**ETL Project**

Ellen Hsu

Gary Schulke

The purpose of this project is to find a public source of data to extract, transform the data into a format that can be saved into a database, and demonstrate that the database can be used to recall and display the date. We used a database found on Kaggle, [https://www.kaggle.com/citylines/city-lines/data#](https://www.kaggle.com/citylines/city-lines/data). It is a group of datasets that include transportation data from a number of cities and countries throughout the world and includes information on thousands of miles of transportation lines. There were seven csv files of data which included information on cities, countries, miles of line, station and location information and historical data.

A Postgres relational data base was chosen because of the number of data files and because “city\_id” was common to all of them. This makes the database easily extendable based on the data that is actually used. We chose two of the datasets, cities, and tracks “cities.csv” and “tracks.csv”. Jupyter Notebook was used to develop and execute Python code.

Data transformation involved renaming columns to be more consistent with relational database naming conventions. Unwanted data was dropped from the dataframes. One column, country\_state, was not used consistently. In the case of the United States, it was used for state names, province names for other countries, and unused for 60% of the data. To achieve total track distance, tracks data was grouped by “city\_id” and summed to create the “length” column. Track lengths were given in meters and converted to miles. A unique city count was used to get the number of stations per city.

The table shows the columns in the source data, name changes and dropped columns.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **cities.csv** | |  | **tracks.csv** | |  | **stations.csv** | |
| **ColumnName** | **Renamed** |  | **ColumnName** | **Renamed** |  | **ColumnName** | **Renamed** |
| id | city\_id |  | id | Dropped |  | id | Dropped |
| name | city\_name |  | geometry | Dropped |  | name | Dropped |
| coords | Dropped |  | opening | Dropped |  | geometry | Dropped |
| start\_year | Dropped |  | closure | Dropped |  | buildstart | Dropped |
| url\_name | Dropped |  | length | length |  | opening | Dropped |
| country | country |  | city\_id | city\_id |  | city\_id | city\_id |
| country\_state | Dropped (NaN) |  |  |  |  |  | unique name count |

Using Postgres we create a data base named **city\_transit\_db** and tables **cities** and **tracks**. Column “city\_id”

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table** | **Columns** | **Type** |  |  | **Table** | **Columns** | **Type** |  |
| cities | city\_id | INT | Primary |  | tracks | city\_id | INT | Primary |
|  | city\_name | VARCHAR |  |  |  | length | INT |  |
|  | country | VARCHAR |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Table | **Columns** | **Type** |  |  |  |  |  |  |
| Stations | city\_id | INT | Primary |  |  |  |  |  |
|  | station\_count | VARCHAR |  |  |  |  |  |  |

To create the Postgres database run the script **table\_create\_queries.sql**.using the pgAdmin Query Tool. This will create the database and the tables **cities** and **tracks**.

If the Postgres database requires a password, edit **config.py** to use your password.

Running the Jupyter Notebook will do and show the following.

1. Pandas and Sqlalchemy is used and imported. The Postgres password is imported by importing password.
2. The cities and tracks files were read in, put in a Pandas dataframe. Column renames, drops, and aggregate functions were performed as shown in the tables above then displayed.
3. The dataframes were merged on city\_id. This merge is duplicated using a sqlachemy query to the Postgres databas. Pandas was used to write the dataframes to their respective tables. Pandas was then used to read the data back and displayed.
4. SQL queries were created using sqlalchemy to read from Postgres, merge and display the data. This produces the same results as running the SQL query in queries.sql.

Running the Flask app queries the Postgres database and returns the tables in json format.

The available routes are:

/ Displays the available routes.

/api/v1.0/transit\_systems Displays the full list of cities.

/api/v1.0/transit\_systems/<city\_name> Displays one result based on city name.

Returns a 404 error if the city is not found.

Submitted to Github: <https://github.com/Gary-Schulke/ETL-Project.git>

Files Submitted: city\_transit\_systems.ipynb, queries.sql, table\_create\_queries.sql

cities.csv, stations.csv, tracks.csv, app.py