Chapter 4

October 28, 2019

```
In [14]: %reload_ext sql
```

```
In [15]: %sql postgresql://postgres:postgres@localhost:5432/analysis
```

```
Out[15]: 'Connected: postgres@analysis'
```

1 Importing and Exporting Data

```
In [15]: # COPY table_name
         # FROM '/Users/ugurtigu/Documents/Learn/Docs/SQL/my_file.csv'
         # WITH (FORMAT CSV, HEADER);
```

- this block of code starts with the COPY keyword followed by the name of the target which must already exist in your database
- "Copy data to my table called table_name"
- the FROM keyword indentifies the full path to the source file
- the WITH keyword lets you specify options surrounded by parameters
- here we specify that the external file should be comma-delimited and that we exclude the header:
  - Input and output file format **FORMAT**
  - Presence of a header row **HEADER** or **HEADER ON**
  - Delimiter **DELIMITER 'character'**
  - Quote character **QUOTE 'quote_character'**

```
In [16]: %%sql

         CREATE TABLE us_counties_2010 (
             geo_name varchar(90),                    -- Name of the geography
             state_us_abbreviation varchar(2),        -- State/U.S. abbreviation
             summary_level varchar(3),                -- Summary Level
             region smallint,                         -- Region
             division smallint,                       -- Division
             state_fips varchar(2),                   -- State FIPS code
             county_fips varchar(3),                  -- County code
             area_land bigint,                        -- Area (Land) in square meters
             area_water bigint,                       -- Area (Water) in square meters
             population_count_100_percent integer,    -- Population count (100%)
```

```
housing_unit_count_100_percent integer,  -- Housing Unit count (100%)
internal_point_lat numeric(10,7),        -- Internal point (latitude)
internal_point_lon numeric(10,7),        -- Internal point (longitude)

-- This section is referred to as P1. Race:
p0010001 integer,   -- Total population
p0010002 integer,   -- Population of one race:
p0010003 integer,       -- White Alone
p0010004 integer,       -- Black or African American alone
p0010005 integer,       -- American Indian and Alaska Native alone
p0010006 integer,       -- Asian alone
p0010007 integer,       -- Native Hawaiian and Other Pacific Islander alone
p0010008 integer,       -- Some Other Race alone
p0010009 integer,   -- Population of two or more races
p0010010 integer,   -- Population of two races:
p0010011 integer,       -- White; Black or African American
p0010012 integer,       -- White; American Indian and Alaska Native
p0010013 integer,       -- White; Asian
p0010014 integer,       -- White; Native Hawaiian and Other Pacific Islander
p0010015 integer,       -- White; Some Other Race
p0010016 integer,       -- Black or African American; American Indian and Alaska
p0010017 integer,       -- Black or African American; Asian
p0010018 integer,       -- Black or African American; Native Hawaiian and Other Pa
p0010019 integer,       -- Black or African American; Some Other Race
p0010020 integer,       -- American Indian and Alaska Native; Asian
p0010021 integer,       -- American Indian and Alaska Native; Native Hawaiian and
p0010022 integer,       -- American Indian and Alaska Native; Some Other Race
p0010023 integer,       -- Asian; Native Hawaiian and Other Pacific Islander
p0010024 integer,       -- Asian; Some Other Race
p0010025 integer,       -- Native Hawaiian and Other Pacific Islander; Some Other
p0010026 integer,   -- Population of three races
p0010047 integer,   -- Population of four races
p0010063 integer,   -- Population of five races
p0010070 integer,   -- Population of six races

-- This section is referred to as P2. HISPANIC OR LATINO, AND NOT HISPANIC OR LATI
p0020001 integer,   -- Total
p0020002 integer,   -- Hispanic or Latino
p0020003 integer,   -- Not Hispanic or Latino:
p0020004 integer,   -- Population of one race:
p0020005 integer,       -- White Alone
p0020006 integer,       -- Black or African American alone
p0020007 integer,       -- American Indian and Alaska Native alone
p0020008 integer,       -- Asian alone
p0020009 integer,       -- Native Hawaiian and Other Pacific Islander alone
p0020010 integer,       -- Some Other Race alone
p0020011 integer,   -- Two or More Races
p0020012 integer,   -- Population of two races
```

```
            p0020028 integer,   -- Population of three races
            p0020049 integer,   -- Population of four races
            p0020065 integer,   -- Population of five races
            p0020072 integer,   -- Population of six races

            -- This section is referred to as P3. RACE FOR THE POPULATION 18 YEARS AND OVER
            p0030001 integer,   -- Total
            p0030002 integer,   -- Population of one race:
            p0030003 integer,       -- White alone
            p0030004 integer,       -- Black or African American alone
            p0030005 integer,       -- American Indian and Alaska Native alone
            p0030006 integer,       -- Asian alone
            p0030007 integer,       -- Native Hawaiian and Other Pacific Islander alone
            p0030008 integer,       -- Some Other Race alone
            p0030009 integer,   -- Two or More Races
            p0030010 integer,   -- Population of two races
            p0030026 integer,   -- Population of three races
            p0030047 integer,   -- Population of four races
            p0030063 integer,   -- Population of five races
            p0030070 integer,   -- Population of six races

            -- This section is referred to as P4. HISPANIC OR LATINO, AND NOT HISPANIC OR LATI
            -- FOR THE POPULATION 18 YEARS AND OVER
            p0040001 integer,   -- Total
            p0040002 integer,   -- Hispanic or Latino
            p0040003 integer,   -- Not Hispanic or Latino:
            p0040004 integer,   -- Population of one race:
            p0040005 integer,   -- White alone
            p0040006 integer,   -- Black or African American alone
            p0040007 integer,   -- American Indian and Alaska Native alone
            p0040008 integer,   -- Asian alone
            p0040009 integer,   -- Native Hawaiian and Other Pacific Islander alone
            p0040010 integer,   -- Some Other Race alone
            p0040011 integer,   -- Two or More Races
            p0040012 integer,   -- Population of two races
            p0040028 integer,   -- Population of three races
            p0040049 integer,   -- Population of four races
            p0040065 integer,   -- Population of five races
            p0040072 integer,   -- Population of six races

            -- This section is referred to as H1. OCCUPANCY STATUS
            h0010001 integer,   -- Total housing units
            h0010002 integer,   -- Occupied
            h0010003 integer    -- Vacant
        );

 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[16]: []

In [17]: %%sql

         SELECT * from us_counties_2010;

 * postgresql://postgres:***@localhost:5432/analysis
0 rows affected.


Out[17]: []
```

- we created a table without inserted data

```
In [18]: %%sql

         COPY us_counties_2010
         FROM '/Users/ugurtigu/Documents/Learn/Docs/SQL/us_counties_2010.csv'
         WITH (FORMAT CSV, HEADER);

 * postgresql://postgres:***@localhost:5432/analysis
3143 rows affected.


Out[18]: []
```

- our target will be our created table us_counties_2010
- we give the full path of the csv file
- we use the WITH keyword to use some parameters

```
In [19]: %%sql

         SELECT geo_name, state_us_abbreviation, area_land
         FROM us_counties_2010
         ORDER BY area_land DESC
         LIMIT 3;

 * postgresql://postgres:***@localhost:5432/analysis
3 rows affected.


Out[19]: [('Yukon-Koyukuk Census Area', 'AK', 376855656455),
          ('North Slope Borough', 'AK', 229720054439),
          ('Bethel Census Area', 'AK', 105075822708)]
```

- we check if the data is loaded correct

```
In [20]: %%sql

         SELECT geo_name, state_us_abbreviation, internal_point_lon
         FROM us_counties_2010
         ORDER BY internal_point_lon DESC
         LIMIT 5;

 * postgresql://postgres:***@localhost:5432/analysis
5 rows affected.


Out[20]: [('Aleutians West Census Area', 'AK', Decimal('178.3388130')),
          ('Washington County', 'ME', Decimal('-67.6093542')),
          ('Hancock County', 'ME', Decimal('-68.3707034')),
          ('Aroostook County', 'ME', Decimal('-68.6494098')),
          ('Penobscot County', 'ME', Decimal('-68.6574869'))]
```

- next we check the longitude

1.1 Importing a Subset of Columns with COPY

```
In [22]: %%sql

         CREATE TABLE supervisor_salaries (
         town varchar(30),
         county varchar(30),
         supervisor varchar(30),
         start_date date,
         salary money,
         benefits money)

 * postgresql://postgres:***@localhost:5432/analysis
Done.


Out[22]: []
```

- because our csv file has just 3 columns and we defined our table with more than this, we will get an error
- to prevent such an error we do this:

```
In [30]: %pwd

Out[30]: '/Users/ugurtigu/Documents/Learn/Docs/SQL'

In [32]: %%sql

         COPY supervisor_salaries (town, supervisor, salary)
         FROM '/Users/ugurtigu/Documents/Learn/Docs/SQL/supervisor_salaries.csv'
         WITH (FORMAT CSV, HEADER);
```

```
 * postgresql://postgres:***@localhost:5432/analysis
5 rows affected.
```

Out[32]: []

- the columns after the table name in the parantheses will just make sql look for data to fill those columns when it reads the CSV

In [33]:
```
%%sql

        SELECT * from supervisor_salaries;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
5 rows affected.
```

Out[33]:
```
[('Anytown', None, 'Jones', None, '$27,000.00', None),
 ('Bumblyburg', None, 'Baker', None, '$24,999.00', None),
 ('Moetown', None, 'Smith', None, '$32,100.00', None),
 ('Bigville', None, 'Kao', None, '$31,500.00', None),
 ('New Brillig', None, 'Carroll', None, '$72,690.00', None)]
```

1.2 Adding a default Value to a Column During Import

In [34]:
```
%%sql

        DELETE FROM supervisor_salaries;
```

```
 * postgresql://postgres:***@localhost:5432/analysis
5 rows affected.
```

Out[34]: []

- we first delete the data of our table we created earlier

In [36]:
```
%%sql

        CREATE TEMPORARY TABLE supervisor_salaries_temp (LIKE supervisor_salaries);
```

```
 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

Out[36]: []

- we first create a temporary table calles supervisor_salaries_temp based on the original passing the keyword LIKE followed by the parent table

```
In [38]: %%sql

         COPY supervisor_salaries_temp (town, supervisor, salary)
         FROM '/Users/ugurtigu/Documents/Learn/Docs/SQL/supervisor_salaries.csv'
         WITH (FORMAT CSV, HEADER);

 * postgresql://postgres:***@localhost:5432/analysis
5 rows affected.


Out[38]: []
```

- then we import the supervisor_salaries.csw fole into the temporary table using the COPY syntax

```
In [39]: %%sql

         INSERT INTO supervisor_salaries (town, county, supervisor, salary)
         SELECT town, 'Some County', supervisor, salary
         FROM supervisor_salaries_temp;

 * postgresql://postgres:***@localhost:5432/analysis
5 rows affected.


Out[39]: []
```

- we use an INSERT statement to fill the salaries table
- insetad of specifying values we employ a SELECT statement to query the temporary table
- for the second column we specify not a column name, but as a string inside single quotes

```
In [40]: %%sql

         DROP TABLE supervisor_salaries_temp;

 * postgresql://postgres:***@localhost:5432/analysis
Done.


Out[40]: []
```

- finally we erase the table with the command DROP TABLE
- the temp table will automatically disappear when you disconnect

```
In [41]: %%sql

         SELECT * from supervisor_salaries;

 * postgresql://postgres:***@localhost:5432/analysis
5 rows affected.
```

```
Out[41]: [('Anytown', 'Some County', 'Jones', None, '$27,000.00', None),
          ('Bumblyburg', 'Some County', 'Baker', None, '$24,999.00', None),
          ('Moetown', 'Some County', 'Smith', None, '$32,100.00', None),
          ('Bigville', 'Some County', 'Kao', None, '$31,500.00', None),
          ('New Brillig', 'Some County', 'Carroll', None, '$72,690.00', None)]
```

- now we have filled the county field with a string value

1.3 Using COPY to Export Data

```
In [42]: %%sql

         COPY us_counties_2010
         TO '/Users/ugurtigu/Documents/Learn/Docs/SQL/us_counties_export.txt'
         WITH (FORMAT CSV, HEADER, DELIMITER '|');

 * postgresql://postgres:***@localhost:5432/analysis
3143 rows affected.


Out[42]: []
```

- **exporting all data**
- we use the copy statement again
- this time we use *TO* instead of *FROM*
- we change the delimiter to |
- we can transform to any other text file format than csv, too

```
In [44]: %%sql

         COPY us_counties_2010 (geo_name, internal_point_lat, internal_point_lon)
         TO '/Users/ugurtigu/Documents/Learn/Docs/SQL/us_counties_latlon_export.txt'
         WITH (FORMAT CSV, HEADER, DELIMITER '|');

 * postgresql://postgres:***@localhost:5432/analysis
3143 rows affected.


Out[44]: []
```

- **exporting particular columns**
- we don't export all the data
- the columns in the paranthesis will be included
- note that the column names should be precisely as they're listed in the original table, to recognize them

```
In [12]: def file_read_from_head(fname, nlines):
             """Shows the n lines of a file."""
             from itertools import islice
```

```
            with open(file) as f:
                for line in islice(f, nlines):
                    print(line)

        file = 'us_counties_latlon_export.txt'
        file_read_from_head(file, 5)
```

```
geo_name|internal_point_lat|internal_point_lon

Autauga County|32.5363818|-86.6444901

Baldwin County|30.6592183|-87.7460666

Barbour County|31.8706701|-85.4054562

Bibb County|33.0158929|-87.1271475
```

```
In [19]: %%sql

         COPY (
         SELECT geo_name, state_us_abbreviation
         FROM us_counties_2010
         WHERE geo_name ILIKE '%mill%'
         )
         TO '/Users/ugurtigu/Documents/Learn/Docs/SQL/us_counties_mill_export.txt'
         WITH (FORMAT CSV, HEADER, DELIMITER '|');

 * postgresql://postgres:***@localhost:5432/analysis
9 rows affected.


Out[19]: []
```

- **Exporting Query Results**
- we use the *COPY* statement and insert a query inside that statement
- which will be fine-tuned with the ILIKE keyword
- so this is what our txt file looks like

```
In [7]: file = 'us_counties_mill_export.txt'
        file_read_from_head(file, 5)
```

```
geo_name|state_us_abbreviation

Miller County|AR

Miller County|GA
```

```
Vermillion County|IN

Mills County|IA
```

1.3.1 Tasks

– 1. Write a WITH statement to include with COPY to handle the import of an – imaginary text file that has a first couple of rows that look like this:
– id:movie:actor – 50:#Mission: Impossible#:Tom Cruise

```
In [18]: %%sql

         DROP TABLE actors;

 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[18]: []
```

```
In [19]: %%sql

         CREATE TABLE actors (
             id integer,
             movie text,
             actor text
         );

 * postgresql://postgres:***@localhost:5432/analysis
Done.
```

```
Out[19]: []
```

- we first create an empty table

```
In [20]: %%sql

         SELECT * from actors;

 * postgresql://postgres:***@localhost:5432/analysis
0 rows affected.
```

```
Out[20]: []
```

```
In [21]: %%sql

         COPY actors
         TO '/Users/ugurtigu/Documents/Learn/Docs/SQL/movies.txt'
         WITH (FORMAT CSV, HEADER, DELIMITER ':', QUOTE '#');

 * postgresql://postgres:***@localhost:5432/analysis
0 rows affected.


Out[21]: []
```

- we export the table into the movies.txt file from our actors table with the format we wish to have
- lets read the two lines of the file

```
In [22]: %%sql

         COPY actors
         FROM '/Users/ugurtigu/Documents/Learn/Docs/SQL/movies.txt'
         WITH (FORMAT CSV, HEADER, DELIMITER ':', QUOTE '#');

 * postgresql://postgres:***@localhost:5432/analysis
0 rows affected.


Out[22]: []

In [24]: %%sql

         SELECT * from actors;

 * postgresql://postgres:***@localhost:5432/analysis
0 rows affected.


Out[24]: []

In [25]: %%sql

         INSERT INTO actors
         VALUES (50,'Mission: Impossible', 'Tom Cruise')

 * postgresql://postgres:***@localhost:5432/analysis
1 rows affected.


Out[25]: []
```

```
In [26]: %%sql

         SELECT * from actors;

 * postgresql://postgres:***@localhost:5432/analysis
1 rows affected.


Out[26]: [(50, 'Mission: Impossible', 'Tom Cruise')]
```

– 2. Using the table us_counties_2010 you created and filled in this chapter, – export to a CSV file the 20 counties in the United States that have the most – housing units. Make sure you export only each county's name, state, and – number of housing units. (Hint: Housing units are totaled for each county in – the column housing_unit_count_100_percent.

```
In [16]: %%sql

         COPY (
         SELECT geo_name, state_us_abbreviation, housing_unit_count_100_percent
         FROM us_counties_2010
         ORDER BY housing_unit_count_100_percent
         DESC LIMIT 20
         )
         TO '/Users/ugurtigu/Documents/Learn/Docs/SQL/us_counties_housing_20.csv'
         WITH (FORMAT CSV, HEADER, DELIMITER ',')
         ;

 * postgresql://postgres:***@localhost:5432/analysis
20 rows affected.


Out[16]: []

In [17]: file = 'us_counties_housing_20.csv'
         file_read_from_head(file, 20)

geo_name,state_us_abbreviation,housing_unit_count_100_percent

Los Angeles County,CA,3445076

Cook County,IL,2180359

Maricopa County,AZ,1639279

Harris County,TX,1598698

San Diego County,CA,1164786

Orange County,CA,1048907
```

```
Kings County,NY,1000293

Miami-Dade County,FL,989435

Dallas County,TX,943257

King County,WA,851261

New York County,NY,847090

Clark County,NV,840343

Queens County,NY,835127

Wayne County,MI,821693

Broward County,FL,810388

Riverside County,CA,800707

Tarrant County,TX,714803

San Bernardino County,CA,699637

Philadelphia County,PA,670171
```