Friedrich-Alexander-Universität Erlangen-Nürnberg

Institut für Geographie

Vertiefte Methodik MA: Geodatenbanken-Analysen

Dozent: Dipl.-Geogr. Christian Bittner

Wintersemester 2014/15

Projekt „rank by OSM“
-Dokumentation-

vorgelegt von:

Jan Gemeinholzer (21480203)

Inhalt

Beschaffung und Beschneidung der Rohdaten

Download der Daten von overpass-turbo

```
overpass-api.de/api/interpreter?data=((area[name="Erlangen"];node(area););(area[name="Erlangen"];(way(area);>;););(area[name="Erlangen"];(rel(area);>;);););out meta;
```

Analog für Münster und Nürnberg. Achtung: Nürnberg muss über das tag „name:de“ gesucht werden. Nachteil: Bei dieser Selektionsmethode: bekommt man alle nodes, ways und relations, die den entsprechenden Namen enthalten (also viel mehr Datensätze, als man u. U. braucht). Vorteil der Selektionsmethode = Nachteil. Ursprünglich hatte ich das Projekt anders geplant, deshalb diese Selektionsmethode.

Hinweis: alle im Folgenden aufgeführten Abfragen funktionieren analog für die anderen beiden Städte. Wenn nicht, dann ist das vermerkt.

Zuschnitt der Daten

Dazu zunächst admin-6-Grenzen (Kreise und kreisfreie Städte) aus OSM extrahieren; analog für Münster und Erlangen. Die Grenzen werden dann benutzt, um die overpass-Daten zuzuschneiden. Erzeugung der Grenzen-Polygone analog zum PostGIS-Script von Christian, S. 22-23 bzw. Hausaufgabe JG No. 3; vgl. http://wiki.openstreetmap.org/wiki/Tag:type%3Dboundary für tag-select

```
create table nuernberg_boundary_admin6_ways as
select * from
(select * from relations
where tags @> 'boundary=>administrative'
and tags @> 'admin_level=>6'
and tags @> 'name:de=>Nürnberg')
as relations
cross join (select * from relation_members where member_type = 'W' and member_role= 'outer')
as members
where relations.id = members.relation_id;
```

*and member_role=‘outer‘, um ein schönes Grenzen-Multipolygon ohne Enklaven zu bekommen (vgl. Ergebnis, wenn man das weglässt. Betrifft in diesem Fall http://www.openstreetmap.org/way/194566187)

Weitere Arbeitsschritte analog zum Script von Christian S. 15 f.

```
create table nuernberg_boundary_admin6_ways_geo as
select * from nuernberg_boundary_admin6_ways
left join (select linestring, id as way_id from ways) as line_geom
on member_id = line_geom.way_id;

create table poly_temp as
select relation_id, (st_polygonize(linestring)) as poly_gc from nuernberg_boundary_admin6_ways_geo
group by relation_id;

create table nuernberg_boundary_admin6_polygons as
select polygons.relation_id, polygons.polygon, relations.version, relations.user_id, relations.tstamp, relations.changeset_id, relations.tags
from
(select relation_id, ST_CollectionExtract(poly_gc, 3) as polygon from poly_temp) as polygons
left join (select id, version, user_id, tstamp, changeset_id, tags from relations) as relations
on polygons.relation_id = relations.id;

select Populate_Geometry_Columns();
```

„Zuschnitt“ mit Funktion ST_Contains (geht genauso mit ST_Intersects, s. HA JG No. 2 und Dokumentation der Funktionen)

```
create table nuernberg_ways as
SELECT n.*
FROM ways n, nuernberg_boundary_admin6_polygons a
WHERE st_contains(a.polygon, n.linestring) = true;

create table nuernberg_nodes as
SELECT n.*
FROM nodes n, nuernberg_boundary_admin6_polygons a
WHERE st_contains(a.polygon, n.geom) = true;
```

Feststellung: Beim Export einer über den DBManager geladenen OSM-Tabelle aus QGis in das Esri-Shapefile-Format geht anscheinend die tag-Spalte (Datentyp hstore) verloren, was das Shapefile für tag-Analysen unbrauchbar macht... zum Test Reimport über:

```
shp2pgsql -I -s 4326 C:\Users\Albager\Documents\QGIS_projekte\Nuernberg_nodes_cut.shp | psql -d nuernberg -U postgres
```

Definition der Suchobjekte

Hinweis: Viele der Abfragen könnte man sicherlich noch viel eleganter lösen, als hier gezeigt! Mit tiefergehenden SQL-Kenntnissen ließe sich die Anzahl der Zwischenschritte (und damit das unweigerlich entstehende Tabellen-Chaos in der Datenbank) vielleicht erheblich reduzieren.

Abfrage Fahrradwege in OSM? -> http://wiki.openstreetmap.org/wiki/DE%3AKey%3Acycleway

Ideales Tool: http://mijndev.openstreetmap.nl/~ligfietser/fiets/index.html

Der Einfachheit halber beschränke ich die Fahrradweg-Abfrage nur auf Objekte vom Typ way. Die relations interessieren mich für dieses Projekt nicht, da ich keine zusammenhängenden Fahrrad-Routen als Fahrradwege erfassen möchte. Meine Abfrage zeigt einen sehr guten Deckungsgrad mit der Opencyclemap (hinreichend überprüft an einigen Stellen in Nürnberg, die ich selbst kenne, da zentrales Objekt meiner Untersuchungen). Fahrradwege definiere ich für meine Untersuchungen in diesem Projekt also folgendermaßen:

```
create table nuernberg_cycleways as
select * from nuernberg_ways
where tags @> 'highway=>cycleway'
or tags @> 'cycleway:moped=>yes'
or tags @> 'cycleway:moped=>no'
or tags @> 'cycleway:mofa=>no'
or tags @> 'cycleway=>track'
or tags @> 'cycleway=>opposite_track'
or tags @> 'cycleway:right=>track'
or tags @> 'cycleway:left=>track'
or tags @> 'cycleway=>lane'
or tags @> 'cycleway=>opposite_lane'
or tags @> 'cycleway:right=>lane'
or tags @> 'cycleway:left=>lane'
or tags @> 'cycleway=>shared_lane'
or tags @> 'cycleway=>opposite'
or tags @> 'bicycle=>yes'
or tags @> 'bicycle=>designated'
or tags ? 'cyclestreet'
```

Gesamtanzahl Fahrradwege:

```
create table nuernberg_cycleways_count as
SELECT COUNT (id)
FROM nuernberg_cycleways
```

„Erhebung“ eigener Variablen als OSM-Vergleich zu den ADFC Variablen:

F14_JG: Abfrage Fahrradwege + Hindernisse im Umkreis von 15 m (Dieser Abstand erfasst erst große Ampelanlagen oder Masten und Telefonzellen, die auf Haupstraßen getaggt sind, aber „in Wirklichkeit“ auf dem Fahrradweg stehen, der nebenan verläuft!) (Auswahl der tags ohne genauere Analyse der vorhandenen Tag-Population nach http://wiki.openstreetmap.org/wiki/DE:Key:barrier und http://wiki.openstreetmap.org/wiki/DE:Tag:highway%3Dtraffic_signals)

Alle Hindernisse (nodes)

```
create table nuernberg_obstacles_nodes as
select * from nuernberg_nodes
where tags @> 'highway=>traffic_signals'
or tags @> 'highway=>bus_stop'
or tags @> 'public_transport=>platform'
or tags @> 'public_transport=>stop_position'
or tags @> 'amenity=>telephone'
or tags @> 'barrier=>bollard'
or tags @> 'barrier=>kerb'
or tags @> 'barrier=>cycle_barrier'
or tags @> 'barrier=>motorcycle_barrier'
or tags @> 'barrier=>kerb'
```

```
or tags @> 'barrier=>sally_port'
or tags @> 'barrier=>sump_buster'
or tags @> 'railway=>tram'

create table nuernberg_obstacles_ways as
select * from nuernberg_ways
where tags @> 'highway=>traffic_signals'
or tags @> 'highway=>bus_stop'
or tags @> 'public_transport=>platform'
or tags @> 'public_transport=>stop_position'
or tags @> 'amenity=>telephone'
or tags @> 'barrier=>bollard'
or tags @> 'barrier=>kerb'
or tags @> 'barrier=>cycle_barrier'
or tags @> 'barrier=>motorcycle_barrier'
or tags @> 'barrier=>kerb'
or tags @> 'barrier=>sally_port'
or tags @> 'barrier=>sump_buster'
or tags @> 'railway=>tram'
```

analog für ways, damit alle tags abgefragt werden, auch wenn sie laut „OSM-Gesetz“ für den entsprechenden Elementtyp gar nicht vergeben werden dürften.

Hindernisse im Umkreis von 15 m eines Fahrradweges (nodes)

```
create table nuernberg_obstacles_nodes_cycleways as
SELECT DISTINCT ON (s.id) s.id, s.version, s.user_id, s.tstamp, s.changeset_id, s.tags, s.geom
FROM nuernberg_obstacles_nodes s
RIGHT JOIN nuernberg_cycleways h ON ST_DWithin(h.linestring, s.geom, 15, TRUE)
ORDER BY s.id
```

Hindernisse im Umkreis von 15 m (ways) -> Abfrage liefert die kompletten ways. Das betrachte ich als positiven Nebeneffekt, weil die Straßenbahnlinien zumindest in Nürnberg idealtypisch verdeutlichen, wo ausgewiesene Radwege fehlen!)

```
create table nuernberg_obstacles_ways_cycleways as
SELECT DISTINCT ON (s.id) s.id, s.version, s.user_id, s.tstamp, s.changeset_id, s.tags, s.nodes, s.bbox, s.linestring
FROM nuernberg_obstacles_ways s
RIGHT JOIN nuernberg_cycleways h ON ST_DWithin(h.linestring, s.linestring, 15, TRUE)
ORDER BY s.id
```

Zählen:

```
create table nuernberg_obstacles_nodes_count as
SELECT COUNT (DISTINCT id)
FROM nuernberg_obstacles_nodes

create table nuernberg_obstacles_ways_count as
SELECT COUNT (DISTINCT id)
FROM nuernberg_obstacles_ways

create table nuernberg_obstacles_nodes_cycleways_count as
SELECT COUNT (DISTINCT id)
FROM nuernberg_obstacles_nodes_cycleways

create table nuernberg_obstacles_ways_cycleways_count as
SELECT COUNT (DISTINCT id)
FROM nuernberg_obstacles_ways_cycleways
```

F19_JG: Abfrage Fahrradwege + surface=* + smoothness=* (http://taginfo.openstreetmap.org/keys/surface#values und http://taginfo.openstreetmap.org/keys/smoothness#values)

Smoothness; erst alle keys auslesen und zwischenspeichern, dann alle values auslesen und zählen (analog für surface):

```
create table nuernberg_smoothness_temp as
select * from nuernberg_cycleways where 'smoothness' = any(array(Select
(each(tags)).key))

create table nuernberg_smoothness_value_count as
```

```
SELECT (each(tags)).value, count(*) FROM nuernberg_smoothness_temp
GROUP BY value
ORDER BY count DESC, value;
```

Recherche „komischer tags“ – z. B. was ist DE:240? ->
http://wiki.openstreetmap.org/wiki/DE:Bicycle/Radverkehrsanlagen_kartieren_L%C3%BCbecker_Methode

Dann Auswahl der passenden tags und Abfrage der zugehörigen Fahrradwege. Ich wähle nur tags, die mindestens 10 mal vorkommen und deren values eine eindeutige Aussage über einen guten baulichen Zustand bzw. gute Befahrbarkeit machen. Dann sehe ich nach, ob diese tags tatsächlich „existieren“ und interpretiere bspw. in das tag smoothness=yes oder smoothness=designated einen user-Fehler, der eigentlich smoothness=good oder smoothness=excellent meint. Beispiel smoothness (http://wiki.openstreetmap.org/wiki/Key:smoothness), analoge Vorgehensweise für surface (http://wiki.openstreetmap.org/wiki/Key:surface). Die mit den Daten aus Nürnberg getroffene Auswahl wird aus Gründen der Einfachheit für die anderen Städte blind übernommen. Man könnte natürlich entsprechend der individuellen tag-Population jeder Stadt entsprechende Auswahlen für das Untersuchungsgebiet treffen, was dann aber der Vergleichbarkeit der Ergebnisse schaden würde…

Abfrage Fahrradwege mit guter surface & smoothness:

```
create table nuernberg_cycleways_goodquality as
select * from nuernberg_cycleways
where tags @> 'smoothness=>excellent'
or tags @> 'smoothness=>asphalt'
or tags @> 'smoothness=>designated'
or tags @> 'smoothness=>yes'
or tags @> 'smoothness=>cycleway'
or tags @> 'smoothness=>good'
or tags @> 'surface=>designated'
or tags @> 'surface=>yes'
or tags @> 'surface=>path'
or tags @> 'surface=>asphalt'
or tags @> 'surface=>paved'
or tags @> 'surface=>cycleway'
or tags @> 'surface=>excellent'
or tags @> 'surface=>paving_stones'
or tags @> 'surface=>good'
```

Abfrage Fahrradwege mit schlechter oder unzuordenbarer surface & smoothness (alles, was nach obenstehender Definition „nicht gut“ ist): Dazu den Join-typ LEFT JOIN mit id auf id verwenden („flip selection“). (http://blog.midnightmonk.com/wp-content/uploads/2012/08/Database_Joins.jpg)

```
select * from nuernberg_cycleways a
left join nuernberg_cycleways_goodquality b
on a.id=b.id
where b.id is null
```

Auswahl der tags:

smoothness		
value	count	
designated	392	drin
asphalt	209	drin
excellent	201	drin
yes	165	drin
path	152	raus
no	103	raus
cycleway	67	drin
DE:240	59	raus
good	38	drin
footway	27	raus
1	23	raus
paving_stones	14	raus
Wöhrder Wiesenweg	13	raus
DE:241	11	raus
intermediate	11	raus

smoothness	
value	
designated	drin
asphalt	drin
excellent	drin
yes	drin
cycleway	drin
good	drin

surface		
vaue	count	
designated	1262	drin
yes	1084	drin
path	740	drin
asphalt	510	drin
no	413	raus
paved	329	drin
DE:240	215	raus
cycleway	212	drin
excellent	200	drin
paving_stones	181	drin
footway	151	raus
unpaved	80	raus
official	76	raus
1	62	raus
gravel	58	raus
track	53	raus
grade2	41	raus
DE:241	40	raus
good	36	drin
service	31	raus
grade1	28	raus
DE:239,DE:1022-10	26	raus
-1	25	raus
HiRes aerial imagery	25	raus
residential	25	raus
cobblestone	23	raus
compacted	23	raus
30	20	raus
ground	20	raus
2	18	raus
pedestrian	18	raus
grade3	16	raus
50	15	raus
lane	14	raus
Wöhrder Wiesenweg	14	raus
both	12	raus
intermediate	12	raus
destination	11	raus
3	10	raus
4	10	raus
driveway	10	raus
opposite	10	raus

surface		
vaue	count	
designated	1262	drin
yes	1084	drin
path	740	drin
asphalt	510	drin
paved	329	drin
cycleway	212	drin
excellent	200	drin
paving_stones	181	drin
good	36	drin

Um sich die Ergebnisse in eine Tabelle zu schreiben, muss man logischerweise wieder die einzelnen Spalten angeben:

```
create table nuernberg_cycleways_badquality as
select a.id, a.version, a.user_id, a.tstamp, a.changeset_id, a.tags, a.nodes, a.bbox, a.linestring
from nuernberg_cycleways a
left join nuernberg_cycleways_goodquality b
on a.id=b.id
where b.id is null
```

Alle „guten“ Fahrradwege zählen:

```
create table nuernberg_cycleways_goodquality_count as
SELECT COUNT (DISTINCT id)
FROM nuernberg_cycleways_goodquality
```

Alle „schlechten“ Fahrradwege zählen:

```
create table nuernberg_cycleways_badquality_count as
SELECT COUNT (DISTINCT id)
FROM nuernberg_cycleways_badquality
```

F20_JG: Abfrage Abstellmöglichkeiten amenity=bicycle_parking und bicycle_parking=*
(http://taginfo.openstreetmap.org/keys/bicycle_parking#values)

Alle Fahrrad-Abstellmöglichkeiten (nodes):

```
create table nuernberg_amenity_bicycle_parking_nodes as
select * from nuernberg_nodes
where tags @> 'amenity=>bicycle_parking'
```

Anzahl aller Fahrrad-Abstellmöglichkeiten (nodes):

```
create table nuernberg_amenity_bicycle_parking_nodes_count as
SELECT COUNT (DISTINCT id)
FROM nuernberg_amenity_bicycle_parking_nodes
```

Alle Fahrrad-Abstellmöglichkeiten (ways):

```
create table nuernberg_amenity_bicycle_parking_ways as
select * from nuernberg_ways
where tags @> 'amenity=>bicycle_parking'
```

Anzahl aller Fahrrad-Abstellmöglichkeiten (ways):

```
create table nuernberg_amenity_bicycle_parking_ways_count as
SELECT COUNT (DISTINCT id)
FROM nuernberg_amenity_bicycle_parking_ways
```

Dann analog zu F19 Auswahl der Tags (http://wiki.openstreetmap.org/wiki/Key:bicycle_parking). -> Art der Abstellmöglichkeiten (für genauere Bewertung, wie „detailgetreu“ und „korrekt“ die OSM-Community Fahrradabstellmöglichkeiten taggt):

Auswahl der tags:

bicycle_parking			bicycle_parking gesamt	
value	count		value	
bicycle_parking	80	drin	bicycle_parking	drin
stands	72	drin	stands	drin
no	55	raus	yes	drin
yes	14	drin	wall_loops	hinzu
			wide_stands	hinzu
			anchors	hinzu
			rack	hinzu
			shed	hinzu
			lockers	hinzu
			bollard	hinzu
			ground_slots	hinzu

building	hinzu
tree	hinzu

Daraus detaillierte Abfrage „ausgewiesene Abstellflächen“ (nodes):

```
create table nuernberg_bicycle_parking_detail_nodes as
select * from nuernberg_nodes
where tags @> 'bicycle_parking=>bicycle_parking'
or tags @> 'bicycle_parking=>stands'
or tags @> 'bicycle_parking=>yes'
or tags @> 'bicycle_parking=>wall_loops'
or tags @> 'bicycle_parking=>wide_stands'
or tags @> 'bicycle_parking=>anchors'
or tags @> 'bicycle_parking=>rack'
or tags @> 'bicycle_parking=>shed'
or tags @> 'bicycle_parking=>lockers'
or tags @> 'bicycle_parking=>bollard'
or tags @> 'bicycle_parking=>ground_slots'
or tags @> 'bicycle_parking=>building'
or tags @> 'bicycle_parking=>tree'
```

detaillierte Abfrage „ausgewiesene Abstellflächen“ (ways):

```
create table nuernberg_bicycle_parking_detail_ways as
select * from nuernberg_ways
where tags @> 'bicycle_parking=>bicycle_parking'
or tags @> 'bicycle_parking=>stands'
or tags @> 'bicycle_parking=>yes'
or tags @> 'bicycle_parking=>wall_loops'
or tags @> 'bicycle_parking=>wide_stands'
or tags @> 'bicycle_parking=>anchors'
or tags @> 'bicycle_parking=>rack'
or tags @> 'bicycle_parking=>shed'
or tags @> 'bicycle_parking=>lockers'
or tags @> 'bicycle_parking=>bollard'
or tags @> 'bicycle_parking=>ground_slots'
or tags @> 'bicycle_parking=>building'
or tags @> 'bicycle_parking=>tree'
```

Zählen wie immer:

```
create table nuernberg_bicycle_parking_detail_ways_count as
SELECT COUNT (DISTINCT id)
FROM nuernberg _bicycle_parking_detail_ways

create table nuernberg_bicycle_parking_detail_nodes_count as
SELECT COUNT (DISTINCT id)
FROM nuernberg_bicycle_parking_detail_nodes
```

F23_JG: Abfrage Stadtzentrum (Punkt OSM) + Radwege im Umkreis von 3 Km / 5km

Dazu zunächst das Stadtzentrum identifizieren: Einfach den Städtenamen bei OSM eingeben, dann nach der ID des entsprechenden nodes suchen und das where-statement mit in die räumliche Abfrage schreiben (für Nürnberg: http://www.openstreetmap.org/node/1569338041):

```
create table nuernberg_cycleways_center3km as
SELECT DISTINCT ON (s.id) s.id, s.version, s.user_id, s.tstamp, s.changeset_id, s.tags, s.nodes, s.bbox, s.linestring
FROM nuernberg_cycleways s
RIGHT JOIN nuernberg_nodes h ON ST_DWithin(h.geom, s.linestring, 3000, TRUE) where h.id=1569338041
ORDER BY s.id

create table nuernberg_cycleways_center5km as
SELECT DISTINCT ON (s.id) s.id, s.version, s.user_id, s.tstamp, s.changeset_id, s.tags, s.nodes, s.bbox, s.linestring
FROM nuernberg_cycleways s
RIGHT JOIN nuernberg_nodes h ON ST_DWithin(h.geom, s.linestring, 5000, TRUE) where h.id=1569338041
ORDER BY s.id
```

Bei dieser Abfrage werden die zusammenhängenden ways selektiert, so kann es z. B. vorkommen, dass ein besonders langer way aus der 3km-Zone weit in die 5km-Zone hineinragt, weil eben mindestens ein Punkt dieses ways dann innerhalb oder genau auf der Grenze des entsprechenden Suchradius der Funktion ST_DWithin liegt.

Zählen:

```
create table nuernberg_cycleways_center3km_count as
SELECT COUNT (id)
FROM nuernberg_cycleways_center3km

create table nuernberg_cycleways_center5km_count as
SELECT COUNT (id)
FROM nuernberg_cycleways_center5km
```

Stadtzentrum als Punkt für Layer-Darstellung:

```
create table nuernberg_osm_center as
select * from nuernberg_nodes
where id=1569338041
```

Das macht aber nichts, weil es in der Kartendarstellung bspw. zeigt, dass Radwege in der Regel zerstückelt sind; der „flow“ wird beim Radfahren also in der Regel unterbrochen bzw. kommt gar nicht erst zustande. Das gibt wiederum einen Hinweis auf die Erreichbarkeit des Stadtzentrums, die ja auch vom ADFC abgefragt wurde. Um das Thema Erreichbarkeit noch besser zu beleuchten frage ich ab, welche und wie viele Fahrradwege innerhalb der 3 km-Zone der Erreichbarkeit länger als 300m, innerhalb der 5 km-Zone länger als 500m sind:

```
create table nuernberg_cycleways_center3km_length300plus as
select * from (
select id, version, user_id, tstamp, changeset_id, tags, nodes, bbox, linestring, ST_Length (ST_Transform(linestring, 32632)) length
from nuernberg_cycleways_center3km s) as length where length >300 and not tags @> 'area=>yes'

create table nuernberg_cycleways_center5km_length500plus as
select * from (
select id, version, user_id, tstamp, changeset_id, tags, nodes, bbox, linestring, ST_Length (ST_Transform(linestring, 32632)) length
from nuernberg_cycleways_center5km s) as length where length >500 and not tags @> 'area=>yes'
```

Hier wäre jetzt eine Abfrage der relations u. U. noch befriedigender (vgl. z. B. Altstadtring in Nürnberg), weil dann zusammenhängende wege erfasst würden. Als Näherung an die ADFC-Variable genügen mir für dieses Projekt aber die ways.

Anzahl der Fahrradwege und user-ids zählen:

```
create table nuernberg_cycleways_center3km_length300plus_count as
SELECT COUNT (id)
FROM nuernberg_cycleways_center3km_length300plus

create table nuernberg_cycleways_center3km_length300plus_user_id_count as
SELECT COUNT (DISTINCT user_id)
FROM nuernberg_cycleways_center3km_length300plus

create table nuernberg_cycleways_center5km_length500plus_count as
SELECT COUNT (id)
FROM nuernberg_cycleways_center5km_length500plus

create table nuernberg_cycleways_center5km_length500plus_user_id_count as
SELECT COUNT (DISTINCT user_id)
FROM nuernberg_cycleways_center5km_length500plus
```

F25_JG: Abfrage Einbahnstraßen bicycle=opposite Anzahl + (highway=* + oneway=yes + oneway:bicycle=no)

```
create table nuernberg_cycleway_opposite as
select * from nuernberg_ways
where tags @> 'cycleway=>opposite'

create table nuernberg_oneway_yes as
select * from nuernberg_ways
where tags @> 'oneway:bicycle=>no'
and tags @> 'oneway=>yes'
```

Zählen:

```
create table nuernberg_cycleway_opposite_count as
SELECT COUNT (id)
FROM nuernberg_cycleway_opposite

create table nuernberg_oneway_yes_count as
SELECT COUNT (id)
```

```
FROM nuernberg_oneway_yes
```
(um herauszufinden, wie viele richtig getaggt sind)

F27_JG: Abfrage Fahrradverleih amenity=bicycle_rental Anzahl
(*Der Einfachheit halber beschränke ich diese Abfrage auf Objekte vom Typ node)*

Alle Fahrradverleih-Stationen
```
create table nuernberg_amenity_bicycle_rental as
SELECT * FROM nuernberg_nodes
where tags @> 'amenity=>bicycle_rental'
```

Anzahl Fahrradverleih-Stationen
```
create table nuernberg_amenity_bicycle_rental_count as
SELECT COUNT (DISTINCT id)
FROM nuernberg_nodes
where tags @> 'amenity=>bicycle_rental'
```

Eigene Variablen:

JG1: Anzahl der edits pro user im Untersuchungsgebiet / Bevölkerung
Die user_id wird immer dann eingetragen, wenn ein edit stattfindet – d.h. die Abfrage „aller" user IDs fragt die Einträge in der Datenbank danach, wie viele verschiedene user_ids die Momentaufnahme des Untersuchungsgebiets enthält. Sie zeigt nicht, welche bzw. wie viele user_ids in der gesamten history von OSM in diesem Gebiet vorhanden sind!

Alle user-IDs (nodes):
```
create table nuernberg_user_id_nodes as
SELECT DISTINCT user_id
FROM nuernberg_nodes
```

Anzahl der user-IDs (nodes):
```
create table nuernberg_user_id_nodes_count as
SELECT COUNT (DISTINCT user_id)
FROM nuernberg_nodes
```

Alle user-IDs (ways):
```
create table nuernberg_user_id_ways as
SELECT DISTINCT user_id
FROM nuernberg_ways
```

Anzahl der user-IDs (ways):
```
create table nuernberg_user_id_ways_count as
SELECT COUNT (DISTINCT user_id)
FROM nuernberg_ways
```

Folgende Abfragen liefern uns alle user_id + alle entsprechenden name (aus users). Jetzt muss die Anfrage entsprechend umgebaut werden, so dass eine geordnete Liste mit der Anzahl der editierten nodes/ways und dem namen des users (aus users) entsteht.
```
create table nuernberg_top5users_ways_count as
SELECT t1.user_id AS id_des_users, t2.name AS name_des_users, COUNT (user_id) AS ways_des_users
FROM nuernberg_ways AS t1
INNER JOIN users AS t2
ON t1.user_id = t2.id
GROUP BY t1.user_id, t2.name
ORDER BY ways_des_users DESC
LIMIT 5;

create table nuernberg_top5users_nodes_count as
SELECT t1.user_id AS id_des_users, t2.name AS name_des_users, COUNT (user_id) AS nodes_des_users
FROM nuernberg_nodes AS t1
INNER JOIN users AS t2
ON t1.user_id = t2.id
GROUP BY t1.user_id, t2.name
ORDER BY nodes_des_users DESC
LIMIT 5;
```

Rausfinden, welche nodes+ways von den top5-usern bearbeitet wurden und in Tabelle speichern für Layerdarstellung:

```
create table nuernberg_top5users_nodes as
select * from nuernberg_nodes a inner join  nuernberg_top5users_nodes_count b on a.user_id=b.id_des_users

create table nuernberg_top5users_ways as
select * from nuernberg_ways a inner join  nuernberg_top5users_ways_count b on a.user_id=b.id_des_users
```

Durchschnittliche Versionsnummer der von den top5-Usern bearbeiteten Elemente im Untersuchungsbiet:

```
select avg(version)from nuernberg_nodes a inner join  nuernberg_top5users_nodes_count b on a.user_id=b.id_des_users



select avg(version)from nuernberg_ways a inner join  nuernberg_top5users_ways_count b on a.user_id=b.id_des_users
```

Abfrage der von den restlichen usern bearbeiteten Elemente (nur für Layerdarstellung, kann man auch über die Summen in Excel rauskriegen; das ist auch eine gute Kontrolle für die Abfragen):

```
create table nuernberg_restusers_nodes as
select * from nuernberg_nodes a
left join nuernberg_top5users_nodes_count b
on a.user_id=b.id_des_users
where b.id_des_users is null

create table nuernberg_restusers_ways as
select * from nuernberg_ways a
left join nuernberg_top5users_ways_count b
on a.user_id=b.id_des_users
where b.id_des_users is null
```

Durchschnittliche Versionsnummern:

```
select avg(version) from nuernberg_nodes a
left join nuernberg_top5users_nodes_count b
on a.user_id=b.id_des_users
where b.id_des_users is null

select avg(version) from nuernberg_ways a
left join nuernberg_top5users_ways_count b
on a.user_id=b.id_des_users
where b.id_des_users is null
```

Anzahl der Objekte, die im Untersuchungsgebiet noch in der ersten Version vorliegen:

```
select count (id) from nuernberg_nodes where version=1

select count (id) from nuernberg_ways where version=1
```

JG2: Abfrage Fahrradläden shop=bicycle / Bevölkerung (http://wiki.openstreetmap.org/wiki/DE:Tag:shop%3Dbicycle, http://taginfo.openstreetmap.org/search?q=service%3Abicycle)
(*Der Einfachheit halber beschränke ich diese Abfrage auf Objekte vom Typ node)*

Alle Fahrradläden:

```
create table nuernberg_shop_bicycle as
SELECT * FROM nuernberg_nodes
where tags @> 'shop=>bicycle'
or tags @> 'service=>bicycle'
```

Anzahl der Fahrradläden:

```
create table nuernberg_shop_bicycle_count as
SELECT COUNT (DISTINCT id)
FROM nuernberg_nodes
where tags @> 'shop=>bicycle'
or tags @> 'service=>bicycle'
```

JG3: Bewertung der Elemente im Untersuchungsgebiet (Berechnung in Excel)

Flächenberechnung der Untersuchungsgebiete: (zum check Vergleich mit Wikipedia. Ergebnis in m², also teilen durch 1000000 für km²)

```
create table nuernberg_admin6_area as
select ST_Area(ST_Transform(polygon, 32632))from nuernberg_boundary_admin6_polygons
```