DS 100/200: Principles and Techniques of Data Science **Date: Feb 7, 2020**

Discussion #3

Name:

SQL Primary and Foreign Key

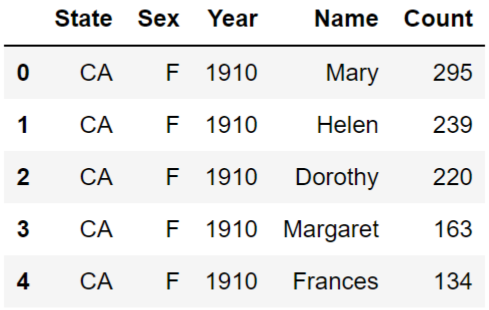
Definitions:

- **Primary key**: The column or minimal set of columns that uniquely determines the values in all the remaining columns. This is a statement about the schema and should hold for all data that could be put in the table.
  Below are some constraints on the primary key:
  - The data within these columns must be unique.
  - No value in the columns can be NULL.
- **Foreign key**: A set of one or more columns in a table that refers to the primary key in another table.
  Foreign keys have the following properties:
  - We can have NULL values in foreign keys.
  - We can have non-unique foreign keys in a table.
  - The foreign key is not null it should reference a particular primary key in another table.

1. **Examples?** What might be good examples of common primary keys and how might they be referenced as foreign keys.

   unique id #, phone #, email, etc.

2. Consider the following *sample* of the baby names table.



	State	Sex	Year	Name	Count
0	CA	F	1910	Mary	295
1	CA	F	1910	Helen	239
2	CA	F	1910	Dorothy	220
3	CA	F	1910	Margaret	163
4	CA	F	1910	Frances	134

What is the primary key?

each count is tied to a (State, Sex, Year, Name) tuple, so that's the p-key. This is because any subset of these cols is non-unique.

SQL Practice

3. For this question, we will be working with the UC Berkeley Undergraduate Career Survey dataset. Each year, the UC Berkeley career center surveys graduating seniors for their plans after graduating. Below is a sample of the full dataset. The full dataset contains many thousands of rows.

j_name	**c_name**	**c_location**	**m_name**
Llama Technician	Google	MOUNTAIN VIEW	EECS
Software Engineer	Salesforce	SF	EECS
Open Source Maintainer	Github	SF	Computer Science
Big Data Engineer	Microsoft	REDMOND	Data Science
Data Analyst	Startup	BERKELEY	Data Science
Analyst Intern	Google	SF	Philosophy

Table 1: `survey` Table

Each record of the `survey` table is an entry corresponding to a student. We have the student's major information (`m_name`), company information (`c_name, c_location`), and the job title (`j_name`).

(a) Write a SQL query that selects all data science major graduates that got jobs in Berkeley. The result generated by your query should include all 4 columns.

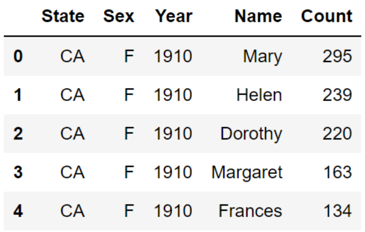
```
SELECT *  FROM survey
WHERE m-name = 'Data Science' AND      } this is all
      c-location = 'BERKELEY'          } info from the question.
```

(b) Write a SQL query to find the top 5 popular companies that data science graduates will work at, from most popular to 5th most popular.

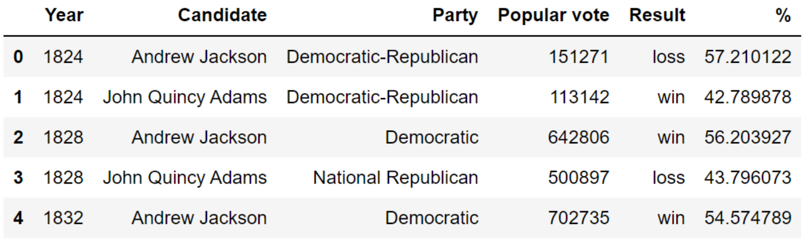
```
SELECT c_name, COUNT(*) as count
FROM survey
WHERE m-name = 'Data Science'
GROUP BY c-name
ORDER BY count DESC   } we want top 5, so
LIMIT 5                 we need to specify descending as
                        SQL's default sort order is
                        ascending
```

Pandas Practice

Throughout this section you'll be working with the babynames (left) and elections (right) datasets as shown below:



	State	Sex	Year	Name	Count
0	CA	F	1910	Mary	295
1	CA	F	1910	Helen	239
2	CA	F	1910	Dorothy	220
3	CA	F	1910	Margaret	163
4	CA	F	1910	Frances	134



	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789

4. (a) Using `groupby.agg` or one of the shorthand methods (`groupby.min`, `groupby.first`, etc.), create a Series `best_result` that gives the highest percentage vote ever attained by each party. For example, `best_result['Libertarian']` should return 3.3. The order of your Series does not matter.

   `best_result =` elections.groupby('Party').max()['%']

   the party col will end up as our index

   get out % col so we get a series

   use shortcut since we want max

   (b) Again using `groupby.agg` or one of the its shorthand methods, create a DataFrame `last_result` that gives the result for a party in its most recent year of participation, with `Party` as its index. For example `last_result.query("Party == 'Whig'")` should give you a row showing that the Whigs last participated in an election in 1852 with Winfield Scott as their candidate, earning 44% of the vote. This might take more than one line of code. Write your answer below.

   ```
   last_result = (elections.sort_values('Year', ascending=False)
                  .groupby('Party')
                  .first())
   ```

   wrapping in parens allows you to break lines to chain commands, e.g.

   sort by year first so that the first value after grouping is most recent year.

(c) Using `filter`, create a DataFrame `major_party_results_since_1988` that includes all election results starting in 1988, but only include a row if the Party it belongs to has earned at least 1% of the popular vote in ANY election since 1988.

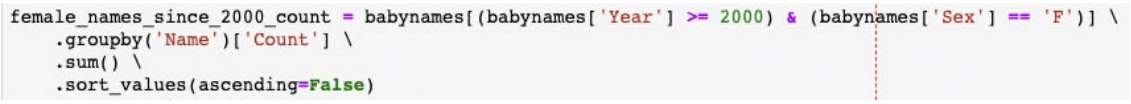
For example, in 1988, you should not include the `'New Alliance'` candidate since this party has not earned 1% of the vote since 1988. However, you should include the 'Libertarian' candidate from 1988 despite only having 0.47 percent of the vote in 1988 because in 2016 the Libertarian candidate Gary Johnson had 3.3% of the vote.

so we need to filter all groups + ensure that MAX val is > 1%

`major_party_results_since_1988 =` elections[elections['Year'] >= 1988] \
.groupby('Party'). filter(lambda f: f['%'].max() > 1.0)

(d) Create a Series `female_name_since_2000_count` which gives the total number of occurrences of each name for female babies born in California from the year 2000 or later. The index should be the name, and the value should be the total number of births. Your series should be ordered in decreasing order of count. For example, your first row should have index "Emily"and value 49605, because 49,605 Emilys have been born since the year 2000 in California.

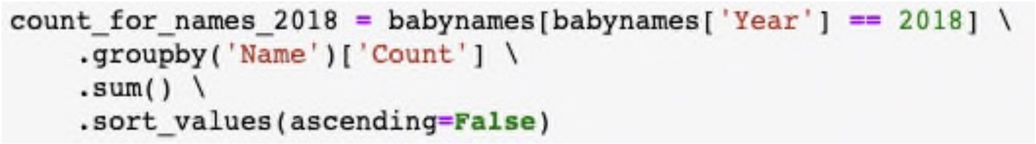
`female_name_since_2000_count =`



```
female_names_since_2000_count = babynames[(babynames['Year'] >= 2000) & (babynames['Sex'] == 'F')] \
    .groupby('Name')['Count'] \
    .sum() \
    .sort_values(ascending=False)
```

(e) Using `groupby`, create a Series `count_for_names_2018` listing all baby names from 2018 in decreasing order of popularity. The result should not be broken down by gender! If a name is used by both male and female babies, the number you provide should be the total across both genders. For example, `count_for_names_2018["Noah"]` should be the number 2567 because in 2018 there were 2567 Noahs born (12 female and 2555 male).

`count_for_names_2018 =`



```
count_for_names_2018 = babynames[babynames['Year'] == 2018] \
    .groupby('Name')['Count'] \
    .sum() \
    .sort_values(ascending=False)
```