Eric Vela

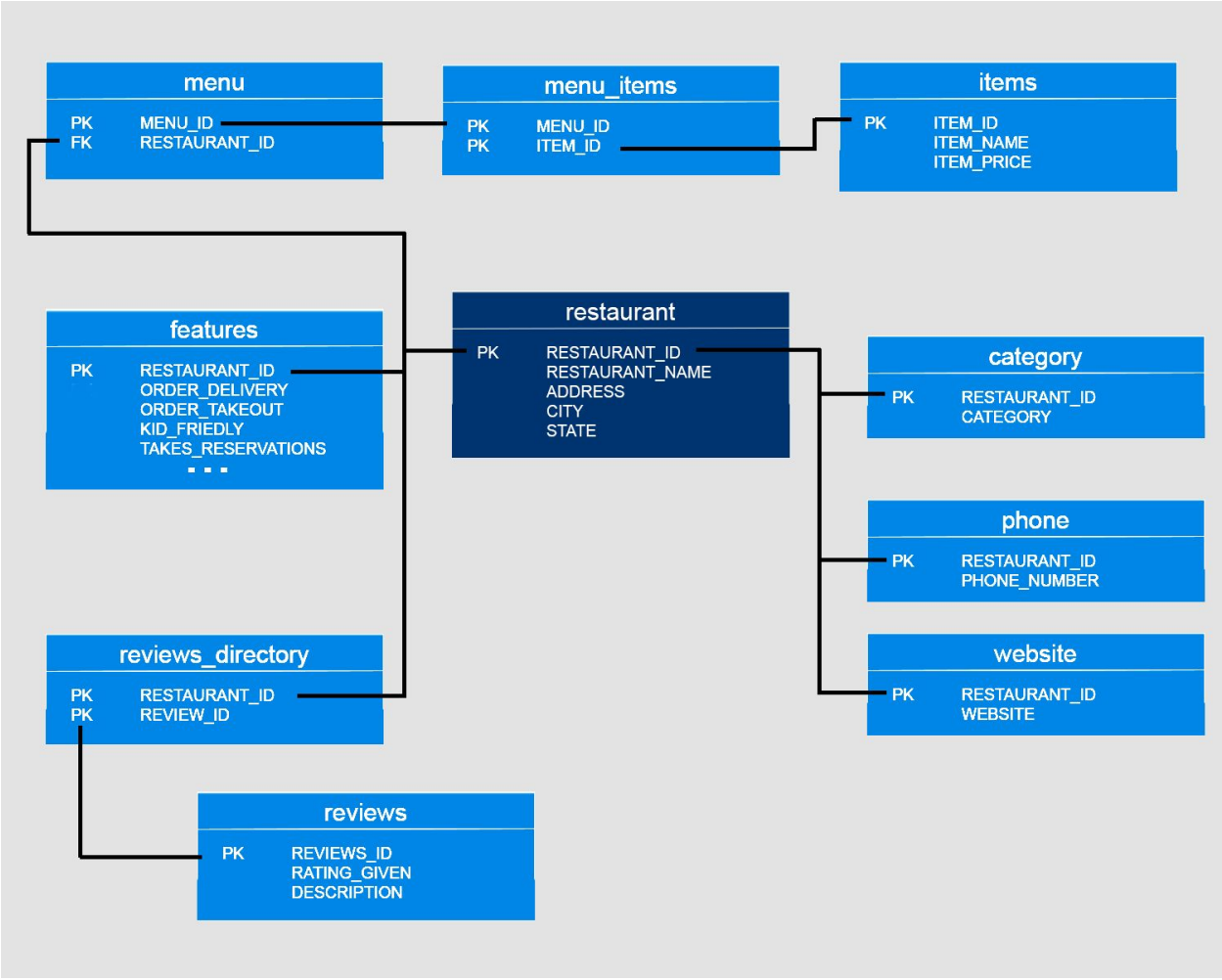
Charlie Liu

Restaurant Database

When trying to find a good place to eat, it is not always easy to find what you are looking for. Looking up a restaurant online, it is easy to find the name of the restaurant, its cuisine, and a phone number or a website. Often, it is difficult to find what is on the menu at any given time or any reviews on the quality of service or food served. Sometimes, it is impossible to know how much the items on the menu will cost until you get there. Moreover, most restaurants may not have many desired features or tailor to one's needs. If one plans to go out with a group of friends, a restaurant that can handle large groups and serves alcohol would be most suitable. If one has a family and wants to take their children out to dinner, it is important that the restaurant is kid-friendly and offers a kid's menu. Or perhaps, one is walking around a shopping mall with their dog. It is important that the restaurant allows dogs and offers outdoor seating. Combining all these elements in a user-friendly and easily navigable user interface is the goal. This user interface provides enough information to make the best judgement on a restaurant so that there will never be a bad restaurant experience again.

Creating this system required us to create a database that would allow us to efficiently and effectively store restaurant data to allow for easy access. Initially, incorporating aspects of restaurants such as the name, location, contact information, menus, and reviews became the focal point of this project. However, creating a system that had no functional dependencies and logical constraints appeared to be a more

daunting task. After several hours configuring the database, and testing if it was set up correctly, the final product was created, featuring more tables than we intended. Although the schema appeared complex, the database was in third normal form and operated exactly as planned. Below is a diagram of the schema:

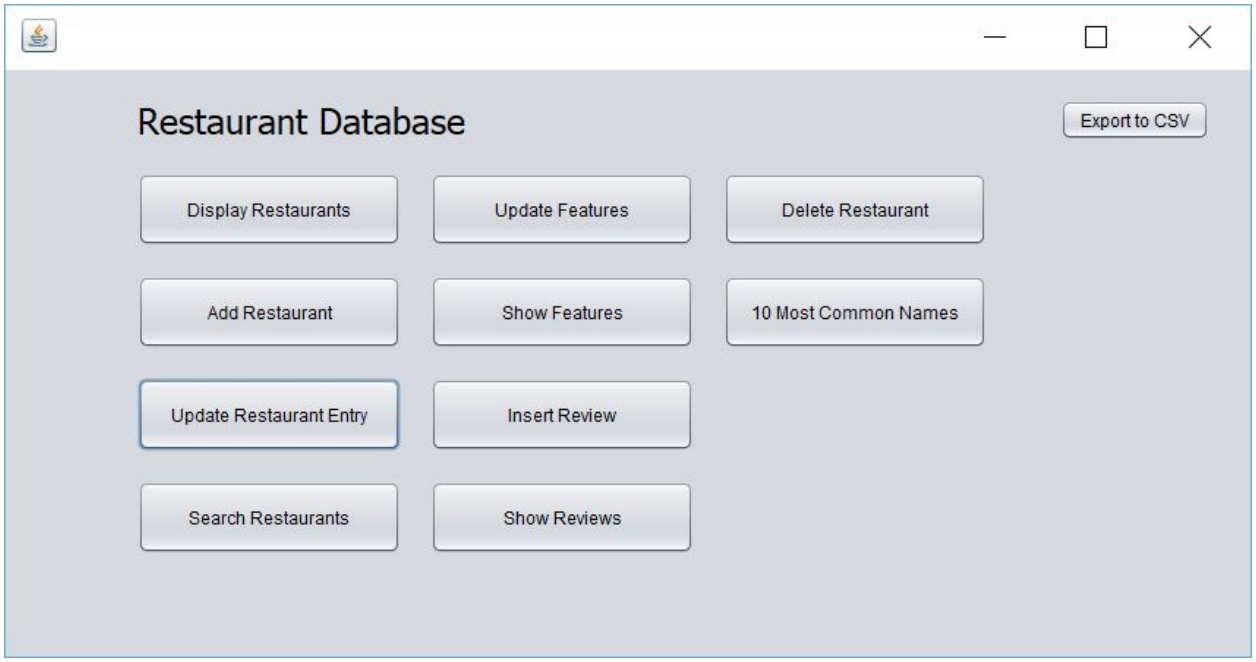


The field RESTAURANT_ID is the primary key across most tables and original comes from the parent table which is “restaurant.” RESTAURANT_ID connects

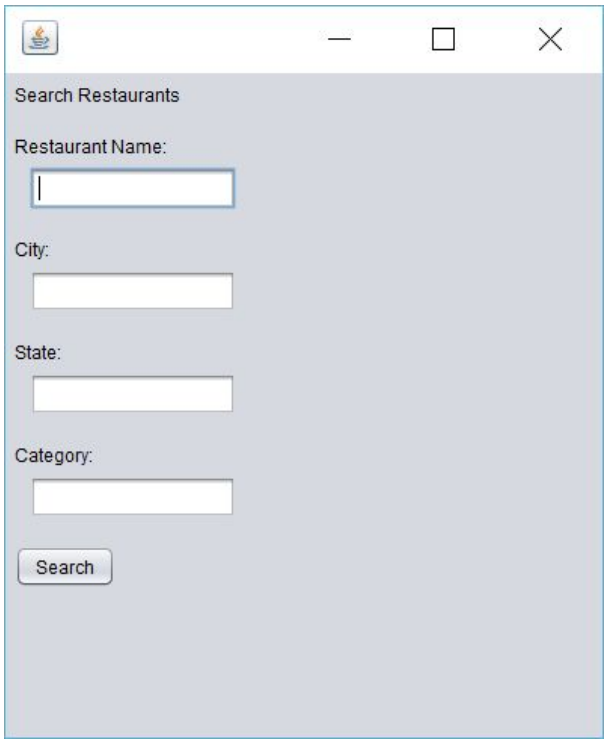
information regarding the physical, contact and website information, as well as a directory to reviews regarding the restaurant. All other tables build off of “restaurant” into tables that are not strictly based off the RESTAURANT_ID such as reviews, menu_items, and items which have their own unique identifiers.

To further elaborate why the tables “reviews,” “menu,” “menu_items,” and “items” have their own key(s) not involving RESTAURANT_ID is due to the following issues. Every review needs its own REVIEW_ID because multiple reviews can be submitted for the same restaurant. Thus, in the “reviews_directory” table, RESTAURANT_ID and REVIEW_ID are a composite key where RESTAURANT_ID is not null and not unique and REVIEW_ID is not null and IS unique. Moreover, the same restaurant can have multiple menus, as seen by the “menu” table. Since different menus have different items on them, setting up “menu_items” to be a non-null composite key was the best option. Multiple restaurants can have the same item. Also restaurants have more than one item in their menu so any given RESTAURANT_ID will show up multiple times in this table because we are pairing RESTAURANT_ID with ITEM_ID. The information regarding the ITEM_ID is stored in the “items” table which gives basic information one what the item is, such as its name and price.

The functionality of the database depends on the user interface’s ability to capture the relevant fields and display information. There is a home screen that allows the user to navigate around the user interface by clicking one of the buttons.



The user is then prompted to perform the specified actions after clicking on one of the buttons, such as “Restaurant Search” in this case.



The user is then required to fill out the form and submit their results. A pop-up window prompts that the operation has been completed.

The backend relational database is MySQL which connects to a Java UI framework, called Netbeans using JDBC. The JDBC connector allows a connection between Netbeans and the MySQL database server setup on our local internet in our house. The queries utilize transactions within Java rather than through the query call itself. JDBC does not allow “START TRANSACTION;” and “COMMIT;” to be used in SQL calls to the database. Thus, a rollback feature has been implemented as an exception if the query fails to modify the tables within the database. The Netbeans IDE allows drag-and-drop capabilities to make simple UIs, such as text fields, labels, buttons, and drop down menus. Once the user pushes one of the buttons, for example: to add a restaurant in the database, the JDBC allows a query to be performed on the MySQL database. Thus, all the features of the user interface have been implemented using SQL queries and JDBC which allows field to be populated.

Netbeans needed some set up to allow the database connection. The MySQL Connector for Java needed to be important as a library and the connection credentials needed to be validated through the IDE. Moreover, handling SELECT Statements that would return views of the database and its subsequent tables was more involved. The rs2xml.jar file imported as a library handled some of the tricky tasks of configuring the table display such as managing what columns and information would populate. Moreover, a dynamically updating Java Table was used to display any table from the database with the help of Object Oriented programming and the rs2xml.jar file.

Furthermore, this Restaurant database and user interface appeared to be more of a challenging task than usual due to tricky setup, configuration, and somewhat difficult queries to create. However, with the amount of progress made on this project, we are proud of what we have created.