The application I decided to develop for this course project is a nutritional calculator for the Publix Deli. I chose to do this because I have been trying to lose weight for several months now and I work at a Publix Deli; the current way Publix displays the nutritional information for items in the deli is unsatisfactory at best and often non-functional. The key components of this project surround the nutrition of the different options for subs customers have for subs in the Publix Deli, and the users of the application that represent shoppers at the Publix Deli.

Database Details

The design for this database went through multiple iterations throughout the development of the project. The final iteration (and the one for this submission) is in BCNF; the tables the project uses are items (used to contain the item's name, nutritional contents, and whether or not the item is a Boar's Head item), subs (which holds the sub's name, id, whether or not it is a whole sub, whether or not it has double meat or cheese, and the ids for the meat, cheese, and bread), subtoppings (which holds the subids and the topping ids for all of the subs in the database), user (which holds the user's email, password, and username), and finally userfavs (which is used to store user's favorite subs). In the end, the main rational behind designing the database was prioritizing the ease of access if the relations in the database. For example, in the first iteration, the nutritional contents of each item were stored in a different table; this meant I would have to perform an inner join on each item to get the basic information. Having to perform a join of any kind to get the basic information of an item would mean the logic behind building subs, storing subs in databases, etc. would have been an absolute nightmare to write. Below is the relational schema and the ER diagram for the application at the time of submission.

Items

temid	Name	Itemtype	Cals	Carbs	Fat	Protein	Na	Cholesterol	Bh

Subs

Subid	Subname	Whole	Doublemeat	Doublecheese	meatid	cheeseid	breadid

Users

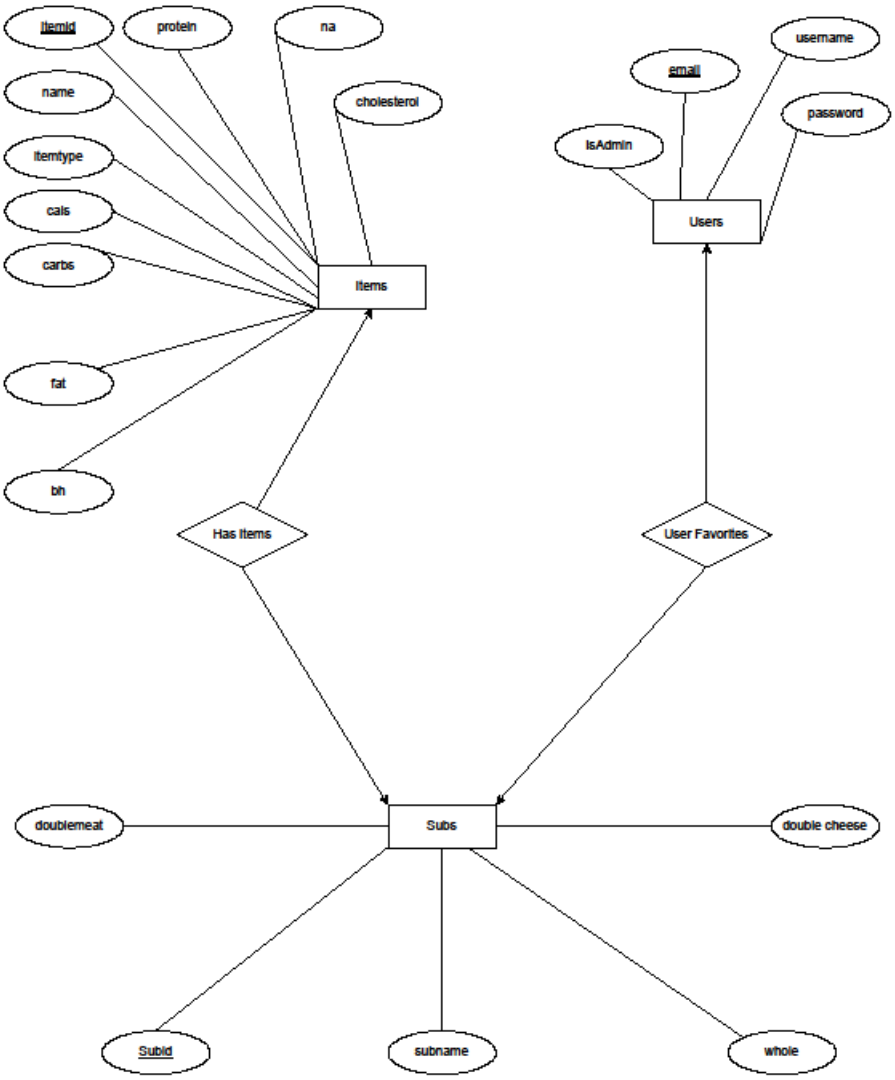
Email	Username	Admin	password

Subtoppings

Toppingid	subid

Userfavs

Email	subid



Functionality Design

The basic functions of this project are looking at the information of a single item, adding items to the database, removing items from the database, editing items in the database, adding users to the database, removing users from the database, and logging into the application. Viewing the items in the database involves a standard “SELECT * FROM items” query to display all the items in the database to allow users to select the item they want to see the nutritional contents of where the nutritional contents of the item is extracted. Adding items to, editing items in, and deleting items from the database can only be done by admin users (more on that in the

next paragraph); adding items is done by getting the next id available for items in the database (I did not know about the SERIAL type at this point in creating the application) and inserting the name, nutritional contents of the item, and other information about the item into the database using an INSERT query. Editing and removing items combine the previous two functionalities: admin users can select an item to either edit or delete in a screen like the one regular users use to select the item they want to view. When deciding to edit an item, users are taken to a page where they can edit all the item's information – with exception of its id as that is the primary key – and submit those changes. To delete an item, the admin user just needs to click the delete button and they are given a confirmation window; if they click "yes" then the item is removed from the database using a DELETE query.

To add a user to the database, a new user would need to click the "Sign up" link on the login screen and enter their username, email address, and password. The registration function performs a query to check if the email address the user enters is already associated with another user; if it is the user is notified and no other queries are performed. However, if there is no user with that email address already in the database then the new user's information is inserted into the database and is logged in. New users are not allowed to give themselves admin permissions as that would defeat the purpose of special permissions. Logging into the application involves checking if the email address the user inputs is in the database, and then checking if the password they entered is the one in the database. If it does then the user is logged in, if not the user is prompted to input their email and password in again. Admin users are allowed to delete users in the database in a manner like how they would delete an item.

For the advanced functionality, I allowed users to build subs in the application, display the nutritional information of the subs, allow users to insert the subs into the database to be viewed later, and allow them to edit and delete their favorite subs. When users go to the page that allows them to build a sub, four queries are performed: one to get all items of each type (meat, cheese, bread, and toppings). Users are allowed to select one of each from the first three groups, and then as many toppings as they would like. Additionally, users can decide on whether they want double meat, double cheese, or a whole sub on this screen. The nutritional information of the sub is updated as they build the sub. If the user is logged into the application, then they are allowed to give the sub a name (with the default name being "My favorite sub") and add the sub into the database. The reason this feature is advanced is due to the complexity of how this accomplished; most of the sub's information is stored in the "subs" table. The only attributes not stored in this table are the toppings and the which user saved the sub, with the former being stored into the "subtoppings" table where only the topping's id and the sub's id are stored and the latter storing the user's email address and the sub's id.

To get the nutritional information of the sub to display to the user, users click on the sub that they would like to view the nutritional information of, and they are redirected to a page that shows the nutritional information. A query is performed that sums the items' nutritional information and joins the tables subs, subtoppings, and items. Editing a sub follows a similar process where users click on the edit button next to the sub they would like to edit, and they re-enter all of the information they want to change; then an UPDATE query is performed on the database to insert the new information into the database. Deleting a sub involves clicking on the delete button next to the sub and then performing a DELETE query on the subs table where the ids match. Since both the userfavs and the subtoppings tables have foreign keys form the subs

table, I added an “ON DELETE CASCADE” to both foreign keys to ensure there are no anomalies in the data.

The reason the features surrounding subs are considered advanced is due to the length of time it took to implement these functionalities and the complexity of these features. My reasoning is evidenced by the SQL queries used to accomplish these features; while attempting to keep all lines of code less than 80 characters, the query to calculate the nutrition of user favorites is over ten lines of code. Most other queries are five lines of SQL or less, meaning that one query has more than double the amount of code than the average query for the project.

Implementation Details

For implementing this application, I used React, NodeJS, Express, and Postgres to use the “PERN” tech stack. The reason behind this decision is I had learned the “MERN” tech stack for another class this semester which uses MongoDB as the database; since MongoDB is a “NoSQL” database, I assumed that it would not work for this project, so I decided to use Postgres due to its popularity. React is used for the frontend because it allows for responsive web applications that are easy to use and popular. React makes the application easier to use, and faster to use as the application does not need to refresh as often as other applications. The front end interacts with Postgres through NodeJS and Express; they allow for an API to run on a different URL with a different port which is used to perform queries on the database. React, NodeJS, and Express all use JavaScript, but React uses HTML and CSS in addition to JavaScript to create web pages. The GitHub repository for the project can be found at: https://github.com/metalBread33/DBMSProj.

Experiences

I learned a great deal about developing front end applications with a relational database while creating this project. The biggest thing I learned however is to think about how features would be implemented when building the database. I spent half of the time I was given to work on this project refactoring the database, and to be honest I am still thinking about how it can be improved. For example, although having a table that stores the nutritional information of each item would mean I would have to perform more joins when writing queries, it would mean the table for food items would be significantly simpler. Maybe it would come down to a matter of preference, but that is something I look forward to experimenting with. Solving the hard problems within this project was a matter of hard work, persistence, and using the resources that I have found such as the Udemy tutorial my other class required and ChatGPT. I plan to submit this project through Publix’s Idea Spot which is where associates can submit ideas that could help the company and continue to refactor the project to make it more secure, faster and have more functionality. For example, the passwords are stored as plaintext in the database, which is a security risk; running the password through a hash function upon submitting it would prevent passwords from being leaked to the public. Submitting the application to Publix would make sure it has accurate information and can be properly embedded into the Publix ecosystem.

References

OpenAI. (2020). ChatGPT. OpenAI. https://openai.com/chatgpt

Mern ecommerce from scratch | Udemy. (n.d.). https://www.udemy.com/course/mern-ecommerce/

SQL data types for mysql, SQL Server, and MS access. (n.d.). https://www.w3schools.com/sql/sql_datatypes.asp

YouTube. (2020, March 19). *Pern Stack course - postgres, Express, react, and node*. YouTube. https://www.youtube.com/watch?v=ldYcgPKEZC8&t=1213s