID = @ingredent),
   (SELECT max(ID) from MenuItem),
   (SELECT ID from Equipment where equipmentID = @equipment));
```

Use case 3: ChangeMenuItem- Jeff

- Function: Allows the user to create a update a menu item for the restaurant menu
- Input: The menu item name of the menu item to be updated
- Input: The ingredient of the menu item to be updated
- Input: The ingredient that replaces the old ingredient
- Input: The equipment of the menu item to be updated
- Input: The equipment that replaces the old equipment
- Because only one ingredient and one equipment can be updated with the use case, in order to update multiple ingredients/equipments corresponding with a single menu item, ChangeMenuItem must be called multiple times, each time for each ingredient/equipment

```
   update Recipe set ingredientID =
       (SELECT ID from Ingredient where ingredientName =@ingredient_replacing)
   where (SELECT ID
           from MenuItem where @menu_item_name = menuItemName) =
           Recipe.menuItemID and Recipe.ingredientID = (SELECT ID
           from Ingredient where ingredientName =@ingredient_replaced);
   update Recipe set equipmentID =
       (SELECT ID from Equipment where equipmentName =@equipment_replacing)
   where (SELECT ID
           from MenuItem where @menu_item_name = menuItemName) =
           Recipe.menuItemID and Recipe.equipmentID = (SELECT ID
           from Equipment where equipmentName =@equipment_replaced);
```

Use case 4: UpdateOrder- Jeff

- Function: Allows the user to update the order of a customer
- Input: The order ID of the customer order to be updated

- Input: A menu item to replace the old order menu item

```
DELETE from OrderedItem where @order_id = ID
INSERT into OrderedItem (orderID, menuItemID)
values(@order_id, (SELECT ID from MenuItem where @menu_item = menuItemName));

UPDATE CustomerOrder Set price = (SELECT price from MenuItem
where @menu_item = MenuItemName) where CustomerOrder.ID = @order_id;
```

Use case 5: DeleteMenuItem - Elwin

- Function: Allows the user to delete a menu item from the restaurant menu
- Input: The menu item name of the menu item that will be deleted

```
DELETE from MenuItem
where @menu_item = menuItemName
```

Use case 6: DeleteIngredient - Elwin

- Function: Allows the user to delete an ingredient that a supplier supplies
- Input: The ingredient name of the supplier ingredient that will be deleted

```
DELETE from Supplies where (SELECT ID from Ingredient
where @ingredient = ingredientName) = ingredientID
DELETE from Ingredient where @ingredient = ingredientName
```

Use case 7: FindMenuItemIngredients - Vishnu

- Function: Allows the user to select all the ingredients that a menu item uses
- Input: The menu item name that the user wants to the ingredients of
- Outpt: All the ingredients that are related to the input menu item

```
Select ingredientName from Ingredient join
    (SELECT * from MenuItem inner join Recipe
    on MenuItem.ID = Recipe.menuItemID
    where @menu_item = menuItemName)
as Item on Ingredient.ID = Item.ingredientId
```

Use case 8: FindOrdersBasedOnCost - Vishnu

- Function: Allows the user to find all orders that are equal to an input cost
- Input: The customer order cost that the user wants to find all orders of
- Output: All customer orders that are of the input cost

```
select @out = ID from CustomerOrder
where price = @price;
```

Use case 9: FindOrderHistory - Edwardo

- Function: Allows the user to find all orders that a customer has placed
- Input: A customer name
- Output: All orders that the customer has placed (order id and the menu item as pairs)

```
select CustomerOrder.ID, menuItemName from Customer
    join Buys on Customer.ID = Buys.customerID
    join CustomerOrder on CustomerOrder.ID = Buys.orderID
    join OrderedItem on CustomerOrder.ID = OrderedItem.orderID
    join MenuItem on MenuItem.ID = menuItemID
    where customerName = @customer_name
```

Use case 10: FindSupplierIngredients - Edwardo

- Function: Allows the user to find all ingredients that a supplier supplies
- Input: A supplier name
- Output: All ingredients that a supplier supplies

```
Select distinct ingredientName from Ingredient
    Join Supplies on Ingredient.ID = Supplies.supplierID
    Join Supplier on Supplies.supplierID = Supplier.ID
Where supplierName = @supplier_name
```

Lessons Learned:

1. In the beginning we had included more relations in our database. However, we had to exclude some of these relations since we realized they were redundant. This helped us free some time to focus on the user cases.
2. In order to create the GUI, we had to review some Java programming concepts we learned in freshman year of college. Some of these concepts include the Scanner class, which helps us take inputs from the user. At the same time, we had to familiarize ourselves with Java Swing, a Java GUI toolkit. We chose Java Swing over JavaFX since the former is more established and has a larger library, making it easier for us to implement our ideas within the determined deadline.
3. Initially, our project was a bit too large for our team size. We had to downsize the number of relations and data points in order to make the project more feasible for us.
4. In the early stages of the project, we experienced some difficulties trying to connect JDBC with Microsoft SQL Server. This delayed our team a lot, so we had to attend office

hours and spend extra time outside of class. This taught us to ask questions earlier and to be proactive.

Restaurant DB- User Manual

In order to open the database, you'll need to open the main.java file and a GUI like the one in figure 1 will appear.

For our database, we created a simple GUI for employees and managers to use.

The white row and blue row are buttons, while each cell in the red row represents the heading of each column.

The user can click on any button and the GUI will display a pop-up prompting the user to input the necessary information like in figure 2. The program will validate the user's input depending on what command was called. For instance, if the user desires to insert an order, the GUI will ask for the customer's name. If the user inputs an empty value or the customer's name doesn't exist in the database, a pop-up will appear indicating that the input was invalid.

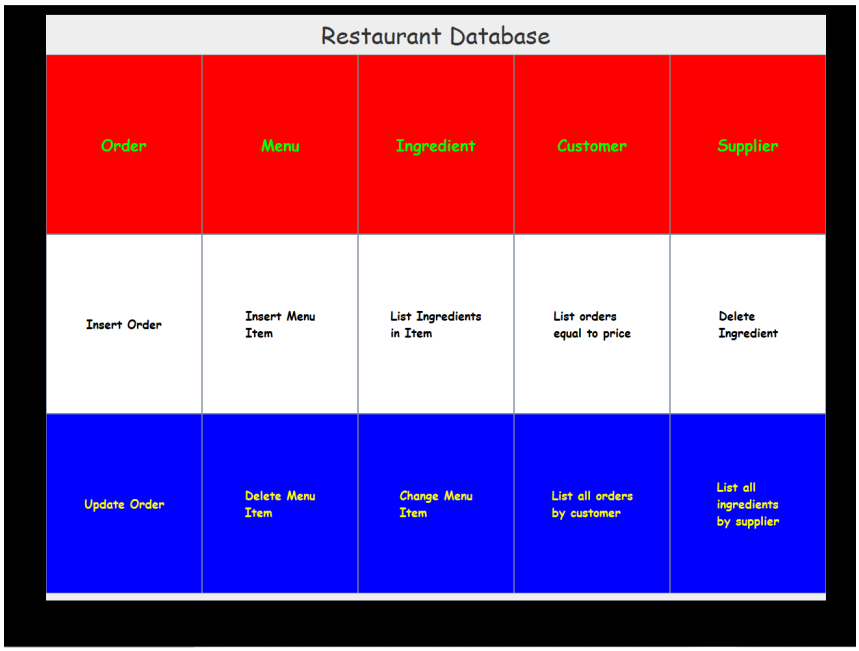

Figure 1

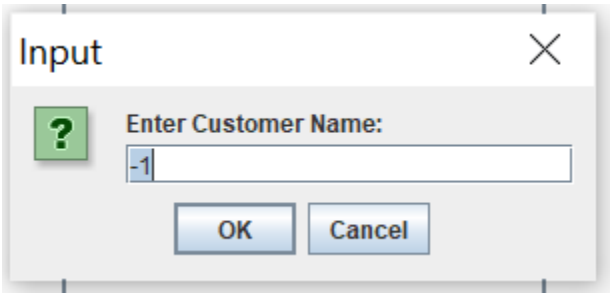

Figure 2

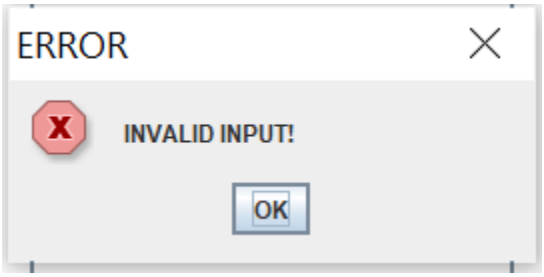

Figure 3

Error Cases-
Below are the invalid actions the user can input resulting in an 'Invalid Input' error:

1. Insert Menu Item- If the number of ingredients of the new menu item is greater or lower than the number of tools.
2. Insert Order- If the input for either the customer name or menu item are empty.
3. List Ingredients in Item- If the user doesn't provide a menu Item.
4. List Order Equal To Price- If user inputs a value that is not a float
5. Delete Ingredient- If the user inputs an empty value for ingredient
6. Update Order- If the user inputs a menu item
7. Delete Menu Item- If the user inputs an empty value for the menu item, ingredient to be replaced, ingredient replacing, equipment to be replaced and the equipment replacing
8. Change Menu Item- If the user doesn't provide a menu Item, ingredient replaced, new ingredient, equipment replaced, or new equipment.
9. Find Order History- If the user doesn't provide a customer name.
10. Find Supplier Ingredients - If the user inputs an empty field for the supplier name.

Operating the DB on the Virtual Machine-
To run the GUI on csdswinlab017, open the 'RestaurantDB' folder located on the machine's desktop in VS code. In the project's src folder there is a 'Main.java' file to run which will open the GUI. Additionally the database can be accessed in Microsoft SQL Management Studio, by connecting to 'CSDSWINLAB017\SQLEXPRESS' by windows authentication and accessing the 'restaurantDB' database in the Databases folder.