University of Missouri-Kansas City

Restaurant Database Report

Group 7:

Henry Fundenberger, Soniya Pathak, Stephen Holeman, Giovanni Hawver, Rachel Schlutz

COMP-SCI 470, Introduction to Database Management Systems

William (Bill) Yerkes

5/9/2022

Introduction:

Restaurants everywhere need a way to manage groups of people and their purchases. Some have different offers such as merchandise, food, or drinks. We will be creating a database that will represent the floor of the restaurant and methods to track location of groups, the total bill, and everything a restaurant would need to keep track of for finance or storage reasons.

This model created was populated with real restaurant items and pricing from various restaurants, and unique people created from an online person generator as to not use random and their private information.

The RDMS which was used was MySQL 8.0.

Division of Labor:

Henry Fundenberger – Attended group meetings, help create Milestone 1 paper, specifically: Introduction, assisted with Information Collection and Requirements, database architecture, Ideas, and assisted with roles. Helped create Milestone 2 paper, specifically: Schema diagram, Site Diagram, and Constraints. Created layout based upon ER Diagram for MySQL schema, filled schema with information, created DB Build commands and Input commands. Created and added all items to GitHub. Helped create Milestone 3 paper.

Soniya Pathak – Attended first group meeting, helped create Milestone 1 paper, specifically: assisted with Information Collection and Requirements, Architecture of the project, and assisted with roles. Helped create Milestone 2 paper, specifically: Relational Algebra Statements.

Stephen Holeman – Attended group meetings, helped create Milestone 1 paper, specifically: Contributed to roles section. Helped create Milestone 2 paper, specifically: ER Diagram. Assigned roles to members not working on anything. Updated and maintained ER Diagram overtime with changes to database. Reviewed documents for upload.

Giovanni Hawver – Helped create Milestone 1 paper, specifically: Initial layout of documents, and assisted with roles. Helped create Milestone 2 paper, specifically: Front End Design. Helped create layout for Milestone 3 paper.

Rachel Schlutz – Attended first group meeting, helped create Milestone 1 paper, specifically: assisted with Information Collection and Requirements, and assisted with roles. Helped create Milestone 2 paper, specifically: Relational Algebra Statements.

Problem You are Solving:

The problem we are attempting to solve is enhancing the ability for employees and management of any restaurant. These kinds of solutions somewhat exist for front of house staff. Point of sale or PoS, systems do already exist, however the purpose of this database for our imaginary restaurant is to expand the abilities of even already existing systems like Toast or Square PoS systems. Including basic information such as the tables available for a host to sit a party at, is very helpful. Being able to see all the items a part has ordered on one specific ticket or broken into many groups will help productivity and enhance the atmosphere of a restaurant giving everybody a central form of communication they can rely on. This database layout also surpasses the needs of the front of house servers or host. It also enables kitchen staff and managers to know what needs to be fixed if equipment is down or if they are running low on stock to order more. This system also can enhance customer experience, if a restaurant were to use QR code menus that are connected to a front-end webpage that displays the menu of the restaurant in the specific season, a webpage could be made to automatically update when the season changes or if new items are added into the database a reactive webpage could be developed to include the new item in a way pleasant to the eyes. This database serves more than a traditional PoS system, but helps not just the front of house, but everybody in the restaurant always.

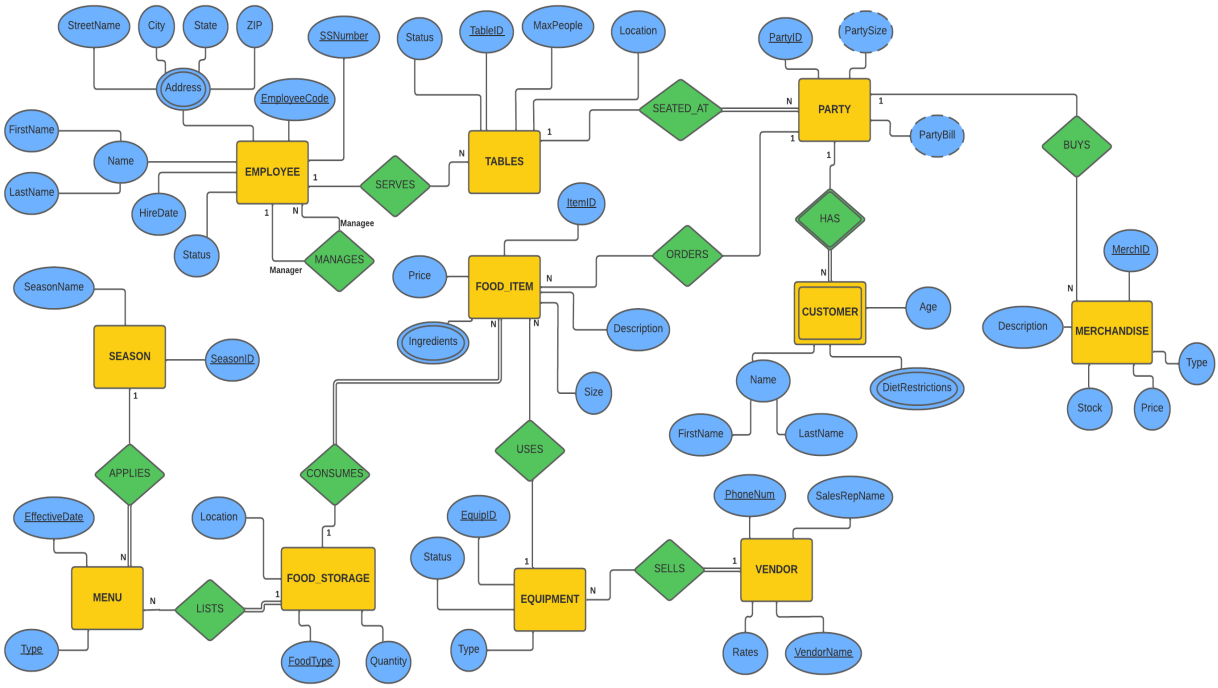
Database Requirements

1.) Employee:
   a. An employee may belong to another employee through a ManagerSSN where one manager can oversee many employees.
   b. An employee is related to one or many tables.
   c. An employee will be uniquely identified by their SSNumber, up to 9 characters.
   d. An employee will be uniquely identified by their EmployeeCode, an integer.
   e. An employee has one address consisting of Street Name, City, State, and Zip.
   f. An employee has a first name, up to 45 characters.
   g. An employee has a last name, up to 45 characters.
   h. An employee will have a hire date, managers will have the earliest higher date.
   i. An employee may have a ManagersSSN assigned to them, up to 45 characters.
   j. An employee will have an integer status to represent hired or not working.
2.) Tables:
   a. A table will be uniquely identified by a table ID integer.
   b. A table will have a status consisting of, in use, not in use, or needs cleaning.
   c. A table will have a maximum number of people allowed to be at the table.
   d. A table will have a location of inside, outside, or bar.
   e. A table will have a server, with the assigned EmployeeCode.
   f. A table relates to at least one party less than or equal to the size of maximum people.

3.) Party:
    a. A party will be uniquely identified by a PartyID integer.
    b. A party will have a PartySize representing the maximum number of people in the party.
    c. A party will have a party bill consisting of their total spent in merchandise and food_item.
    d. A party will have a specific TableID where the party belongs.
    e. A party consists of a group of customers.
    f. A party is related to merchandise through a merchandise tab.
    g. A party is related to food_item through a food_items tab.
4.) Merchandise:
    a. Merchandise is uniquely identified by its Merch_ID integer.
    b. Merchandise has a description of up to 500 characters.
    c. Merchandise has a type consisting of, hats, shirts, hoodies, etc… of up to 45 characters.
    d. Merchandise has a stock integer representing the number of items purchased.
    e. Merchandise has a price, representing the decimal value of the merchandise.
    f. Merchandise is related to a party through PartyMerchOrder integer.
5.) Customer:
    a. A customer has an age integer.
    b. A customer has diet restrictions, up to 45 characters.
    c. A customer has a first name, up to 45 characters.
    d. A customer has a last name, up to 45 characters.
    e. A customer is related to party through CustomerPartyID.
6.) Food_Item:
    a. A food_item is uniquely represented by an ItemID integer.
    b. A food_item has a price represented by a decimal.
    c. A food_item is made up of ingredients, up to 500 characters.
    d. A food_item is related to food_storage through a description of up to 500 characters.
    e. A food_item has a size of personal or more, up to 45 characters.
    f. A food_item is related to equipment through a tools integer ID.
    g. A food_item is related to a party through DeliveryParty integer.
7.) Equipment:
    a. Equipment has an equipmentID integer.
    b. Equipment has a type, up to 45 characters.
    c. Equipment has a status of working or not working.
    d. Equipment is related to vendor through a manufacture, up to 45 characters.

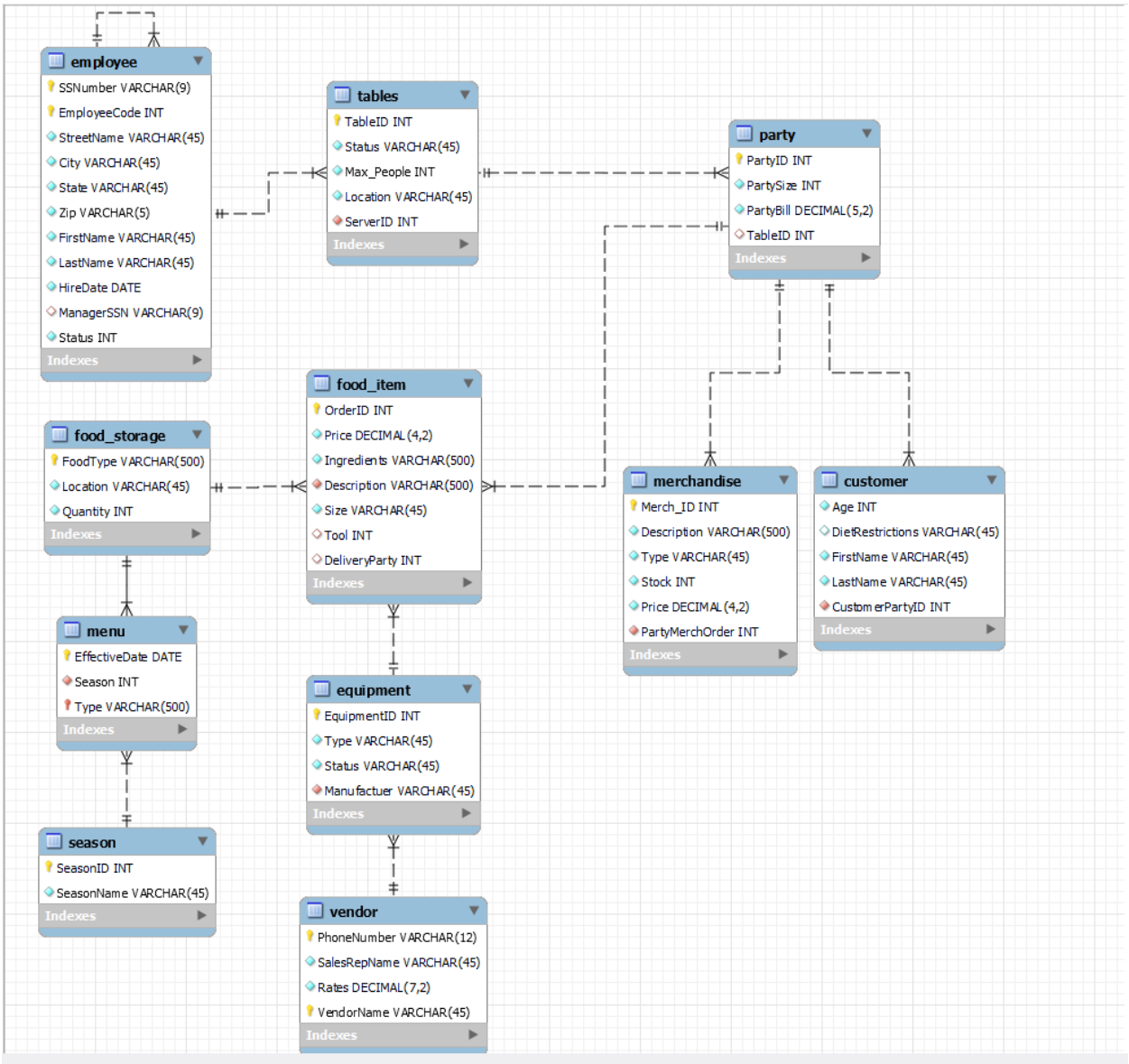
8.) Vendor:
    a. A vendor is uniquely identified by a phone number, up to 12 characters.
    b. A vendor has a sales representative name, up to 45 characters.
    c. A vendor has specific rates represented by a decimal.
    d. A vendor is uniquely identified by a vendor name, up to 45 characters.
9.) Food_Storage:
    a. The food_storage is uniquely identified by FoodType, up to 500 characters.
    b. The food_storage relates to food_item by its FoodType.
    c. The food_storage has a location description, up to 45 characters.
    d. The food_storage has a quantity of items represented by an integer.
10.) Menu:
    a. A menu is uniquely identified by an effective date.
    b. A menu relates to season through a season's integer number.
    c. A menu relates to food_storage through an items type, up to 45 characters.
11.) Season:
    a. A season is uniquely identified by a seasonID integer.
    b. A season has a season name consisting of spring, summer, fall, or winter.

ER Diagram



Link for better quality image: https://lucid.app/lucidchart/4951e0d2-ca11-4a08-8af3-c8d3ff0fc54c/edit?invitationId=inv_d34ba4ea-8184-449f-ad1d-2eab26a4d387&page=e16.L~w7n9dT#

Schema Diagram



Database Model Features

This database incorporates several queries and constraints that will be highlighted.

Show Currently Unoccupied Tables and The Server

```
SELECT
   employee.FirstName,
   employee.LastName,
   tables.Location,
   tables.TableID,
   tables.Status
FROM
   employee
      JOIN
   tables ON employee.EmployeeCode = tables.ServerID
WHERE
   tables.Status <> 'In Use'
ORDER BY tables.TableID;
```

This stored procedure gets the name of the full name of the employee and joins this information with the appropriate tables that the server belongs to, the table numbers, location, and the status of all the tables that are currently ready for a new customer or need to be cleaned to be ready for a new customer.

Show all Working Kitchen Equipment and Vendor Information

```
SELECT
   equipment.Type AS 'Equipment',
   vendor.VendorName,
   vendor.PhoneNumber
FROM
   equipment
      JOIN
   vendor ON equipment.Manufactuer = vendor.VendorName
WHERE
   equipment.status = 'Working';
```

This stored procedure returns the kitchen equipment being used joined with the Vendor Name of the vendor that made the equipment and the phone number of the appropriate vendors. This procedure allows the kitchen manager to know what equipment is currently working, and with the change of equipment_status to find machines that are not working, the kitchen manager then has the phone number of the vendor to call.

Return All Makeable Foods

```
SELECT
   food_storage.FoodType
FROM
   food_storage,
   food_item
      JOIN
   equipment ON (food_item.Tool = equipment.equipmentID)
WHERE
   food_storage.Quantity > 0
GROUP BY food_storage.FoodType;
```

This stored procedure returns only the foods that have the ingredients to be made, if the kitchen runs out of ingredients this items will be removed from the relation, and thus the menu.

Future Scope

There are many items that can be added into our system in the future that would increase the functionality of our database, and even our front end when we build it. Such as the ability for the DBMS to automatically update the values of the party's bills, when a new entry is added into "food_item" or "merchandise" tables. This can be maintained through the code connecting the front end to the back end of our software; however, I would prefer this feature to be apart of the DBMS and not have the potential of getting lost in the code.

Another feature that would allow the customers more customization in how they place orders would be, allowing parties to have sub-checks or sub-bills so they can split their bills if there is more than one group in a party. Such as two families coming together to form one party, but each family paying for their own food. Right now, our database allows for one bill per party, allowing more bills per party to split up the items purchased, would allow customers to choose what they are paying for, and would make bill management easier for employees.

Deficiencies

The goal to build a useable front end connected to the database would not achieve.

There are currently potential issues as all values had to be computed by hand, so there are potential user errors.

Currently the items that appear on the menus are set and stone even if they run out, the menu could be a view that shows all items in season that have the ingredients to be made.

Github Link: https://github.com/ItsHenryF/CS470-DB-Project