Introduction

We are tasked at creating a database application that manages data for an upcoming election in the (made up) country of Wonderland. The database contain various tables, but they can be categorized into 6 major groups:

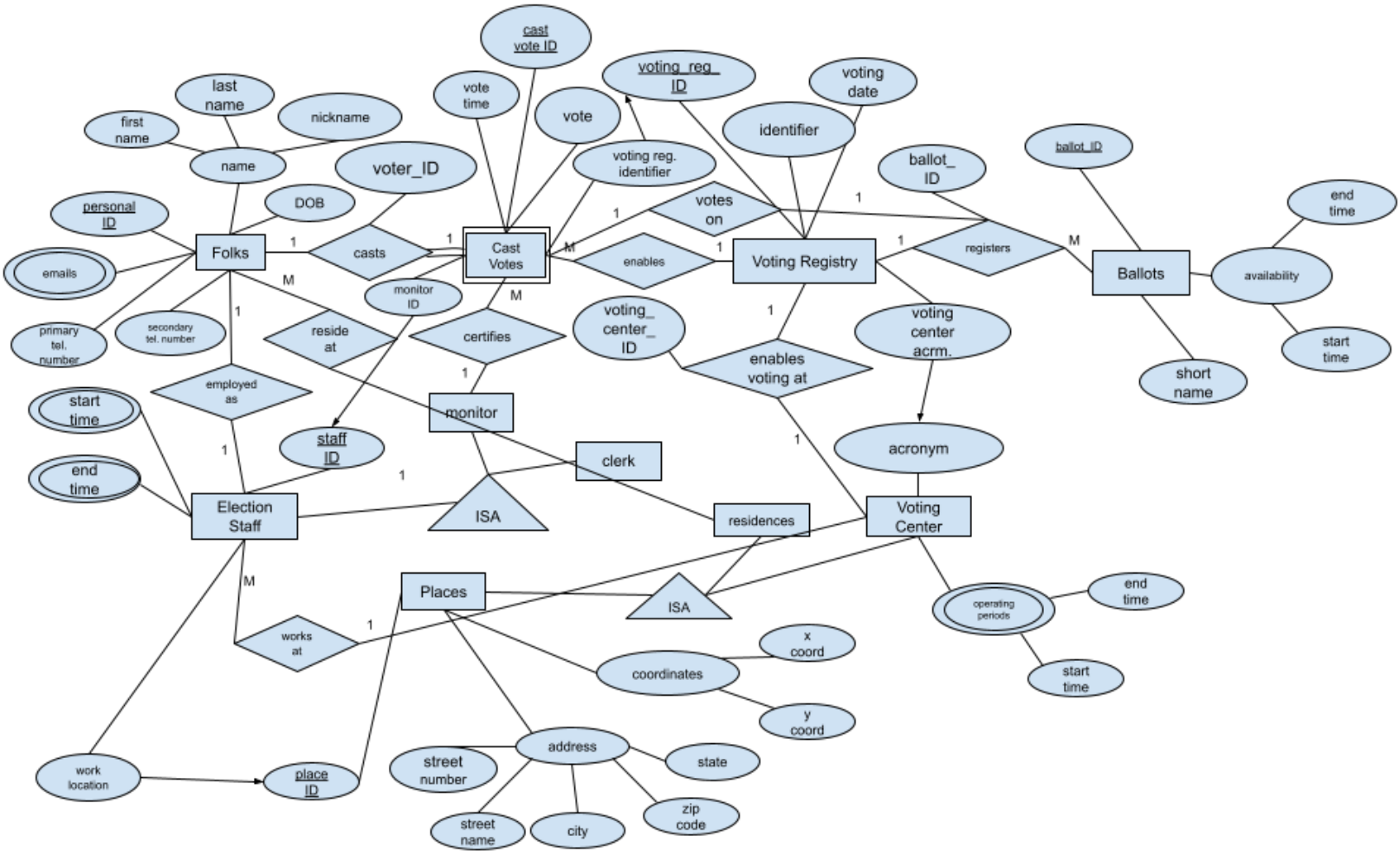
1. Folks (the people eligible to vote)
2. Elections Staff (election workers)
3. Places (representing either a person's residence or voting center)
4. Ballots (the election people vote on)
5. Voting Registry (a confirmation that someone is allowed to vote)
6. Cast Votes (the tally to a specific ballot)

The project presents various complexities. For instance, multivalued and compound attributes must be implemented to allow (for instance) folks having multiple emails or a way to represent their names respectively. Not only that, but election staff can be categorized into 2 groups: clerks and monitors. It's assumed that election monitors have all of a clerk's responsibilities in addition to confirming a folk's casted vote. As mentioned earlier, places can either represent a folk's residence or the physical location of where people vote (voting centers). Each place is mapped onto a 2D Cartesian coordinate along with a derived distance from the capital city, Megapolis, located at the origin (0, 0). Ballots represent the actual elections people vote on where the 4 options are "YES", "NO", "ABSTAIN", or NULL. Adding a voting registry contains various checks such as validating that the folk's ID exists, the ballot they want to vote for exists, the voting date is within the chosen voting center's operating periods, etc. Cast votes on the other hand is fairly straightforward compared to the rest. It'll just represent the tallied votes for its respective ballot. The only thing to keep in mind is that all information relating to a casted vote (e.g. the voter) has to exist or at least not conflict with other information (for data integrity reasons).

Project Organization

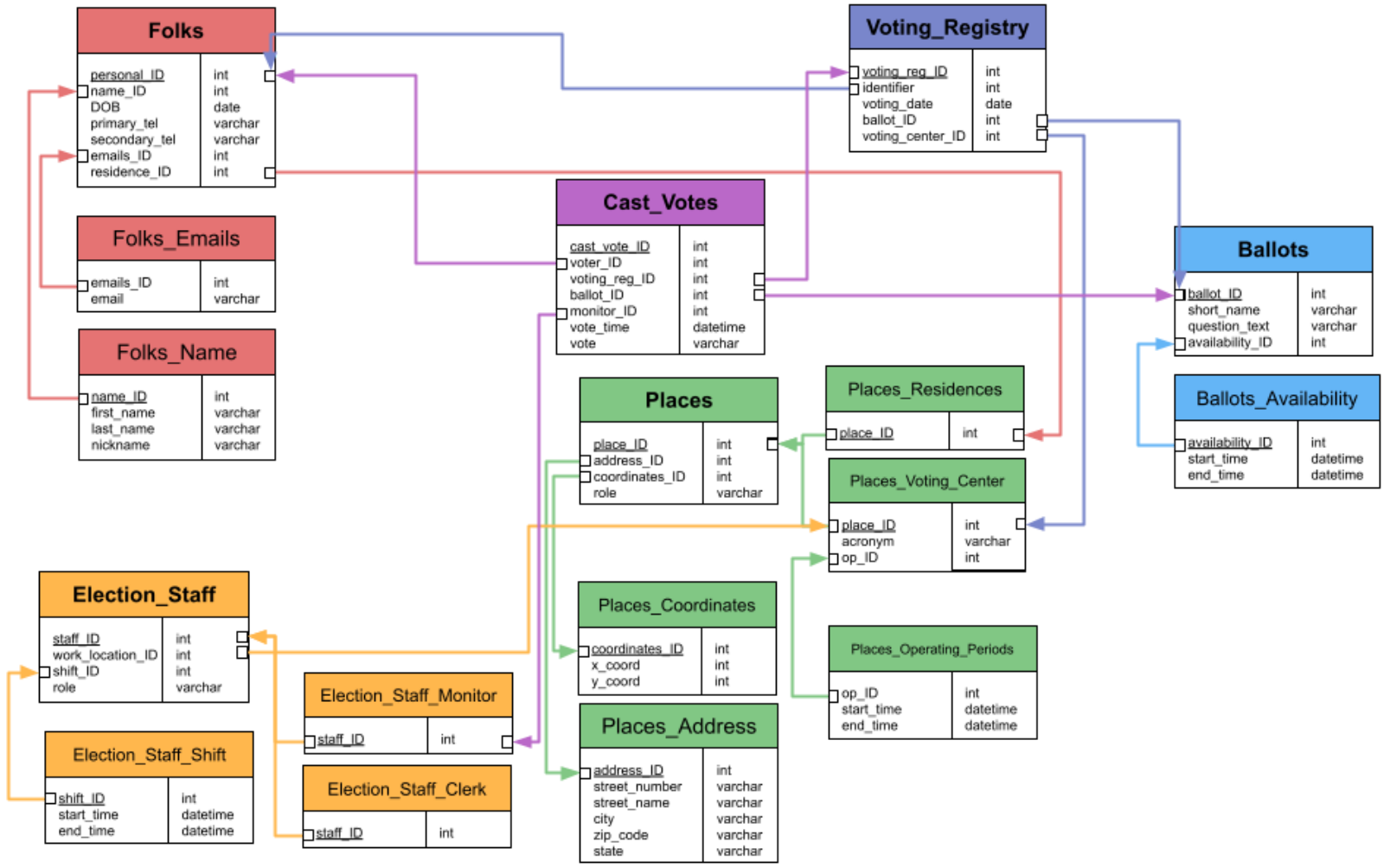
Organizing a database application often involves an entity relationship (ER) diagram and a relational data model. Below are the models I've developed.

ER Diagram



The entity relationship diagram is more focused on displaying the attributes and relationships for all tables in the database.

Relational Data Model



The relational data model on the other hand focuses more on how specific attributes from 1 table relate to 1 from another table (foreign key references).

Files Developed

To accomplish the election database model, I've developed various SQL/SQLite files along with a Jupyter Notebooks files (with a functions.py library) to run scripts.

- createAll.sql
  - SQL scripts to create all the tables necessary for this project
  - Implements an index on a Folk's DOB (line 68)
- loadAll.sql
  - SQL scripts to load user-defined data for the election database
- dropAll.sql
  - SQL scripts to drop all tables

    - elections.ipynb contains a cell at the very bottom that allows you to run this file. You'll have to confirm that you actually want to drop all files by entering "YES"
- transaction.sql
    - includes a code snip it for how I implemented a SQLite version of a transaction. Whenever a registered folk casts a vote, a lot of checks has to be performed for data integrity. If any validations/checks fails, then the casted vote is rejected.
- queryAll.sql
    - Contains code for all query-related portions of the project. Some questions require user input, though. This file isn't meant to be run on its own, but rather just contains code snip its from elections.ipynb.
- elections.ipynb
    - The Jupyter Notebook file to perform all activities and queries/reports outlined in the project document.
    - The file *should* work if you were to run each cell sequentially, once. Any cell that involves deleting either a table or row should be approached with caution since it might try to delete data that no longer exists. As for the other cells (e.x. involving only inserting data), they *should* able to be executed more than once. However, try to **stick to executing each cell sequentially, once to simplify grading**.
- functions.py
    - A user-defined library to simplify the code in elections.ipynb. Essentially puts repeated tasks here to reduce code clutter.

Accomplishments and Pitfalls

For the most part, the project does work. Loading SQL files, inserting, deleting, and altering data in tables, and querying all work (as far as I'm aware). I intended on the elections.ipynb to be run sequentially, once. I've found that it doesn't run into an issue after running the file like that multiple times.

As for pitfalls, they're mostly room for improvement. I feel like I could've expanded more into implementing an effective indexing system as well as a more robust user defined function. Both features *do* exist in the project. However, they're rather basic and were only implemented because they were required. If I had been given more time, I would expand further into these 2 areas. Right now, I only have an index for a folk's DOB. I did this because we might want to get useful information about voter age demographics. At the moment, it just seems very bare bones

Regardless of the pitfalls, I'm still very proud of how the project turned out. I feel like I learned a lot in implementing a database application. I learned 2 programming languages: SQL and SQLite. I'm not entirely sure if I enjoyed doing the project, but I can say that I liked the learning aspect of it.