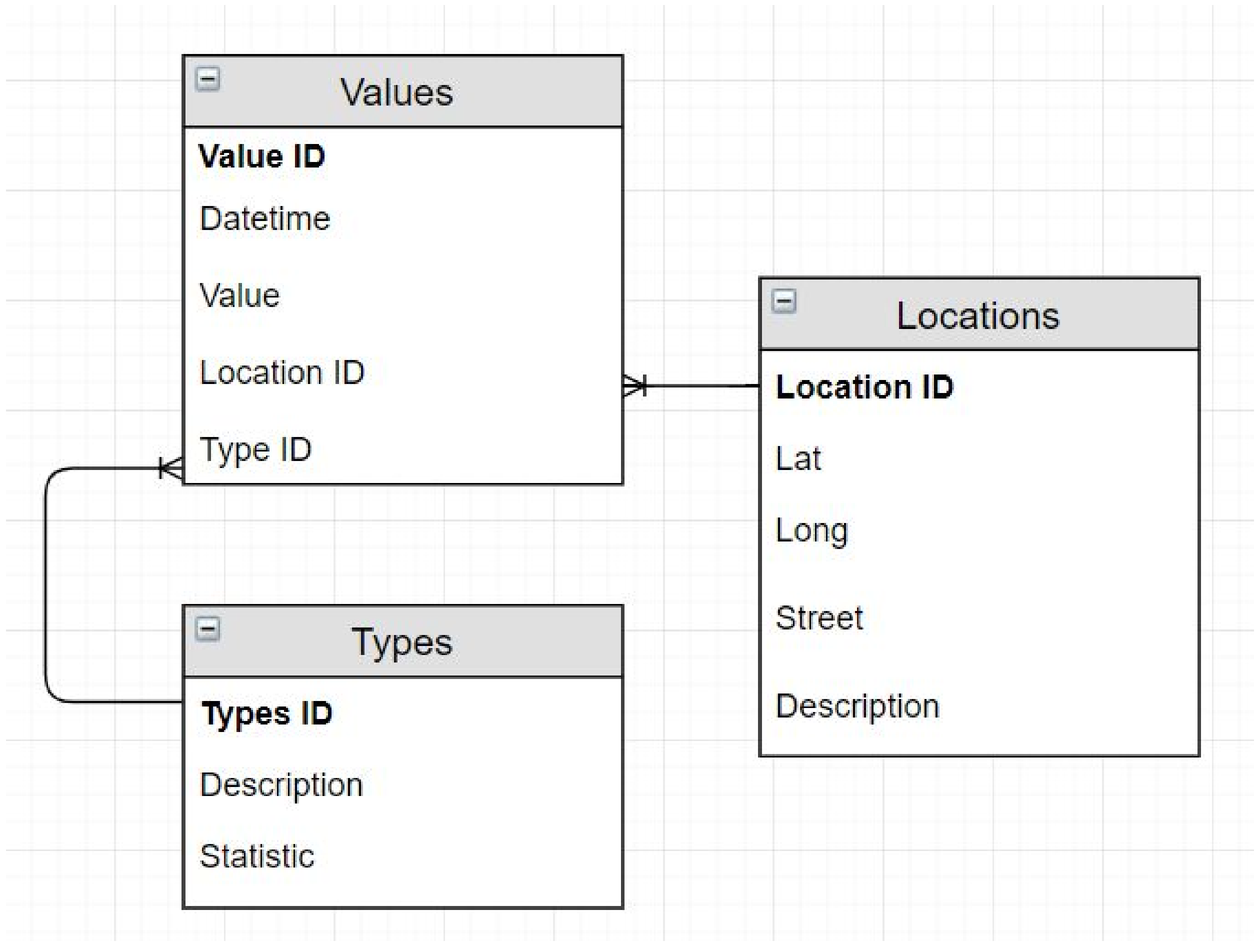
<u>Design around data points</u>



Description:
Value Table: Each individual data point will be entered in the value table along with its timestamp, Location ID, and Type ID. This is essentially each column in the current for_rf_model data set on top of each other to make one long column. This will have millions of entries.

Locations Table: This has a one-to-many relationship with the values table. There are a couple thousand locations in the data set. Each location will have a location ID, a street name, and a description attached to it. The description and the street name does not yet exist in the data set.

Types: This has a one-to-many relationship with the values table. There are currently 15 types in the data set. Each type will have one of the following descriptions: Rainfall, Tide Level, Wind Speed, Groundwater, and Topographic. Each type will also have a statistic that is the same as the column header in the current data set such as rh or max15.

Pros

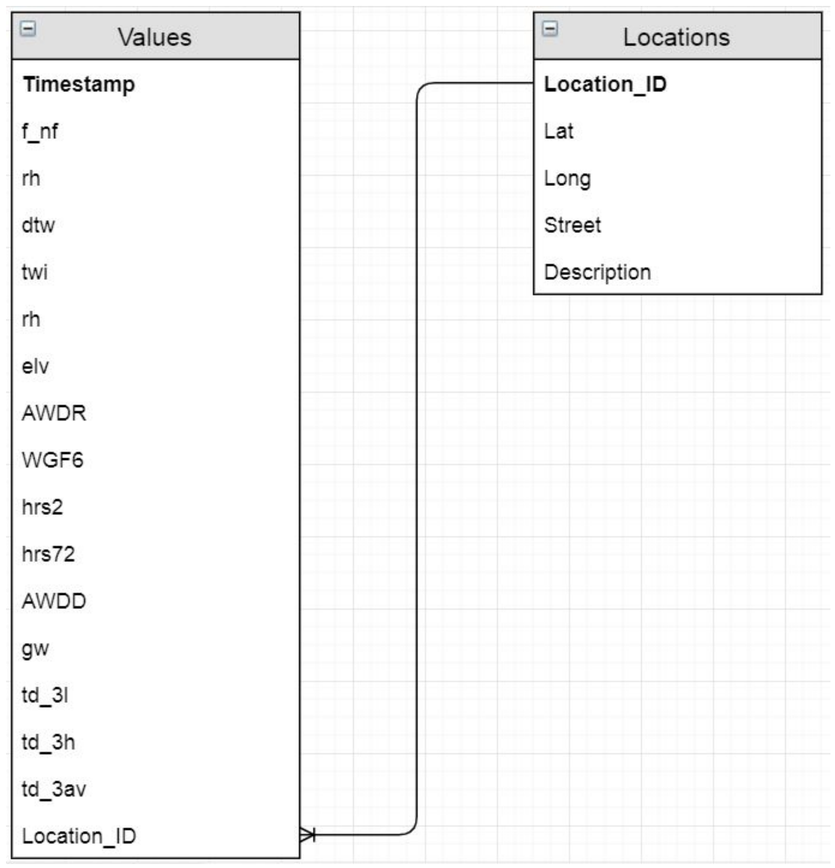
- More flexible should database expand later down the road

- Easy to query general information, does not require knowledge of the entire database for querying
- Easy to collect and add more individual data points
- Allows for description and statistics for data types

Cons

- Far more rows total, that could limit processing capabilities and implementation
- Requires more code in Python first to process the raw data

<u>Design around columns</u>



Description:

Values Table: Each data type will be its own column in a table, along with a primary key and a location ID. This table could be broken up into separate table by value type (i.e. Rainfall, Groundwater, Tide)

Locations Table: This has a one to many relationship with the values table. There are a couple thousand locations in the data set. Each location will have a location ID, a street name, and a description attached to it. The description and the street name does not yet exist in the data set.

Pros

- Simple design

- Simple implementation
- Requires little work to create Faria’s current input csv for the random forest model

Cons

- Limited application/flexibility later down the road
- Requires more knowledge about the contained data for querying (must know exactly what the columns are)
- Little flexibility on data type description

Final Decision

Ultimately, it was decided that the database set up for design around data points was the best choice. We believe that the flexibility of the design will allow for greater implementation further down the road. Although the larger volume of rows will use more memory and computational capacity, it is important for a wide variety of users to be able to query the data according to their individual needs. We hope that the database generated will have the potential for further expansions than the current Random Forest Model, and can easily accommodate data generated by our colleagues’ microcontroller without adjusting the design. The design around the columns is situationally appropriate for Faria’s current Random Forest Model. While this makes for simple implementation, it limits usability should the model be expanded upon and built out further.

Next Steps

The next steps is to create a code in Python that will take the for_rf_model csv and import it into the database according to the design around data points. Ideally, we could design an application that can take any csv and import it into the database, with the column headers being different type statistics, assuming all future datasets contain a lat/long column in the same format. Another important step is to convert the current “event_date” data into a timestamp.