**Extract: your original data sources and how the data was formatted (CSV, JSON, pgAdmin 4, etc).**

* We used 2 data sources for comparing public parks to population, in the City of Chicago, by zip code.

1. [Parks in the City of Chicago](https://data.cityofchicago.org/Parks-Recreation/Parks-Chicago-Park-District-Park-Boundaries-curren/ej32-qgdr) (data.cityofchicago.org)
   1. Exported as a CSV
   2. Imported and manipulated via Python/Jupyter Notebook
2. [Census Data](https://www.census.gov/data/developers/guidance/api-user-guide.html) (census.gov)
   1. We used an API call

**Transform: what data cleaning or transformation was required.**

**Census:**

* Remove NA and negative data (such as -66666666 for population number)
* Dropped several column names not related to the scope of the project
* Rename column names, save csv file as ‘cleaned\_census\_data.csv’

**Parks:**

* Column titles were in ALL CAPS. We used “columns.str.strip().str.capitalize() to capitalize the first letter of each column title and lowercase the rest.
* Dropped several column names not related to the scope of the project
  + i.e.) the type of activities available at the park (bocci ball, baseball field, croquet, etc)
* There was was originally looked like an additional set of numbers to identify each park, we dropped that and set ‘Park\_No’ as the index.
* Changed a few column names to make more sense (i.e. Location —> Address, since it is the physical address provided)
* There were a lot of problems with the format of the zipcodes. They originally were appearing as “60611.0”. We needed to change it from an OBJ to an INT. It got to the point where we were working on manipulating the data in a CSV to try to get to the root of the issue. Eventually, we were able to do this by using the following code:

*parks = pd.read\_csv('CPD\_Parks.csv')*

*parks.dropna(inplace=True)*

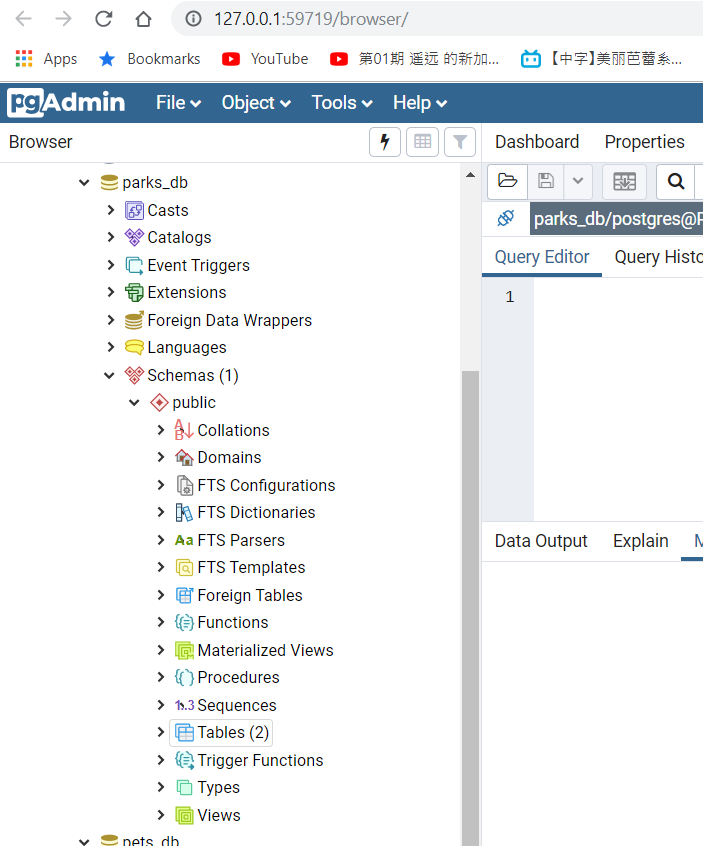
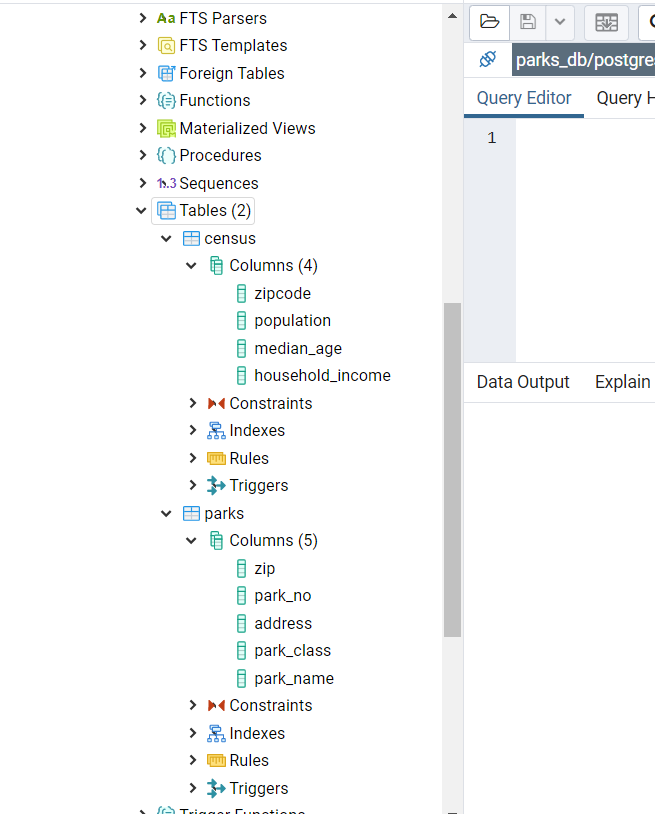
*parks['ZIP'] = parks['ZIP'].astype(int)*

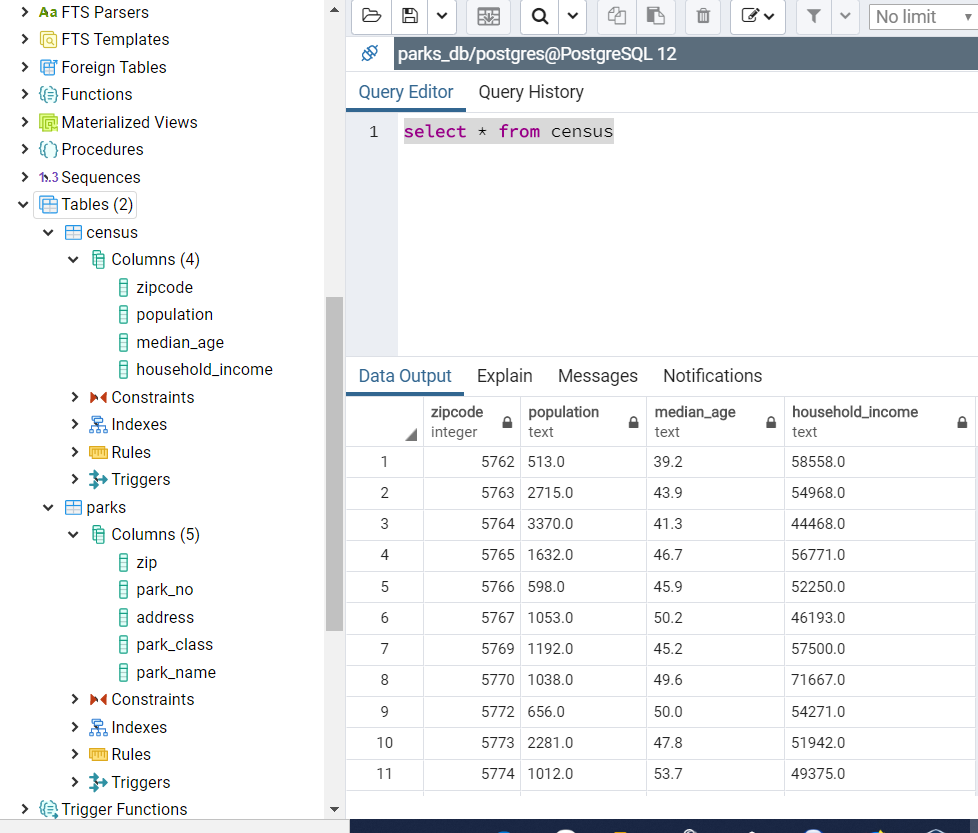
*Parks.dtypes*

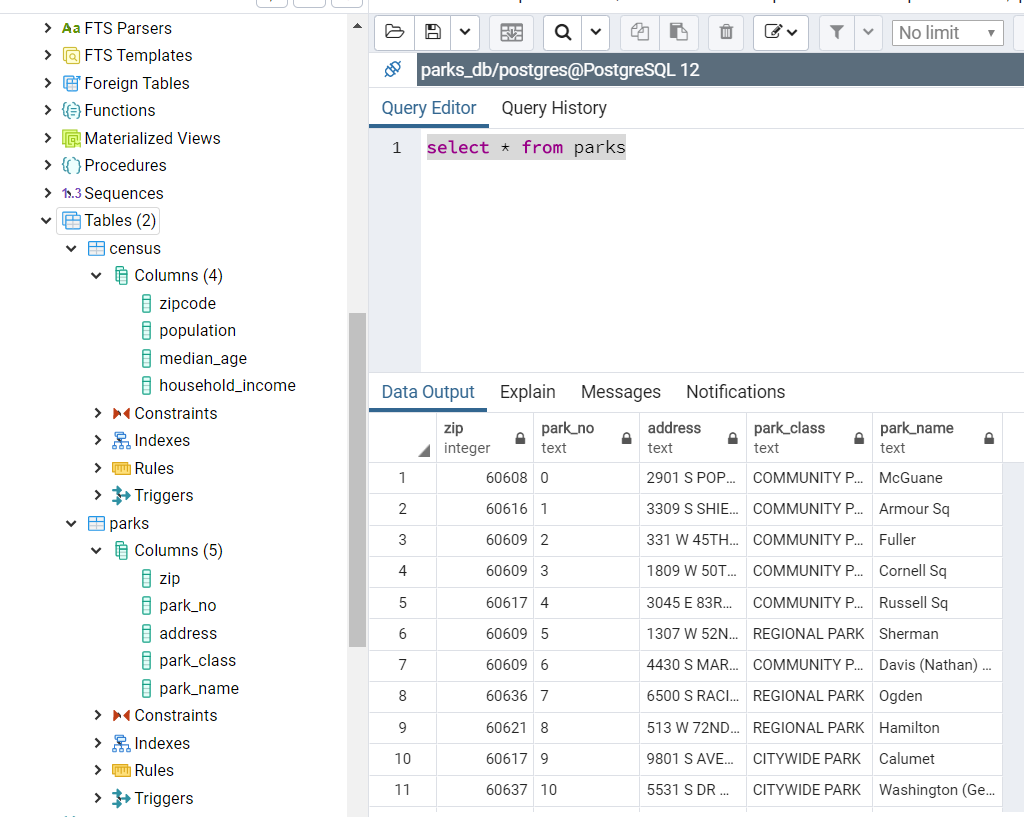
* Exported both files as CSVs and uploaded into SQL
* When we went to merge the two tables (census and parks) in SQL, we learned we were unable to do so. When importing the *parks.csv* as the foriegn key linking to the zipcode in the census table (Primary key), error messages would pop up. This was due to the fact that there were zip codes appearing in the foreign key that did not exist in the Census data’s table.

**Load: the final database, tables/collections, and why this was chosen.**

We chose to initially read and cleanup the data in python because we are more comfortable with the program and we found it to be faster than SQL. Once the data was ‘clean’, we created queries in SQL which merged 2 tables: census and parks. We merged on the zip code, as that was the only unique identifier between the two datasets. We were unable to create a Primary Key and Foreign Key, relating to Zip Code, because the parks data contained a few zip codes that were not in the census data.









This would be more relevant if SQL would enable our Primary Keys to work