SQL_Notebook_1_Elli Linek

June 21, 2018

1 SQL Notebook 1: What are the Characteristics of Jobs by Census Block?

In these workbooks, we will start with a motivating question, then walk through the process we need to go through in order to answer the motivating question. Along the way, we will walk through various SQL commands and develop skills as we work towards answering the question.

As you work, there will be headers that are in **GREEN**. These indicate locations where there is an accompanying video. This video may walk through the steps or expand on the topics discussed in that section. Though it isn't absolutely necessary to watch the video while working through this notebook, we highly recommend watching them at least once on your first time through.

If you have not yet watched the "Introduction to pgAdmin" video, watch it before you proceed!

You will also run into some headers that are in **RED**. These headers indicate a checkpoint to practice writing the code yourself. You should stop at these checkpoints and try doing the exercises and answering the questions posed in these sections. Overall, in the notebook examples, I'll use data from California, but I encourage you to use data from another state, which you can download from the LEHD website (see below). You can copy the same analysis that I do with your state of choice, then try to explore a little bit more, altering the code slightly to see what it does.

As you work through the notebook, make sure you have pgAdmin open and try running the queries as we go. That is, whenever we show some code, try copying and pasting it into the query tool in pgAdmin running it. This way, you can see the output as we work through the code, as well as write your own code when we get to the checkpoints.

1.1 Longitudinal Employer-Household Dynamics (LEHD) Data

In these workbooks, we will be using LEHD data. These are public-use data sets containing information about employers and employees. Information about the LEHD Data can be found at `https://lehd.ces.census.gov/`.

We will be using the LEHD Origin-Destination Employment Statistics (LODES) datasets in our applications in this workbook. Each state has three main types of files: Origin-Destination data, in which job totals are associated with a home and work Census block pair, Residence Area Characteristic data, in which job totals are by home Census block, and Workplace Area Characteristic data, in which job totals are by workplace Census block. In addition to these three, there is a "geographic crosswalk" file with descriptions of the Census Blocks as they appear the in the LODES datasets.

You can find more information about the LODES datasets here. **Understanding how these datasets are structured is crucial to being able to successfully construct queries and gain insight into the data, so make sure you take some time to read through the documentation and get familiar with what you're working with!**

In these examples, I will use data from California with "All Segments" and "All Jobs." At the checkpoints, I highly encourage you to think about what questions might be interesting and try downloading other data to use so you can get more practice coding in SQL.

1.2 Motivating Question (VIDEO)

The LODES data has a wealth of information about jobs at the census block level. We want to explore this, so that we can characterize the data that is available to us. That is, for any given state, we want to answer, for example, some of the following questions:

- **How many census blocks contain workplaces?**
- **What were the most jobs in a census block?**
- **How many census blocks had over 50 jobs? Over 100?**
- **Among census blocks containing workplaces, what is the average number of jobs per census block?**

These, as well as other questions about the data we might answer, can help us better understand the distribution of jobs by location. In this notebook, try to keep these types of questions in mind as we explore the data.

1.3 Starting Out: Introduction to SQL and Relational Databases

SQL is a language designed for a very specific purpose: to interact with relational databases.

- **Database**: A database is a structured collection of data. There are various different ways of structuring the database, and there may or may not be information about the relationship between entities in the database.
- **Query**: A query is a request for data from the database.
- **Database Management System (DBMS)**: A DBMS is a system of storing and managing databases, including querying the database.
- **Relational Database Management System (RDBMS)**: In an RDBMS, data records are stored in *tables*, each of which has a predefined set of *columns*, the pieces of information captured for each record in a table, and *rows* in the table, where each row has a place to store a value for every column in the table.

Tables, including their columns, column types and relationships with other tables, are defined in a database **schema**. Many times, tables will contain a **primary key**, one or more columns that uniquely define a row. You can think of the primary key as a kind of ID, in which each row is given a unique ID. Tables can also contain **foreign keys**, which are column(s) that comprise the primary key in another table and, thus, provides a way of matching between multiple tables.

In this class, we'll use data stored using the PostgreSQL RDBMS. In practice, there are many others that you could use, each with its own unique characteristics, but the basic principles of relational data will remain the same.

1.4 Writing a Basic Query (VIDEO)

In order to analyze the data in a database, we need to query it, or request specific information about the data.

Let's start with some basics. We'll start by retrieving all columns from the California Workplace Area Characteristic (`ca_wac_2015`) table. Try running the following query in pgAdmin:

```
SELECT * FROM lodes.ca_wac_2015 LIMIT 10;
```

You should see 10 rows of the `ca_wac_2015` dataset. Let's go over the basics of this SQL command.

- **SELECT:** We start out with the `SELECT` statement. The `SELECT` statement specifies which variables (columns) you want.
  - Here, we used `SELECT *`. The "`*`" just says that we want all the variables.
  - If we wanted a few columns, we would use the column names separated by commas instead of "`*`" (for example, `w_geocode, createdate`).
- **FROM:** Now, let's look at the next part of the query, "`FROM lodes.ca_wac_2015`". The data that we want is in the `ca_wac_2015` table in the `lodes` schema. This part of the query specifies the schema, `lodes`, and the table, `ca_wac_2015`, from which we want to retrieve the data, with a "`.`" between the two. Most of your queries will begin in this fashion, describing which columns you want, from which table, making sure to include schema information.
- **LIMIT:** We typically include a `LIMIT` statement at the end of our query so that we don't get overloaded with rows being output. Here, `LIMIT 10` means that we just want the first ten rows. Many times, the `LIMIT` that you want will be higher than 10 -- you might generally prefer to use 1000 or so. Having a `LIMIT` for all queries is highly recommended even if you know only a few rows will be shown, since it acts as a safety precaution against (for example) displaying millions of rows of data.

In this case, we've put everything in one line, but that's not necessary. We could have split the code up into multiple lines, like so:

```
SELECT *
FROM lodes.ca_wac_2015
LIMIT 10;
```

This gives the same output as our original query. Generally, once queries start getting longer, breaking up the code into multiple lines can be very helpful in organizing your code and making it easier to read.

Along those lines, note that we used a semi-colon at the end of the query to mark the end of the query. That isn't absolutely necessary when using PostgreSQL, but it does help mark the end of a query and is required in other applications of SQL, so it's good practice to use it.

1.4.1 Side note about capitalization

If you notice, we've been using all caps for SQL commands and all lowercase for data table and schema names. This is simply a convention, as SQL is not case sensitive. For example, we could have run "`select * from lodes.ca_wac_2015 limit 10;`" and it would have given us the exact same output as the first query.

This does mean you need to be careful when using column names. If your column name has capital letters in it, you need use double quotes (e.g. `"C000"`) to preserve the capitalization. For this reason, you might find that using all lowercase letters in column names is preferable, which is what we've done here.

Now, consider the following query. What do you think it will do?

```
SELECT w_geocode, createdate
FROM lodes.ca_wac_2015
LIMIT 100;
```

We've changed the original query by using "`w_geocode, createdate`" instead of "`*`", so we'll only get the values from two columns, `w_geocode` and `createdate`. In addition, we've changed the value after `LIMIT` to be 100 instead of 10, so we'll get the first 100 rows instead of the first 10 rows.

1.5 Checkpoint 1: Running Basic Queries

Consider the following queries. What do you think they will do? Try figuring out what the output will look like, then run the code to see if you're correct.

1.5.1 -> short note on the selected data: since i am not working ith the virtual box i had to load the data on my own, in order to follow your examples in a better way idecided to load al the csv files for CA, if there is enough time i will load New york files as well and play around with the data a bit.

- `SELECT * FROM lodes.ca_wac_2015 LIMIT 25;`

-> selecting all columns from the california wac data set, limited to the first 25 rows, which are going to be presented as some kind of preview within the "editor" if i could name it like that

- `SELECT c000,ca01,ca02,ca03 FROM lodes.ca_wac_2015 LIMIT 1000;`

-> selecting only the four selected or named columns (total number of jobs and 3 columns regarding the number of jobs per age groups) from the california wac data set, the limit is set to the first 1000 rows

- `SELECT * FROM lodes.ca_od_2015 LIMIT 100;`

-> again all columns from the california od file/table, limited to the first 100 rows

- `SELECT * FROM lodes.ca_rac_2015 LIMIT 40;`

-> again all columns but that time from the california rac table, limited to the first 40 rows
Think about the following scenarios. What is the query you would use to answer these questions? Try them out.

- You want to see the first 100 rows of the origin and destination geocodes for each census block in California.

queries for ca and ny

```
SELECT w_geocode, h_geocode
FROM lodes.ca_od_main_jt00_2015
LIMIT 100;

SELECT w_geocode, h_geocode
FROM lodes.ny_od_main_jt00_2015
LIMIT 100;
```

- You want to see the top 1000 rows of census blocks containing workplaces and the number of jobs for workers of each race.

queries for ca and ny

```
SELECT C000, CR01, CR02, CR03, CR04, CR05, CR07
FROM lodes.ca_wac_s000_jt00_2015
LIMIT 1000;

SELECT C000, CR01, CR02, CR03, CR04, CR05, CR07
FROM lodes.ny_wac_s000_jt00_2015
LIMIT 1000;
```

(i included the total number of jobs)

1.6 Checking Number of Rows and Duplicates (VIDEO)

Let's say we want to find out how many rows there are. You can do this by using a COUNT.

```
SELECT COUNT(*)
FROM lodes.ca_wac_2015
LIMIT 1000;
```

Here, we used COUNT(*), which does a count of all rows, regardless of NULL values. We can instead do a count of all non-NULL values of a certain variable by including that variable instead of *.

```
SELECT COUNT(w_geocode)
FROM lodes.ca_wac_2015
LIMIT 1000;
```

But wait; what if there are duplicates in the data? We can check for them by using DISTINCT.

```
SELECT DISTINCT(w_geocode)
FROM lodes.ca_wac_2015
LIMIT 1000;
```

This shows us all of the rows with distinct `w_geocode` values; that is, all of the distinct census block ids. Let's count how many there are. To count them, all we have to do is put `COUNT()` around the `DISTINCT` part.

```
SELECT COUNT(DISTINCT(w_geocode))
FROM lodes.ca_wac_2015
LIMIT 1000;
```

1.6.1 Building Up a Query

Notice that we wanted to count the number of distinct rows, but we first started from querying the rows with distinct `w_geocode` first before adding in the `COUNT`. Though this is a simple example, this process of building up a query as we go is important, especially when we get to more complicated tasks. When writing a query, try to think about the basic parts first, and feel free to run intermediate steps (making sure to include `LIMIT`) as you go.

1.7 Using Conditional Statements (VIDEO)

Suppose we want to look at a subset of the data. We can use conditional statements to do this.

```
SELECT *
FROM lodes.ca_wac_2015
WHERE c000 < 100
LIMIT 1000;
```

Here, we use `WHERE` to insert a conditional statement. The above code will give all rows with `c000` less than 100.

Using a query like the one above can be useful for finding if there are any data entry errors or missing values. Since it's not possible to have a number of jobs less 0, if there are any rows with negative values, this is likely an error or the method used to code missing values (e.g. `-1`).

We can also use more complicated conditional statements.

```
SELECT count(*)
FROM lodes.ca_wac_2015
WHERE (c000 > 50) AND (c000 < 100)
LIMIT 1000;
```

This subsets to rows in which `c000` is greater than 50 and `c000` is less than 100. That is, this subsets to census blocks with between 50 and 100 total jobs. Using `OR` works in the same way.

```
SELECT count(*)
FROM lodes.ca_wac_2015
WHERE (c000 <= 50) OR (c000 >= 100)
LIMIT 1000;
```

This subsets to rows in which `c000` is less than or equal to 50 or `c000` is greater than or equal to 100. This query should, in other words, capture the rest of the rows.

1.7.1 Common Comparison Operators

Though there are some more complicated comparison operators (if you're curious, feel free to look up what `LIKE` and `IN` do), these should cover most of what you want to do. - "=": equal to - "!=" or "<>": not equal to - "<": less than - "<=": less-than-or-equal-to - ">": greater than - ">=": greater-than-or-equal-to - "`IS NULL` and "`IS NOT NULL`": The signifier of a row in a column not having a value is a special keyword: `NULL`. To check for `NULL`, you use "`IS NULL`" or "`IS NOT NULL`", rather than "=" or "!=". For example, to count the number of rows with `NULL` values for `c000` we might use the following:

```
SELECT count(*)
FROM lodes.ca_wac_2015
WHERE c000 IS NULL
LIMIT 1000;
```

1.8 Creating Variables

Suppose we want to create a new column in the table that acts as a "flag" for which rows fit a certain condition, so that you can use them later. We can do this using the `ALTER TABLE` statement. Try running the following code:

```
ALTER TABLE lodes.ca_wac_2015 ADD over100 BOOL;
UPDATE lodes.ca_wac_2015 SET over100 = False;
UPDATE lodes.ca_wac_2015 SET over100 = True WHERE c000 > 100;
```

Let's break this down line by line. First, we use `ALTER TABLE`, then specify the table we want to alter. In this case, we want to alter the `ca_wac_2015` table in the `lodes` schema. Then, we `ADD` a new column, `over100`. We designate this as a boolean (that is, a TRUE/FALSE value) column.

After we create this new column, we need to fill it with the appropriate values. First, we're going to set everything in the column to be False. To do this, we use `UPDATE`, specify the appropriate table, then use `SET over100 = False`. Then, we replace the value with True if the value in `c000` for that row is above 100. We again use `UPDATE` in a similar manner, except we add a `WHERE` clause, so that it only set the value to TRUE if a certain condition is met -- in this case, that `c000 > 100`.

1.9 Checkpoint 2: Counting Rows, Using Conditional Statements and Creating Variables

We've included the California 2015 OD, RAC, and WAC, as well as the geography crosswalk data for you in tables. Try using the methods described in this section to further explore the tables. Feel free to bring in data from a different state (using the document about bringing in data into PostgreSQL), or use the California data provided for you. Answer the questions below, making sure to write out the queries used to answer the questions.

From here on the results of the queries are listed as comments below each query

- How many census blocks contain more than 200 jobs?

queries for ca and ny

```
SELECT COUNT(*)
FROM lodes.ca_wac_s000_jt00_2015
WHERE c000 > 200
LIMIT 1000;
-- "16093"

SELECT COUNT(*)
FROM lodes.ny_wac_s000_jt00_2015
WHERE c000 > 200
LIMIT 1000;
-- "7813"
```

- How many census blocks contain residences of fewer than 25 workers?

queries for ca and ny

```
SELECT COUNT(*)
FROM lodes.ca_rac_s000_jt00_2015
WHERE c000 < 25
LIMIT 1000;
-- "198469"

SELECT COUNT(*)
FROM lodes.ny_rac_s000_jt00_2015
WHERE c000 < 25
LIMIT 1000;
-- "140267"
```

- How many census blocks contain workplaces with more than 10 workers with a Bachelor's degree or higher? combination of two variables

queries for ca and ny

```
SELECT COUNT(*)
FROM lodes.ca_wac_s000_jt00_2015
WHERE c000 >10 AND cd04 > 0
LIMIT 1000;
-- "97629"

SELECT COUNT(*)
FROM lodes.ny_wac_s000_jt00_2015
WHERE c000 >10 AND cd04 > 0
LIMIT 1000;
-- "51607"
```

- How many counties are there?

queries for ca and ny

```
SELECT COUNT(DISTINCT(cty))
FROM lodes.ca_xwalk
LIMIT 1000;
-- "61"

SELECT COUNT(DISTINCT(cty))
FROM lodes.ny_xwalk
LIMIT 1000;
-- "63"
```

- How many total census blocks are there?

queries for ca and ny -- All blocks are listed in the xwalk table. tabblk2010 uniquely identifies the census blocks.

```
SELECT COUNT(*)
FROM lodes.ca_xwalk
LIMIT 1000;
-- "710145"

SELECT COUNT(*)
FROM lodes.ny_xwalk
LIMIT 1000;
-- "350169"
```

- How many Metropolitan/Micropolitan areas are there?

queries for ca and ny -- I got different results by counting cbsa and cbsaname. Looking closer at the data i discovered counting cbsa only I would count one more item which is 99999 i.e. an invalid value. The cbsaname in these cases is null. Since COUNT does not count null values hence we can omit an additional WHERE clause.

```
SELECT COUNT(DISTINCT(cbsaname))
FROM lodes.ca_xwalk
LIMIT 1000;
-- "36"


SELECT COUNT(DISTINCT(cbsaname))
FROM lodes.ny_xwalk
LIMIT 1000;
-- "26"
```

1.10 Using Aggregation Functions (VIDEO)

We've created a variable that indicates whether the census block had over 100 jobs or not. What if we wanted to know how many blocks had over 100 jobs and how many didn't? We can now use the `GROUP BY` statement.

```
SELECT over100, COUNT(over100)
FROM lodes.ca_wac_2015
GROUP BY over100
LIMIT 1000;
```

Here, the `GROUP BY` statement groups it into the categories of the variable. Since we've chosen to display the count, we can see the counts. We can also change the order in which the results are displayed so that it's in increasing order.

```
SELECT over100, COUNT(over100)
FROM lodes.ca_wac_2015
GROUP BY over100
ORDER BY COUNT(over100)
LIMIT 1000;
```

The `ORDER BY` statement orders the rows that it displays according to whatever you put after it. In this case, we chose the count of `over100`.

1.10.1 Using GROUP BY with Multiple Variables

For the next few queries, let's try using a different table. The `ca_xwalk` table in the same `lodes` schema contains information about each of the census blocks in California. We can use this to, for example, look at aggregation by CBSA (metropolitan/micropolitan area) name and by county name.

```
SELECT cbsaname, ctyname, COUNT(*)
FROM lodes.ca_xwalk
GROUP BY cbsaname, ctyname
ORDER BY COUNT(*) DESC
LIMIT 1000;
```

This first groups by CBSA (`cbsaname`) name, then it groups by county (`ctyname`), in that order. In this case, county is nested completely in the metropolitan/micropolitan area. In other cases in which we don't have complete nesting, we would be able to see all possible combinations that exist in the data.

Further, notice that we used `DESC` after `ORDER BY`. This orders in descending order instead of ascending order, so that we can see the areas with the most census blocks at the top.

1.10.2 Conditional Statements After Aggregation

Suppose we wanted to display only certain counts. We can use `HAVING` to do this.

```
SELECT ctyname, cbsaname, COUNT(cbsaname)
FROM lodes.ca_xwalk
GROUP BY ctyname, cbsaname
HAVING count(cbsaname) > 20000
ORDER BY COUNT(*) DESC
LIMIT 1000;
```

This will only display the counts for which the count of `cbsaname` is greater than 20000. Note that this is different from using `WHERE`, since the conditional statement comes after the `GROUP BY` statement. Basically, `HAVING` gives us a way of using the same types of conditional statements after we do our aggregation.

1.10.3 Using Different Aggregation Functions

What if we wanted to find the sum within each group, or the minimum or maximum value? We can use the appropriate aggregation function. To show this, let's go back to our `ca_wac_2015` table.

```
SELECT over100, COUNT(over100), AVG(c000) AS avg_jobs, MIN(c000), MAX(c000)
FROM lodes.ca_wac_2015
GROUP BY over100
ORDER BY over100
LIMIT 1000;
```

Here, we're finding the counts, average, minimum, and maximum value of the total jobs in each census block within each group. Now, we're not doing anything very insightful here, since the groups already split the blocks by how many jobs there are. However, as we'll see later on, these aggregation functions can be very useful. For example, suppose we had the county data that's in `ca_xwalk` in this table. We could find the average number of jobs per census block for each county in this way.

1.10.4 Side Note: Aliasing

You may have noticed that we included a part using "`AS`," followed by a new name, in the first line. When you ran the code, you might have noticed that the column labels were changed to these new names. This is called aliasing, and is done for readability and ease of access. Later on, aliasing will also help us more easily reference tables within the same query.

1.11 Checkpoint 3: Checking Your Dataset

Using the above methods, explore the tables we've provided or your own state's data to answer the questions below. As before, make sure to include the queries with your answers.

- Which county has the most census blocks?

queries for ca and ny -- Limiting to just one result which is the answer.

```sql
SELECT ctyname
FROM lodes.ca_xwalk
GROUP BY ctyname
ORDER BY COUNT(*) DESC
LIMIT 1;
-- "Los Angeles County, CA"

SELECT ctyname
```

```
FROM lodes.ny_xwalk
GROUP BY ctyname
ORDER BY COUNT(*) DESC
LIMIT 1;
-- "Suffolk County, NY"
```

- Which Metropolitan/Micropolitan area has the most census blocks?

queries for ca and ny -- Ignoring non Metropolitan/Micropolitan areas by using the where clause.

```
SELECT cbsaname
FROM lodes.ca_xwalk
WHERE cbsaname is not NULL
GROUP BY cbsaname
ORDER BY COUNT(*) DESC
LIMIT 1;
-- "Los Angeles-Long Beach-Anaheim, CA"

SELECT cbsaname FROM lodes.ny_xwalk
WHERE cbsaname is not NULL
GROUP BY cbsaname
ORDER BY COUNT(*) DESC
LIMIT 1;
-- "New York-Newark-Jersey City, NY-NJ-PA"
```

- Which Origin census block - Destination census block combination has the most workers? How many workers are in this combination?

queries for ca and ny

- s000 represents the number of workers.

```
SELECT w_geocode, h_geocode, s000
FROM lodes.ca_od_main_jt00_2015
ORDER BY s000 DESC
LIMIT 1;
-- "060730083051009"    "060730083632000"   213

SELECT w_geocode, h_geocode, s000
FROM lodes.ny_od_main_jt00_2015
ORDER BY s000 DESC
LIMIT 1;
-- "360610203002000"    "360610207012001"   184
```

- How would you find all counties containing at least 1000 census blocks?

queries for ca and ny

```
SELECT ctyname
FROM lodes.ca_xwalk
GROUP BY ctyname
HAVING COUNT(*) > 1000
ORDER BY COUNT(*)
LIMIT 1000;
-- 57 counties were returned. results listed in the appendix.

SELECT ctyname
FROM lodes.ny_xwalk
GROUP BY ctyname
HAVING COUNT(*) > 1000
ORDER BY COUNT(*)
LIMIT 1000;
-- 62 counties were returned. results listed in the appendix.
```

- For California, how many census blocks are there with a latitude above +36?

queries for ca

```
SELECT COUNT(DISTINCT(ctyname))
FROM lodes.ca_xwalk
WHERE blklatdd > 36
LIMIT 1000;
-- 50
```

- For California, which county has the most census blocks above the +36 latitude line? Which county has the most below?

queries for ca some googleing later I found https://www.w3resource.com/sql/aggregate-functions/max-count.php

```
SELECT ctyname,count(ctyname)
FROM lodes.ca_xwalk
WHERE blklatdd > 36
GROUP BY ctyname
HAVING count(ctyname) = (
    SELECT MAX(sub.mycount)
    FROM (
        SELECT count(ctyname) as mycount
        FROM lodes.ca_xwalk
        WHERE blklatdd > 36
        GROUP BY ctyname
    ) as sub
) OR count(ctyname) = (
    SELECT MIN(sub.mycount)
    FROM (
        SELECT count(ctyname) as mycount
        FROM lodes.ca_xwalk
        WHERE blklatdd > 36
        GROUP BY ctyname
    ) as sub
)
-- "Alameda County, CA" "23956"
-- "Lake County, OR"    "1"
```

1.12 Appendix

1.12.1 Results of 'How would you find all counties containing at least 1000 census blocks?' (ca)

```
"Amador County, CA"
"Mariposa County, CA"
"Sierra County, CA"
"Del Norte County, CA"
"Colusa County, CA"
"San Benito County, CA"
"Sutter County, CA"
"Napa County, CA"
"Glenn County, CA"
"Calaveras County, CA"
"Yuba County, CA"
"Mono County, CA"
"Trinity County, CA"
"Yolo County, CA"
"Marin County, CA"
"Nevada County, CA"
"Plumas County, CA"
"Tuolumne County, CA"
"Madera County, CA"
"Santa Cruz County, CA"
"Lake County, CA"
"Modoc County, CA"
"Inyo County, CA"
"Tehama County, CA"
"Lassen County, CA"
"El Dorado County, CA"
"Kings County, CA"
"Butte County, CA"
"Merced County, CA"
"San Francisco County, CA"
"Mendocino County, CA"
"Stanislaus County, CA"
"Imperial County, CA"
"Placer County, CA"
```

```
"San Mateo County, CA"
"Humboldt County, CA"
"Siskiyou County, CA"
"Solano County, CA"
"Santa Barbara County, CA"
"Sonoma County, CA"
"Shasta County, CA"
"Monterey County, CA"
"San Luis Obispo County, CA"
"San Joaquin County, CA"
"Tulare County, CA"
"Ventura County, CA"
"Contra Costa County, CA"
"Sacramento County, CA"
"Fresno County, CA"
"Santa Clara County, CA"
"Alameda County, CA"
"Kern County, CA"
"Riverside County, CA"
"Orange County, CA"
"San Diego County, CA"
"San Bernardino County, CA"
"Los Angeles County, CA"
```

1.12.2 Results of 'How would you find all counties containing at least 1000 census blocks?' (ny)

```
"Orleans County, NY"
"Hamilton County, NY"
"Schuyler County, NY"
"Yates County, NY"
"Wyoming County, NY"
"Seneca County, NY"
"Cortland County, NY"
"Fulton County, NY"
"Genesee County, NY"
"Putnam County, NY"
"Tioga County, NY"
"Montgomery County, NY"
"Schoharie County, NY"
"Greene County, NY"
"Warren County, NY"
"Washington County, NY"
"Tompkins County, NY"
"Lewis County, NY"
"Chemung County, NY"
"Franklin County, NY"
"Essex County, NY"
```

```
"Schenectady County, NY"
"Madison County, NY"
"Livingston County, NY"
"Chenango County, NY"
"Clinton County, NY"
"Cayuga County, NY"
"Wayne County, NY"
"Ontario County, NY"
"Columbia County, NY"
"New York County, NY"
"Allegany County, NY"
"Herkimer County, NY"
"Otsego County, NY"
"Rockland County, NY"
"Niagara County, NY"
"Delaware County, NY"
"Rensselaer County, NY"
"Richmond County, NY"
"Cattaraugus County, NY"
"Sullivan County, NY"
"Oswego County, NY"
"Bronx County, NY"
"Broome County, NY"
"Ulster County, NY"
"Saratoga County, NY"
"Chautauqua County, NY"
"Jefferson County, NY"
"Dutchess County, NY"
"Steuben County, NY"
"Albany County, NY"
"St. Lawrence County, NY"
"Oneida County, NY"
"Kings County, NY"
"Onondaga County, NY"
"Orange County, NY"
"Monroe County, NY"
"Erie County, NY"
"Queens County, NY"
"Westchester County, NY"
"Nassau County, NY"
"Suffolk County, NY"
```