Data Wrangling

Project: Hawaii OpenStreetMap Data

Marvin Lüthe

Data Analyst Nanodegree 2018/2019

Table of Contents

List of Abbreviations

API	Application Programming Interface
CSV	Comma Separated Values
OSM	OpenStreetMap
XML	Extensible Markup Language

1 Introduction

I selected the state of Hawaii for the Data Wrangling Project and downloaded the *.osm.bz2* data. (cf. https://download.geofabrik.de/north-america.html)

The size of the *hawaii_original.osm* file is 226 MB unzipped. For test purposes two additional sample files *hawaii_medium.osm* (58.4 MB) and *hawaii_small.osm* (7.5 MB) were created.

This project only deals with the file *hawaii_medium.osm.*

The reason why I chose Hawaii is that, one day, I want to visit Hawaii and recap “Yes, I data wrangled these streets!”

This project was conducted in separate steps.

1. Download and unzip the OpenStreetMap data in XML format
2. Understand the XML structure (look in the data using Terminal and OpenStreetMap documentation)
3. Data Auditing
4. Prepare the data for the database creation
    a. Define cleaning procedures
    b. Clean the data as part of the conversion process to *.csv* files
    c. Validate the data against a pre-defined data schema
5. Create sqlite3 database and tables
6. Import the *.csv* files into the database
7. Analyse the data via SQL

In the following, I will precise these steps.

2 OpenStreetMap Data

OpenStreetMap is a collaborative project to create a free editable map of the world. The data is accessible for anybody.

The data consists of the following elements:

- Nodes hold geographical information as points. These information are represented by coordinates (attributes *lat* and *lon*). Every node is identified by its own *id*.
- Ways describe linear feature such as streets and rivers. They are ordered list of nodes which represents a polyline.
- Relations are ordered lists of nodes, ways and relations. Accordingly, relations show the relationship between nodes and ways.
- Tags are key-value pairs. These tags are either linked to a node or to a way object.

3 Data Auditing and Cleaning

The Data Auditing phase aims at determining dirty data in the given dataset. In a second step the data will be cleaned up iteratively to increase the data quality.

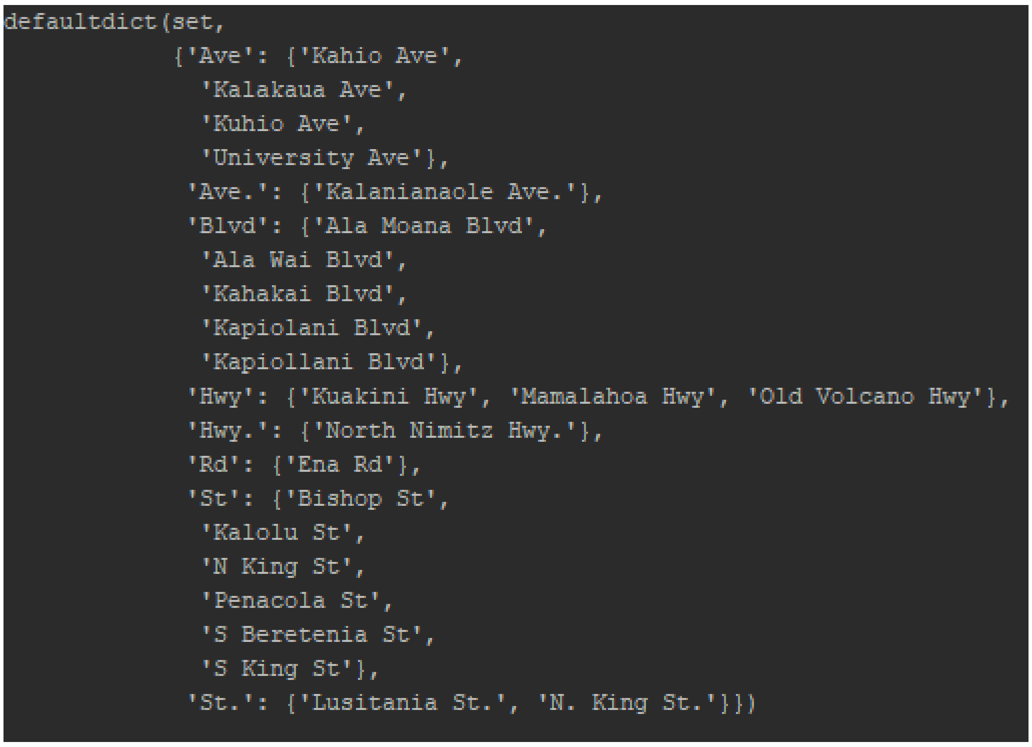
Data quality is measured by:

- Validity: the data conforms to a defined schema.
- Accuracy: the data conforms to gold standard (trusted external source).
- Completeness: all records are available in the data.
- Consistency: the fields, that represent the same data across systems, are consistent to each other.
- Uniformity: the same units are used for a given field.

In this project I decided to audit the streets, elevation indications and postcodes.

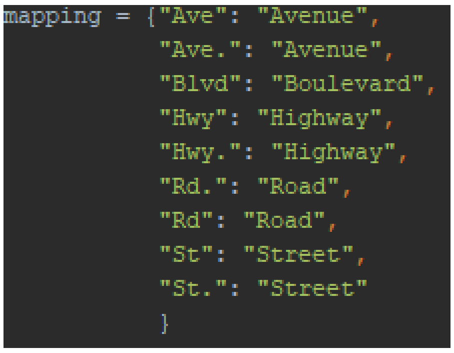
3.1 Audit Streets

The address information are stored in the *tag* tags of the node/way objects. I investigated if the street types were consistent and valid. I wanted to clean abbreviations e.g. “Ave” for Avenue or “Blvd” for Boulevard. The result of the auditing procedure *audit_streets(…)* for *hawaii_medium.osm* was:



```
defaultdict(set,
            {'Ave': {'Kahio Ave',
              'Kalakaua Ave',
              'Kuhio Ave',
              'University Ave'},
             'Ave.': {'Kalanianaole Ave.'},
             'Blvd': {'Ala Moana Blvd',
              'Ala Wai Blvd',
              'Kahakai Blvd',
              'Kapiolani Blvd',
              'Kapiollani Blvd'},
             'Hwy': {'Kuakini Hwy', 'Mamalahoa Hwy', 'Old Volcano Hwy'},
             'Hwy.': {'North Nimitz Hwy.'},
             'Rd': {'Ena Rd'},
             'St': {'Bishop St',
              'Kalolu St',
              'N King St',
              'Penacola St',
              'S Beretenia St',
              'S King St'},
             'St.': {'Lusitania St.', 'N. King St.'}})
```

Accordingly, the following mapping was created for the cleaning procedure:

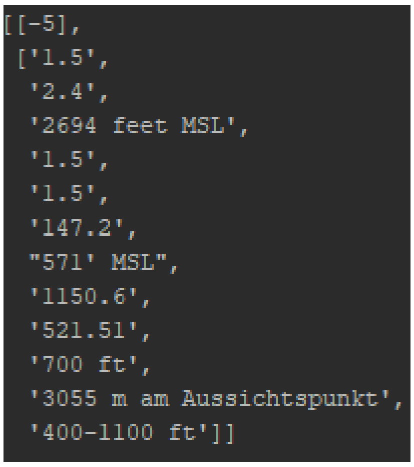


```
mapping = {"Ave": "Avenue",
           "Ave.": "Avenue",
           "Blvd": "Boulevard",
           "Hwy": "Highway",
           "Hwy.": "Highway",
           "Rd.": "Road",
           "Rd": "Road",
           "St": "Street",
           "St.": "Street"
           }
```

3.2 Audit Elevation Indications

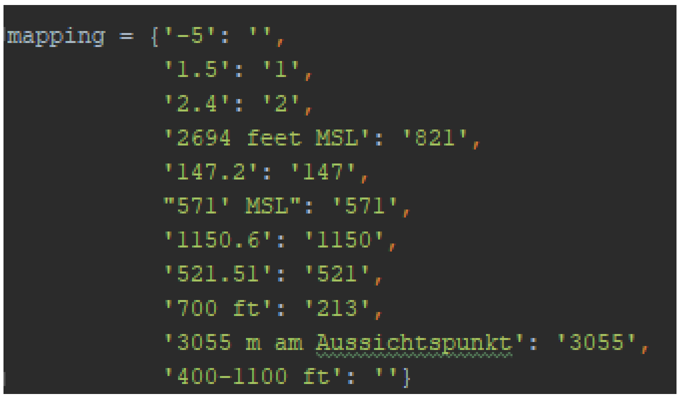
The auditing procedure for the elevation indications checks the accuracy and uniformity of the dataset. The lowest point is sea level, the highest point is 4205 meter above sea level on the mountain Mauna Kea. I checked, if the elevation indications exceed this range and will clean up these data, if necessary. Finally, it became apparent that one data point did not lie within the range [0 m;4205 m] (cf. first list)

The output of the *audit_elevation(…)* procedure was:



```
[[-5],
 ['1.5',
  '2.4',
  '2694 feet MSL',
  '1.5',
  '1.5',
  '147.2',
  "571' MSL",
  '1150.6',
  '521.51',
  '700 ft',
  '3055 m am Aussichtspunkt',
  '400-1100 ft']]
```

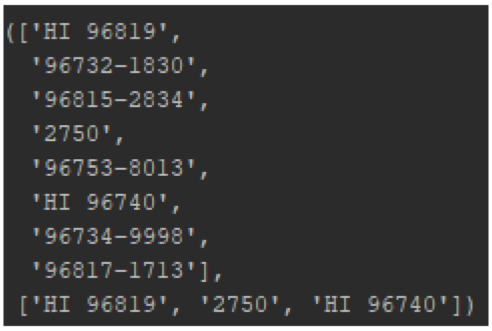
For these data I created a mapping to delete invalid values, decimal points and convert feet into meters.



```
mapping = {'-5': '',
           '1.5': '1',
           '2.4': '2',
           '2694 feet MSL': '821',
           '147.2': '147',
           "571' MSL": '571',
           '1150.6': '1150',
           '521.51': '521',
           '700 ft': '213',
           '3055 m am Aussichtspunkt': '3055',
           '400-1100 ft': ''}
```

3.3 Audit Postcodes

The postcodes are five digit numbers. This makes it easy to check if the given data comply with this schema. Additionally, we know that every Hawaiian postcode starts with “96”. We use this information to determine if the data is accurate. The auditing procedure *audit_zip(…)* identified the following data points:



```
(['HI 96819',
  '96732-1830',
  '96815-2834',
  '2750',
  '96753-8013',
  'HI 96740',
  '96734-9998',
  '96817-1713'],
 ['HI 96819', '2750', 'HI 96740'])
```

It becomes visible that every of these postcodes can be converted to a valid postcode, except for “2750”.

3.4 Preparing for SQL Database

The Python file *data.py* takes the OpenStreetMap data and performs the data cleaning and conversion to appropriate *.csv* files. The procedure *shape_element(…)* refers to the audit files and calls the "update" functions to clean the data for further database queries. Furthermore, the data is validated against the pre-defined data structure in *schema.py*.

4 SQLite Database

SQLite is a relational database management system. Relational databases are often used to store lots of information. The data is stored in tables which are linked by Foreign Keys. The Primary Keys are used to uniquely identify all table records. (cf. Appendix 1)

These databases use SQL (and variants of SQL) to access the data.

The Python file *sql.py* creates the database *osm.db* and the required tables.

By using the Terminal, we populate these tables with data using the following commands:

```
Marvins-iMac:My_Hawaii_Project marvin-luethe$ sqlite3 SQLite version 3.22.0 2018-01-22
18:45:57 Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open osm.db
sqlite> .mode csv
sqlite> .import node_tags.csv node_tags
sqlite> .import nodes.csv nodes
sqlite> .import ways.csv ways
sqlite> .import ways_nodes.csv way_nodes
sqlite> .import ways_tags.csv way_tags.
```

The sizes of the tables for the *osm.db* are as follows:

▪ *Hawaii_medium.osm:*	57 MB
▪ *nodes.csv:*	21 MB
▪ *node_tags.csv:*	0.4 MB
▪ *ways.csv:*	1 MB
▪ *ways_tags.csv:*	2.6 MB
▪ *ways_nodes.csv:*	9.5 MB

The database queries are realized in the Python file *sql_queries.py*. The first step was to create a connection and cursor object using the SQLite3 module as a Database Application Programming Interface (API).

```
db = sqlite3.connect("osm.db")
c = db.cursor()
```

The Hawaii OpenStreetMap data comprises the attributes “user” and “uid”. However, the downloaded dataset does not contain any values for these attributes. Therefore, due to a lack of information, it is not possible to calculate the number of unique users.

For exploration purposes, it is interesting to figure out the amount of entries per table.

```
tables = ['nodes', 'node_tags', 'ways', 'way_nodes', 'way_tags']


for table in tables:
    QUERY = "SELECT count(*) from {}".format(table)
    c.execute(QUERY)
    print table
    print c.fetchone()
```

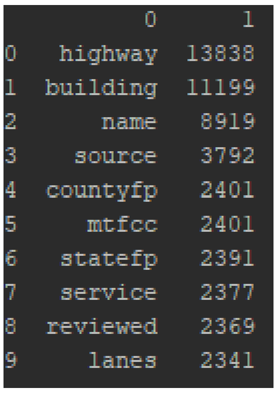
The result for this query:

```
nodes
(348477,)
node_tags
(12128,)
ways
(29101,)
way_nodes
(401508,)
way_tags
(78747,)
```

The tables *node_tags* and *way_tags* contain information which we would like to use for exploratory purposes. In order to figure out which keys are most oftenly used in these tables we will use the following query. This query counts the occurrence of the top ten key values in the *node_tags* and *way_tags* table:

```
QUERY = "SELECT tags.key, count(*) as num " \
        "FROM (SELECT * FROM node_tags UNION ALL SELECT * FROM way_tags)
         as tags " \
        "GROUP BY tags.key " \
        "ORDER BY num DESC " \
        "LIMIT 10;"
c.execute(QUERY)
df = pd.DataFrame(c.fetchall())
print df
```

The result for this query:



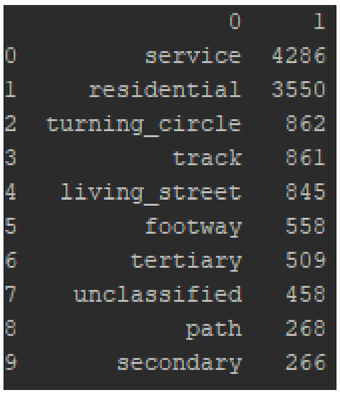
```
          0      1
0   highway  13838
1  building  11199
2      name   8919
3    source   3792
4  countyfp   2401
5     mtfcc   2401
6   statefp   2391
7   service   2377
8  reviewed   2369
9     lanes   2341
```

Since highways and buildings are tagged very often in this data set, I decide to further investigate these keys using the following queries:

```
#Further investigate highways
QUERY = "SELECT tags.value, count(*) as num " \
        "FROM (SELECT * FROM node_tags UNION ALL SELECT * FROM way_tags) as
         tags " \
        "WHERE tags.key='highway' " \
        "GROUP BY tags.value " \
        "ORDER BY num DESC " \
        "LIMIT 10;"
c.execute(QUERY)
df = pd.DataFrame(c.fetchall())
print df
```

```
#Further investigate buildings
QUERY = "SELECT tags.value, count(*) as num " \
        "FROM (SELECT * FROM node_tags UNION ALL SELECT * FROM way_tags) as
         tags " \
        "WHERE tags.key='building' " \
        "GROUP BY tags.value " \
        "ORDER BY num DESC " \
        "LIMIT 10;"
c.execute(QUERY)
df = pd.DataFrame(c.fetchall())
print df
```

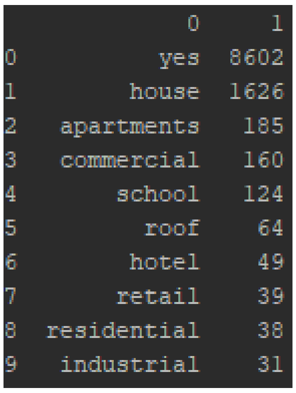
Result for highways:



```
                0     1
0         service  4286
1     residential  3550
2  turning_circle   862
3           track   861
4   living_street   845
5         footway   558
6        tertiary   509
7    unclassified   458
8            path   268
9       secondary   266
```

The most frequent highways are service highways which are basically access roads to any building such as petrol stations, beaches, camping, industries or parkings. This is followed by residential highways which are streets in settlement areas.

Result for buildings:



```
             0     1
0          yes  8602
1        house  1626
2   apartments   185
3   commercial   160
4       school   124
5         roof    64
6        hotel    49
7       retail    39
8  residential    38
9   industrial    31
```

The most frequently used tag schema for buildings is building = yes. This does not provide any information concerning the building type. However, the result shows that most of the buildings are houses, apartments and commercial buildings e.g. offices for banks, assurances and administration.

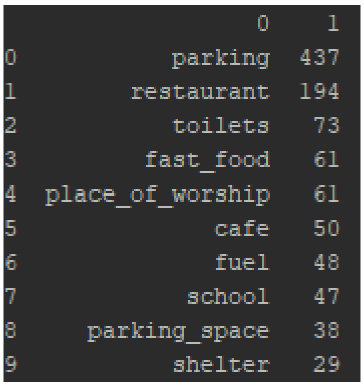
The remaining TOP Ten keys in *the node_tags* and *way_tags* table do not cause much curiosity which is why I continue to look at more interesting key, value pairs such as amenities and cuisines.

```
#Further investigate amenities
QUERY = "SELECT tags.value, count(*) as num " \
        "FROM (SELECT * FROM node_tags UNION ALL SELECT * FROM way_tags) as
         tags " \
        "WHERE tags.key='amenity' " \
        "GROUP BY tags.value " \
        "ORDER BY num DESC " \
        "LIMIT 10;"
c.execute(QUERY)
```

```
df = pd.DataFrame(c.fetchall())
print df
```

```
#Further investigate cuisine
QUERY = "SELECT tags.value, count(*) as num " \
        "FROM (SELECT * FROM node_tags UNION ALL SELECT * FROM way_tags) as
         tags " \
        "WHERE tags.key='cuisine' " \
        "GROUP BY tags.value " \
        "ORDER BY num DESC " \
        "LIMIT 10;"
c.execute(QUERY)
df = pd.DataFrame(c.fetchall())
print df
```

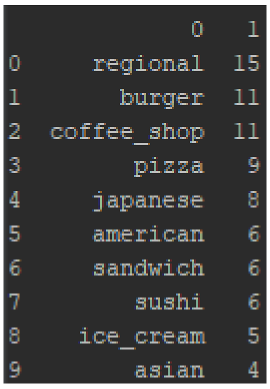
Result for amenities:



```
                 0    1
0          parking  437
1       restaurant  194
2          toilets   73
3        fast_food   61
4  place_of_worship   61
5             cafe   50
6             fuel   48
7           school   47
8    parking_space   38
9          shelter   29
```

The most frequent amenities are parkings, restaurants and toilets. This statistics unveils an inconsistency between *parking* and *parking_space*. Does it really make sense to separate these values or is it more convenient to merge them together? This question could be subject to further discussions around the quality of this dataset.

Result for cuisine values:



```
             0   1
0     regional  15
1       burger  11
2  coffee_shop  11
3        pizza   9
4     japanese   8
5     american   6
6     sandwich   6
7        sushi   6
8    ice_cream   5
9        asian   4
```

The query statement for cuisines did not provide many results in comparison with the preceding queries. Nevertheless, it can be shown that the Hawaiian people and tourists prefer regional food and burgers.

5 Additional Ideas

The modules *audit_streets, audit_elevation_data* and *audit_postcode* propose procedures to clean the Hawaii OpenStreetMap data to increase the data quality.

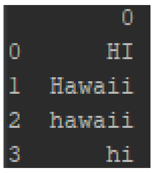
5.1 Suggestions for Improvement

Nevertheless, this dataset is very large and I would like to present three tag keys which could be cleaned in additional data auditing and cleaning steps. It becomes clear that data wrangling is an iterative process and the data which is stored in the *osm.db* database is definitely not fully free from inconsistencies.

We will take a look at the *node_tags* table. The first query identifies issues related to state indications.

```
QUERY = "SELECT distinct(value) " \
        "FROM node_tags " \
        "WHERE key='state';"
c.execute(QUERY)
df = pd.DataFrame(c.fetchall())
print df
```

Since we downloaded only the OpenStreetMap data for Hawaii we expect to receive one value. However, due to abbreviations and capitalization/case sensitivity within the data we receive four differing values:



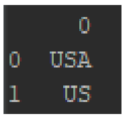
```
        0
0      HI
1  Hawaii
2  hawaii
3      hi
```

Similar issues can be encountered for country and city indications.

Query for country data:

```
QUERY = "SELECT distinct(value) " \
        "FROM node_tags " \
        "WHERE key='country';"
c.execute(QUERY)
df = pd.DataFrame(c.fetchall())
print df
```

Output:



```
     0
0  USA
1   US
```

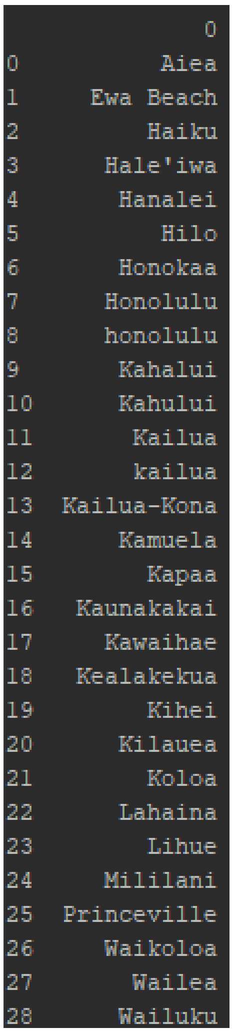
The state and country queries identified issues in the dataset but we can fix these issues easily (because we know that the state is Hawaii and is part of the USA) by adding cleaning procedures and modules which can be called by the *shape_element()* function in *data.py*.

The following statement queries city values from the table *node_tags*.

Query for city data:

```
QUERY = "SELECT distinct(value) " \
        "FROM node_tags WHERE key='city' " \
        "ORDER BY value COLLATE NOCASE ASC;"
c.execute(QUERY)
df = pd.DataFrame(c.fetchall())
print df
```

Output:



```
                 0
0             Aiea
1        Ewa Beach
2            Haiku
3         Hale'iwa
4          Hanalei
5             Hilo
6          Honokaa
7         Honolulu
8         honolulu
9          Kahalui
10         Kahului
11          Kailua
12          kailua
13     Kailua-Kona
14         Kamuela
15           Kapaa
16      Kaunakakai
17        Kawaihae
18      Kealakekua
19           Kihei
20         Kilauea
21           Koloa
22         Lahaina
23           Lihue
24        Mililani
25     Princeville
26        Waikoloa
27          Wailea
28         Wailuku
```

This result illustrates that the same content is stored in different values which finally leads to inconsistencies. This is exemplified by the city Honolulu (Honolulu and honolulu) which is one of the easier examples to deal with as we can easily ignore case sensitivity in SQL language.

It becomes more difficult if we throw our attention to cities like Kahului (which seems to be the correct city, instead of Kahalui) as this is a first sign that we cannot trust our data completely due to a lack of accurateness (the city Kahalui does not exist).

We could use the Google Maps API to improve the OpenStreetMap dataset if we compared our dataset with Google Maps data. Invalid values such as the Kahului/Kahalui city names can be fixed by using the Google Maps API because Google Maps data helps us decide which city is correct and update the data accordingly.

5.2 Benefits

- We can determine the accuracy of our dataset since Google Maps serves as a trusted external source.
- We can clean our given dataset before database creation as we compare our data with the Google Maps data and assume Google Maps to have a better level of completeness.

5.3 Anticipated Problems

- The completeness of Google Maps data may also lead to problems while wrangling the OpenStreetMap data. If the comparison between both datasets discloses that the OpenStreetMap data is not complete, it will be necessary to decide if the missing data is filled with Google Maps data or not. Are the contributors to the OpenStreetMap data allowed to make use of the richness of Google Maps?
- If yes, this might lead to another problem related to data validity. It is possible that Google stores its Map data in a different schema and database structure. This aspect does not only relate to the data cleaning process but might already be subject to discussions during the comparison process.
- If no, the OpenStreetMap initiative will have to find opportunities to complete the dataset by motivating local people and tourists to share their knowledge about the region and contribute to the OpenStreetMap project.

6 Conclusion

The results of the preceding queries once again underline, that the already modified and improved OpenStreetMap data for the state Hawaii still contains quality issues which need to be corrected for further data exploration phases.

The aim of this project was to propose solutions for data auditing and cleaning. Additionally, the data has been transferred to a database which makes it easily accessible and explorable for deeper investigations.

Bibliography – Index of Literature and Sources

http://book.pythontips.com/en/latest/enumerate.html

http://www.sqlitetutorial.net/sqlite-python/creating-database/

http://www.sqlitetutorial.net/sqlite-union/

http://zetcode.com/db/sqlitepythontutorial/

https://docs.python.org/2/library/sqlite3.html

https://download.geofabrik.de/north-america.html

https://en.wikipedia.org/wiki/OpenStreetMap

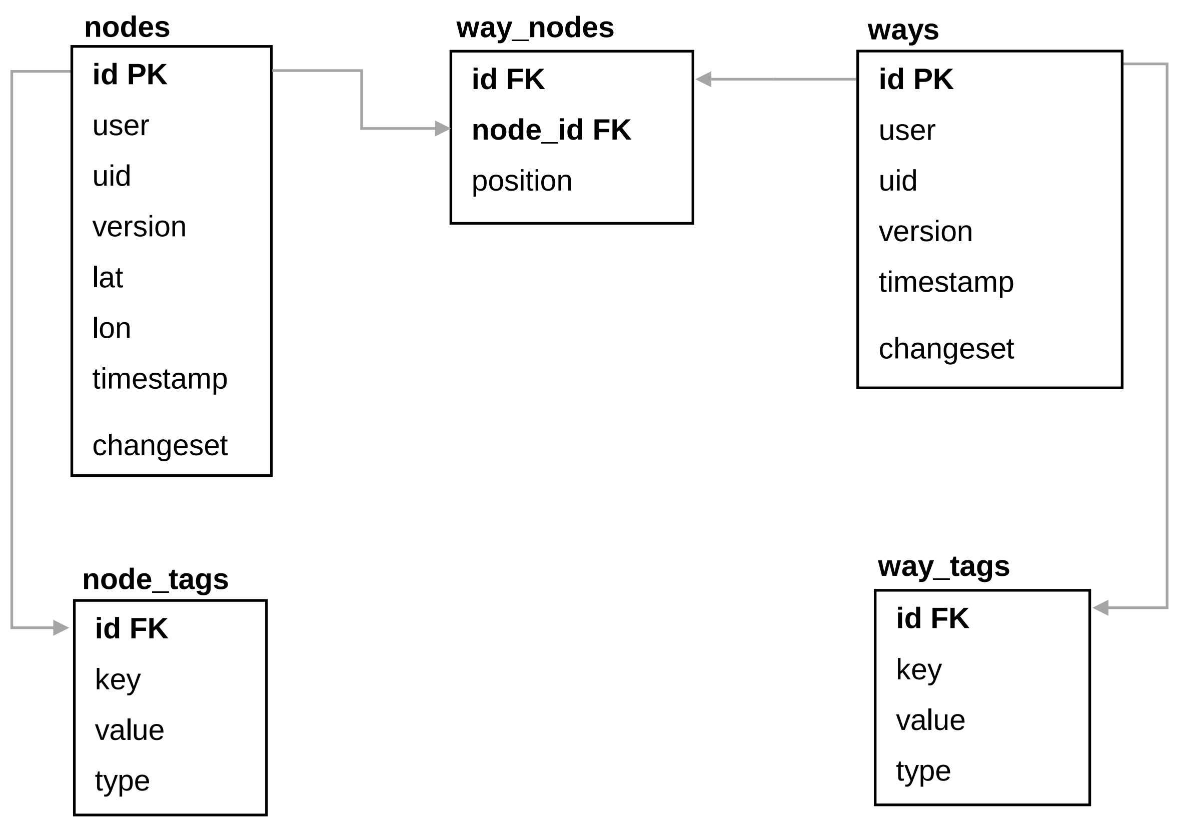
https://stackoverflow.com/questions/

https://wiki.openstreetmap.org/wiki/DE:Key:building

https://wiki.openstreetmap.org/wiki/DE:Tag:highway%3Dservice

Appendices

Appendix 1 OSM Database



PK: Primary Key

FK: Foreign Key