Let's Talk Politics…

CIS 550 Final Project

Nicole Profit | Jesse Berliner-Sachs | Brooke Behrbaum | Rohan Shah

Introduction and project goals

Our project was conceived after recognizing a very specific political problem and grew into an app to solve that problem as well as provide general information about the political landscape in the US.

Imagine there is an important vote about to happen on some congressional subcommittee that you are very passionate about. You want to try and persuade the representatives on that subcommittee to vote a certain way (either by contacting them, lobbying them, etc.) so you want to find the members of the committee who will be most susceptible to your persuasion. Presumably, those members on the committee who are in the closest re-election races will be most likely to listen to you. But how do you find which representative is in the closest re-election bid? That's where our app, Let's Talk Politics, comes in solving this issue and providing general information on the American Political System.

Our web app accomplishes this goal in 5 sections:

1. The home page includes an overview of the app and a very simple, easy-to-follow breakdown of the 4 main features of the website.
2. The "Committee Members" page solves the initial problem we recognized by finding members of a committee in the closest races. Here, the user is able to choose a committee from a dropdown of all congressional committees and then choose a subcommittee under that committee. The user will then see a) who on the committee is in the closest race b) who on the committee is least likely to be reelected and c) who are all the members of the committee and what is their contact information.
3. The "Reps Per State" page allows the user to view a listing of all of the states with the number of representatives per state. This allows the user to see which of the states have the most political representation and compare how their states stacks up to the rest of the nation.

4. The “Who’s Running” page allows the user to enter their (or someone else’s) state and congressional district to see all of the candidates running. Additionally, the user is able to see which of the candidates, if any, are incumbents (currently hold the office).
5. Lastly the “Tight Race Watch” page allows the user to find all races nationally which are within a specific margin/ For example, a user may select 2% to see a list of all of the races polling data that indicates that the race is within the margin, specified. Note that these are not vote share percentages but likelihood of winning as model by FiveThirtyEight, more info on this data source below. On the “Tight Race Watch” page, we limit the tight races to 10% or less because we figured anything more than that is no longer considered a tight race.

We designed our app to solve a specific problem in american politics and provide constituants easier access their representative. We feel our app accomplishes this and more in an elegant and intuitive platform that can help increase political participation and hold representatives more accountable to their districts.

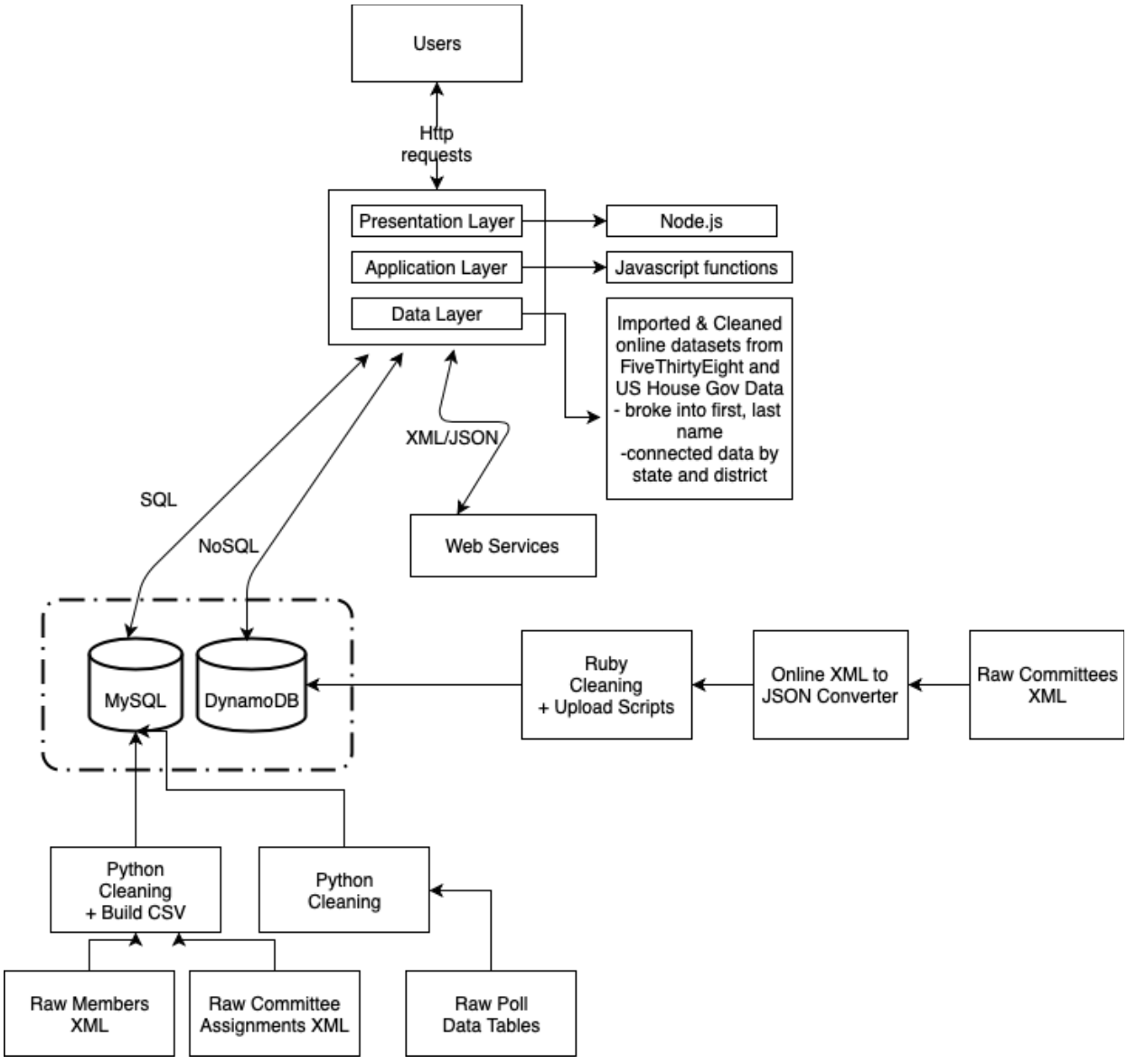
Data sources

Technologies Used

1. AWS DynamoDB (NoSQL Solution)
2. RazorSQL (app for working with MySQL databases)
3. AWS MySQL
4. HTML/CSS/Node

Diagram and description of system architecture

Note, the dotted line around MySQL and DynamoDB indicates that these databases are hosted on AWS.



How We Addressed Required Features

Requirement	How We Addressed It
Information successfully drawn from at least two large datasets, of at least two different types (e.g. CSV, HTML, XML, Word…)	We took data from 2 sources: US Gov House (Clerk of the US House Database) and FiveThirtyEight.com US House Provided 2 XML files. First describing the 435 members of the 114th US house, with their contact information, election info, and committee membership. Second provided information on committees including names, party membership breakdown and any subcommittees present. FiveThirtyEight provided 3 CSV files for every congressional election in the US including information on each candidates win percentage. They perform these calculations by aggregating polls according to 3 different models (named PollClassic, PollLite, and PollDeluxe). We input all of these models into our data app to allow users to select which model they would like calculations to be based on.
Data cleaning: frequently, online data is incorrect and incomplete. As you import the data, you should run simple scripts to check for errors.	Using Python and Ruby scripts (as well as some work in Excel) we cleaned the data, making sure names had no weird characters, districts numbers were the same across sources, win percentages were stored as ints, and more. Script attached in zip file folder 'cleaning_scripts'. We also transformed win percentage from a decimal between 0 and 1 to an integer between 0 and 100.
Entity resolution: Since you are importing data from multiple sources, you will need to "connect" the data. For example, in the Summer Olympic example the athlete name (and possibly country information) can be used to relate entries in different sources,	While originally we tried to merge data sets on member name we soon found this was impractical as there were too many discrepancies. We then decided to connect based on state and district which uniquely identifies the member representing or running

although names may be spelled differently in different sources.	in that congressional district. Additionally, committees and subcommittees had unique IDs.
Normalized relational schema with more than four relations.	See SQL DDL and Entity Relation Diagram below
More than one web page interacting with the database.	Our web app included the five pages described in the intro
Look and feel of web pages.	We sketched out how we wanted the pages to look like on a whiteboard, then we created a rough mockup, using Adobe XD, a design software (https://xd.adobe.com/spec/d78106c7-78a5-4352-5862-cbb98db3e6ed-a1b0/). We attempted to implement those designs, but ultimately went with manipulating the HTML and CSS by hand to create our own theme. As seen in our demo, we put a lot of effort into making the web application as intuitive to use as possible, especially because our overall goal was to decrease the difficulty of finding political information.
Complex SQL queries, i.e. with multiple joins, subqueries, aggregation etc. These can be in the application queries, or part of the schema design (e.g. triggers).	See SQL and AWS DynamoDB queries below. Our queries utilized subqueries, grouping, aggregation, and multiple joins.
Consideration of performance, including indexing and caching.	See Performance Evaluation Table below. We added an index on (Member.state, Member.district). We also added caching on the "Who's Running" page so if a member where to search for 'WA9', then 'WA10', and then 'WA9' again, the initial 'WA9' result would be cached.
Performance measurements, showing the effect of your optimizations on the running time of your queries and other operations.	See information in table below

Relational Schema

SQL DDL

```
CREATE TABLE Member (
        state VARCHAR(2),
        district TINYINT,
        firstname VARCHAR(255) NOT NULL,
        lastname VARCHAR(255) NOT NULL,
        middlename VARCHAR(255),
        party VARCHAR(1),
        caucus VARCHAR(1),
        state_fullname VARCHAR(30),
        office_building VARCHAR(10),
        office_room VARCHAR(10),
        office_zip INT,
        office_zip_suffix INT,
        phone VARCHAR(20),
        elected_date VARCHAR(25),
        PRIMARY KEY (state, district),
        CHECK (state IN ('AL',
                'AK','AZ','AR','CA','CO','CT','DE','DC','FL','GA','HI','ID','IL','IN','IA','KS','
                KY','LA','ME','MD','MA','MI','MN','MS','MO','MT','NE','NV','NH','NJ',
                'NM','NY','NC','ND','OH','OK','OR','PA','RI','SC','SD','TN','TX','UT','VI',
                'VT','VA','WA','WV','WI','WY')
        )
);

CREATE TABLE CommitteeAssignment (
        state VARCHAR(2),
        district TINYINT,
        committee_id VARCHAR(4),
        subcommittee VARCHAR(4) NOT NULL,
        PRIMARY KEY (state, district, committee_id, subcommittee),
        FOREIGN KEY (state, district) REFERENCES Member(state, district)
);

CREATE TABLE PollLite (
        state VARCHAR(2),
```

```
        district TINYINT,
        candidate_first VARCHAR(255),
        candidate_last VARCHAR(255),
        party VARCHAR(1),
        is_incumbent INTEGER(1),
        win_probability DECIMAL,
        expected_voteshare DECIMAL,
        p10_voteshare DECIMAL,
        p90_voteshare DECIMAL,
        PRIMARY KEY (state, district, candidate_first, candidate_last),
        FOREIGN KEY (state, district) REFERENCES Member(state, district)
);

CREATE TABLE PollDeluxe (
        state VARCHAR(2),
        district TINYINT,
        candidate_first VARCHAR(255),
        candidate_last VARCHAR(255),
        party VARCHAR(1),
        is_incumbent INTEGER(1),
        win_probability DECIMAL,
        expected_voteshare DECIMAL,
        p10_voteshare DECIMAL,
        p90_voteshare DECIMAL,
        PRIMARY KEY (state, district, candidate_first, candidate_last),
        FOREIGN KEY (state, district) REFERENCES Member(state, district)
);

CREATE TABLE PollClassic (
        state VARCHAR(2),
        district TINYINT,
        candidate_first VARCHAR(255),
        candidate_last VARCHAR(255),
        party VARCHAR(1),
        is_incumbent INTEGER(1),
        win_probability DECIMAL,
        expected_voteshare DECIMAL,
        p10_voteshare DECIMAL,
        p90_voteshare DECIMAL,
```

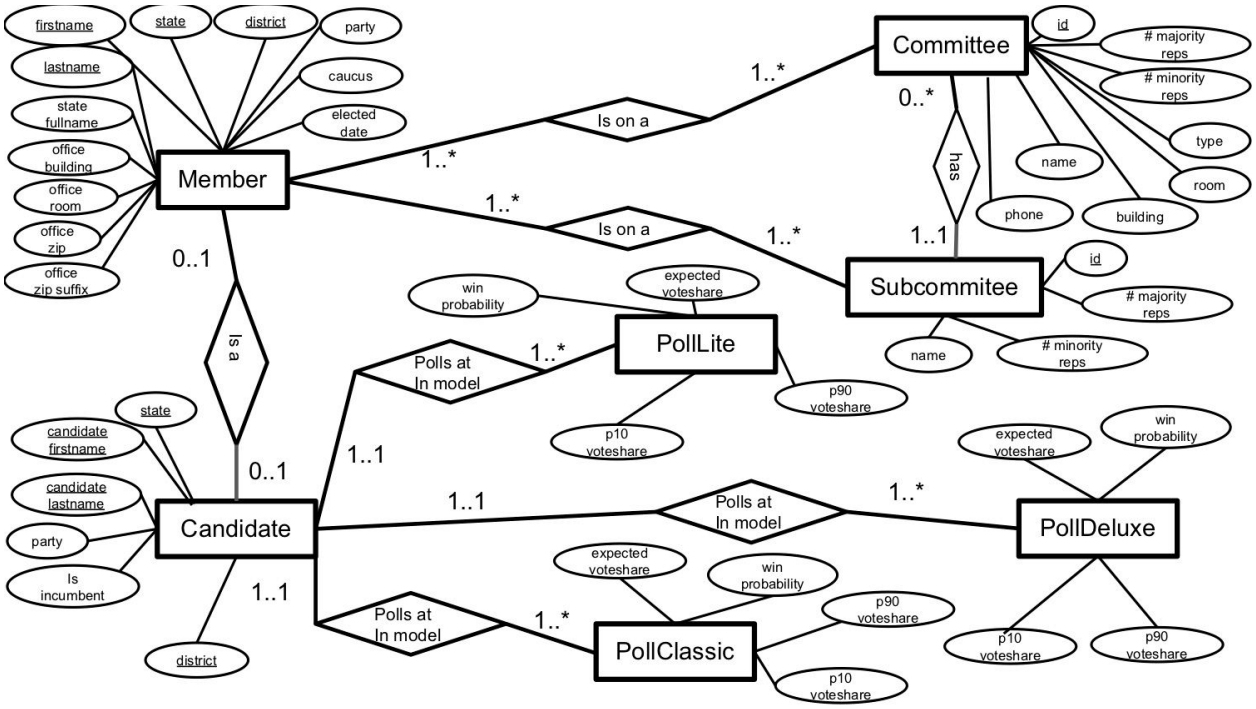
```
        PRIMARY KEY (state, district, candidate_first, candidate_last),
        FOREIGN KEY (state, district) REFERENCES Member(state, district)
);
```

DynamoDB NoSQL Document Schema

Committees:

```
{
    committee_id: string
    full_committee_name : string
    num_majority_members : int
    num_minority_members : int
    type : string
    building : string
    room : string
    phone : string
    subcommittees : [{
        subcommittee_code : string
        full_subcommittee_name : string
        num_majority_members : int
        num_minority_members : int
    }]
}
```

ER Diagram


firstname
state
district
party
lastname
caucus
state fullname
elected date
office building
office room
office zip
office zip suffix
Member
Is on a
1..*
1..*
Committee
id
majority reps
minority reps
type
room
building
phone
name
has
0..*
1..1
Is on a
1..*
Subcommitee
id
majority reps
name
minority reps
0..1
Is a
win probability
expected voteshare
PollLite
Polls at In model
1..*
p90 voteshare
p10 voteshare
state
candidate firstname
candidate lastname
party
Is incumbent
Candidate
0..1
1..1
1..1
Polls at In model
1..*
expected voteshare
win probability
PollDeluxe
1..1
district
Polls at In model
1..*
expected voteshare
win probability
p90 voteshare
PollClassic
p10 voteshare
p10 voteshare
p90 voteshare

Backend Queries

For all queries requiring polling data, a polling model must be selected from the following PollLite, PollClassic, PollDeluxe.

1. `get_least_likely_member_of_committee_to_win_reelection(cid, scid, poll_model)`

   Given a committee with id %1 and subcommittee id %2, return the first and last name of the member of the committee least likely to retain her seat given that she is up for reelection using polling model %3. NOTE: some committee members may not be seeking another term, these will not be included.

   SQL Query:

```
SELECT DISTINCT m.firstname, m.lastname,
m.state_fullname, p.district, p.win_probability
FROM (SELECT *
     FROM CommitteeAssignment
     WHERE committee_id = %1' AND subcommittee =%2) ca
     JOIN Member m ON ca.state = m.state AND
     ca.district = m.district
     JOIN %3 p ON m.state = p.state AND m.district =
     p.district
WHERE p.win_probability = (
     SELECT MIN(p.win_probability)
     FROM (
          SELECT * FROM CommitteeAssignment
          WHERE committee_id = %1 AND subcommittee =%2
     ) ca
     JOIN Member m ON ca.state = m.state AND
          ca.district = m.district
     JOIN %3 p ON m.state = p.state AND m.district =
          p.district
     WHERE p.is_incumbent = 1
     );
```

2. `get_member_of_committee_in_closest_race(cid, scid, poll_model)`

   Given a committee with id %1 and subcommittee id %2, return the first and last name of the member of the committee in the closest reelection bid, that is who's win probability is closest to 50% using polling model %3

   SQL Query:

```
SELECT DISTINCT m.firstname, m.lastname,
     p.win_probability as winProbability
FROM(
     SELECT * FROM CommitteeAssignment
     WHERE committee_id = %1 AND subcommittee = %2) ca
JOIN Member m ON ca.state = m.state AND ca.district =
m.district JOIN %3 p ON m.state = p.state AND
m.district = p.district
WHERE p.is_incumbent = 1 AND p.win_probability + 50 =
(
     SELECT MIN(ABS(p.win_probability - 50))
     FROM (SELECT * FROM CommitteeAssignment
          WHERE committee_id = %1 AND subcommittee =
          %2) ca
          JOIN Member m ON ca.state = m.state AND
          ca.district = m.district
          JOIN %3 p ON m.state = p.state AND
     m.district
          = p.district)
     OR p.win_probability - 50 = (
          SELECT MIN(ABS(p.win_probability - 50))
          FROM (SELECT * FROM CommitteeAssignment
               WHERE committee_id = %1 AND
               subcommittee = %2) ca
          JOIN Member m ON ca.state = m.state AND
          ca.district = m.district JOIN %3 p ON
          m.state = p.state AND m.district =
          p.district
          )
```

3. get_candidates_in_district(state, district)

Given a state, %1, and district, %2, return the list of candidates running in the upcoming election, their party and probability of winning. State is the 2 letter state abbreviation, and district is a string i.e. ‘5’.

SQL Query:

```
SELECT candidate_first as firstname, candidate_last as
     lastname, party, win_probability as
     winProbability, is_incumbent as isIncumbent
FROM PollLite
WHERE state = %1 'AND district = %2;
```

4. get_info_for _member(state, district)

Given a state, %1, and district, %2, return all the information about the member from that district. State is the 2 letter state abbreviation and district is of the form ‘5’.

SQL Query:

```
SELECT *
FROM Member
WHERE state = %1 AND district = %2;
```

5. get_all_committees()

Return a list of all committees with their id and name

DynamoDB NoSQL Query:

```
dynamo.scan({
      TableName: ‘Committees’,
      AttributesToGet: ['committee_id',
     'full_committee_name']
  }
);
```

6. get_all_subcomittees_on _committee(cid)

Given a committee with id, %1,return name and id for all subcommittees under that committee

DynamoDB NoSQL Query:

```
dynamo.getItem({
      TableName: ”Committees”,
      Key:{
        "committee_id": {
          S: %1
        }
      },
      ProjectionExpression: "subcommittees",
     AttributesToGet:
          ['subcommittee_id',
          'full_subcommittee_name']
  });
```

7. get_all_members_on_committee(cid,vscid)

Given a committee with id %1 and subcommittee id %2, ,return the district, state, first and last name of all members of that committee

SQL Query:

```
SELECT DISTINCT m.firstname, m.lastname,
m.state_fullname, m.district, m.phone
FROM Member m JOIN CommitteeAssignment ca ON
     m.district = ca.district AND m.state = ca.state
WHERE ca.committee_id = %1 AND subcommittee = %2;
```

8. get_all_members_up_for_reelection_on_committee(cid)

Given a committee with id, %1 return the district, state, first and last name of all members of that committee who are in a contested election, that is they are running for reelection and have at least 1 opponent or their win percentage is less than 100

SQL Query:

```
SELECT DISTINCT m.firstname, m.lastname, p.district,
p.state
FROM (SELECT * FROM CommitteeAssignment
     WHERE committee_id = %1) ca
JOIN Member m ON m.state = ca.state AND m.district =
```

```
        ca.district
    JOIN PollLite p ON m.state = p.state AND m.district =
        p.district
    WHERE p.win_probability < 100 AND p.is_incumbent = 1;
```

9. get_all_races_within(num_percentage_points, poll_model)

Given a number of percentage points, %1, return state, district, lead candidate first name, lead candidate last name, lead candidate win probability, trailing candidate first name, trailing candidate last name, and trailing candidate win probability, for every race where the leading candidate is %1 percentage points or fewer ahead of the trailing candidate using polling model %2

SQL Query:

```
    SELECT DISTINCT p1.state,
        p1.district,
        p1.candidate_first as leadFirstname,
        p1.party as leadParty,
        p1.win_probability as leadWinProbability,
        p2.candidate_first as behindFirstname,
        p2.party as behindParty,
        p2.win_probability as behindWinProbability
    FROM %2 p1 JOIN %2 p2
        ON p1.state = p2.state AND p1.district =
    p2.district
    WHERE abs(p1.win_probability - p2.win_probability) <=
        %1
        AND (p1.win_probability > p2.win_probability
            OR (p1.win_probability = 50 and
            p1.candidate_first > p2.candidate_first))
        AND p1.candidate_first <> p2.candidate_first
        AND (p1.state, p1.district, p1.win_probability)
        IN (
            SELECT w.state, w.district,
                MAX(w.win_probability)
            FROM %2 w
            WHERE w.state = p1.state AND w.district =
            p1.district)
```

10. `get_num_districts_in_state(state)`

Given a state, %1, return the number of congressional districts in that state

SQL Query:

```
SELECT MAX(district)
FROM PollLite
WHERE state = %1;
```

11. `get_member_from_district(state, district)`

Given a state, %1, and district %2, return the name of the current member.

SQL Query:

```
SELECT firstname, lastname
FROM Member
WHERE state = %1 AND district = %2;
```

12. `get_num_subcommittees_on_committee(cid)`

Given a committee code %1, return the codes of subcommittees on the given committee.

SQL Query:

```
SELECT DISTINCT subcommittee
FROM CommitteeAssignment
WHERE committee_id = %1
ORDER BY subcommittee ASC;
```

13. `get_num_reps_per_state()`

Return the number of reps in each state.

SQL Query:

```
SELECT state, COUNT(*)
FROM Member
GROUP BY state ASC;
```

14. `get_num_districts_per_state`(state)

Given a state abbreviation %1, return all congressional districts in that state.

SQL Query:

```
SELECT DISTINCT district
FROM Member
WHERE state = %1
ORDER BY state ASC';
```

15. `get_all_states`()

Return all state abbreviations that have associated representatives

SQL Query:

```
SELECT DISTINCT state, state_fullname
FROM Member
ORDER BY state_fullname ASC
```

Performance Evaluation

Index	Function	Before Index (ms)	After Index (ms)	Difference (ms)
Member.state	Reps per state	23.238	20.898	-2.34
	Get all states	20.008	20.811	0.803
	Get districts for state (WA)	24.388	24.903	0.515
	Who's running (WA, 9)	24.57	26.324	1.754
	Tight Race (threshold = 3, PollLite)	25.745	26.245	0.5
	Get committee codes	21.689	21.629	-0.06
	Populate subcommittee codes (AG)	23.819	27.13	3.311
	Closest Race (AG, 16, PollLite)	44.328	43.087	-1.241
	Least Likely (AG, 16, PollLite)	25.642	25.876	0.234
	All members on committee (AG, 16)	63.936	61.695	-2.241
Member.state &	Reps per state	23.238	23.062	-0.176
Member.district	Get all states	20.008	19.521	-0.487
	Get districts for state (WA)	24.388	22.287	-2.101
	Who's running (WA, 9)	24.57	24.901	0.331
	Tight Race (threshold = 3, PollLite)	25.745	23.11	-2.635
	Get committee codes	21.689	22.426	0.737
	Populate subcommittee codes (AG)	23.819	23.933	0.114
	Closest Race (AG, 16, PollLite)	44.328	42.09	-2.238
	Least Likely (AG, 16, PollLite)	25.642	23.98	-1.662
	All members on committee (AG, 16)	63.936	58.851	-5.085

We first attempted to just index on Member.state; this did not provide latency decrease. We then added an index on (Member.state, Member.district). This provided some latency decrease. The results here are averaged over 10 trials. Many of these decreases are not very significant. Our dataset is relatively small. There are around 435 members in office. Each position has 2 or 3 candidates running for it, thus each polling model dataset contains just over 1,000 tuples. Because our dataset is relatively small, the

indexing does not improve latency that much. If our dataset was larger, we believe that the indeing would provide significant decrease in latency.

Additionally, we added caching to the "Who's Running" page. If a user queries for who is running in Washington district 9, then Washington district 10, and then again Washington district 9, our application caches the initial Washington district 9 search so the query does not need to be executed again.

Technical challenges and how they were overcome

There were a number of technical challenges that we needed to overcome. Firstly, there was some difficulty with loading times of the databases that we had been integrating. As such, we needed to work on optimizing the load times and the querying, such that it would load the data as quickly as possible. We also decided that it would make sense to cache results of queries of candidates running in district. This cache is stored client side as we assume each user will want to query results for a small number of districts - presumably the ones they live in or near. Nextly, it was quite hard to connect the backend (originally an Oracle database) to our original Angular frontend. As such, we switched the backend architecture to MySQL, along with our frontend (we switched to Node, DynamoDB, RazorSQL, MySQL, and maintained some Angular). In terms of data cleaning, win probabilities was stored as a decimal between 0 and 1. To be more readable and easier to use in queries, we translated these decimals to integers between 0 and 100. Lastly, we learned late in the development process that, with an AWS Educate account, DynamoDB needs access keys to be regenerated every hour. This led to a slight hiccup in our presentation where we had to regenerate the codes.

Extra credit features

1. We implemented a NoSQL solution in DynamoDB requiring a new querying language we learned
2. We used AWS to store both our MySQL and DynamoDB tables.
3. We used an advanced web database stack (MEAN Stack) of Angular, Node (which we later migrated to Angular), and Dynamo all built on JavaScript.