CS 470: Final Group Milestone Report

Bryan Davis, Cory Free, Benaiah Kilen

May 3, 2020

Requirements

The purpose of our group's database is to hold information pertaining to multiple restaurants, their employees, the positions each restaurant offers, and the food items offered by each restaurant.

Each restaurant should have a numeric ID (their primary means of identification) a name, a location (city or town), and a derived count of how many employees that restaurant has. A restaurant requires at least 10 employees to operate, but this is not strictly enforced; however, at least one employee must be employed at a restaurant at any given time. No restaurant can hire more than 30 employees at a time.

For each employee, we will store their numeric ID (their primary means of identification), their full name, their home address, and the ID of their position. An employee can work at any number of restaurants, but they must at least work at one.

Each position will contain a numeric ID and a title (both together are the primary means of identification) along with the pay that comes with the position.

We will store a list of all possible menu items, containing each item's name (the primary means of identification), its price, and what type of food it is. A food's type can only be Food or Drink. As for the individual restaurant menus, we will store the restaurant's numeric ID and the item name (both together are the primary means of identification), the count, and the item's status. If the count is 0, the status must be no_stock. If the count is under 30, the status must be lo_stock. Else, the status will be in_stock. The count must be a positive integer or zero.

Technical Summary

Our group opted to use XAMPP to host the database, as well as PHP and MySQL to create and modify the schema. We used Visual Studio and C++/CLI to create the windows form our model used. The model requires a connection to the database in order to access it, and the connection string was stored separate from our code within the application's app.config. As we did not expect to see these characters in our entries, we chose to make the characters "**();**" illegal input within the application, meaning users could not manually search for entries if their input contained these characters; this was to protect from SQL injection. We also chose to use procedure calls for all of the button functionality within our model for further security.

Future Scope

As of right now, our model only supports the viewing of information found within our database. The model could be further modified to add the ability to modify and create entries within our database fairly easily; the most difficult part would just be finding room to place the buttons in a neat and organized manner. We would just need to create procedure calls for each table we'd support the modification/creation of entries for that used the information within each attribute's respective text boxes for input. If we were to do this, we would also need to implement a secure login & role system that would allow and disallow certain roles to modify, or even view, the information within the database.

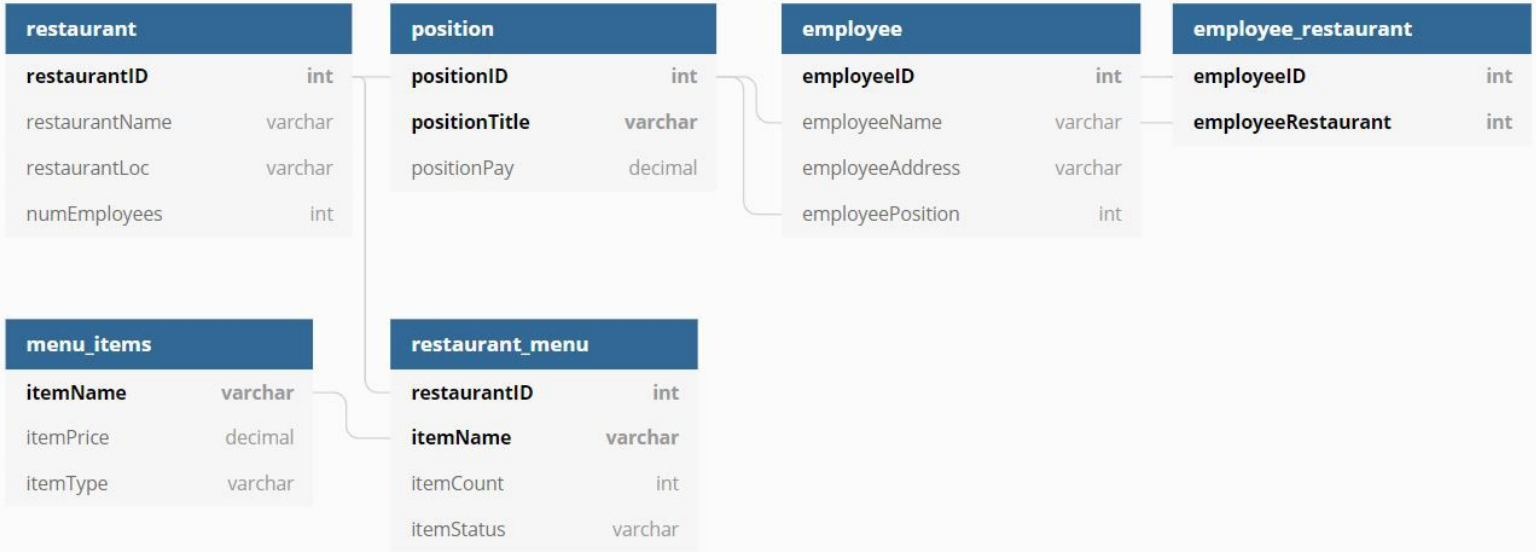
Deficiencies

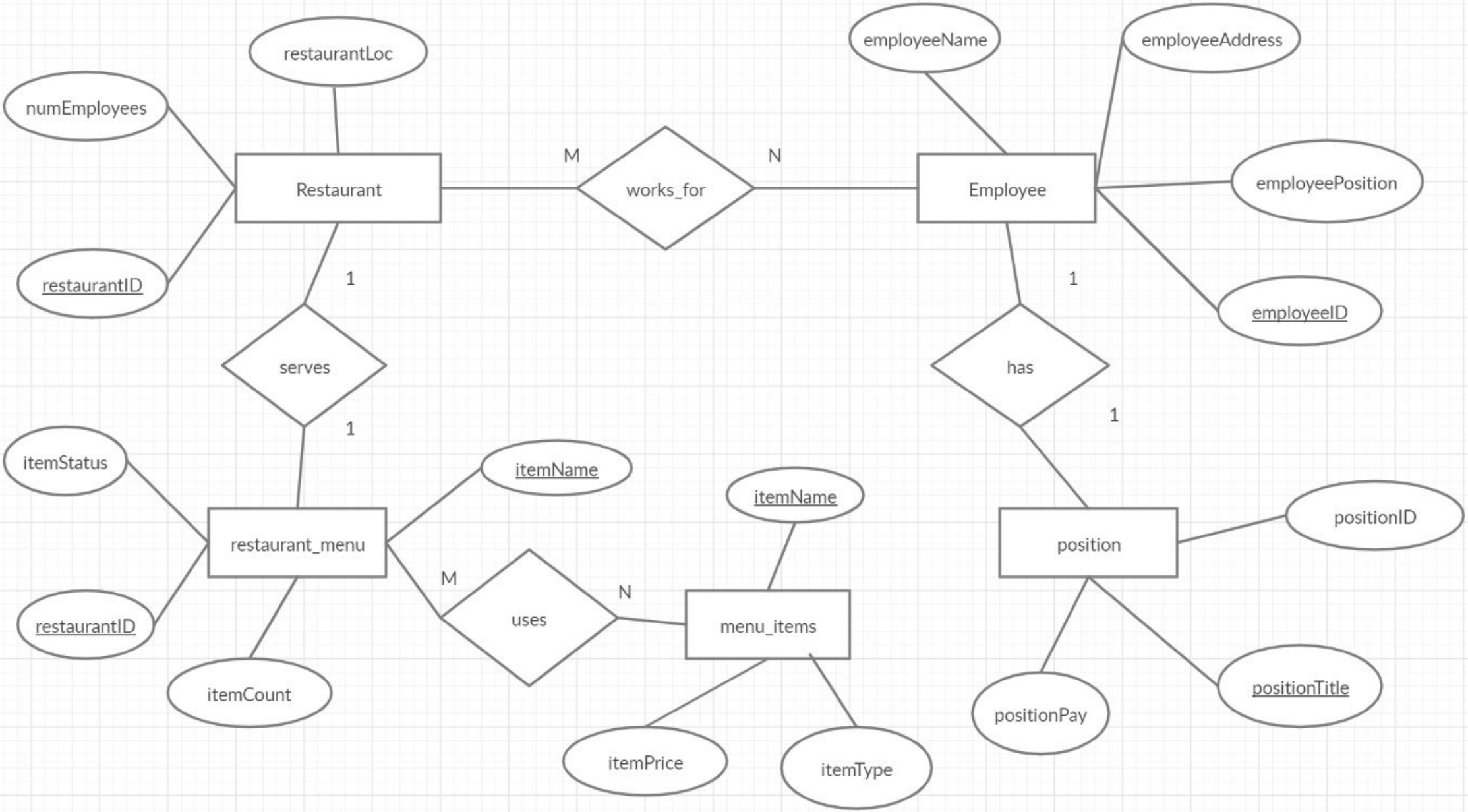
Our model's security could be improved. While the illegal character system works, it still takes away from some potential functionality of our application. While these characters aren't used often, this does not mean that the characters are not used at all. We also did not encrypt the app.config file containing the connection string to our database, and doing so would add another layer of protection to our system as a whole.

Schema & ER Diagrams

See Attached

```
Table restaurant as R {
  restaurantID int [pk, not null, unique]
  restaurantName varchar [not null]
  restaurantLoc varchar [not null]
  numEmployees int [not null, note: 'Must be 1-30']
}

Table position as P {
  positionID int [pk, not null, unique]
  positionTitle varchar [pk, not null, unique]
  positionPay decimal [not null, note: 'Must be >= 0.']
}

Table employee as E {
  employeeID int [pk, not null, unique]
  employeeName varchar [not null]
  employeeAddress varchar [not null]
  employeePosition int [not null, ref: > P.positionID]
}

Table employee_restaurant as ER {
  employeeID int [pk, unique, not null, ref: > E.employeeID]
  employeeRestaurant int [pk, unique, not null, ref: > R.restaurantID]
}

Table menu_items as MI {
  itemName varchar [pk, not null, unique]
  itemPrice decimal [not null, note: 'Must be >= 0.']
  itemType varchar [not null, note: 'Drink or Food']
}

Table restaurant_menu as RM {
  restaurantID int [pk, not null, unique, ref: > R.restaurantID]
  itemName varchar [pk, not null, unique, ref: > MI.itemName]
  itemCount int [not null, default: 0]
  itemStatus varchar [not null, note: 'lo_stock, no_stock, or in_stock']
}
```




restaurantLoc
numEmployees
Restaurant
M
works_for
N
employeeName
employeeAddress
Employee
employeePosition
restaurantID
1
employeeID
serves
1
has
1
itemStatus
itemName
itemName
positionID
restaurant_menu
position
M
N
uses
menu_items
restaurantID
itemCount
positionPay
positionTitle
itemPrice
itemType