Final Report - Team050 - Produce 101

Team Members:

Jingyu Li (jingyul9), Yutao Zhou (yutaoz3), Junda Shen (jundas2), Yuhao Feng (yuhaof3)

1. Please list out changes in directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).

The only change in direction of our project is removing the fuzzy search functionality. Instead, we implemented a search system based on the keywords in each dish's name.

Our final project is a customer-oriented dish ordering and recommendation system that can be used by restaurants. We implemented the core functionalities with a vivid interface as proposed in the original proposal, which includes food searching, dish ordering, food recommendations, and checkout. However, the fuzzy search functionality in the original proposal is not implemented due to the extra workload, but our customers can still search for the dishes they want with high confidence.

2. Discuss what you think your application achieved or failed to achieve regarding its usefulness.

We've implemented the core idea of our application which is helping customers make better decisions and accelerate the ordering process. This is important to both restaurant's efficiency and the customer's quality of experience. The idea is achieved by two functions. The first one is a recommendation function that shows the recommended dishes to customers on the menu according to their preferences and history of purchases. Another is a TOP-10 food list, which lists the most popular dishes in the current restaurant. Both functions help customers to make better choices.

Although the final version application achieves most of the usefulness we proposed in the proposal, there is one we miss, which is the fuzzy search of ingredients. In our initial design, we want to improve users' experience by allowing them to search for food via ingredients. This is useful in their ordering process and especially important for those people who are allergic to certain ingredients. We did not achieve this usefulness due to the time limitation.

3. Discuss if you changed the schema or source of the data for your application

We do not change the schema of the data but strictly follow the UML diagram to create the relational database. Our schema is delicately designed so no modification is required. As for the source of data, originally we plan to use the Python Web Crawler to collect data from different restaurant websites (e.g. https://www.mcdonalds.com/us/en-us.html). However, we find a wonderful food dataset in Kango which fits the requirement of our application. The food dataset includes food names, food descriptions, images, ingredients, and so on. We perform some data cleaning and processing work and use this dataset for our project. The reason why we use an existing dataset is that it reduces the workload of searching and processing data and improves our efficiency. As for customer data, we plan to generate some random data

originally, but we find a good customer dataset just on the 411-course website. We perform some tricky combinations and extend the entry length to 1000.

4. Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?

There's no change to our ER diagram or our table implementation. However, some tables (e.g. Recipes, OrderDishes) serve as the connection of other tables so that the web pages (e.g. the order page) can show a list of items. To show the list of items we need to join several tables, which can be very inefficient for large tables. It would be more suitable to use databases that can store lists to store the connections, for example, MongoDB can be a good choice.

5. Discuss what functionalities you added or removed. Why?

Compared with our initial design, we have some functionalities added and removed.

We add two functions in our final version of the application, both of which serve to benefit the customer's experience. Firstly, we implement the auto-calculated discount for each payment, achieved with the stored procedure. The discount is based on their order history including how much money they've spent in the past and how many times they've visited. The reason for designing discounts in this pattern is that it is essential to maintain a good relationship with frequent customers, who contribute to the major part of revenue. Setting up discount can make them more willing to drop by. Secondly, we add a TOP-10 food list to our application. The motivation behind it is we also care much about the new customers. A well-designed TOP-10 list could leave a great impression on customers' first visit and make their ordering process easier.

As for the removed one, we did not implement a personal page for each user where they can see his/her own information and order history. Because we want to make our application minimal and clean. A user's personal page tends to be visited very infrequently and thus not central to the core function of our application.

6. Explain how you think your advanced database programs complement your application.

Stored Procedure: The checkout function is implemented in the Stored Procedure of MySQL. This stored procedure will search users' order history and calculate the discount according to how much money they spent in the past and how many times they eat here. The reason why we use stored procedure is we can embed these complex queries into one function. It provides better productivity and it is much easier to use. Also, the procedure calls are quick and efficient because stored procedures are compiled once and stored in executable form. So the response is quick. Also, the executable code is automatically cached, which lowers the memory requirements. Thus, this function fits the application very well and improves its performance.

Trigger: The registration function is implemented with a trigger. We assume the phone number for each customer is unique, so we use the trigger to check if the input phone number has been stored in the database. If so, the database would reject the insertion and raise a user-defined exception (SQLSTATE = 45000) with a message to the backend. Otherwise, assign the new customer with a CustomerId and return it to the backend. The registration check is handled by the database since MySQL can efficiently and

accurately handle concurrent requests. It is crucial to assure the uniqueness of phone numbers and it is harder to implement such functionality on the backend.

(Notes: You can access the code of advanced programs in our repo: /order_system/query/)

7. Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.

Yutao Zhou: Stored procedures often depend on other objects in the database, such as tables, views, and other stored procedures. This can make it difficult to make changes to the stored procedure without breaking dependencies and potentially causing errors in the application. During my implementation of the stored procedure, the queries about calculating discounts is complex since it involves multiple tables (Customers, Orders, Dishes, OrderDishes) and multiple controlling conditions. Managing the dependencies in such a situation is challenging. There were many times that the program violate the pre-defined dependencies, for example, the OrderId in OrderDishes is a foreign key reference to Order. The solution is to use temporary tables. The temporary tables hold these intermediate results, which can help to simplify the stored procedure and reduce the dependencies between tables.

Junda Shen: The trigger for the registration function can detect duplicate phone numbers and reject the insertion, but the problem is how to know that the insertion fails. The first solution was to query the database one more time to check if the customer information associated with the phone number is correct, but this is inefficient and can be inaccurate when the database had the same information originally. After some research, we found that we can signal an SQLSTATE to indicate an unhandled exception and return a message to the backend so that the backend can take appropriate actions. This turned out to be an elegant solution and simplified the development.

Jingyu Li: The technical challenge we encounter is about the transmission of the parameter between different functions in React. After the user successfully login the system, information about the user should be passed to the following function to perform user-specific work. For example, the customer id should be passed to the food search function to add dishes for specific customers, and passed to the order function for checkout. However, we use a router to route different pages and the router is created at the very beginning. So after the user login, the corresponding user id cannot be passed to other pages in the router, since it's not dynamic (or at least I didn't find a way to pass it in function). We think of some solutions to the technical problem. Store the customer id in the database, store the customer id in a text file, or store the customer id information in the URL. We choose the last method, where the customer id serves as a suffix of the URL, and the function can directly extract the information from the URL. However, this solution is not perfect since users can modify the URL randomly and access other user's information, so some data isolation work should be performed to ensure security.

Yuhao Feng: When designing the recommendation feature in the food search page, I want to show both the result of food recommendations and the food search result in one single table. The design of concatenation is a challenge because I want to make the result of the table displayed in one pass and avoid any kind of refreshing/flickering. The simplest approach I found is to do concatenation inside the SQL query using UNION, and then return the entire result from the cloud database to the backend. In this way, we do not

need to do concatenation in backend, and thus makes the program simple to read and easy to maintain.

8. Are there other things that changed comparing the final application with the original proposal?

As we mentioned before, we do not implement the fuzzy search (search ingredient) function as our proposal proposed. We have implemented the top-10 food recommendation and top-10 frequent customer functions for a better user experience. Besides, for the advanced SQL program, we originally planned to implement a store procedure for food recommendation, and implement a trigger for order checkout. After considering the application scenario of the store procedure and the trigger, we finally use the store procedure for order checkout, since there are multiple if-else conditions. We implement triggers for user registers since registration requires inserting new data into the database and triggers can work in this case. Finally, the low-fidelity UI mockup is different from the application interface, where we make the front-end more clean and easy to use.

9. Describe future work that you think, other than the interface, that the application can improve on

As for the application feature, adding the ingredient search function is a good choice, which provides customers with more flexibility when ordering food. One of the advanced future works of feature is to recommend food with some machine learning model. As for the database design, we can add more attributes to make the table more complete and inclusive. For example, we can add the food calorie and customer gender. As for system optimization, we can integrate the backend and the database instance, since currently our backend and database are deployed on different GCP instances. Putting the two staff in one place can increase the performance and reduce the connection overhead.

10. Describe the final division of labor and how well you managed teamwork.

Yutao Zhou: The tasks I am responsible for include: design database, implement advanced queries, add index to database and analyze, add dish to an order and checkout with auto-calculated discount according to user's order history based on stored procedure.

Junda Shen: The tasks I am responsible for include: data generation (ingredients, recipes, dishes), order page design (update order information), registration page design, trigger (checks duplicate phone number and ask backend to take appropriate actions), GCP deployment

Jingyu Li: The tasks I am responsible for include: schema design, SQL instance deployment on GCP, project framework construction, login page design, home page design, food search page design, and top customer page design.

Yuhao Feng: The tasks I am responsible for include: user interface design, UML diagram design, generating raw data for relations (Orders, Orderdishes, Seatings), order page design, customized food recommendation feature design (in the food search page)

In a word, we managed teamwork pretty well. We had the online/onsite meeting once in two weeks on average. We discussed application design, project requirements, and workload distribution during the meeting and recorded the information in a notion note. We will also make our own deadline to synchronize the work. Everyone gets a reasonable workload and collaborates wonderfully. There is no free rider.