Lab 2: Fact Checking with Relational Databases

In this lab exercise, you will build on HW 2 to practice more with SQL queries for processing and exploring relational databases. You should work in a group of 4-5 students of your choice; you will only need to submit once as a group (see submission instructions below). Remember that labs are graded for sincere effort: if you do not finish the entire lab assignment before the end of class, turn in what you have so far.

This lab will be about the `congress` database that you worked with for HW 2. For your convenience, we have packaged the entire database together in one file, already formatted for `SQLite`, inside of the lab 2 box folder. It is, however, the same database as from HW 2.

Name and NetID

Nathan Kim (njk24) and Eugene Kim (yk171)

Part 1: More Ways to Use SQL

For HW 2, you worked with the `SQLite` command line interface. You are welcome and encouraged to continue using the command line interface in this lab (remember the reccomended setup writing scripts in a plain text file and executing them at the command line), but we will also introduce you to executing `SQL` queries in your Python programs. This is especially useful when you want to pull data from `SQL` queries into your programs (for example, we will pull `SQL` query results into Pandas dataframes) for subsequent computation.

We will use the `sqlite3` Python library https://docs.python.org/3.7/library/sqlite3.html (https://docs.python.org/3.7/library/sqlite3.html) to connect to a database and run SQL queries, and we will store the results as `pandas` dataframes. This library should already be included with your Anaconda distribution, so you should simply be able to run the code cell below to import it and pandas.

In [3]:
```
import sqlite3
import pandas as pd
```

Connecting to the Database

First we need to establish a connection to the relational database. Assuming you have downloaded the file `congress` from box and it is in the same working directory as this notebook, you should simply be able to run the following code cell to initialize a connection with the database.

If you don't want to redownload the database, you can easily create it using the files from HW 2 as follows: move this Jupyter notebook to the same working directory as `load.sql` and the `load` folder from HW 2. In your terminal, navigate to this directory and then initialize the `SQLite` command line interface and create a new `SQLite` database file named congress by typing `sqlite3 congress` in your terminal. Then, inside of the command line interface, simply type `.read load.sql` as before. The file `congress` should now be ready for use, and you can proceed to run the following code cell to connect to it from this notebook.

In [4]:
```
conn = sqlite3.connect("congress")
cursor = conn.cursor()
```

Integrating SQL and Pandas

Now that you are connected to the database, we want to store the results of `SQL` queries as `Pandas` dataframes. The cursor object allows you to directly execute `SQL` on the database, but we will more often be interested in getting the results of a query for use in our program. `Pandas` has built-in functionality for doing just this, and we show a simple example below of creating a dataframe `df` which is the result of an `SQL` query that we run on the database.

In [ ]:
```
# simple query
simple_query = """
SELECT id, first_name, last_name, birthday
FROM persons p
LIMIT 5
"""
df = pd.read_sql(simple_query, con=conn)
print(df)
```

Note that our new dataframe `df` is just like any other Pandas dataframe, and we can perform subsequent analysis on it in Python directly. For example, if we wanted to pull out the years of birth from the above as a separate column of our dataframe containing integers instead of strings, we could run the following.

In [ ]:
```
birth_year = []
for birthday in df['birthday']:
    birth_year.append(int(birthday.split('-')[0]))
df['birth_year'] = birth_year
print(df)
```

Temporary Views using WITH Statements

Another trick: for a complex query, you can use the `WITH` clause to define a sequence of temporary views that each build onto other temporary views. This is helpful when it is logically more straightforward to create a series of views to get your desired result, but you don't actually care about keeping the intermediate views for future use. Here is the syntax:

```
WITH tmp_view_1 AS
  (... query that defines tmp_view_1 ...),
tmp_view_2 AS
  (... query that defines tmp_view_2 ...),
tmp_view_3 AS
  (... query that defines tmp_view_2 ...)
-- your actual query then starts below:
SELECT *
FROM tmp_view_3
WHERE ...;
```

Here is an example that uses such temporary views in calculating the percentage of members of the house of representative identified as female in each Congress in U.S. history. It pulls out counts of those identified as male and female grouped by start_date, creates two separate "tables," one for each gender, and then joins the two on start_dates in order to finally compute the percentages row-wise.

Note that this solution is not exact: most members of a particular congress for the House of Representatives take office on the same day, but there are some special elections and vacant seats filled at other dates. This simplistic solution simply groups together the days on which at least three members identified as men or women took office, and uses those to build the comparison.

In [ ]:
```
gender_query = """
WITH house_gender_counts AS
    (SELECT gender, start_date, end_date, count(*) as num
    FROM persons,
         person_roles
    WHERE persons.id = person_roles.person_id
      AND type = 'rep'
    GROUP BY gender, start_date),
large_counts_male AS
    (SELECT *
    FROM house_gender_counts
    WHERE num >= 3
        AND gender='M'),
large_counts_female AS
    (SELECT *
    FROM house_gender_counts
    WHERE num >= 3
        AND gender='F')
SELECT f.start_date, f.end_date,
    f.num AS women, m.num AS men,
    CAST(f.num AS float)/CAST((f.num+m.num) AS float) AS percentage_wome
n
FROM large_counts_female f
LEFT JOIN large_counts_male m
    ON f.start_date = m.start_date
ORDER BY f.start_date;
"""
df = pd.read_sql(gender_query, con=conn)
print(df.head())
```

Visualizing SQL Results

Just to drive home the point about integrating `SQL` and `Python`, let's now use these results along with `Seaborn` and `matplotlib`, `Python` libraries for data visualization, to visualize the trend of female participation in the House of Representatives over time. We'll reuse the Python code from above for pulling out the numeric years from start_date, and then plot a line graph of the percentages against these start years. The visual makes the point powerfully that female participation started a dramatic increase around the 1990s, although it still remains low overall.

In [ ]:
```
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt

start_year = []
for start_date in df['start_date']:
    start_year.append(int(start_date.split('-')[0]))
df['start_year'] = start_year
```

In [ ]:
```
sns.lineplot(x=df['start_year'], y=df['percentage_women'])
plt.show()
```

Part 2: Fact-Checking using a Relational Database

All of the following "facts" are in some sense true, and in another sense not true, or misleading. For each, use SQL queries to investigate two tasks: (1) verify the sense in which it is technically correct, and (2) determine to what extent it is not correct, or misleading. You can use the `SQLite` command line interface or execute your queries directly in this notebook through Pandas and the connection to the database as above. Either way, for each problem you should provide at least two sequal queries (answering tasks 1 and 2) and a brief expalanation of your results (that is, the sense in which the claim is true, and the sense in which it is not true or misleading).

(A): There are very few Christians in Congress Among all current members of the Congress, only 5 are Christians!

In [8]:
```
# A, Task 1
# fill in the string for query
query_a_1 = """
SELECT r.religion, COUNT(r.religion)
FROM cur_members r
WHERE r.religion = 'Christian'
"""
df = pd.read_sql(query_a_1, con=conn)
print(df)
```

```
    religion  COUNT(r.religion)
0  Christian                 5
```

The way this SQL query is implemented, when we specifically look for Christians with the string 'Christian', we are shown that there are only 5 Christians. However, this is misleading as there are denominations of Christianity.

In [9]:
```
# A, Task 2
# fill in the string for query
query_a_2 = """
SELECT r.religion, COUNT(r.religion)
FROM cur_members r
GROUP BY r.religion
"""
df = pd.read_sql(query_a_2, con=conn)
print(df)
```

	religion	COUNT(r.religion)
0		381
1	African Methodist Episcopal	3
2	Baptist	20
3	Catholic	14
4	Christian	5
5	Christian Scientist	2
6	Church of Christ	1
7	Episcopalian	12
8	Jewish	13
9	Latter Day Saints	6
10	Lutheran	5
11	Methodist	15
12	Nazarene	1
13	Presbyterian	13
14	Protestant	4
15	Roman Catholic	36
16	Seventh Day Adventist	1
17	Southern Baptist	2
18	United Methodist	1
19	Unknown	5

This SQL query was written so that we are shown the counts of all the members' religions. This is a better implementation of showing the count of Christians as it shows all possible denominations of Christianity.

(B) Texas is more bipartisan than California Amongst the newly elected members of the house in 2017 (i.e., never held office in either senate or house before), Texas had as many Democrats as Republicans, but California didn't have a single Republican!

In [51]:
```
# B, Task 1
# fill in the string for query
query_b_1 = """
WITH texas AS(
SELECT p.state, p.party, COUNT(p.party)
FROM cur_members p, person_roles r
WHERE (p.state = 'TX') AND r.person_id = p.id AND NOT(r.start_date<2017)
GROUP BY p.party),
california AS
(SELECT p.state, p.party, COUNT(p.party)
FROM cur_members p, person_roles r
WHERE (p.state = 'CA') AND r.person_id = p.id AND NOT(r.start_date<2017)
GROUP BY p.party)
SELECT*
FROM texas t, california c
"""
df = pd.read_sql(query_b_1, con=conn)
print(df)
```

	state	party	COUNT(p.party)	state	party	COUNT(p.party)
0	TX	Democrat	77	CA	Democrat	226
1	TX	Democrat	77	CA	Republican	91
2	TX	Republican	191	CA	Democrat	226
3	TX	Republican	191	CA	Republican	91

Explanation of B, Task 1

In [18]:
```
# B, Task 2
# fill in the string for query
query_b_2 = """
WITH texas AS(
SELECT p.state, p.party, COUNT(p.party)
FROM cur_members p, person_roles r
WHERE (p.state = 'TX') AND r.person_id = p.id AND NOT(r.start_date<2017)
GROUP BY p.party),
california AS
(SELECT p.state, p.party, COUNT(p.party)
FROM cur_members p, person_roles r
WHERE (p.state = 'CA') AND r.person_id = p.id AND NOT(r.start_date<2017)
GROUP BY p.party)
SELECT*
FROM texas t, california c
"""
df = pd.read_sql(query_b_2, con=conn)
print(df)
```

	party	COUNT(p.party)
0	Democrat	77
1	Republican	191

Explanation of B, Task 2

(C) Democratic leadership agrees a lot with Republican party line "Nancy Pelosi voted yes almost 50% of the time in a Republican congress during 2015-2016!" This is surprising because votes in the house are controlled by the majority party, which was Republican in 2015-16, and being the leader of the democratic party in the house Nancy Pelosi should be voting No for a lot of things.

In [57]:

```
# C, Task 1
# fill in the string for query
query_c_1 = """
SELECT COUNT(v.votes)
FROM persons p, person_votes v, votes o
WHERE p.last_name = 'Pelosi' AND v.person_id = p.id AND o.date='2015'
"""
df = pd.read_sql(query_c_1, con=conn)
print(df)
```

```
---------------------------------------------------------------------------
----
OperationalError                          Traceback (most recent call l
ast)
~/opt/anaconda3/lib/python3.7/site-packages/pandas/io/sql.py in execute
(self, *args, **kwargs)
   1594             else:
-> 1595                 cur.execute(*args)
   1596             return cur

OperationalError: no such column: v.votes

During handling of the above exception, another exception occurred:

DatabaseError                             Traceback (most recent call l
ast)
<ipython-input-57-68dbeaecd5d8> in <module>
      6 WHERE p.last_name = 'Pelosi' AND v.person_id = p.id AND o.date=
'2015'
      7 """
----> 8 df = pd.read_sql(query_c_1, con=conn)
      9 print(df)

~/opt/anaconda3/lib/python3.7/site-packages/pandas/io/sql.py in read_sq
l(sql, con, index_col, coerce_float, params, parse_dates, columns, chun
ksize)
    408             coerce_float=coerce_float,
    409             parse_dates=parse_dates,
--> 410             chunksize=chunksize,
    411         )
    412

~/opt/anaconda3/lib/python3.7/site-packages/pandas/io/sql.py in read_qu
ery(self, sql, index_col, coerce_float, params, parse_dates, chunksize)
   1643
   1644         args = _convert_params(sql, params)
-> 1645         cursor = self.execute(*args)
   1646         columns = [col_desc[0] for col_desc in cursor.descripti
on]
   1647

~/opt/anaconda3/lib/python3.7/site-packages/pandas/io/sql.py in execute
(self, *args, **kwargs)
   1608                 "Execution failed on sql '{sql}': {exc}".format
(sql=args[0], exc=exc)
   1609             )
-> 1610             raise_with_traceback(ex)
   1611
   1612     @staticmethod

~/opt/anaconda3/lib/python3.7/site-packages/pandas/compat/__init__.py i
n raise_with_traceback(exc, traceback)
     44     if traceback == Ellipsis:
     45         _, _, traceback = sys.exc_info()
---> 46     raise exc.with_traceback(traceback)
     47
     48
```

```
~/opt/anaconda3/lib/python3.7/site-packages/pandas/io/sql.py in execute
(self, *args, **kwargs)
   1593                 cur.execute(*args, **kwargs)
   1594             else:
-> 1595                 cur.execute(*args)
   1596             return cur
   1597         except Exception as exc:

DatabaseError: Execution failed on sql '
SELECT COUNT(v.votes)
FROM persons p, person_votes v, votes o
WHERE p.last_name = 'Pelosi' AND v.person_id = p.id AND o.date='2015'
': no such column: v.votes
```

Explanation of C, Task 1

In [ ]:
```
# C, Task 2
# fill in the string for query
query_c_2 = """

"""
df = pd.read_sql(query_c_2, con=conn)
print(df)
```

Explanation of C, Task 2

(D) Women on Abortion H. R. 7 of the 114th Congress is also known as the "No Taxpayer Funding for Abortion Act." (The bill was introduced in the 112th session of the Congress and was passed by the House on Janary 28, 2014 and on Jan 22, 2015, but was never passed by the Senate.) Women were strongly against this bill---61 of them voted against it while only 22 of them voted for it. The vote id (which you can use as given) for the bill is `h45-114.2015` .

In [ ]:
```
# D, Task 1
# fill in the string for query
query_d_1 = """

"""
df = pd.read_sql(query_d_1, con=conn)
print(df)
```

Explanation of D, Task 1

In [ ]:
```
# D, Task 2
# fill in the string for query
query_d_2 = """

"""
df = pd.read_sql(query_d_2, con=conn)
print(df)
```

Explanation of D, Task 2

In [ ]:
```
# Finally, don't forget to close the connection to the database
conn.close()
```

Submitting Lab 2

1. Double check that you have written all of your answers along with your supporting work in this notebook. Make sure you save the complete notebook.
2. Double check that your entire notebook runs correctly and generates the expected output. To do so, you can simply select Kernel -> Restart and Run All.
3. You will download two versions of your notebook to submit, a .pdf and a .py. To create a PDF, we reccomend that you select File --> Download as --> HTML (.html). Open the downloaded .html file; it should open in your web broser. Double check that it looks like your notebook, then print a .pdf using your web browser (you should be able to select to print to a pdf on most major web browsers and operating systems). Check your .pdf for readability: If some long cells are being cut off, go back to your notebook and split them into multiple smaller cells. Also, make sure that it is a reasonable length; print statements which are truncated inside of the notebook may come to many pages in the pdf. To get the .py file from your notebook, simply select File -> Download as -> Python (.py).
4. Upload the .pdf to gradescope under lab 2 report and the .py to gradescope under lab 2 code. Only submit once per group, but be sure to add your partner using the group feature on gradescope (https://www.gradescope.com/help#help-center-item-student-group-members).

In [ ]:
```

```