Final Project: Marielle Riveros

INTRODUCTION

A few years ago I did a co-op at a non-profit where one of my tasks was updating the Resource Directory as a 296-page Word document to make sure the information for each organization was accurate. After that was completed, the Resource Directory was used to find organizations that were best suited to assist clients. It was an asset to have all the information in a central location, but the format of a long Word document was sub-optimal to best locate information quickly and efficiently. For this project I wanted to design a better solution.

This database stores information on different non-profit organizations in Massachusetts. The information was gathered from the Massachusetts Department of Mental Health "Multicultural Populations Resource Directory" (accessible here). The most recent version I could find is from 2014, so I had to independently validate the information before adding it to the database.

Upon accessing the database using the designed application, the system administrator is able to search for organizations based on criteria such as organization name, services offered, and languages offered. The administrator is also able to add new organizations and modify or delete existing ones. Languages and service types can also be added independently as criteria which multiple organizations can use. If an organization is removed these underlying criteria remain, though the link between the organization and the criteria is removed The system administrator is also able to add languages and services to the database and delete them as long as they are not associated with any organization. Addresses can also be added to or deleted from organizations, and can be modified by the administrator.

Based on my experiences working with this data set at a non-profit, the most important role for the use of this application would be that of a central system administrator – one person who is able to not only search the organizations, but also modify *all* the organizations in the database. This project is designed for that role.

I. README

- Application: Massachusetts Resource Directory Management Tool
- Creator: Marielle Riveros
- To Run:
    - Install PyMySQL, stdiomask, cryptography (if not already installed)
    - Terminal Command: python3 mass_charities.py
    - Log in with MySQL credentials
- Languages: Python (application code), MySQL (database)
- Python version: Python 3.7
- Preferred Environment: Terminal (Mac)
- Python modules: sys, textwrap, re
- Additional packages:
    - pymysql
        - Install*: pip install pymysql
        - Used for: communicating with MySQL
        - More information: https://pypi.org/project/PyMySQL/ - installation
    - stdiomask
        - Install*: pip install stdiomask

- Used for: masking passwords in the terminal
- More information: https://pypi.org/project/stdiomask/

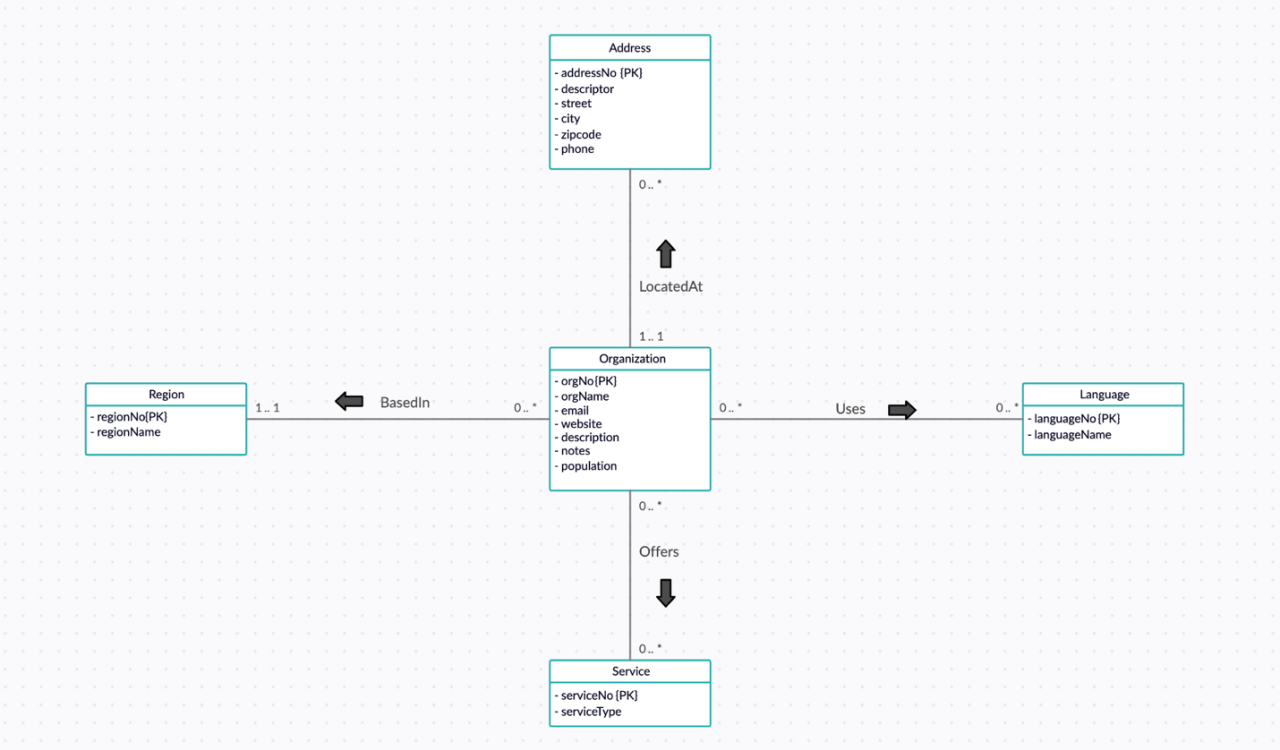
- cryptography
  - Install*: pip install cryptography
  - Used for: password authentication (triggered by Python)
  - More information: https://pypi.org/project/cryptography/
- * If these commands do not work, preface them with 'python3 -m' (no quotes; only if Python 3 is being used)

II. Technical Specifications

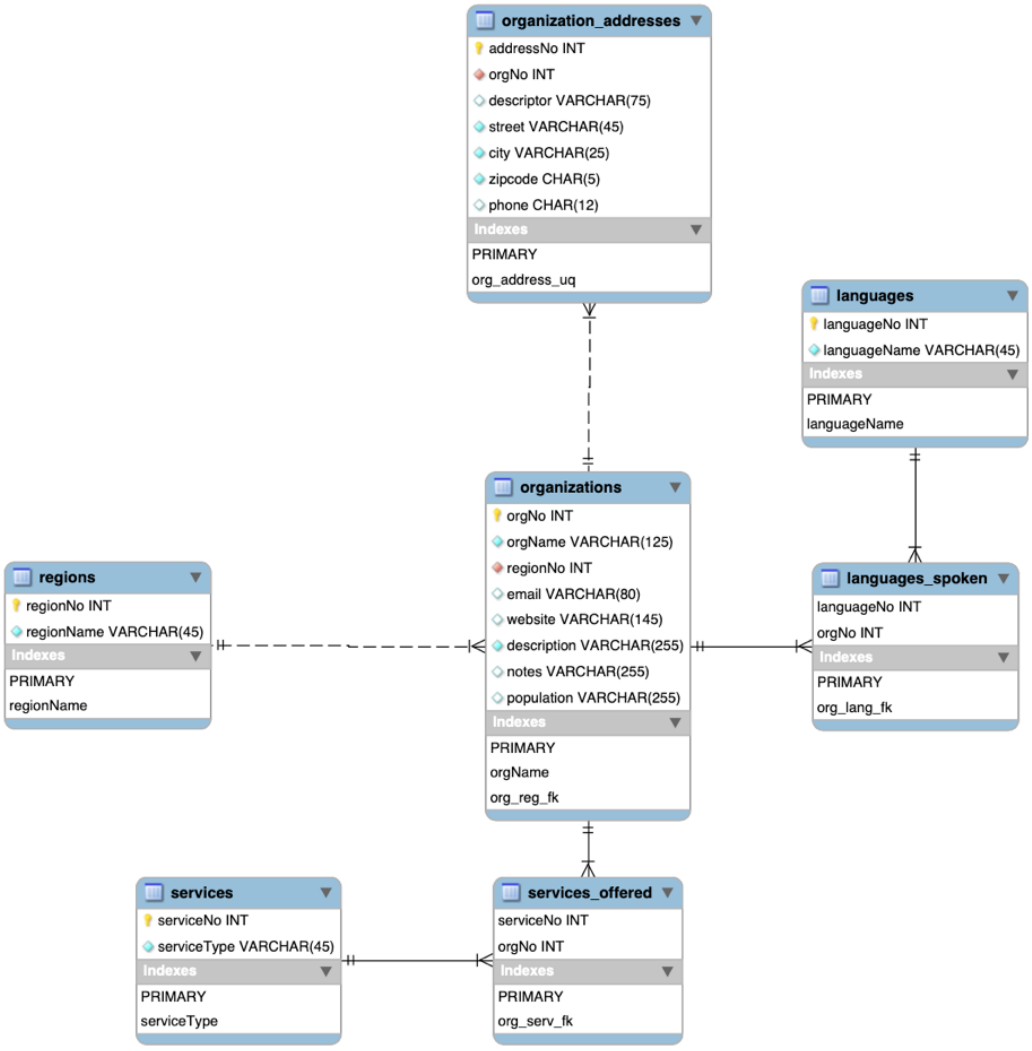
- Machine used: Mac (OS: Catalina)
- Software used: PyCharm (application code), SQL Workbench (database)
- Languages: Python (3.7), MySQL
- Preferred environment: Terminal (Mac)
- I have no reason to believe that the project will not work on Windows, but I do not have the facilities to test this and cannot confirm
- Python modules: sys, textwrap, re
- Additional packages: pymysql, stdiomask (cryptography dependency also needs to be installed but is not used explicitly in the project; it seems to be triggered by Python)

III. Conceptual and Logical Design

Conce*ptual Design:*



Logical Design:



IV. User Flow

Please see page 7.

V. Lessons Learned

A. Technical expertise gained

There were many instances in which I gained technical expertise through this project. Much of this expertise came from working with the communication between the database (MySQL) side and the client application (Python) side. One example was when I was trying to figure out MySQL error handling. In addition to learning how to actually handle errors in MySQL stored procedures, I learned how to balance using the messages generated by MySQL error handlers with writing my own conditionals to validate information on either the MySQL or the Python side. Messages generated by MySQL error handlers can be extremely valuable, but like other error handling protocols they can

lack specificity (the same error can be prompted by issues with multiple inputs, for example). I found that while in some cases it was perfectly fine to use the messages I generated by error handlers to convey information to the user, it was often better to use input validation on the client side to prevent the errors or to use conditionals within a stored procedure to return error messages. In many cases I was able to display more specific error messages using methods other than the error handlers. Furthermore, input validation was extremely important because (in addition to it generally being better to prevent errors), I was able to ensure that values used as arguments in a stored procedure matched the parameter constraints of the procedure – this error could not be handled by the stored procedure as it would prevent the procedure from even running.

In using PyMySQL, the interface used to connect the database to Python, I learned how to commit changes to a database so that they remain persistent. I believe my IDE (PyCharm) was committing changes that were made during an aborted program session, so at first I did not realize that having a commit action at the end of the program was not sufficient to ensure persistence – I thought that the database management system was ensuring that the changes were committed. After learning that this was not the case, I found that I needed to commit the connection in the client code after every successful transaction.

Some of the technical expertise I gained from the the Python side was a better understanding of static methods and experience in creating regular expressions and using them for pattern matching. In creating a project of this size, I also had reason to think more about modularization of my code and the abstraction of functions.

B. Insights

In my original design for this database, I included Target Population as a different entity type with a many to many relationship with organizations. The Target Population would also have been a parent class with sub-classes corresponding to different different types of attributes that were often referenced in defining a population (age, gender, sexual orientation, and immigrant status). To determine the sub-classes I looked at the data and tried to pick out the most common criteria included in the listed populations. I did this because I felt that it was the best way to be able to search by population type, which is something that I would have found useful in my own work with the data. However, as the project progressed I realized that the target populations listed by organizations were even more individualized than I originally thought, often including specificity to the level of countries of origin. This makes it much harder to generalize, and as a result I decided to include target population as an attribute of each organization.

Something else I realized about the data is that each organization can have multiple addresses, rather than just one. I was also surprised by the number of organizations that do not list an email, even on their websites – and some organizations do not even list websites. The resource directory is from 2014, which is old enough to mean some of the data is out of date but obviously not so old that websites and emails were uncommon as methods of information dissemination. While this is not something that has an enormous impact on the database design (other than the fact that neither website nor email can have a “Not Null” constraint), it was certainly interesting to me.

I also had some insights beyond those regarding data domain. Though I did not fully realize it while creating the application, in this project I used elements of different programming paradigms. Though I primarily used procedural programming, there was at

least one point in the program that is based in object-oriented design. I did this simply because I felt that there while there was a particular case where it was extremely beneficial to have an object with shared attributes, in other cases it made more sense to me to organize functions into different files that could be imported as needed and pass the necessary data as parameters. This was primarily distinguished by the OOD class corresponding to a real-world object with multiple attributes (the creation of an organization), while in other cases the program was more directly performing actions on the database. I do not know if it is considered "bad practice" to incorporate different paradigms in the same program, but it did give me a greater understanding of how each paradigm has its own benefits.

I learned quite a bit about the process of working on a larger project and gained a much greater appreciation for the importance of planning and organization, as much from my failings as from my successes. One example of this is the user flow model. Before working on the database or on the application I did think about how the user would interact with the database and what types of operations would be necessary for that particular user, but I made a mistake in writing this out in bullet points rather than creating an initial flow diagram. Without seeing the flow and having that visual diagram to refer to, it was much easier for me to get caught up in working on the functionality and focus less on the flow of user interactions. In creating the flow diagram for this report, I realized that I might have designed it more simply by having a central menu that maps to operations for organizations, languages, and services, rather than having a central menu that groups the create, search, update, and delete operations for all different items together by operation type. With better planning and organization in general I could also have abstracted a number of my functions from the beginning.

C. Realized/contemplated alternative designs, approaches

As stated above, I originally had an alternative design for how to deal with the target population for an organization in the database. On the application side, I originally intended to make a GUI using Tkinter but I was unable to complete this before the deadline. If I had the GUI functional, I intended to allow the user to search by multiple fields (including languages, services, population) rather than only by a single field.

D. Code not working

I tested my program extensively and did not find any code that was not working. However, there are aspects of the code that I would like to improve. Specifically, I would like to make some of the functions relating to language and service more abstract, rather than having highly similar functions in different modules. I did start doing this when I realized that it was a viable option, but I was unable to do this for many of the functions due to time constraints. Furthermore, I was hesitant to undertake a massive code refactoring task close to the project deadline due to the risk of introducing errors.

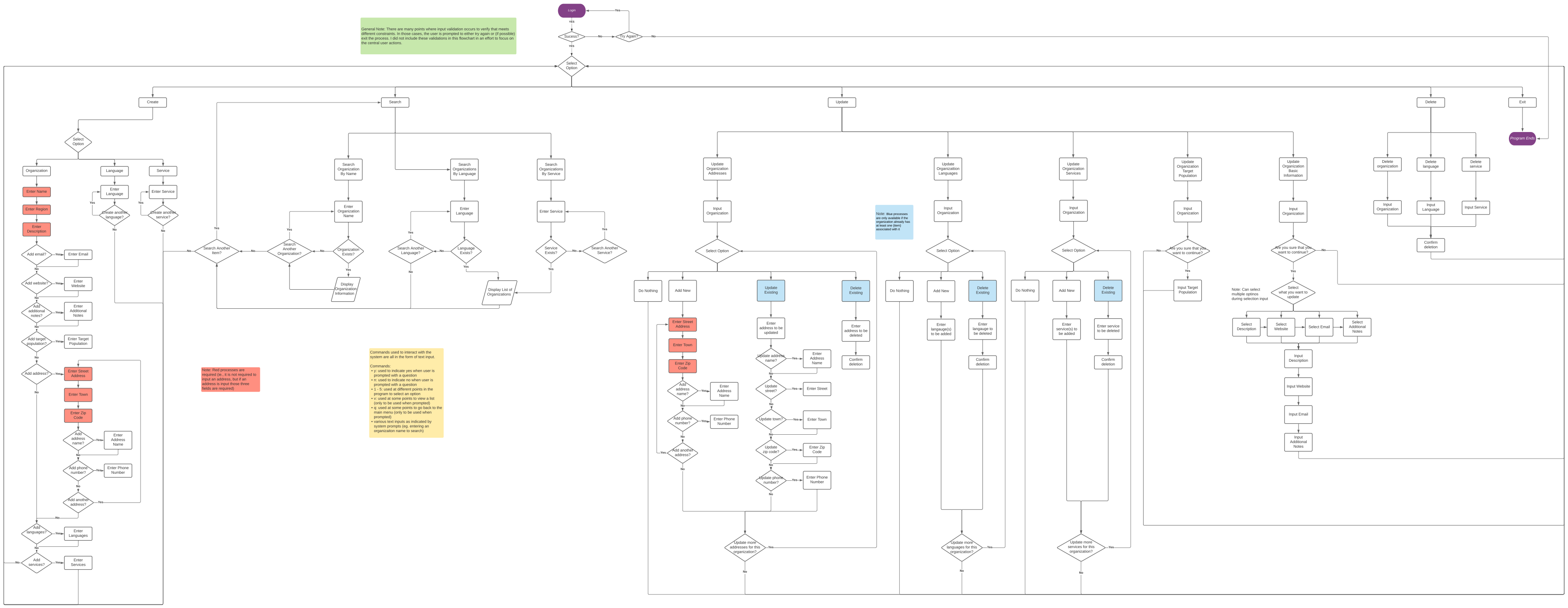
VI. Future work

A. Planned uses of the database

I currently do not have any planned uses for the database. I recently spoke to the organization that I used to work for and I believe that they are currently using some third-party databases rather than the static directory that they had previously. However, I do intend to use this database and application as a way to continue to improve my coding and learn new skills and approaches.

B. Potential areas for added functionality

There are multiple areas for added functionality that I would like to address. I would like to implement a GUI to make usage easier and more streamlined. To go along with this I intend to figure out dynamic SQL to allow for searches on multiple criteria (at minimum, language(s) and service(s) at the same time). One area of functionality that I did not really explore but would like to look into is the ability to have multiple users and user levels within the database. For example, I would like to allow for users who are able to use the search functionality but not the create, update, or delete operations. These users would also be able to store some sort of list of "favorite" organizations.


Login
Success?
Try Again?
Select Option
General Note: There are many points where input validation occurs to verify that meets different constraints. In those cases, the user is prompted to either try again or (if possible) exit the process. I did not include these validations in this flowchart in an effort to focus on the central user actions.
Create
Search
Update
Delete
Exit
Program Ends
Note: Red processes are required (ie., it is not required to input an address, but if an address is input those three fields are required)
Commands used to interact with the system are all in the form of text input.
Commands:
• y: used to indicate yes when user is prompted with a question
• n: used to indicate no when user is prompted with a question
• 1 - 5: used at different points in the program to select an option
• v: used at some points to view a list (only to be used when prompted)
• q: used at some points to go back to the main menu (only to be used when prompted)
• various text inputs as indicated by system prompts (eg. entering an organizaiton name to search)
Note: Blue processes are only available if the organization already has at least one (item) associated with it