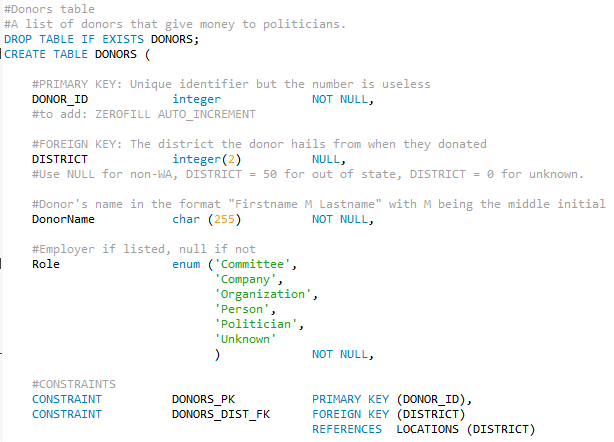
Donations to Votes Executive Summary

**Objective**: Donations to votes is a utility that our team has created to better allow individuals and groups alike to see where their local lawmakers’ money is coming from, and how that could be influencing how their representatives are voting on new bills. The project shows what bills representatives have voted for or against, what the overall topic of the bill was, companies and individuals that have donated to representatives’ campaigns, and how much those donations are. This is to simply show voters important information that could, or could not, be influencing how their representatives vote because of the pull from donors. We do not wish to influence anybody or attempt to push an opinion on anyone but, simply provide a tool that anyone can access that displays this information clearly and in an easily readable manner.

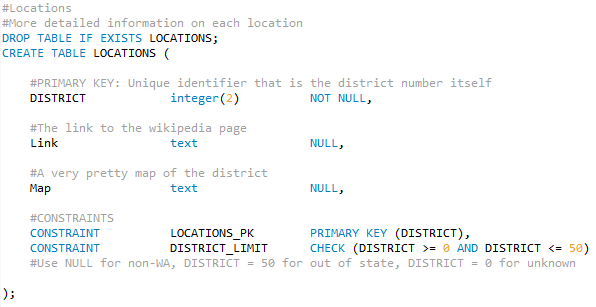
**Professional Summary**: The project began with the idea of creating a tool that would show correlations between interest groups, companies, and individuals that donate to campaigns and the representatives voting patterns while in office. The first obstacle the team had to conquer was where to get the data we needed, and how wide we wanted to throw the net. The team decided for simplicity sake we would only cover Washington State representatives in the House and Senate. This was because the information was available through a state website already, just in a very unorganized and hard to understand layout. This was a very direct link to the raw data we needed as it was available for all to download and use which gave the team many options for sorting through the data to find correlations and ways that votes could be linked to donor’s interests. Once we secured the necessary data, the team began organizing the data into tables which could be sorted through more easily by algorithms once ported into a database. To achieve this the team decided to create seven separate tables which would interact well with one another. These tables are a donations table, a keyword and bills table, and a votes table.



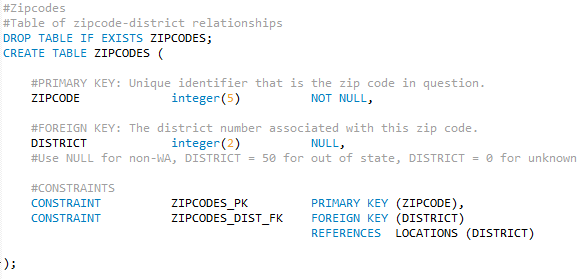
The first table is the donors table. This table is designed to identify an entity and communicate that information to other tables. Each donor is assigned a donor number which is given by our team simply for back end identification. The donor table also has the district number that the donor belongs to, if it applies. Additionally, each donor carries a name, which could be a person’s name, company name, group name, etc., as well as a role identifier that defines whether that donor is a person, group, company, etc. The role identifier is assigned through and algorithm that detects patterns in naming convention to determine what kind of role this donor has.



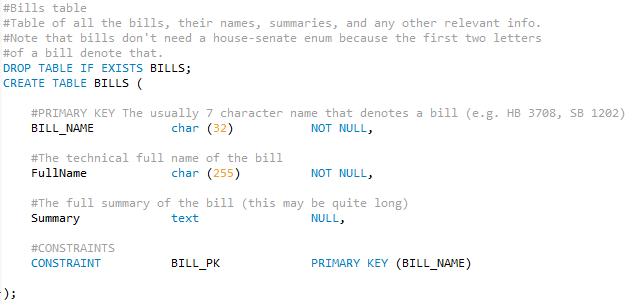
The next table is the politicians table which behaves similarly to the donors table in the manner of how it is used to define a single entity. Each politician is again assigned an arbitrary number that is used for identifying them in the back end system. This is followed by the district the politician belongs to, the politician’s name, the chamber in which the politician belongs to, i.e. House or Senate, the party the politician belongs to, i.e. Republican or Democrat, as well as the district that politician represents. There is also an embedded link for each politician that directs the user to the politicians own webpage which is also highly educational to users. Finally, the politicians table also contains an out of district donation index that has been calculated using an algorithm to evaluate the sources of a politician’s funds. Politicians who obtain a high amount of funds from districts other than their own, who seem to vote more for or against bills that could harm or help companies and interest groups, among other things are evaluated. Once the evaluation is complete the index is assigned to show an estimated level of corruption that the politician in question might have.



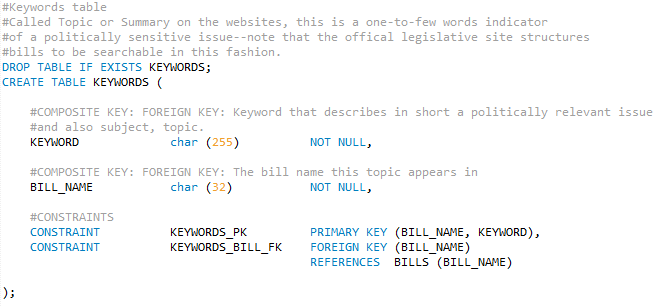
To aid users in gathering information we added a table called locations which contains the district numbers in the state of Washington with a link and a map that will direct users to another webpage with information on that district number. The linked page contains zip codes, a detailed map, among other potentially useful information.



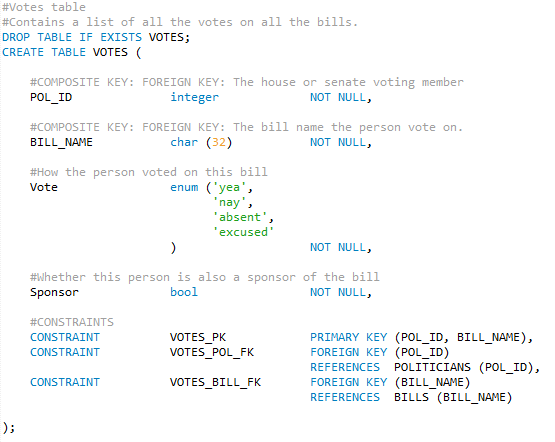
For enhanced utility for location information and for queries into the database there is also a table for zip codes. This table contains two pieces of information, one of which is the zip code that is in question and the other is the zip code that is correlated to that zip code. This table allows queries that display all the elected officials from the same district. This allows the flow of information to be more fluid and easier to read for the user. Additionally, this makes the queries much simpler when a user enters a zip code. For our teams purposes this expanded the usefulness of our tool while also allowing us to not reinvent the wheel by creating redundant data that is already well organized elsewhere.



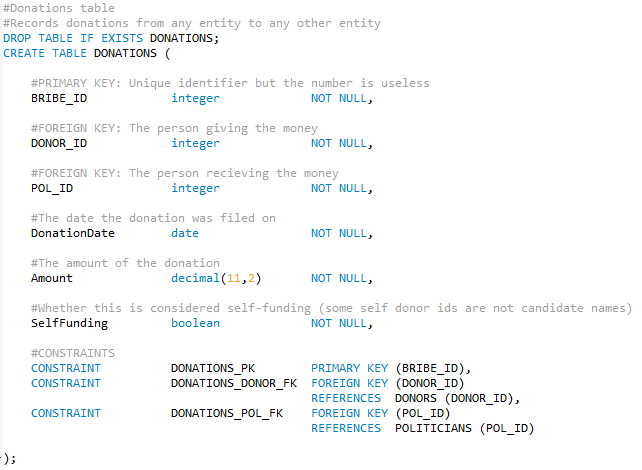
There is a bills table that contains the shortened names of a bill in their code and/or number format, the full length name of the bill, and a quick summary of what the bill covered. The summary is used to display to the user when inquiring about specifics of a bill, while the shortened bill code and/or number is also displayed, it is used in back end identification of a singular bill or cluster of bills. This again allows interconnectivity between tables and pages that link to one another, increasing the efficiency of the project.



In association with the bills table we established a keywords table that contains a bill name which is shared with the bills table as a unique identifier, and the keyword or category that bill is associated with. The keyword is again assigned through an algorithm we developed that searched for keywords in the full title of the bill and then assigns a keyword for its topic that is then used to make it easy for the user to identify what a bill was about quickly.



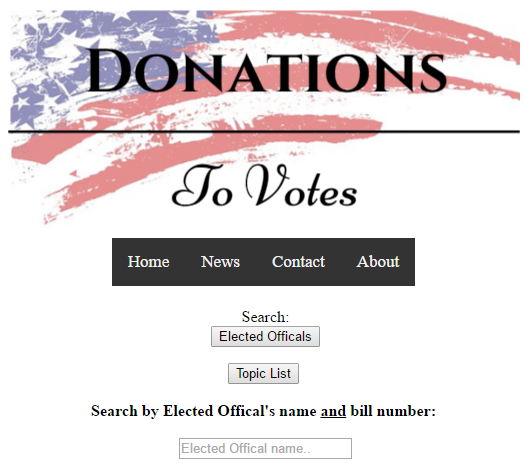
The votes table contains four pieces of information. It contains the unique politician identifier which was previously assigned, the bill code name which is shared with the bills table, whether or not that representative voted for or against the bill, and if that politician was a sponsor of the bill. Each line is one politician along with the for or against vote. However, the bill code and/or number stretches across multiple lines of the table. This is for full coverage of a bill and how every individual voted while also allowing connections to the bill itself.



The donations table contains information regarding a donation given to a representative, one donation on each line. Donations are assigned a bribe identifier which is used for back end identification. Each donation has the donor identifier shared with the donor table, the politician identifier shared with the politician table which identifies who the recipient was, the date (if applicable), the amount of the donation, and whether or not this donation was a self-fund donation from the politician themselves. The organization of these tables are important in ensuring proper connections between the tables to allow them to be created in a database with keys that point towards each other.

The connections that were laid out in the tables are incredibly important to the success of the utility. Being able to connect the keywords and bills table to the votes table would allow the tool to pull information, if requested, such as what the topic of the bill was when a user searches for their representative and how they voted. This connection works both ways so that the bill code and/or number can be queried when a user searches for a topic or category and clicks on the bill to see how all the representatives were cast. This two-way street is vital in creating an effective tool in which users can search, find, click, explore, and come full circle to where they began, allowing information to be given when a user finds something they want to learn more about.

All of these tables are filled in the database by a web crawler which pulls data from the state website once a day to update the database. Once the crawler pulls the information it needs it is then processed and split into CSV files which organize the information into a much more database friendly manner. Then those CSV files are processed into the database as the database fills the tables with the proper information needed. This database is connected to a website that our team created with a graphical user interface for users to interact with. Once the user has submitted the politician and bill they are looking for, the website will display the information requested by the user in addition to the other two politicians that belong in the same district as the politician searched.

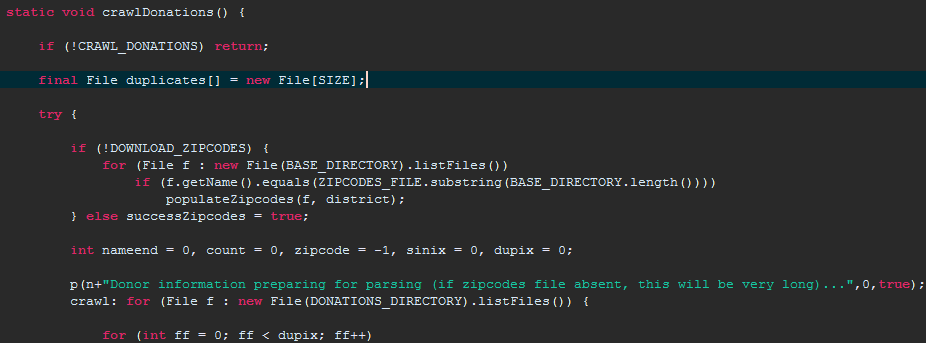


Many people do not know a specific bill to search for, and many want to see a broader range of bills with a common topic, which is exactly what the topic list selection allows. The topic list allows a user to view a list of topics with their associated bills that have been voted on. From there a user can be rerouted to the search method with the bill information being shared across pages to auto fill the bill search. Additionally, while in the topic search, users can click directly on a bill to view its summary and other details.

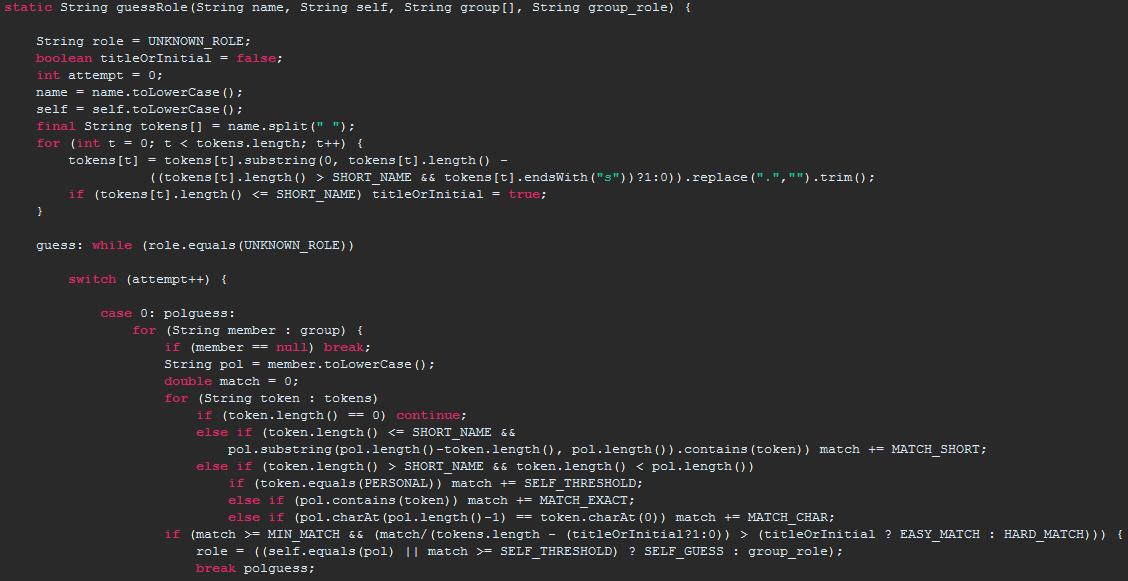
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Insert photo of final result here\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Technical Summary**:

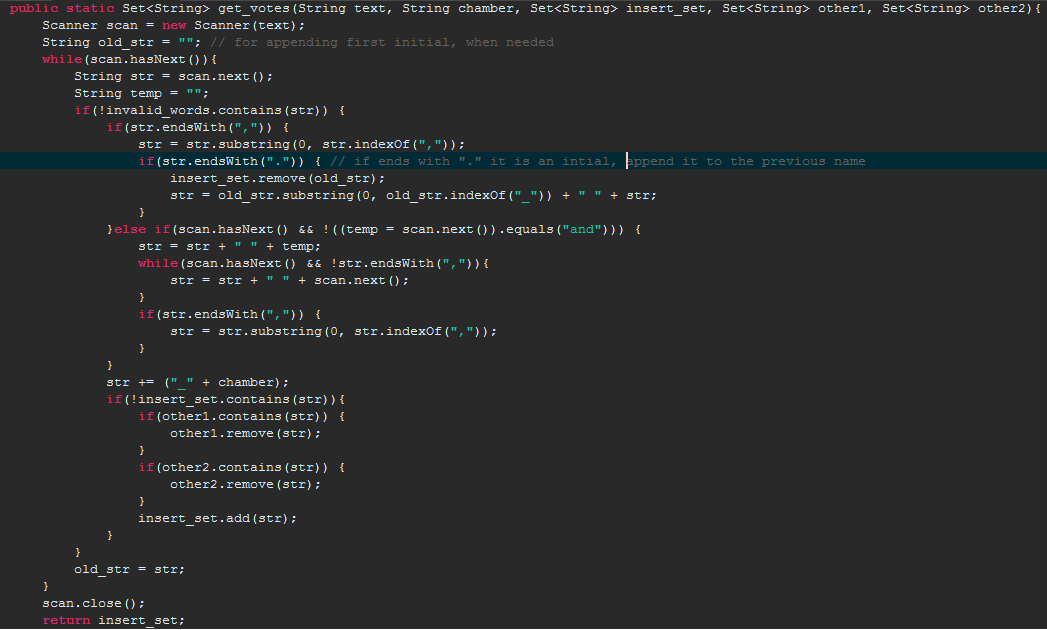
**Showcase**: To showcase the skills of our team we have selected several key pieces of what made our project a success. Between algorithms, database creations, webpage creations, SQL queries, and overall design choices, there are many obstacles that our team has overcome. We are proud to present our teams greatest achievements in product development.



This is the beginning of the crawler method contained inside the WACrawl class. This method was the actual crawler developed to use loops and Boolean variables to sort and pull the information about the politicians, donors, and the donations made the donors. Using the Boolean values this crawler searches the CSV file that we complied to automatically add information into the database. However, the crawler detects information that is not about a politician, donor, or donation, and prints out helpful statements when that information is found, and does not include that into the gathered information. That information is used in a different place, so it would have been inefficient and redundant to include that again.

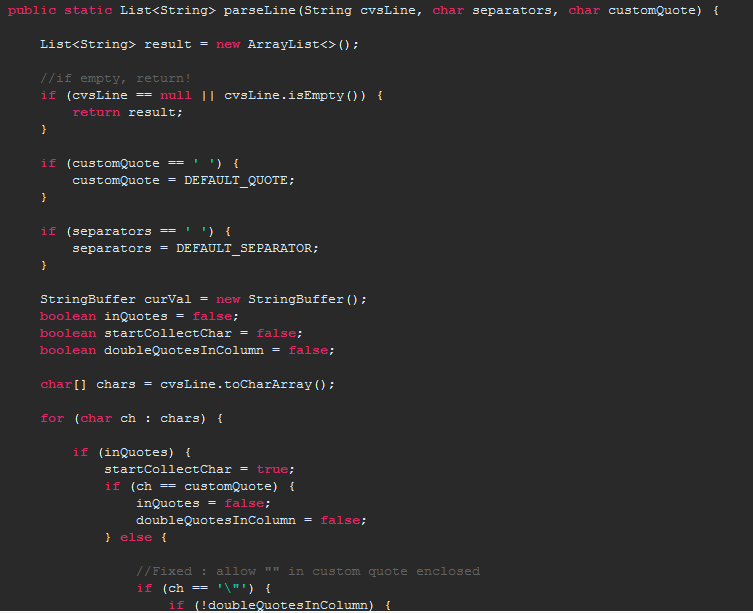


This is the nameGuess class which uses heuristic algorithms to attempt to determine the type of entity given that entity's name. Closely matching names will be identified as the same person (using the self-input). Group input allows the inputted name to be matched as a member of that group. This algorithm was extremely useful in being able to sort donors into more searchable groups. This effect was felt on both the back end searching, and the simplicity of searching from the user perspective.

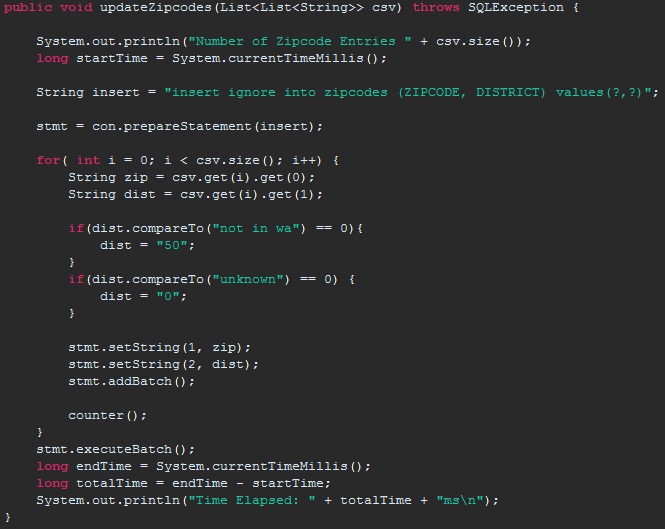


This is the java code that was developed to take the information we gathered about politicians from the internet and compress them into a readable and organized CSV file. It achieves this by cutting invalid strings from the list of voting officials and reformats names with first initials. This specific method returns a set of strings containing: Only officials voting separated by commas, text the parsed and cleaned string as a string of names separated by commas, the chamber of the elected official which is used to prevent legislators with the same names in different chambers from being marked as duplicates, a set of Y/N/O to insert the data into, another set of Y/N/O to check for duplicate/changed votes, the second of the remaining sets to check for duplicate/changed votes.

This code was very useful because it allowed us to take in almost any information off of the government website that we could need, and ensure it was sortable for our purposes, while also ensuring data integrity by checking for erroneous markings of duplicates. Data integrity is vital to provide accurate information to our end users, if our data is incorrectly processed then our entire project has failed.



This is the utility that was developed to parse each line of a CSV file that contained the voting patterns and information about bills and politicians. Again using techniques to implement detection of what information was being read, using Boolean values, it made it possible to evaluate one line at a time, and put the information where it belonged. This provided the team with a versatile tool that aided in keeping data integrity true, as well as not needing to worry as much about possible characters that could crash the program. There was a double quotation char issue that presented a challenge but was fixed as you can see by the comments above.



This is the database insertion code that took the crawled information which has been sorted and analyzed and populated that data into our database. It is a tool to insert values into a remote SQL server connection. Prepared Statements and small private batch sizes are used to combat SQL injection attacks, which is an important security measure to take when using internet connection database protocols. The above function is an updater for zip code data used in district information among others. A loop is performed over the input string to glean values for a prepared Statement, each line is added to the batch with the correct format, a check is performed to see if the internal class batch size has been hit, if so, the batch is executed to the server. There are more functions that follow the same logic to update the database once a day, or more depending on future need.