Sam Brandt

Professor Kalpakis

CMSC 461

December 15, 2021

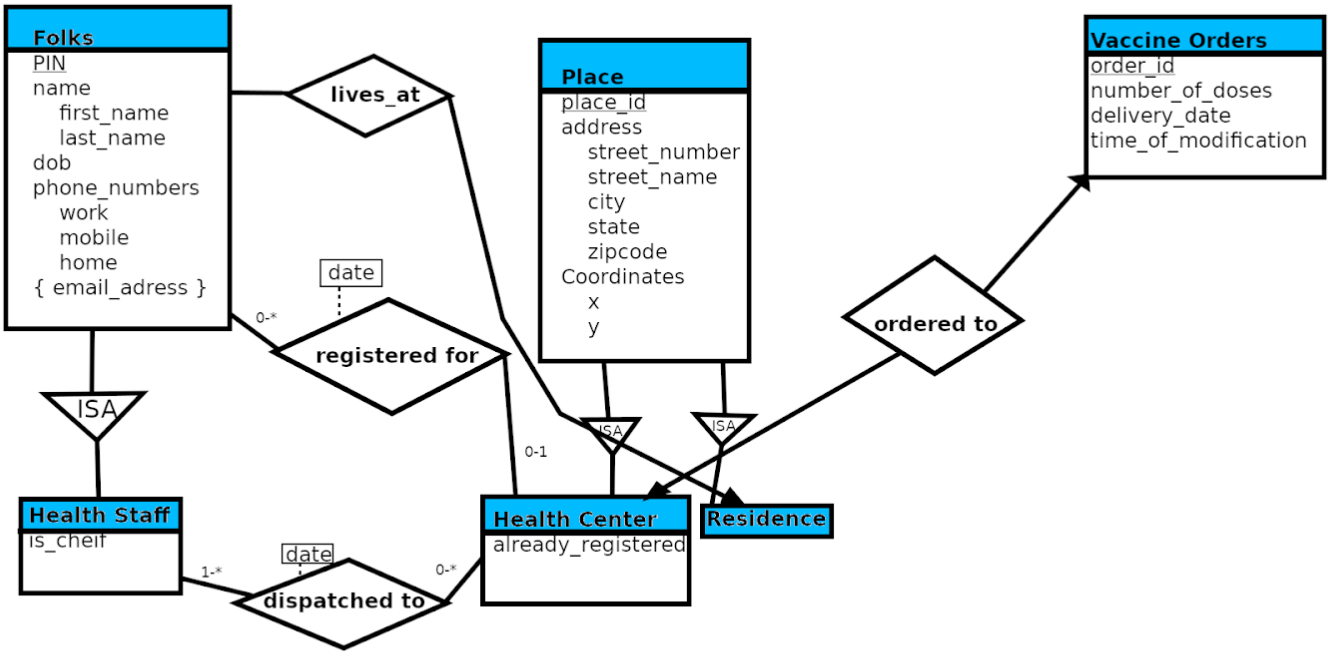
Project 1 Final Report

Introduction

In this project, I was tasked with designing a database to facilitate vaccine administration to the members of a fictional country, writing SQL scripts to create it, and designing an interface application to demonstrate its utility. A common theme throughout the process were my initial design ideas in the earlier stages ending up being proved either useless or detrimental to the final application, which is exemplary of the importance of constant critical thinking and testing stages in application development.

Phase B

Entity Relationship Diagram I created to represented the described data and relationships:



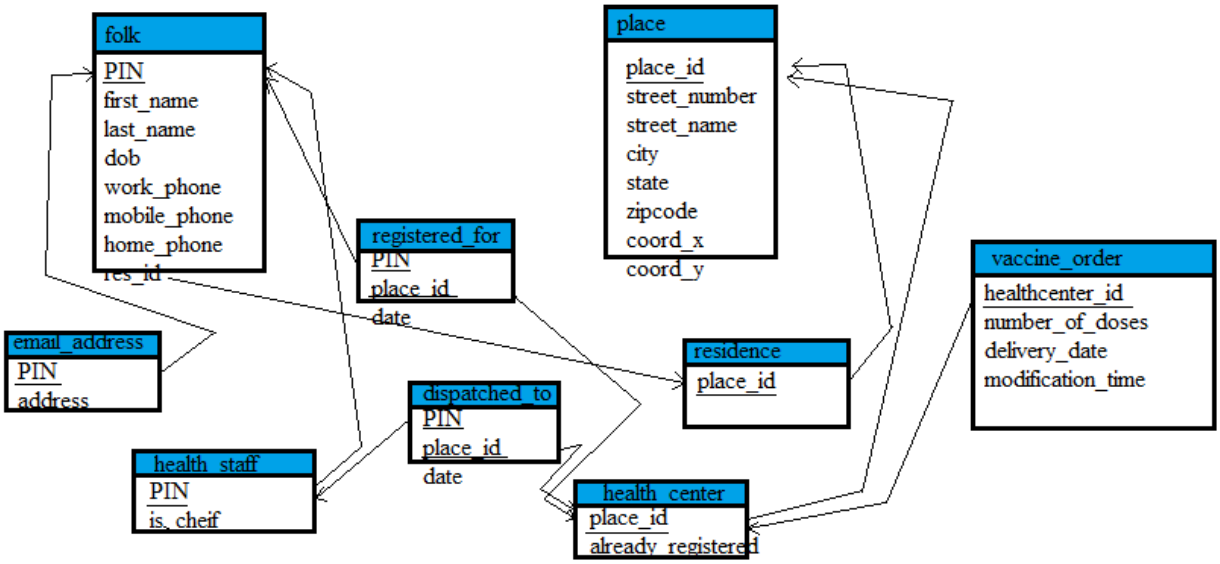
Primary keys are underlined. Relationship cardinality is represented by either directed/undirected lines or ranges written near the lines. In “Folk” the PIN must be 16 digits and the phone numbers must be 10 digits. In “Place” the address and coordinates must be unique.

I chose this design approach because I recognized that much of the data could be represented as relationships between those three major entity types and there seems to be a very natural way that a lot of the data falls into relationship categories and subtype categories. Also there are some fields like the date of a vaccine appointment that are easier to think about as being attributes of a relationship rather than attributes of an entity in that relationship.

Phase C

In this phase, I mapped my above database concept onto a relational model. The first part of the above model I mapped was the Place entity, which was simple to adapt since it had no dependencies on any other part of the database. The resulting ‘place’ table looks exactly the same as the one above. Then, I mapped the Residence and Health Center entities by creating respective tables with a unique foreign key link to the ‘place_id’ in the place table to represent the ISA relationship. After that, I mapped the Folk entity by creating two tables, one for all of the non email data, and one with the email data and a non unique foreign key link to the ‘PIN’ in the folk table to allow the possibility of a variable number of email addresses per folk. I mapped the Health Staff entity by creating a table with a unique foreign key link to the ‘PIN’ in the folk table to, again, represent the ISA relationship. I mapped the lives_at relationship by including a non unique foreign key link in the folk table to the ‘place_id’ in the residence table. I mapped the registered_for and dispatched_to relationship by creating respective tables with two foreign key links to the respective folk and place subclass tables that the relationships are supposed to be relating. Finally, I combined the Vaccine Orders entity and the order_to relationship into one table, since each vaccine order is only associated with one health center.

An illustration of the result of this mapping is shown below:



Phase D

For this phase, I created SQL scripts to create, populate, and delete instances of the above tables. I implemented the tables exactly as described above, and made use of the "unique" keyword for both the 'res_id' foreign key in the folk table and the foreign keys that I used to represent ISA relationships. I put the create statements in order of dependency. I also included a statement at the top of the create script that would create a new database if one didn't already exist. I put the drop statements in reverse order of the create statements so that I wouldn't run into the error caused by another table's foreign key being deleted. In the script for populating the table, I used simple numbers for the id fields to make them easier to track.

Phases E and F

For this phase, I wrote SQL query code and SQL transaction code to fulfill the runtime needs of the application, and then I created a Jupyter Notebook application to function as a user interface for the database. The first issue I noticed with my database design from the prior stages was that I never set up any functionality for foreign keys to change if primary keys were changed, which I fixed by adding "on update cascade" to the foreign key declarations. I also had to fix a mistake I made in the loadAll.sql file where I wrongly thought that a user could only be registered for the next day, which made many of the date-based queries pretty useless. For some of the SQL code that involves "Today's Date," I wanted to use the CURDATE() function in MySQL but I realized this makes testing the passage of time difficult so I instead made it it so a user can manually set the "current" date and time in one of the cells in my Jupyter notebook. Writing all 9 of those queries, plus my own, for other parts of my application, proved that your intuition can easily lead you astray with a declarative language like SQL, but by the end I feel like I mastered both the art of thinking about SQL the right way, and also quickly deducing the source of bugs.

The application that I created has a cell that allows a Wonderland user to "Log in" with their PIN, which allows a real life tester of the application to enter any PIN and pretend to be that fictional wonderland user. Once that cell is run, the other cells will keep track of that PIN for later needs so the user will not have to enter it in again. If a tester wants to change which wonderland user they are pretending to be, they simply can just run the cell again at any time and change their PIN. I also added a cell that allows the chief and only the chief to assign new health staff dispatchments for the next day. All the other cells implement specific functional requirements described in the assignment document, and can be run in any order as many times as a user likes.

I created four indices, one on the date in ‘registerered_for’ because it gets searched through a lot, especially in the queries that are searching for registrations in a specific month. I also created indices on the city, state, and coordinates in ‘place’ since it is common for a query to want to isolate which places are in a city, state, or have or don’t have specific coordinates.

Executive Summary

Overall the process of completing this project involved:

- Thinking about the relationships between the different types of data that would be necessary to track in a database with this purpose.
- Deciding what data can be thought of as part of entities, what data can be thought of as part of relationships, what relationships can be represented with extra fields in existing tables or require an entirely separate table for representation, and what entities need multiple tables for representation.
- Considering the restraints that are necessary on the tables to make sure that the data in the database will always make sense.
- Writing scripts to create the tables in an instance of MySQL Workbench.
- Designing a Jupyter Notebook application to allow a user to interface with the database in a useful way for the purpose of vaccine administration management.

Conclusion

This project taught me a lot about coding and the process of software development and it also taught me that I actually like the kind of thinking that goes into both designing databases and also writing SQL code which makes me feel more open to careers of that type in the future.