OpenStreetMap Data Project

Map Area

Erlangen, Germany

I used an extract from mapzen (https://mapzen.com/data/metro-extracts/). The exact phrase for the region was Erlangen, Deutschland (neighbourhood).

It is a map of my hometown, and I was quite interested, what the database queries would reveal, since I already know the area.

Problems Encountered

When I ran the map data against some scripts, the following problems appeared:

- phone numbers not according to german standards
- invalid/inconsistent urls
- different ways of storing information for traffic zones (but which are totally legit, according to the openstreetmaps wiki)

Phone Numbers

Phone numbers should be according to DIN 5008, which offer guidelines for how to print phone numbers.
Area Codes are separated by a single space from the actual phone number:
0873 376461
03748 37682358
Area Codes consist of 3-5 digits.

An extension number is added to the number with a hyphen, like this:
05444 347687-350
0764 812632-41

The country code for Germany is +49. In this case, die 0 in the area code is dropped and separated from the country code by a single space:
+49 30 3432622-113
Most phone numbers in this dataset were without country code, and I decided, to keep it that way.

Invalid/inconsistent URLs

There were a lot of invalid URLs in the dataset, that would lead to nowhere when clicked. They would miss the https:// or http:// .

Traffic zones

The information, if a street lies in a traffic zone would sometimes be found in the key and sometimes in the value of the xml Element, for example:

```
<tag k="source:maxspeed" v="DE:zone30"/>

<tag k="source:maxspeed" v="DE:zone:30"/>

<tag k="zone:maxspeed" v="DE:30"/>
```

All of the elements store the same kind of information, but are not structured the same way, but also both ways seem legit, so I made no changes to that. But it is important to keep that in mind when running queries.

http://wiki.openstreetmap.org/wiki/Key:zone:maxspeed

http://wiki.openstreetmap.org/wiki/Key:source:maxspeed

We will talk about this problem later on.

Challenges during data wrangling

For the validation and standardizing the data for URLs, the wrangling was very easy, because the validators module could be used.
However, cleaning the phone numbers was challenging. There are different ways to print a phone number in Germany, but I wanted them the way, the DIN 5008 proposes.
All non-digit characters had to be removed, except the plus and minus signs. But the minus sign should only be kept, if it was separating the phone number from the extension number. In some cases, it was impossible to clean the data programmatically, because there anything in the phone number could be an extension number and there are no rules out there for extension numbers that could be used to clean the data programmatically.
It was also tricky, to deal correctly with the assignment of spaces. The only thing which could be cleaned programmatically, was the separation of country code, area code and phone number by a space.
After all, there were also outputs, which made no sense and couldn’t even be found in a google research.

Overview of the data

File sizes

ex_h59MB33V6XrsLWjzXhs7CWHwY3NCz.osm: 89 MB
openstreetmaps.db: 53 MB
nodes.csv: 28 MB
nodes_tags.csv: 2 MB
ways.csv: 4 MB
ways_nodes.csv: 1 KB
ways_tags.csv: 9 MB

Number of unique users

```
SELECT COUNT(DISTINCT(uid))
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways)
```

953

Number of nodes

```
SELECT COUNT(*)
FROM nodes
```

361110

Number of ways

```
SELECT COUNT(*)
FROM ways
```

70080

Number of traffic signals

```
SELECT COUNT(*) as num
FROM(SELECT value FROM nodes_tags UNION ALL SELECT value FROM ways_tags)
WHERE value = 'traffic_signals'
```

```
470
```

Number of ways that lie in a traffic zone

```
SELECT COUNT(*) as num
FROM ways_tags
WHERE value LIKE '%zone%'
OR key LIKE '%zone%'
```

```
457
```

Since there seem to be two different keys for this kind of information, I wanted to know, how often they appear.

```
SELECT key, COUNT(*) AS num
FROM ways_tags
WHERE value LIKE '%zone%'
OR key LIKE '%zone%'
GROUP BY key
ORDER BY num DESC
```

```
"zone:maxspeed" "224"
"source:maxspeed" "147"
"zone:traffic" "82"
"description" "2"
"name" "2"
```

The result was surprising, since the first two results where known to me, but the others weren't. The result for a query, where the key was equal to "name" was as follows:

```
SELECT key, value
FROM ways_tags
WHERE value LIKE '%zone%'
AND key = 'name'
"name" "Venzonebrücke"
"name" "Pflegezentrum Venzone Stiftung"
```

As can be seen, the result has nothing to do with a traffic zone. This is problematic, because it distorts the results of our queries.

Road surface of ways that lie in a traffic zone

```
SELECT A.value, COUNT(*) AS count
FROM ways_tags A, ways_tags B
WHERE A.id = B.id
AND A.key = 'surface'
AND (B.key LIKE '%zone%' OR B.value LIKE '%zone%')
GROUP By A.value
ORDER BY count DESC
```

```
"asphalt" "243"
"paved" "17"
"paving_stones" "8"
"concrete" "3"
"cobblestone" "1"
```

It seems like the surface attribute is not maintained for many of the roads, because the count of all the ways in a traffic zones, where a surface value is available, evaluates to 272. Compare that to the 475 ways, which lie in a traffic zone, you have a difference of 185 without values for the surface.

Parking Opportunities

```
SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='parking') AS i
ON nodes_tags.id=i.id
WHERE nodes_tags.key='parking'
GROUP BY nodes_tags.value
ORDER BY num DESC
```

```
"surface" "15"
"underground" "13"
"multi-storey" "5"
"lane" "2"
```

Other ideas about the dataset

There are two different ways, postal codes are stored. It doesn’t make that big of a difference,
but it worth considering.

```
SELECT key, COUNT(*) AS count
FROM ways_tags
WHERE key LIKE '%post%'
GROUP BY key
ORDER BY count DESC
"addr:postcode" "21576"
"postal_code" "5"
```

It has also an effect on the type of the tag in the database. The first example is listed as type ‘addr’, the second as ‘regular’. You always have to filter by several criteria.
Both of these problems cannot be solved with just some data cleaning, but only with standardizing the openstreetmap library.

When it comes to the traffic zones, it would make most sense, to standardize data like this:

```
<tag k="zone:maxspeed" v="DE:30"/>
```

According to the openstreetmap wiki, it would make no sense, to use the “source” key for this, since this key should be used to indicate the source of some information.
http://wiki.openstreetmap.org/wiki/Key:source

For the postcodes, the key "addr:postcode" should definitely be used, since the Karlsruhe Schema is proposed by openstreetmaps.
http://wiki.openstreetmap.org/wiki/Key:postal_code
http://wiki.openstreetmap.org/wiki/Proposed_features/House_numbers/Karlsruhe_Schema

To deal with these problems, it might be useful, to provide a system, which suggests a schema for user input. It should check for the input to match a certain schema, and if the input is not valid, the user should be informed about this. He should also get a suggestion of which schema to use instead. It might be difficult, to implement such a system in terms of user acceptance. Since they are used to work like this all the time, they might not be willing to change their habits.
For it is hard to get the user to accept certain changes, especially when the changes require following certain quality standards (which in many cases seems to mean more work for the user), it might be safer, to implement a system or an editing bot, which automatically converts invalid user input to a certain schema. The problem with this would be, that it had to be implemented by humans, and no

human would be able to foresee all eventualities and exceptions that a system like this would need to take into account.
So combining both of these systems might be sense. In this case, the autocorrect system should only check for the most common patterns of “wrong” user input. All the other input that is still not valid and does not fit the most common “wrong” inputs, should generate a flag for this node or tag. These flags would indicate, that someone should have a look at this data, because it might still be valid, but an exception, that is just not known to the system. Flags also would make it easy for anyone, to find data that might be problematic.

Conclusion

Most of the problems I had to deal with, did not appear because the data would be incomplete but because of a lack of standardization. I think a system that validates input and assists the user in making the correct input would really improve the quality of the data.