OpenStreetMap Data Wrangling Project with SQL:

Map Area:

I have extracted OSM file from openstreemap.org and it is for my residential area 'Milwaukee County'. Total this file is more than 2GB, hence I have reduced the area locality to get a smaller size of file which is around 150 MB. I am interested to know more about my area through this project and want to see what reveals about my locality via query.
We will now go through the data wrangling process.

Problems Encountered in the file:

I downloaded the file and named as “Milwaukee_Map.osm”. Size of the file is 153.16 MB. File size is big enough, so first prepared a sample of the file named 'milwaukee_sample.osm'. But the first problem I found that, running code with the sample file was not giving the idea about the main file. Hence used the main file for rest of the process. This made the execution time a little longer.

To analyze the data of the file, first I have found the number of available “tags” in the file. Did some more finding on the file in InitialCheck.py and got some idea on how many records are there, found out the different attributes and their values. This initial check shows some problems present in the file. Like

- Improper street name abbreviation
- Different format of phone number’s
- Different format of postal codes.
- Some issues in city name:
    - Same city with different spelling
    - Few city names are in full capitols
    - Few city name with state extension

To get the details of error, I have audited one by one.

Street name audit and corrections:

Iterate over each item of OSM file and found out the improper street name using AuditStreet.py and did the mapping to proper name.

In [9]:
```
if __name__ == '__main__':
    #Get all the street name not with proper suffix
    st_types = audit_streetname(street_types_post,street_type_regex_post, expected_post)
    #Print those street name
    pprint.pprint(dict(st_types))
```

```
{'100': set(['South Highway 100']),
 '164': set(['N6620 HWY 164', 'State Road 164']),
 '18': set(['Highway 18']),
 '190': set(['State Highway 190; W Capitol Dr; State Highway 190']),
 '400': set(['West Park Place Ste. 400']),
 'Ave': set(['6817 W North Ave',
             'E North Ave',
             'E Ogden Ave',
             'E Wisconsin Ave',
             'Harwood Ave',
             'N Teutonia Ave',
```

In [14]:
```
if __name__ == '__main__':
    #Get all the street name not with proper suffix
    st_types1 = audit_streetname(street_types_pre,street_type_regex_pre, expected_pre)
    #Print those street name
    pprint.pprint(dict(st_types1))
```

```
{'E': set(['E Moreland Blvd',
           'E North Ave',
           'E Ogden Ave',
           'E Wisconsin Ave']),
 'E.': set(['E. North Ave.']),
 'N': set(['N 124th St',
           'N 35th St',
           'N Grandview  Blvd',
```

And the mapping corrected the name as below.

In [12]:
```
#Update street name
def update_name(name, street_mapping,street_type_regex):
    m1 = street_type_regex.search(name)
    if m1:
        street_type = m1.group()
        if street_type in street_mapping:
            name = re.sub(street_type_regex, street_mapping[street_type], name)
    return name
```

In [13]:
```
if __name__ == '__main__':
    #Main function to Correct the street name
    for st_type, ways in st_types.iteritems():
        for name in ways:
            better_name = update_name(name, mapping_street_post,street_type_regex_post)
            print name, "=>", better_name
```

```
Melody Ln => Melody Lane
N. Weil St. => N. Weil Street
Burleigh St. => Burleigh Street
N. Commerce St. => N. Commerce Street
North 51st St. => North 51st Street
State Highway 190; W Capitol Dr; State Highway 190 => State Highway 190; W Capitol Dr; State Highway 190
W Watertown Plank Rd => W Watertown Plank Road
W Bluemound Rd => W Bluemound Road
S Moorland Rd => S Moorland Road
N Green Bay Rd => N Green Bay Road
```

```
E. North Ave. => East North Ave.
E Wisconsin Ave => East Wisconsin Ave
E North Ave => East North Ave
E Moreland Blvd => East Moreland Blvd
E Ogden Ave => East Ogden Ave
N. Industrial Rd. => North Industrial Rd.
N. Prospect Ave. => North Prospect Ave.
N. Prospect Ave => North Prospect Ave
```

Postal code audit and corrections:

Audit of postal is done using AuditPostcode.py. Postal codes were not in proper pattern. Though most of them are with 5 digits pattern, but some were with 5 digits plus 4 digits, few were with city name. Using the 'postcode_correction' function, all the postal code has been converted in same 5 digits pattern.

9]:
```
def postcode_correction(postcode):
    if "-" in postcode and "WI" not in postcode:
        postcode = postcode.split("-")[0].strip()
    if "WI" in postcode:
        postcode = postcode.split("WI")[1].strip('WI ')
        if ',' in postcode:
            postcode = postcode.split(",")[1].strip(', ')
        if '-' in postcode:
            postcode = postcode.split("-")[1].strip('- ')
    return postcode
```

0]:
```
#Removing state initials from the postal codes if any and the 4 digit extension part to make all of them with 5 digit
if __name__ == '__main__':
    for zip_type, ways in zp_types.iteritems():
        for name in ways:
            better_name = postcode_correction(name)
            print name, "=>", better_name
```

```
Milwaukee WI, 53222 => 53222
53202-1552 => 53202
53217-2030 => 53217
53202-1614 => 53202
53202-2001 => 53202
```

Phone number audit and corrections:

Phone numbers are listed in many different format and looks very messy. In AuditPhone.py file, the correction of phone numbers is done as below. But it is very doubt full which format of phone number is correct. This kind of correction is done as part of this project. There might be different format.

In [9]:
```
if __name__ == '__main__':
    for phone_type, ways in ph_type.iteritems():
        for name in ways:
            edited_phonenumber = update_phone(name)
            print name, '=>', edited_phonenumber
            #print(edited_phonenumber)
```

```
414-272-0543 => 414 272 0543
+1-414-277-0851 => 414 277 085
(262) 784-7708 => 262 784 7708
+1.414.354.1919 => 414 354 1919
+1-262-789-6300 => 262 789 6300
262-691-0900 => 262 691 0900
+1-414-962-8809 => 414 962 8809
+1-414-509-0530 => 414 509 0530
+1-414-342-0215 => 414 342 0215
+1-414-934-4600 => 414 934 4600
(262) 422-6124 => 262 422 6124
+1-414-304-6100 => 414 304 6100
```

To save this corrected data in database, I have created csv file first and then used those files to create data base tables. This part is done in DataUpdate_ToCSV.py file. And cleaned data is then saved in tables.

Data Base Query

Database query reveals many interesting things about the location.

Finding the NODE count, WAY count and other entry count in each table:

In [20]:
```
curs.execute("SELECT COUNT(*) FROM nodes")
all_rows = curs.fetchall()
print('Number of NODES are:{}').format(all_rows)
con.commit()
```

```
Number of NODES are:[(683659,)]
```

In [21]:
```
curs.execute("SELECT COUNT(*) FROM nodes_tags")
all_rows = curs.fetchall()
print('Number of NODES_TAGS are:{}').format(all_rows)
con.commit()
```

```
Number of NODES_TAGS are:[(72942,)]
```

In [22]:
```
curs.execute("SELECT COUNT(*) FROM ways")
all_rows = curs.fetchall()
print('Number of WAYS are:{}').format(all_rows)
con.commit()
```

```
Number of WAYS are:[(80966,)]
```

In [23]:
```
curs.execute("SELECT COUNT(*) FROM ways_nodes")
all_rows = curs.fetchall()
print('Number of WAYS_NODES are:{}').format(all_rows)
con.commit()
```

```
Number of WAYS_NODES are:[(879494,)]
```

In [24]:
```
curs.execute("SELECT COUNT(*) FROM ways_tags")
all_rows = curs.fetchall()
print('Number of TAGS in ways are:{}').format(all_rows)
con.commit()
```

```
Number of TAGS in ways are:[(355503,)]
```

Top 10 mostly mentioned postal code in the area:

In [45]:
```
curs.execute("""SELECT tags.value, COUNT(*) as count
                FROM (SELECT * FROM nodes_tags UNION ALL
                      SELECT * FROM ways_tags) tags
             WHERE tags.key='postcode'
              GROUP BY tags.value
              ORDER BY count DESC LIMIT 10;""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)
#con.commit()
```

```
[('53202', 3288),
 ('53212', 770),
 ('53205', 382),
 ('53203', 269),
 ('53233', 165),
 ('53151', 147),
 ('53211', 103),
 ('53217', 97),
 ('53204', 95),
 ('53215', 84)]
```

All the postal codes were not in same format. We have already found the issues of postal code during data wrangling phase, when checking on OSM file. And here in data base those records have been saved after the correction. Correction has make it better.

Query on phone numbers:

These phone numbers were in a very unorganized pattern. There were many formats. During the data analysis phase with OSM file, coding is done to correct the data and to bring in same format.

In [44]:
```
curs.execute("""SELECT COUNT(*) as count
              FROM (SELECT * FROM nodes_tags UNION ALL
                    SELECT * FROM ways_tags) tags
            WHERE tags.key='postcode';""")
all_rows = curs.fetchall()
print("Total phone numbers:", all_rows)


curs.execute("""SELECT tags.value, COUNT(*) as count
              FROM (SELECT * FROM nodes_tags UNION ALL
                    SELECT * FROM ways_tags) tags
            WHERE tags.key='phone'
             GROUP BY tags.value
             ORDER BY count DESC LIMIT 5;""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)
```

```
('Total phone numbers:', [(6330,)])
[('414 342 0215', 11),
 ('414 482 4270', 9),
 ('414 281 6289', 8),
 ('414 461 0839', 7),
 ('414 672 5052', 7)]
```

Cities with count

In [28]:
```
curs.execute("""SELECT COUNT(*) as count
FROM (SELECT * FROM nodes_tags UNION ALL
      SELECT * FROM ways_tags) tags
WHERE tags.key LIKE '%city';""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)

curs.execute("""SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags UNION ALL
      SELECT * FROM ways_tags) tags
WHERE tags.key LIKE '%city'
GROUP BY tags.value
ORDER BY count DESC LIMIT 5;""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)
```

```
[(6357,)]
[('Milwaukee', 5354),
 ('New Berlin', 152),
 ('Sussex', 74),
 ('Wauwatosa', 68),
 ('Oak Creek', 60)]
```

Milwaukee is mostly mentioned city and it's almost 84% of total record with city names.

Number of Unique User(Contributor):

In [29]:
```
#Number of Unique Users
curs.execute("""SELECT COUNT(DISTINCT(e.uid))
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;""")
all_rows = curs.fetchall()
print("Number of Unique user::")
pprint.pprint(all_rows)
```

```
Number of Unique user::
[(609,)]
```

Top Most contributor:

In [31]:
```
#Top 10 Contributors
curs.execute("""SELECT e.user, COUNT(*) as num
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
GROUP BY e.user
ORDER BY num DESC
LIMIT 10;""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)
```

```
[('shuui', 202112),
 ('ItalianMustache', 167359),
 ('woodpeck_fixbot', 90798),
 ('erikwanta', 23474),
 ('bbauter', 23074),
 ('Gary Cox', 21356),
 ('iandees', 19367),
 ('jcowdy', 15823),
 ('cncr04s', 13865),
 ('Mulad', 13726)]
```

In [30]:
```
#Total Contribution Count
curs.execute("""SELECT COUNT(*) as num
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
;""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)
```

```
[(764625,)]
```

These top 10 contributers out of total 609, have contributed more than 77% of total contribution. And user "shuui" as top most contributer has contributed more than 26% of total contribution.

In [33]:
```
#Number of users appearing only once (having 1 post)
curs.execute("""SELECT COUNT(*)
FROM
    (SELECT e.user, COUNT(*) as num
     FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
     GROUP BY e.user
     HAVING num=1)  u;""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)
```

```
[(130,)]
```

There are total 130 user those who have contributed only once.

Idea about Amenities:

Most available amenities:

In [34]:
```
#Amenities
curs.execute("""SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key='amenity'
GROUP BY value
ORDER BY num DESC LIMIT 5;""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)
```

```
[('parking_entrance', 1623),
 ('school', 417),
 ('bench', 368),
 ('restaurant', 254),
 ('bicycle_parking', 127)]
```

Least available amenities:

In [35]:
```
#Amenities
curs.execute("""SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key='amenity'
GROUP BY value
HAVING num = 1
ORDER BY num LIMIT 5;""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)
```

```
[('amphitheatre', 1),
 ('bathroom', 1),
 ('bbq', 1),
 ('bureau_de_change', 1),
 ('childcare', 1)]
```

It is quite unclear what is meant by few of the amenities name like 'bathroom', 'bbq'. Need to do some detail finding on these.

Religion:

In [46]:
```
#Religion
curs.execute("""SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='place_of_worship') i
    ON nodes_tags.id=i.id
WHERE nodes_tags.key='religion'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 2;""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)
```

```
[('christian', 32), ('buddhist', 2)]
```

We can see from the all available amenities, that there are total 43 counts of "place of worship". And 32 out of that is for "christian". This result is as expected.

Banks:

In [38]:
```
#Banks
curs.execute("""SELECT nodes_tags.value, COUNT(*) as num
        FROM nodes_tags
            JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='bank') i
            ON nodes_tags.id=i.id
        WHERE nodes_tags.key='name'
        GROUP BY nodes_tags.value
        ORDER BY num DESC LIMIT 5;""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)
```

```
[('Associated Bank', 5),
 ('BMO Harris Bank', 3),
 ('Guaranty Bank', 3),
 ('US Bank', 3),
 ('Chase', 2)]
```

Associated Bank has maximum branch in this area.

Cuisines:

```
In [39]: #Cuisines
curs.execute("""SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='restaurant') i
    ON nodes_tags.id=i.id
WHERE nodes_tags.key='cuisine'
GROUP BY nodes_tags.value
ORDER BY num DESC LIMIT 5;""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)
```

```
[('american', 23),
 ('pizza', 16),
 ('italian', 15),
 ('chinese', 14),
 ('burger', 11)]
```

As expected, American cuisines are majority here.

```
In [40]: curs.execute("""SELECT COUNT(*) FROM nodes_tags WHERE value LIKE '%McDonald%';""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)
```

```
[(15,)]
```

Above query reveals the count of Mac'D, which is 15.

Going to find wheel chair availability.

```
In [41]: curs.execute("""SELECT COUNT(*) FROM nodes_tags WHERE key='wheelchair' AND value='yes';""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)

curs.execute("""SELECT COUNT(*) FROM nodes_tags WHERE key='wheelchair';""")
all_rows = curs.fetchall()
pprint.pprint(all_rows)
```

```
[(49,)]
[(82,)]
```

```
In [43]: float(float(49)/float(82))
```

```
Out[43]: 0.5975609756097561
```

Wheel chair accessibility is approximately 60%, which is not too good. This facility can be improved.

Problems faced during the overall process:

There are few places which needs very deep research to provide the correct data. Like phone numbers. In this specific case, mostly the phone numbers and postal codes are messed up. There are still scopes to correct those. Some street names which are having improper part in middle section, those are not covered here. Some are with too specific errors to be handled. Same city names are spelled differently. Like 'N6620 HWY 164'. HWY can be modified to ‘Highway’. These are not handled here.

Before doing implementation of any correction, there should be some defined standard which needs to be followed. Like the format of phone numbers. Format of postal codes. If these are defined, the correction can be made accordingly. The wrong data is another big problem. It can lead to some wrong analysis.

Using the sample map of big data is sometimes misleading. It restricts the idea about all actual errors which does not comes under any group. Some data are very specifically wrong. Need to handle them separately to make the map more accurate.

The above kind of data analysis helps us to find the scope of improvement, like presence of bank or school or some other public facilities. The finding like availability of wheel chair can be analyzed and increased to make the locality better.

Conclusion:

This Milwaukee OpenStreetMap dataset is large one, but it's not too much messed up. Data is not 100% clean for sure. I have learned a lot during this project. I have gone through many new things like processing OSM file to CSV and then to data base. Also doing the audit part and finding the problems in data is interesting. In this process, few part of the data has been corrected as well.