Project Report – Database Design and SQL Implementation

**Introduction**

Last week I wrote the UI for the hypothetical IMPACT volunteer organization’s website, including a form to input data. This week I am designing and implementing a database which will eventually interact with that UI. The requirements include creating entities for officers, volunteers, and events, as well as deciding on how these entities will interact in the database schema. The organization will need a simple way to track information regarding its employees, office locations, volunteers, and event locations and details. Officers and Volunteers will have contact information stored on the server, as well as any assigned Events and/or office locations. Events will be tracked as well as the Volunteers and Officers assigned to them.

I approached the problem by creating an initial sketch of what my database will look like, then by making a series of trial-and-error adjustments until I was confident with the design. I turned that design into an ER Diagram, then translated that into a Relational Schema. I then used that to write a SQL script which is used by a MySQL server to generate the database. I will further describe this process in the remainder of this report, after which I will include a quick user manual to get this database running on a MySQL server.

**Work Done**

***Entities and Relationships***

I began by making entities for Officer, Event, Volunteer, and Office. I decided that because Officer entities need to keep track of when they start and end work at an office, making an Office entity would be convenient. In addition, it allows the organization to keep track of data related to the office itself, and not necessarily related to any officers in charge of it, such as the Office’s allotted budget.

My Entity relationship (ER) diagram initially included an entity for Address and a quaternary relationship which every other entity belonged to. I did this because my background in Object Oriented Programming told me that I was repeating information by including an address in almost every entity, and that I should encapsulate that behavior somehow. This quickly became needlessly complex in my implementation of a quaternary relationship, however, and so I scrapped that idea. I decided that the small scope of this project does not require such considerations, and that I should focus on functionality first and foremost. Thus, every entity has its own address information as a part of its table in my final design.

While designing the volunteer entity, I recognized a need for some sort of leadership among volunteers. In the example of an ER Diagram from the lecture notes, an Employee entity is given a foreign key assignment to its own entity, illustrating that some employees manage other employees. I decided to use this design for my volunteer entity: some volunteers (senior volunteers) will be in a position to lead groups of other volunteers (junior volunteers). This means that within the organization itself, officers will not need to accompany every group of volunteers in the field, while there can still exist some chain of accountability and leadership.

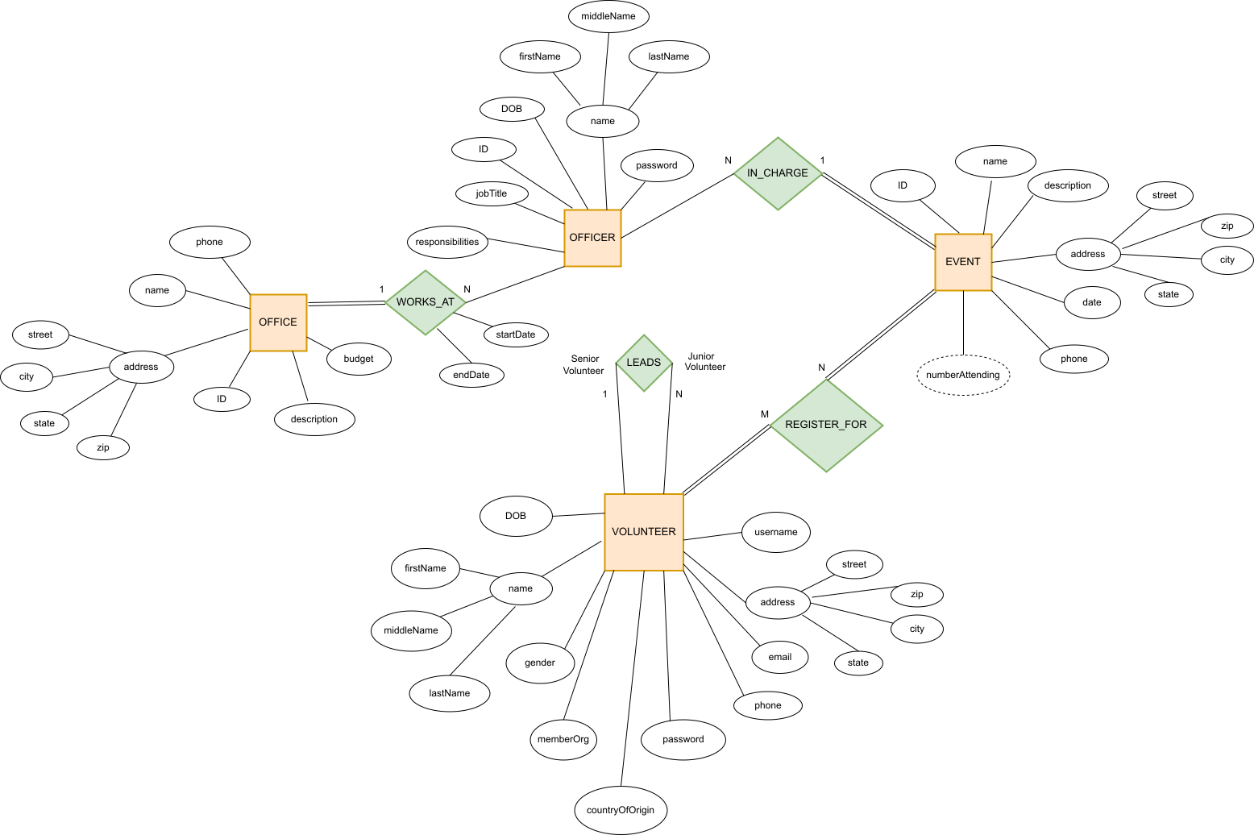
***Cardinality and Participation***

I next decided on relationship cardinality. One Office may have one or more Officers working at it, and so the Works\_At relationship is one-to-many. Likewise, any number of Officers may be assigned to a single event, but an event needs at least one Officer to be in charge. Similarly, one volunteer in a “Senior Volunteer” role may lead a group of “junior” volunteers. Finally, I decided that a volunteer can volunteer for any number of events, and that of course these events can have any number of volunteers, and so Registers\_For is a many-to-many relationship.

In terms of participation in these relationships, I decided an Office must have an officer, an Event must have an Officer, A Volunteer must register for an Event, and that an Event cannot exist without at least one Volunteer to attend.

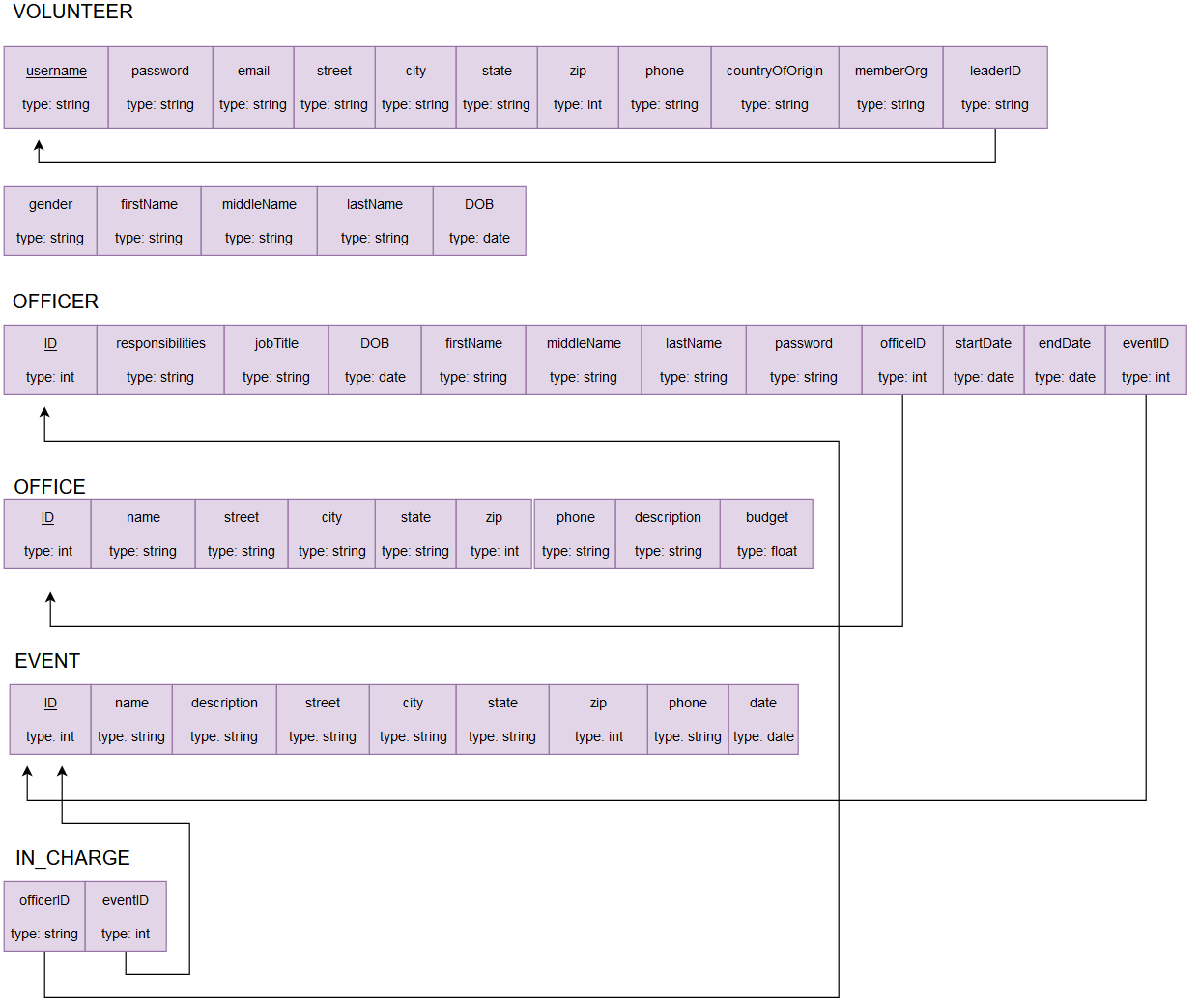
***Constructing the ER Diagram***

My diagrams were designed using [draw.io](https://app.diagrams.net/), after finding easy templates to work with. My ER Diagram up to this point was a sketch on paper with pencil, but my final ER Diagram is below:



***Translation of ER Diagram to Relational Schema***

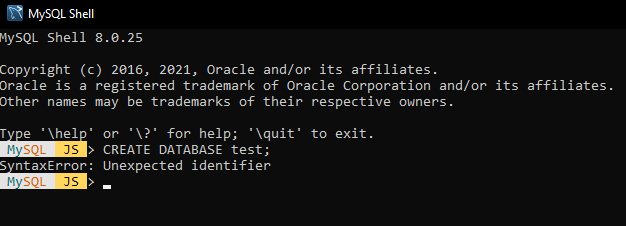
Creating a Relational Schema was fairly straightforward. Following the algorithm given in the video lecture, my process was as follows. I first mapped regular entity types to tables, which are Office, Officer, Event, and Volunteer. I don’t have any weak entity types in this design, nor do I have any 1:1 relationship types, and so I next mapped 1:N relationship types using foreign keys. These are Works\_At, In\_Charge, and Leads. Finally I mapped Registers\_For as its own table, as it is the only M:N relationship type in my ER Diagram. I then connected foreign keys with arrows on the diagram and decided on data types. My final Relational Schema is below:



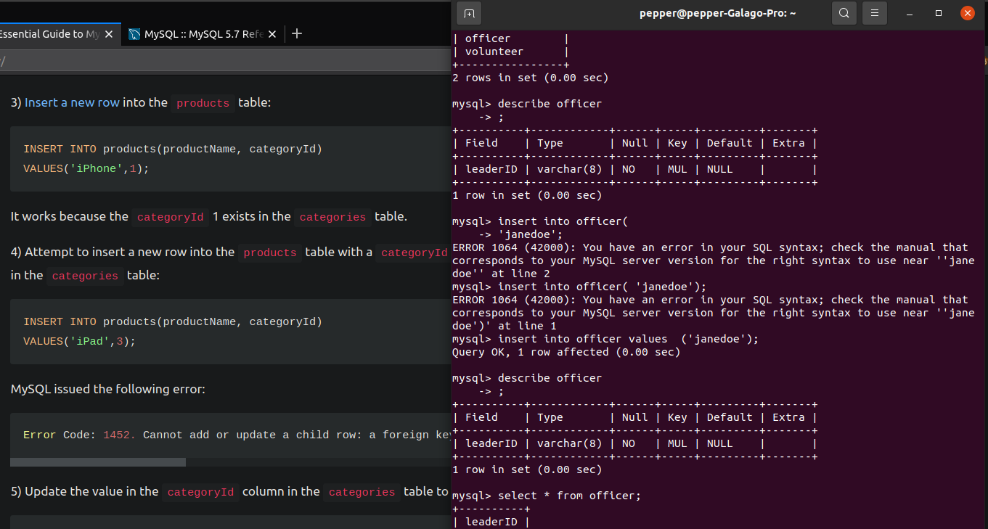
***Translating into SQL***

At this point I began the process of translating my design into a SQL script. This again was a process of trial and error, and I began by using a test database to write my initial scripts. I used code examples from [Derek Banas’s Youtube tutorial](https://youtu.be/yPu6qV5byu4) on MySQL, which was an immense help in recalling how to work with MySQL queries.

I ran into trouble getting MySQL to work on a Windows command line, however. It seemed that my commands were not being recognized even within the MySQL command line environment:



I decided to move my work to my Linux machine to see if the CLI would work on Bash, and lo and behold it worked perfectly fine. I began working in Linux and learning how I would design my database in SQL, including learning how MySQL handles foreign keys, as well as what Constraints are:



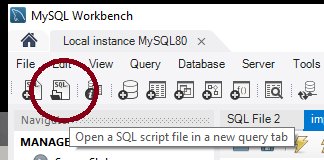
I decided that, now that I was working on two different machines on this project, I should set up version control to make my workflow easier to manage, and so I set up a [Github repository](https://github.com/Pepper37/IFT458Project-webServer) for my project. This repository will also include the most current version of any future deliverables and assignments.

Without much more trouble I was able to write my script file containing all the CREATE queries necessary to make the database, including working foreign keys where necessary. The final script creates a database called impact, sets the server to use it, then creates all the tables outlined in the Relational Schema. Foreign key constraints are set to cascade on update and delete so that there hopefully aren’t any references left hanging when changes are made to the database. I successfully tested this behavior briefly but not extensively.

My final bit of trouble lied with running the MySQL server via MySQL Workbench on my Windows machine. Once again, Windows was being difficult while Linux had no problems. I had to uninstall MySQL Server and re-install and re-configure it to get the Server software to start on my Windows machine. In the end however, it works as expected.

**User Manual**

Within the attached zip file is a sql file called impactDatabase.sql. Extract that file, open MySQL Workbench, and connect to a MySQL server. With the server up and running, select “open a SQL script file in a new query tab”



Navigate to impactDatabase.sql and select it. Click on the lightning bolt icon for “Execute Script.” A database should now be created with tables for Volunteers, Officers, Offices, Events, and Registers\_For, as well as foreign key constraints handling relationships between tables.

**Conclusion**

Through the process of completing this deliverable I have designed an Entity-Relationship diagram given a specific problem description. I then went on to crate an outline of a relational database in the form of a Relational Schema, including data types. I used this schema to write the SQL code itself, which when run on a MySQL Server successfully creates the database according to my design.

I learned through this process the ease of creating a simple database, as every step seemed to consist of simply following an algorithm and thinking about the needs of the organization. Even the source code itself is relatively straightforward: just a series of CREATE queries.

In terms of writing SQL code, I learned about Constraints and how to reference foreign keys in MySQL. My challenges faced were mostly in getting the necessary software tools working in Windows, and as a result I learned that I have even more reason to prefer a Linux-based development environment.

My database design is very simple, and I can certainly use more complexity to manage more difficult problem descriptions. I’m sure in a real-world scenario, the ER diagram would be much more complex, and the database would need to be designed with optimization in mind, rather than pure functionality which was my approach here. I believe complexity and scalability are two fundamental ways that this database design can be improved.