DBMS Final Project Report

Sarah Groark, Devon Reing, Erin Zahner

December 2, 2024

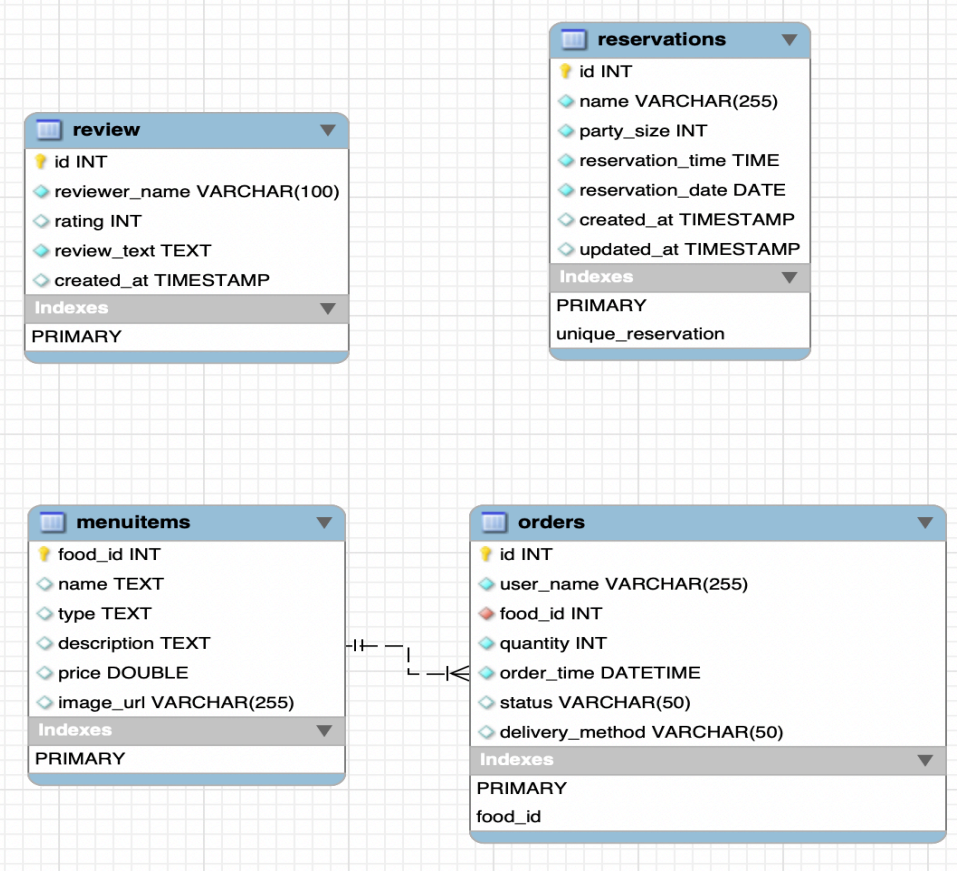
Contents

1. Project Outline

The intended purpose of this project was to create a functional business website for a local restaurant. We aimed to incorporate features that would enhance both user experience as well as business process efficiency. The functionality included the following – reservation and online ordering features as well as menu and customer reviews visibility features. We approached the development of this project using a three-tiered client/server approach: MySQL for the backend, Node as the server, and HTML, CSS, and JavaScript for the frontend. The entire outline of the project, its development, and features will be outlined in this report.

2. Backend Information

ER Diagram:



The structure of the database consists of four tables – review, menuitems, orders, and reservations. They all exist semi-independently and are the underlying data structures for their respective pages on the website.

Important Model Components =

1. Constraints
    a. The reservations table has one unique constraint that prohibits the presence of more than one reservation at one time slot of a day. This constraint assists in front end features such as displaying the availability of reservation times for a given day when a user is booking a reservation.
2. Indexes & Keys
    a. The reservations table also utilizes a primary index to enhance query execution performance, which aids when a user inputs their reservation code (the primary key) on the site and the query must search for that specific record.

3. Site Pages

   a. Home:

      The home page serves as the landing page for all users. Here, users can find the restaurant's hours of operation, customer reviews, and a menu bar for navigation across the site. The navigation bar was crafted with a nav attribute with the following options: menu, orders, and reservation information (built with a dropdown class attribute to include pages for making, updating, and deleting a reservation).

   b. Reservations:

      The three pages falling under the 'Reservations' tab all utilize similar logic to manage the functionality of reservations for the restaurant. All three incorporate error handling, preventing users from submitting invalid data or attempting to submit empty form fields. Any action dealing with a form will notify the user in the form of success, failure, invalid, or other messages to guide the user. All three directly manipulate the reservations table found in the database via select, insert into, update, and delete statements.

      i. Make a Reservation

      The 'Make a Reservation' tab under the Reservations option of the main toolbar allows users to book a reservation at the restaurant. Several user steps are taken to accomplish this task, including the population of the following fields: name, date, time, and party size. The name field consists of a text box, date field is of date type, time is a drop down menu with the available reservation times for that day, and the party size field is a number input with a minimum of 1 and a maximum of 10. The crucial functionality of this page is the population of available times in the time drop down select menu for the chosen booking date. This is accomplished with a three-step process. First, there is a listener implemented on the front end that is responsible for triggering the fetchAvailableTimes() function as soon as the user selects a date. Second, this function (with the user-entered date as a parameter) calls the fetch('/available-times') from server.js which queries the database to return the times already booked for that specific day. The returned result is a JSON structure of times that are not permitted to be reserved. The front-end JavaScript then processes the times to display to the user by pushing the

usually available times onto the time field option menu unless they are in that result list that was sent from the back end. Finally, once the user selects a time, they can press the ‘Submit’ button and if successful, they will receive a message thanking them for booking with us. In theory, they would receive a confirmation code and the information for the upcoming reservation through some means of communication (text, email, etc.)

ii. Update Existing Reservation

This page allows the user to make changes to their existing reservation. They are prompted to enter the reservation code, which is responsible for pulling their reservation record from the database. This is accomplished through the front end listening for the user to submit their reservation code, which then triggers getResInfo(). This function handles the data submitted (reservation code) and triggers the fetch(‘/get-res-info’) get call from the backend server. This queries the database using a SELECT statement to locate the reservation record associated with the reservation code that the user entered (which is the primary key). Once this get method finds the reservation, the record information is returned to the front end in JSON format. The front-end then displays the previous reservation information in a form format, which allows the user to change the fields and modify their reservation details. The same logic for the time field that was mentioned above is used here. Once the user submits their updated information, the front-end triggers the fetch(‘/update-res-info’) post method, which uses an UPDATE statement to change the values of the attributes associated with that reservation record. If successful, the user is prompted with a success message and returned back to the site’s home page.

iii. Delete Existing Reservation

This page allows the user to delete their existing reservation. Similar to the update reservation page, they are prompted to enter their reservation code, which pulls their reservation record from the database. This is accomplished with the same logic using fetch(‘/get-res-info’) and the select statement to pull the record. Once this get method finds the reservation, the record information is returned to the front end in JSON format. The front-end then displays the previous reservation

information in a text format. The user is able to click ‘Delete Reservation’ which then triggers a pop-up confirmation asking them to confirm their deletion. If the user selects yes, the front-end triggers the function deleteReservation(), with the reservation ID as a parameter. This calls the fetch(‘/delete-reservation’) post method, which uses a DELETE FROM statement to make changes to the database. If successful, the user is prompted with a message indicating that their reservation has been canceled and is returned back to the site’s home page.

c. Order Online:

This page loads the menu items in a similar manner to the menu page. Through an API call that executes a select statement to select all the menu items from the database in the backend, all the menu items are loaded in a table structure on the page. All the menu items are organized by type (appetizer, entree, dessert, beverage) in the orders javascript to prevent needing to make multiple SQL query calls with where clauses to separate the items by category. Once the sections are created for each type of menu item, each row of the table features a picture of the menu item, the title and description of the menu item, the price, and an add to cart button. Once the add to cart button is clicked, a cart table is created at the bottom of the page that includes the item in the cart, quantity, price, and total price. Once the customer is satisfied with their cart they can then scroll down where they will be prompted to enter their credit card information and a username. The username will be used in the orders table and all the fields in that form are required for the user to be able to continue placing their order. If all the fields are correctly filled out, the order is sent to the server to be carried out and added to the orders table in the database. The username and cart items are sent as variables in the URL which are then retrieved as part of a post request to be used as the values in the insert into query. If the query is not successful, the user will be alerted through an error message alerting them to which portion of the query contains illegal values so that they may be adjusted on the online order page to attempt to place the order again. If the query is successful, the user will be redirected to a page alerting them that the order is confirmed.

d. Menu

This page loads the menu items through an API call that executes a select statement to retrieve all the menu items from the database on the backend. The menu items include details such as name, description, price, image URL, and type, and are dynamically displayed on the page. To optimize performance, all the menu items are organized by type (e.g., appetizer, entree, dessert, beverage) in the

JavaScript, preventing the need to make multiple SQL queries with where clauses to separate items by category. Once the menu items are grouped, sections are dynamically created for each type of menu item. Each section features a header with the category name and displays a grid of menu items. Every menu item includes an image, the name of the item, a brief description, and the price, all formatted for clarity and ease of navigation. These elements are added to the page dynamically using JavaScript to allow for efficient updates and flexibility. The CSS for this page enhances the user experience by creating an interactive and visually appealing design. Each menu item is styled as a card, with hover effects applied to reveal additional information. When a user hovers over a menu item, the CSS transitions bring the details into focus by slightly enlarging the card and fading in the item description and price. This effect is achieved using the :hover pseudo-class along with transform and opacity properties in CSS. For example, the menu-item class applies basic styling, while the menu-details class initially hides the item's text with low opacity, making it fully visible when hovered. Additionally, the layout uses a responsive grid to ensure that menu items are displayed neatly on various screen sizes. The grid automatically adjusts the number of columns based on the device width, maintaining a clean and accessible design. If there is an error fetching the menu items, an error message is logged to the console to ensure any issues can be addressed promptly. This design not only organizes the menu effectively but also provides a user-friendly browsing experience. It allows users to interact with the menu intuitively, while the dynamic loading and CSS effects ensure a modern, polished presentation.

e. Reviews

This page loads customer reviews through an API call that executes a SELECT statement to retrieve all the reviews from the database on the backend. The reviews include the reviewer's name, rating, review text, and the date the review was created. These reviews are dynamically displayed on the page in individual review cards. Once the reviews are fetched, each review is displayed in its own review-card container. The review card includes the reviewer's name as a heading, a star rating displayed with filled and empty stars (based on the rating), the review text, and the date the review was created. This information is dynamically inserted into the page through JavaScript, which ensures that any changes to the database are automatically reflected on the website without requiring manual updates to the HTML. The CSS allowed for each card to be styled similar to a Google review with a white card and filled-in yellow stars to accompany the name and review information.

4. Ranking Statement (can also be found in ranking.html under the report folder in the GitHub repository)

Erin Zahner:

My teammates and I agree that I handled 33% of the overall project. My specific tasks included:

1. I set up the initial SQL database, filling it with general table structures and information needed to get the restaurant website started.
2. I set up the initial basic HTML pages for the website. I created each page with a general header, body, and footer section and included links to each page in each header.
3. I handled inserting all of the menu items needed into the table in SQL, this included names, prices, types, descriptions and pictures to be displayed.
4. I then handled dynamically displaying the menu items on the menu page for the website. This involved setting up an API to retrieve the menu items from the database, including details like name, description, price, and image. I then created the HTML structure and used JavaScript to display the items in sections, grouped by type. To ensure the menu is always up-to-date, I ensured the items were pulled directly from the database instead of being hardcoded. I also integrated the page with the server, so menu updates are automatically reflected without modifying the code manually.
5. I also handled the website's home page. This involved adding the hours of operation and integrating the customer reviews section. I set up an API to fetch reviews from the database, which includes the reviewer's name, rating, review text, and date. Using JavaScript, I dynamically displayed the reviews on the page in individual cards. This setup ensures that the reviews are always up-to-date and displayed correctly without requiring manual updates. The reviews are automatically fetched and shown in a clean, organized layout.
6. I did much of the CSS styling including the general page styles with the color theme, as well as styling for the header of each page, the drop-down menu, and the more involved styling of the home and menu pages.

Sarah Groark:

My teammates and I agree that I handled 33% of the overall project. My specific tasks included:

1. I drafted the final report document and outlined the layout for the report. I also wrote the project outline, backend information, and home page of the site pages components.
2. I handled the portion of database edits that involved making the database compliant with the reservations functionality. This involved altering the date and time attributes, as well as developing the constraint that would only allow one reservation to be booked at one time of one day.
3. I handled the reservations functionality of the site. This was the bulk of my work and included a few things: research, front-end design, and server handling.

a. Research mainly focused on the mechanism in which I could manipulate the backend database using forms in html. This allowed me to utilize insert into, select, update, and delete statements across all three pages for the reservations feature (make a reservation, update a reservation, delete a reservation).
b. Front-end design allowed me to manipulate the style, appearance, and functionality of the forms that the user would use. For example, I wanted the user to be able to select a date from the date field and then have the time field populate with the available times for the chosen day. This required some front-end manipulation to work in tandem with the server that would fetch these available times. For this feature, I added a few different listeners to the front-end to make this kind of dynamic approach possible. Additionally, error handling/input validation and success messaging was used on the front-end (when possible) and prevented users from submitting faulty data, empty form fields, and informed them of the status of the actions.
c. The main portion of the work for the reservations feature of the project was dealing with the server and get/post statements. The select statement was utilized in the make a reservation page as well as the update a reservation page. This statement was responsible for querying the database for the open times for a reservation on the user-chosen date. The update statement was used in the update a reservation page and required the user to enter their 'reservation code', which was simply just the primary id of the record. Finally, the delete statement also required the user's reservation code, and deleted that record from the table.

Devon Reing:

My teammates and I agree that I handled 33% of the overall project. My specific tasks included:

1. I wrote the initial one-page proposal. We had all agreed upon a restaurant website with a menu, reservations, and online orders, but had yet to figure out the exact ways to incorporate all the select, insert into, update and delete functionality into the website. Part of writing this proposal included finding specific ways to incorporate all of those elements.
2. I handled setting up the GitHub repository for us to use for the project. This included adding access to both Erin and Sarah so that we could all add our work to the repository.
3. I handled editing the SQL file once it was uploaded onto the Github repository. When it was uploaded, some extra edits were added by Github or MySQL Workbench, making the file hard to read and use efficiently. To fix this, I went through the file to remove any odd comments and updated the syntax of the tables being created so that we could continue with the rest of the project tasks. In doing so, I also added sample queries for us to use in the NodeJS server file according to the select, insert into, update, and delete statements we had planned to include.

4. For the website, I handled the online orders functionality. This involved creating the orders and order confirmation HTML pages with the CSS styling we used for the entire site. In addition to the HTML pages, I added the pages to the server with both post and get methods so that they were accessible at the appropriate time on the site. On the server, I also modified the API Erin set up to get the menu items to include more information needed for inserting new food items into the database once an online order was placed. Using this API, I finally set up NodeJS pages for the orders page. This allowed the site to be more functional with additional capabilities such as adding to the cart and setting up the menu based on the current state of the menuitems table instead of hardcoding the menu on the site.