Max Smith
CSCE-608
03/21/2024

Project 1 Report

Project Description

This application is a legislative search system that displays information about bills, legislators, committees, roll call votes, and related legislative data from the 115th-119th sessions of the United States Congress (the 119th legislative session is ongoing, so data from this session is somewhat limited). It provides a web-based interface for users to search, view, and analyze legislative data.

The application allows users to search for bills using several parameters: partial match of the bill title, the bill number, its status, and partial matches of bill sponsors (separated by commas). Results are displayed in a table that can be sorted on any of its attributes (title, bill number, committee, session, status, status date, bipartisanship score). The bipartisanship score is calculated through a query that will be described below.

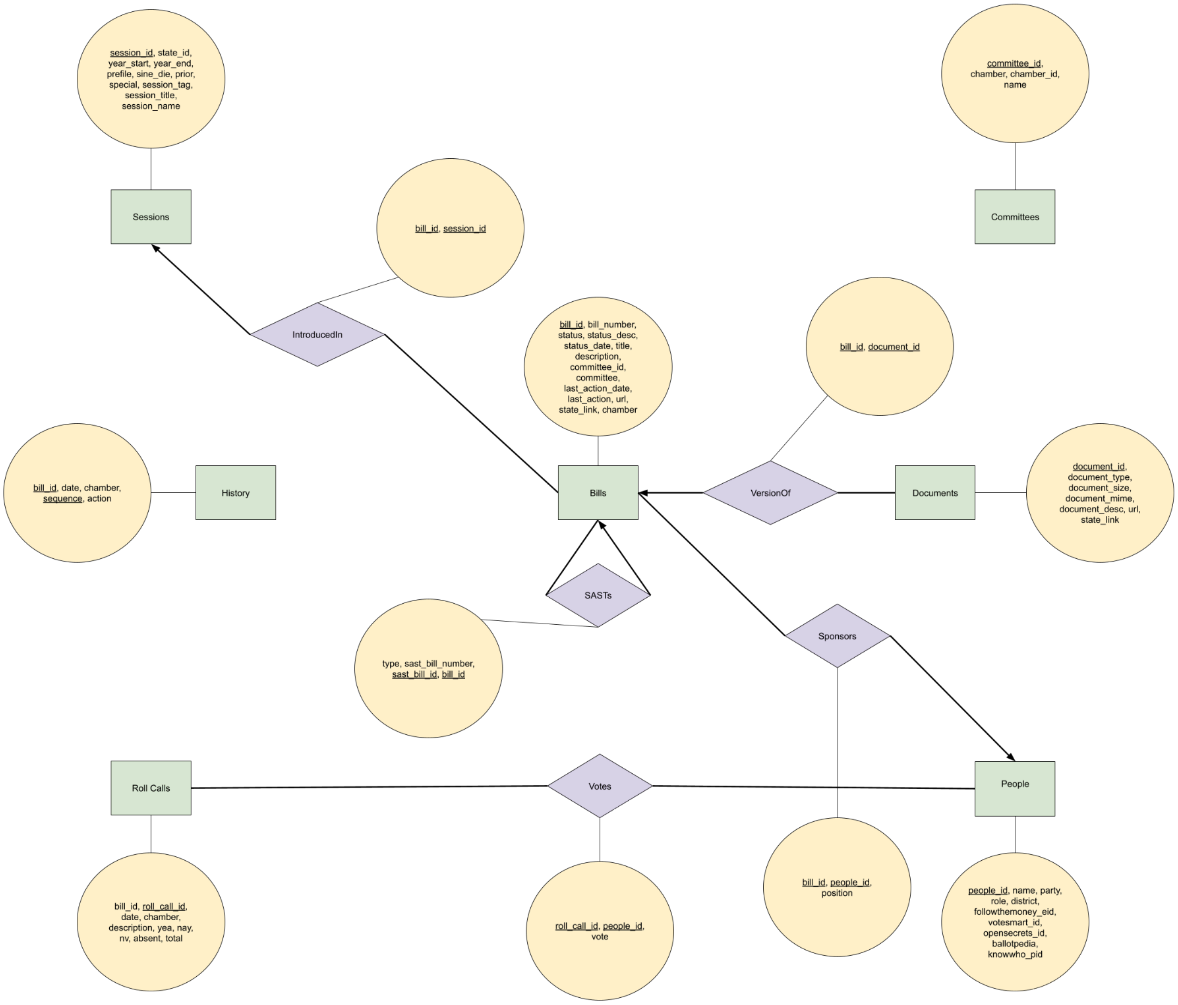
Users can also search for legislators through the following parameters: partial match of a name, party, role (senator/representative), and partial match of a district. Similarly to the bills search page, the results are displayed in a table.

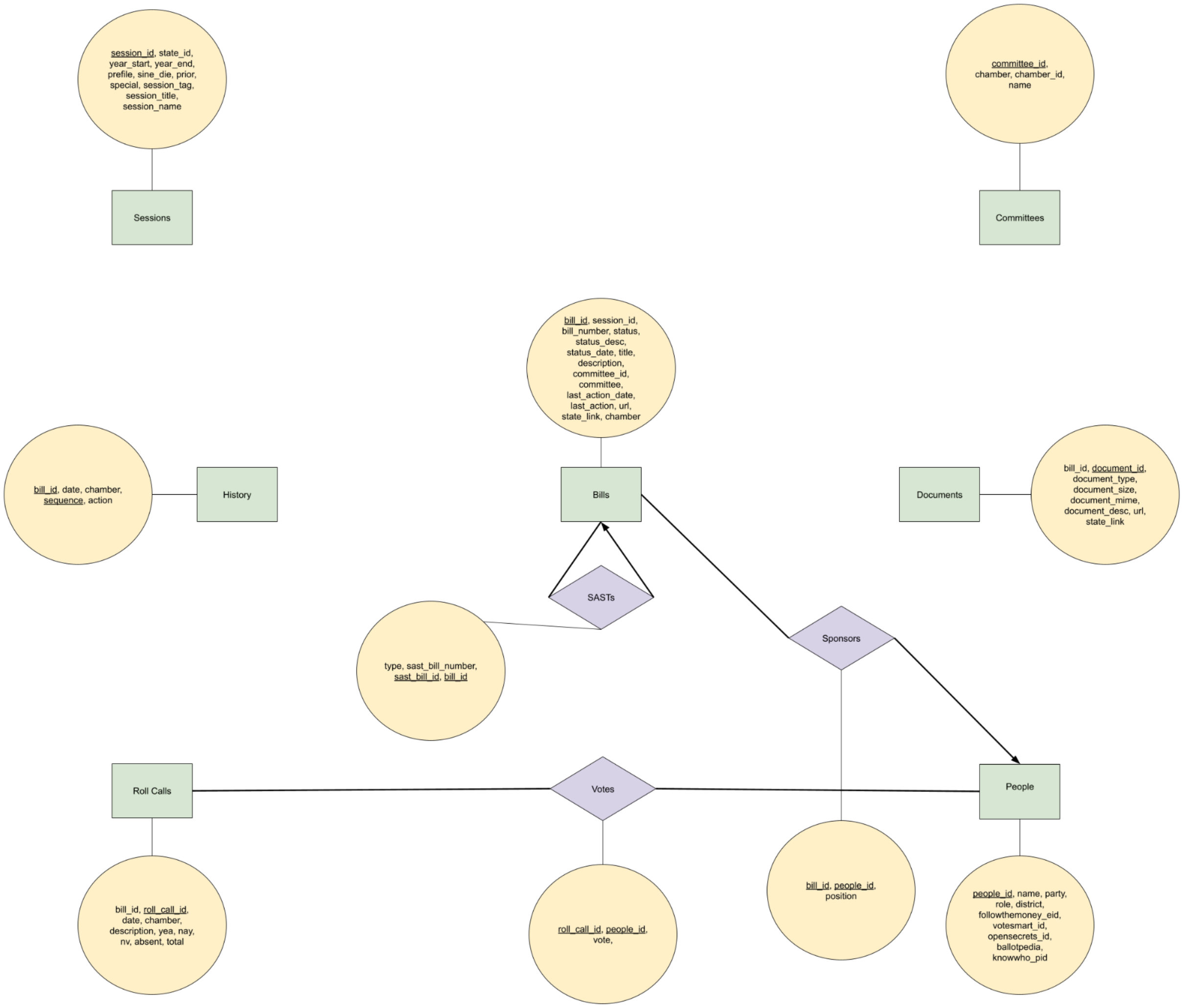
Search results for bills and legislators each link to pages describing each in more detail. The page for a given bill shows detailed information about that bill, including: title, bill number, session, committee, status, sponsors, partisan breakdown, related bills (same as/similar to), roll calls, history, and associated documents (prior versions). Related bills link to the pages for those particular bills, roll calls link to pages showing the details of a given roll call, and bill sponsors link to pages describing a given legislator

A legislator's profile page displays detailed information about a specific legislator, including: name, party, role, and district, sponsored bills with links to their details, and a comprehensive voting history with links to the associated bills and roll calls.

A roll call page shows details of a specific roll call, including its date, chamber (may differ from the chamber that the bill was initially proposed in, so this is not functionally determined by the bill_id), description, and vote counts (yea, nay, not voting, absent). Some details of the associated bill are also displayed, as well as a tally of the individual votes with links to each of the legislators' profiles.

E/R Diagram


session_id, state_id, year_start, year_end, prefile, sine_die, prior, special, session_tag, session_title, session_name
Sessions
committee_id, chamber, chamber_id, name
Committees
bill_id, session_id
IntroducedIn
bill_id, bill_number, status, status_desc, status_date, title, description, committee_id, committee, last_action_date, last_action, url, state_link, chamber
bill_id, document_id
bill_id, date, chamber, sequence, action
History
Bills
VersionOf
Documents
document_id, document_type, document_size, document_mime, document_desc, url, state_link
SASTs
type, sast_bill_number, sast_bill_id, bill_id
Sponsors
Roll Calls
Votes
People
bill_id, roll_call_id, date, chamber, description, yea, nay, nv, absent, total
roll_call_id, people_id, vote
bill_id, people_id, position
people_id, name, party, role, district, followthemoney_eid, votesmart_id, opensecrets_id, ballotpedia, knowwho_pid


session_id, state_id, year_start, year_end, prefile, sine_die, prior, special, session_tag, session_title, session_name
Sessions
committee_id, chamber, chamber_id, name
Committees
bill_id, session_id, bill_number, status, status_desc, status_date, title, description, committee_id, committee, last_action_date, last_action, url, state_link, chamber
bill_id, date, chamber, sequence, action
History
Bills
Documents
bill_id, document_id, document_type, document_size, document_mime, document_desc, url, state_link
SASTs
Sponsors
type, sast_bill_number, sast_bill_id, bill_id
Roll Calls
Votes
People
bill_id, roll_call_id, date, chamber, description, yea, nay, nv, absent, total
roll_call_id, people_id, vote,
bill_id, people_id, position
people_id, name, party, role, district, followthemoney_eid, votesmart_id, opensecrets_id, ballotpedia, knowwho_pid

The second E/R diagram combines a few of the relations shown in the first E/R diagram (combining a "one" into a "many"). The IntroducedIn relation is combined with Bills, since each bill is introduced in only a single congressional session. Likewise, the VersionOf relation was combined into Documents, since each document is a version of only a single bill. This diagram loosely matches the CSVs in the original dataset with some additional relations (Committees, Sessions, VersionOf, IntroducedIn) added. Each of the relations shown provides important information and context about US Congressional legislation.

Relational Schema

Un-normalized:

bills(bill_id, session_id, bill_number, status, status_date, title, description, committee_id,
committee, last_action_date, last_action, url)

bill_id → session_id, bill_number, status, status_date, title, description, committee_id, committee, last_action_date, last_action, url

committe_id → committee

committees(committee_id, chamber, name)

committe_id → committee_id, chamber, name

documents(document_id, bill_id, document_type, document_size, document_mime, document_desc, url)

document_id → bill_id, document_type, document_size, document_mime, document_desc, url

history(bill_id, sequence, date, chamber, action)

bill_id, sequence → date, chamber, action

people(people_id, name, party, role, district, followthemoney_eid, votesmart_id, opensecrets_id, ballotpedia, knowwho_pid)

people_id → name, party, role, district, followthemoney_eid, votesmart_id, opensecrets_id, ballotpedia, knowwho_pid

rollcall(roll_call_id, bill_id, date, chamber, description, yea, nay, nv, absent, total)

roll_call_id → bill_id, date, chamber, description, yea, nay, nv, absent, total

sasts(sast_bill_id, bill_id, type, sast_bill_number)

sast_bill_id, bill_id → type, sast_bill_number
sast_bill_id → sast_bill_number

sessions(session_id, year_start, year_end, prefile, sine_die, prior, special, session_tag, session_title, session_name)

session_id → year_start, year_end, prefile, sine_die, prior, special, session_tag, session_title, session_name

sponsors(bill_id, people_id, position)

bill_id, people_id → position

votes(roll_call_id, people_id, vote)

roll_call_id, people_id → vote

BCNF Relations:

bills(bill_id, session_id, bill_number, status, status_date, title, description, committee_id, last_action_date, last_action, url)

bill_id → session_id, bill_number, status, status_date, title, description, committee_id, committee, last_action_date, last_action, url

committees(committee_id, chamber, name)

committe_id → committee_id, chamber, name

documents(document_id, bill_id, document_type, document_size, document_mime, document_desc, url)

document_id → bill_id, document_type, document_size, document_mime, document_desc, url

history(bill_id, sequence, date, chamber, action)

bill_id, sequence → date, chamber, action

people(people_id, name, party, role, district, followthemoney_eid, votesmart_id, opensecrets_id, ballotpedia, knowwho_pid)

people_id → name, party, role, district, followthemoney_eid, votesmart_id, opensecrets_id, ballotpedia, knowwho_pid

rollcall(roll_call_id, bill_id, date, chamber, description, yea, nay, nv, absent, total)

roll_call_id → bill_id, date, chamber, description, yea, nay, nv, absent, total

sasts(sast_bill_id, bill_id, type)

sast_bill_id, bill_id → type

sessions(session_id, year_start, year_end, prefile, sine_die, prior, special, session_tag, session_title, session_name)

session_id → year_start, year_end, prefile, sine_die, prior, special, session_tag, session_title, session_name

sponsors(bill_id, people_id, position)

bill_id, people_id → position

votes(roll_call_id, people_id, vote)

roll_call_id, people_id → vote

Applying BC normalization removed the “committee” attribute from the “bills” relation. Since the functional dependency involving this attribute is already represented in the “committees” relation, a new relation wasn’t added. Removal of this attribute both reduces the space required for the “bills” relation and avoids update and deletion anomalies.

Similarly, removal of “sast_bill_number” from “sasts” reduces redundancy and avoids anomalies. Removal of the associated nontrivial functional dependency did not result in a new relation because this functional dependency is already represented in “bills”.

Initially, it was assumed that history.bill_id → history.chamber and rollcall.bill_id → rollcall.chamber were functional dependencies, but this is not the case. The chamber of a given tuple in the “history” relation is dependent on both the bill and the sequence number of actions associated with the bill. Similarly, the chamber of a given rollcall vote depends on the vote, not on the chamber that the bill was initially proposed in (a bill can be proposed in one chamber and eventually be voted on by the other chamber).

Data Collection

Data for the application was collected from LegiScan’s datasets and only includes data from the 115th-119th US congressional sessions. LegiScan offers both JSON-formated datasets, as well as more limited CSV-formatted datasets. Since the CSVs were formatted more closely to my relations, those were used; however, the data was supplemented by additional fields that were only present in the JSON datasets. Some fields in the CSVs were either redundant or not useful for this application, so they were removed. In some tables there were missing fields, so either sentinel values were added to avoid issues (such as with missing dates and committee IDs), or values were generated using other attributes (such as with missing bill titles).

Since several datasets were joined to produce each table (one for each session of congress), duplicate tuples were removed by creating a list of unique keys and deleting any tuple with a repeated key. Since this was a real dataset, the tables already had many overlapping attributes that could be used to produce interesting joins.

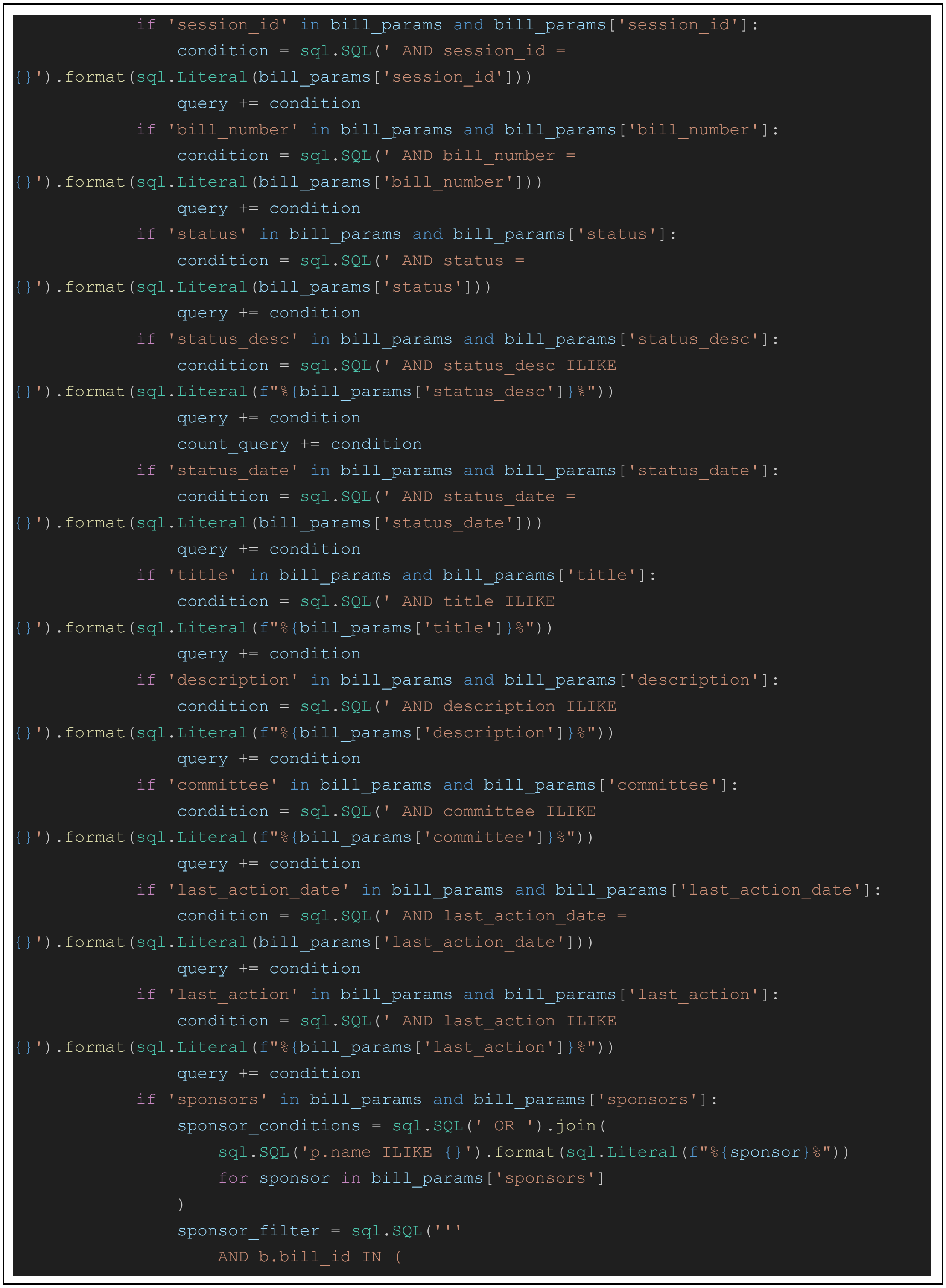
User Interface

The user interface for this application is written in Python and uses the Django framework. It is run as a web application and users can navigate using buttons and hyperlinks. SQL queries are built in python using the `psycopg2` library.

Functions:
Get Bills
This function is used to search bills using filters and return a paginated, sorted list of tuples. The base query defines a bipartisanship score for each bill that increases as the balance of Republican and Democratic sponsors gets closer to 50-50 (the score is scaled by the total number of sponsors to weight bills that have many sponsors over those that do not).

```
def get_bills(self, bill_params, page=1, page_size=50, sort='bill_id', order='asc'):
        # Base query for filtering bills
        query = sql.SQL('''
            SELECT b.bill_id, b.bill_number, b.status, b.status_date, b.title,
                ss.session_name, c.name AS committee,
                (1 - ABS(COALESCE(sc.D, 0) - COALESCE(sc.R, 0))
                / NULLIF(COALESCE(sc.D, 0) + COALESCE(sc.R, 0), 0))
                * (COALESCE(sc.D, 0) + COALESCE(sc.R, 0)) AS bipartisanship_score
            FROM bill b
            LEFT JOIN (
                SELECT s.bill_id,
                    COUNT(CASE WHEN p.party = 'D' THEN 1 END) AS D,
                    COUNT(CASE WHEN p.party = 'R' THEN 1 END) AS R
                FROM sponsor s
                JOIN person p ON s.people_id = p.people_id
                GROUP BY s.bill_id
            ) sc ON b.bill_id = sc.bill_id
            LEFT JOIN session ss ON b.session_id = ss.session_id
            LEFT JOIN committee c ON b.committee_id = c.committee_id
            WHERE TRUE
        ''')

        # Add filters based on bill_params
        if bill_params:
            if 'bill_id' in bill_params and bill_params['bill_id']:
                condition = sql.SQL(' AND b.bill_id = {}').format(sql.Literal(bill_params['bill_id']))
                query += condition
```



```
            if 'session_id' in bill_params and bill_params['session_id']:
                condition = sql.SQL(' AND session_id =
{}').format(sql.Literal(bill_params['session_id']))
                query += condition
            if 'bill_number' in bill_params and bill_params['bill_number']:
                condition = sql.SQL(' AND bill_number =
{}').format(sql.Literal(bill_params['bill_number']))
                query += condition
            if 'status' in bill_params and bill_params['status']:
                condition = sql.SQL(' AND status =
{}').format(sql.Literal(bill_params['status']))
                query += condition
            if 'status_desc' in bill_params and bill_params['status_desc']:
                condition = sql.SQL(' AND status_desc ILIKE
{}').format(sql.Literal(f"%{bill_params['status_desc']}%"))
                query += condition
                count_query += condition
            if 'status_date' in bill_params and bill_params['status_date']:
                condition = sql.SQL(' AND status_date =
{}').format(sql.Literal(bill_params['status_date']))
                query += condition
            if 'title' in bill_params and bill_params['title']:
                condition = sql.SQL(' AND title ILIKE
{}').format(sql.Literal(f"%{bill_params['title']}%"))
                query += condition
            if 'description' in bill_params and bill_params['description']:
                condition = sql.SQL(' AND description ILIKE
{}').format(sql.Literal(f"%{bill_params['description']}%"))
                query += condition
            if 'committee' in bill_params and bill_params['committee']:
                condition = sql.SQL(' AND committee ILIKE
{}').format(sql.Literal(f"%{bill_params['committee']}%"))
                query += condition
            if 'last_action_date' in bill_params and bill_params['last_action_date']:
                condition = sql.SQL(' AND last_action_date =
{}').format(sql.Literal(bill_params['last_action_date']))
                query += condition
            if 'last_action' in bill_params and bill_params['last_action']:
                condition = sql.SQL(' AND last_action ILIKE
{}').format(sql.Literal(f"%{bill_params['last_action']}%"))
                query += condition
            if 'sponsors' in bill_params and bill_params['sponsors']:
                sponsor_conditions = sql.SQL(' OR ').join(
                    sql.SQL('p.name ILIKE {}').format(sql.Literal(f"%{sponsor}%"))
                    for sponsor in bill_params['sponsors']
                )
                sponsor_filter = sql.SQL('''
                    AND b.bill_id IN (
                        SELECT bill_id
                        FROM sponsor s
                        JOIN person p ON s.people_id = p.people_id
                        WHERE {}
                    )
                ''').format(sponsor_conditions)
                query += sponsor_filter

        # Add sorting
        query += sql.SQL(' ORDER BY {} {} NULLS LAST').format(
            sql.Identifier(sort),
            sql.SQL(order.upper())
        )

        # Add LIMIT and OFFSET for pagination
        offset = (page - 1) * page_size
        query += sql.SQL(' LIMIT {} OFFSET {}').format(sql.Literal(page_size),
sql.Literal(offset))
```

Get People

This function is used to search legislators using filters and return a paginated list of tuples.

```
def get_people(self, people_params):
        query = sql.SQL('SELECT people_id, name, party, role, district FROM person WHERE
TRUE')
        if people_params:
            if 'party' in people_params and people_params['party']:
                query += sql.SQL(' AND party =
{}').format(sql.Literal(people_params['party']))
            if 'role' in people_params and people_params['role']:
                query += sql.SQL(' AND role =
{}').format(sql.Literal(people_params['role']))
            if 'name' in people_params and people_params['name']:
                query += sql.SQL(' AND name ILIKE
{}').format(sql.Literal(people_params['name']))
            if 'district' in people_params and people_params['district']:
                query += sql.SQL(' AND district ILIKE
{}').format(sql.Literal(f"%{people_params['district']}%"))
```

Get Person

Retrieves detailed information about a specific legislator by people_id.

Get Bill

Retrieves detailed information about a specific bill by its bill_id and joins the session table to include the session name.

```
def get_bill(self, bill_id):
        query = sql.SQL('''
            SELECT b.*, ss.session_name
            FROM bill b
            JOIN session ss ON b.session_id = ss.session_id
            WHERE bill_id = {}
        ''').format(sql.Literal(bill_id))
```

Get SASTs

Retrieves "same as" or "similar to" relationships for a specific bill (to other bills) and joins the bill table to include additional details about the related bills.

```
query = sql.SQL('''
            SELECT s.type, sb.sast_bill_number, s.sast_bill_id,
s.bill_id, b.title, b.bill_number
            FROM sast s
            JOIN (
                SELECT bill_id, bill_number, title
                FROM bill
            ) b ON s.bill_id = b.bill_id
            JOIN (
                SELECT bill_id, bill_number AS sast_bill_number
                FROM bill
            ) sb ON s.sast_bill_id = sb.bill_id
            WHERE s.bill_id = {}
        ''').format(sql.Literal(bill_id))
```

Get Rollcalls

Retrieves all roll calls associated with a specific bill.

```
query = sql.SQL('''
            SELECT r.roll_call_id, r.date, r.chamber, r.description,
r.yea, r.nay, r.nv, r.absent, r.total
            FROM rollcall r
            WHERE r.bill_id = {}
        ''').format(sql.Literal(bill_id))
```

Get History

Retrieves the history of actions taken on a specific bill.

```
query = sql.SQL('''
            SELECT h.date, h.chamber, h.sequence, h.action
            FROM history h
            WHERE h.bill_id = {}
        ''').format(sql.Literal(bill_id))
```

Get Bill Documents

Retrieves all documents associated with a specific bill (each document is a version of the bill).

```
query = sql.SQL('''
            SELECT d.document_id, d.document_type, d.document_size,
d.document_mime, d.document_desc, d.url
            FROM document d
            WHERE d.bill_id = {}
        ''').format(sql.Literal(bill_id))
```

Get Bill Committee

Retrieves the committee associated with a specific bill.

```
query = sql.SQL('''
            SELECT c.chamber, c.name
            FROM committee c
            JOIN bill b ON c.committee_id = b.committee_id
            WHERE b.bill_id = {}
        ''').format(sql.Literal(bill_id))
```

Get Votes

Retrieves all votes cast by a specific legislator and joins the rollcall, bill, and session tables to include details about the associated rollcalls and bills.

```
query = sql.SQL('''
            SELECT v.vote, r.date, r.description, b.title,
b.bill_number, b.bill_id, s.session_name, r.roll_call_id, r.chamber
            FROM vote v
            JOIN rollcall r ON v.roll_call_id = r.roll_call_id
            JOIN bill b ON r.bill_id = b.bill_id
            JOIN session s ON b.session_id = s.session_id
            WHERE v.people_id = {}
        ''').format(sql.Literal(people_id))
```

Get Sponsored Bills

Retrieves all bills sponsored by a specific legislator and joins the bill and session tables to include details about the associated bills.

```
query = sql.SQL('''
            SELECT b.title, b.bill_number, b.bill_id,
b.last_action_date, b.last_action, ss.session_name
            FROM sponsor s
            JOIN bill b ON s.bill_id = b.bill_id
            Join session ss ON b.session_id = ss.session_id
            WHERE s.people_id = {}
        ''').format(sql.Literal(people_id))
```

Get RollCall

Retrieves details of a specific roll call by its roll_call_id and joins the bill and session tables to include details about the associated bill and session.

```
query = sql.SQL('''
            SELECT r.roll_call_id, r.date, r.chamber, r.description,
                r.yea, r.nay, r.nv, r.absent, r.total, b.bill_id,
b.title, b.bill_number, s.session_name
            FROM rollcall r
            JOIN bill b ON r.bill_id = b.bill_id
            JOIN session s ON b.session_id = s.session_id
            WHERE r.roll_call_id = {}
        ''').format(sql.Literal(role_call_id))
```

Get Rollcall Votes

Retrieves all individual votes for a specific roll call and joins the person table to include details about the legislators who voted in the rollcall.

```
query = sql.SQL('''
            SELECT v.vote, p.name, p.party, p.role, p.district,
p.people_id
            FROM vote v
            JOIN person p ON v.people_id = p.people_id
            WHERE v.roll_call_id = {}
        ''').format(sql.Literal(role_call_id))
```

Get Sponsors

Retrieves all sponsors for a specific bill and joins the person table to include details about the sponsor.

```
query = sql.SQL('''
            SELECT p.name, s.position, p.party, p.role, p.district,
s.people_id
            FROM sponsor s
            JOIN person p ON s.people_id = p.people_id
            WHERE s.bill_id = {}
        ''').format(sql.Literal(bill_id))
```

Get Partisan Breakdown

Calculates the partisan breakdown of sponsors for a specific bill by grouping sponsors by party and counting the number of sponsors in each party.

```
query = sql.SQL('''
            SELECT p.party, COUNT(*) as count
            FROM sponsor s
            JOIN person p ON s.people_id = p.people_id
            WHERE s.bill_id = {}
            GROUP BY p.party
        ''').format(sql.Literal(bill_id))
```

Source Code

https://github.com/maxsmith271346/csce-608-project1

Discussion

Developing this application was a great opportunity to play with an interesting real-world dataset. While developing this application, I unexpectedly ran into issues due to assumptions I made about the dataset. I initially expected that the data was clean, but quickly realized that many fields contained missing entries. This was handled by either adding sentinel values or filling in the empty cells using data generated from other attributes. Additionally, there were both redundant attributes, as well as some that just were not useful to my application in the original data that I removed. Lastly, I ran into an issue by assuming an incorrect functional dependency (these incorrectly assumed FDs were discussed in the normalization section). This caused me to have to restructure my schema and make modifications to both my frontend and backend to handle the change.

The most important thing that I learned from this project is that incorrect assumptions about a dataset can cause serious issues, so one must be careful when handling datasets that they are unfamiliar with.