

P3 Wrangle Data

Udacity Data analyst Training

Marcus Pauli | p3 Wrangle Data | 26.02.2017

### Reviewer notes:

As the project did not state out to use an explicit technology, I did all that with a virtual Ubuntu version. How to accomplish the whole task with the Anaconda/Notebook looked far too complicated and would have needed even more time to accomplish.

Thus I did it with below mentioned tools and the additions within the appendix to accomplish the actual SQL statements and results to get the results to complete the project.

The SQL statements are written down within this document.

The actual SQL-DB would be available but I can’t put it on GitHUB due to the size of 98MB exceeding the 25MB upload size per file.

### Target 1: Choose Your Map Area

Choose any area of the world from [**https://www.openstreetmap.org**](https://www.openstreetmap.org/#map=5/51.500/-0.100), and download a XML OSM dataset. The dataset should be at least 50MB in size (uncompressed). We recommend using one of following methods of downloading a dataset:

* Download a preselected metro area from [**Map Zen**](https://mapzen.com/data/metro-extracts/).
* Use the [**Overpass API**](http://overpass-api.de/query_form.html) to download a custom square area.

Explanation of the syntax can found in the [**wiki**](http://wiki.openstreetmap.org/wiki/Overpass_API). In general you will want to use the following query:(node(minimum\_latitude, minimum\_longitude, maximum\_latitude, maximum\_longitude);<;);out meta; e.g. (node(51.249,7.148,51.251,7.152);<;);out meta; the meta option is included so the elements contain timestamp and user information. You can use the Open Street Map [**Export Tool**](http://www.openstreetmap.org/export#map=5/42.618/-7.559) to find the coordinates of your bounding box. Note: You will not be able to use the Export Tool to actually download the data, the area required for this project is too large.

### Solution:

The Area of southern Bavaria in Germany was chosen as data source and the data was

downloaded accordingly from: <http://download.geofabrik.de/europe/germany/bayern/oberbayern-latest.osm.bz2>

## Target 2: Process your Dataset

It is recommended that you start with the problem sets in your chosen course and modify them to suit your chosen data set. As you unravel the data, take note of problems encountered along the way as well as issues with the dataset. You are going to need these when you write your project report.

## SQL

Thoroughly audit and clean your dataset, converting it from XML to CSV format. Then import the cleaned .csv files into a SQL database using [**this schema**](https://gist.github.com/swwelch/f1144229848b407e0a5d13fcb7fbbd6f) or a custom schema of your choice.

## Solution

With postgresql and PostGIS

Prepare postgres for the data

**sudo** -u postgres -i

createuser gis

createdb -E UTF8 -O gis gis

psql -f **/**usr**/**share**/**postgresql**/**9.3**/**contrib**/**postgis-2.1**/**postgis.sql -d gis

psql -f **/**usr**/**share**/**postgresql**/**9.3**/**contrib**/**postgis-2.1**/**spatial\_ref\_sys.sql -d gis

psql -f **/**usr**/**share**/**postgresql**/**9.3**/**contrib**/**postgis-2.1**/**postgis\_comments.sql -d gis

**echo** "alter table geometry\_columns owner to gis; alter table spatial\_ref\_sys owner to gis;" **|** psql -d gis

**echo** "create extension hstore;" **|** psql -d gis

Get the raw data:

Get OpenStreet data for my region of Germany:

Wget http://download.geofabrik.de/europe/germany/bayern/oberbayern-latest.osm.bz2

Import the raw data:

osm2pgsql -c -d gis -C 2048 --hstore --slim /home/mpauli/P3\_Data\_Wrangle/oberbayern-latest.osm.bz2

* node cache: stored: 13663936(100.00%), storage efficiency: 50.82% (dense blocks: 1387, sparse nodes: 12734263), hit rate: 100.00%

**13663936 Nodes imported from the datafile of southern Bavaria.**

Postactions on the postgres to enhance performance (creating indices)

**sudo** -u postgres -i psql gis

CREATE INDEX idx\_planet\_osm\_point\_tags ON planet\_osm\_point USING gist**(**tags**)**;

CREATE INDEX idx\_planet\_osm\_polygon\_tags ON planet\_osm\_polygon USING gist**(**tags**)**;

CREATE INDEX idx\_planet\_osm\_line\_tags ON planet\_osm\_line USING gist**(**tags**)**;

\q

## Answering Questions on the datasource:

Find restaurants within southern bavaria:

**SELECT** name, ST\_AsText(ST\_Transform(way,4326)) **AS** pt\_lonlattext

**FROM** planet\_osm\_point

**WHERE** amenity='restaurant';

gis=# SELECT name, ST\_AsText(ST\_Transform(way,4326)) AS pt\_lonlattext

gis-# FROM planet\_osm\_point

gis-# WHERE amenity='restaurant';

name | pt\_lonlattext

-------------------------------------------------------------------------+------------------------------------------

Gletschergarten | POINT(10.9795116856019 47.4131929633555)

Gletscherrestaurant SonnAlpin | POINT(10.9799992013066 47.4133826872785)

Panorama Gipfelrestaurant | POINT(10.9840178147301 47.4213887055371)

Knorrhütte | POINT(11.0127748632684 47.4100231779253)

Reintalangerhütte | POINT(11.0356812741927 47.4053133154913)

Eibsee-Alm | POINT(10.9928391809886 47.4546240801471)

Restaurant Alpspitze | POINT(11.0510024006895 47.4391943885217)

Höllentaleingangshütte | POINT(11.0444883572382 47.4483891182407)

Kreuzjochhaus | POINT(11.0734764524676 47.4515120675542)

Bockhütte | POINT(11.0943020956994 47.4181706040471)

| POINT(11.1130838021651 47.4193425032333)

Lisa Hütte | POINT(11.0860896973719 47.4430368647164)

Wettersteinalm | POINT(11.1442002757972 47.4297560035039)

Lautersee Stub'n | POINT(11.2324476252056 47.4381316931787)

**Results were truncated …**

Find restaurants serving pizza within southern bavaria:

**SELECT** name, ST\_AsText(ST\_Transform(way,4326)) **AS** pt\_lonlattext

**FROM** planet\_osm\_point

**WHERE** amenity='restaurant' **AND** tags @> 'cuisine=>pizza';

**name | pt\_lonlattext**

**------------------------------------+------------------------------------------**

**Angelina's Pizza-Heim-Service | POINT(10.8846801343185 47.8113651190073)**

**Colosseo | POINT(11.0901564604947 47.492625918016)**

**Ristorante da Noi | POINT(11.2003933104169 47.6773875120891)**

**Ristorante Pizzeria Galeria | POINT(10.8007638323895 48.069122202831)**

**Grillos Holzofenpizza | POINT(10.874744228287 48.0515311634334)**

**Antica Roma | POINT(10.8848790213224 48.0460879780449)**

**Landhaus | POINT(11.1264243231235 47.8301202705061)**

**Bella Italia | POINT(11.1463494052829 47.8449419612061)**

**Pizzeria Da Pietro | POINT(11.1000289454671 47.9472109690419)**

**Sportheim | POINT(10.9336016660395 48.0664203116721)**

**Ristorante Pizzeria Mediterraneo | POINT(11.2481527814863 48.0744285037287)**

**Pizzeria Vier Jahreszeiten | POINT(11.1856463870498 48.8925111688232)**

**Da Alfonso | POINT(11.3144232966264 47.6555265713339)**

**El Lago | POINT(11.8654722599559 47.7330864638232)**

**La Locanda Locanda | POINT(12.1855551110908 47.6116677888042)**

**Tropea Da Bobby | POINT(12.1878468930436 47.6099318651742)**

**Results were truncated …**

Find restaurants in Ottobrunn (my hometown):

**SELECT** name, tags, ST\_AsText(ST\_Transform(way,4326)) **AS** pt\_lonlattext

**FROM** planet\_osm\_point

**WHERE** tags @> 'addr:city=>Ottobrunn';

gis=# SELECT name, ST\_AsText(ST\_Transform(way,4326)) AS pt\_lonlattext

FROM planet\_osm\_point

WHERE amenity='restaurant' AND tags @> 'addr:city=>Ottobrunn';

**name | pt\_lonlattext**

**-----------------------------------+------------------------------------------**

**Nefeli im Phönix | POINT(11.6523698658383 48.0569127702025)**

**Asia Garden | POINT(11.6646628613438 48.0593236676815)**

**Giannis | POINT(11.6642053493696 48.0595751140524)**

**Asahi | POINT(11.6662243129706 48.0587087956779)**

**Taj Palace | POINT(11.6673393019013 48.0594653015453)**

**Das Nimrods | POINT(11.6639590313187 48.0614998869332)**

**Chop Stick - Running Sushi | POINT(11.6624574973213 48.062809464647)**

**Villa Meraviglia | POINT(11.6623044243969 48.0642946690474)**

**Happy Quynh | POINT(11.6627057018343 48.0639727052979)**

**Bistro Luigi Enoteca & Ristorante | POINT(11.6635988967213 48.0645567171164)**

**Vu Garden | POINT(11.6641324061685 48.0648421768968)**

**Quo Vadis | POINT(11.6641951984069 48.0647619121301)**

**Ayinger Alm | POINT(11.6652234999126 48.0649007094637)**

**Das Wirtshaus am Rathausplatz | POINT(11.6655286576146 48.0650581767467)**

**Bella Roma | POINT(11.6585540479172 48.0679117150216)**

**Taverna Artemis | POINT(11.6568928832938 48.0706521850695)**

**Ristorante Pattio D'oro | POINT(11.6625857767438 48.0655903681733)**

**VINO e CUCINA | POINT(11.6641748964815 48.0666143934543)**

**Ristorante Cristalina | POINT(11.6662887221765 48.0655047016339)**

**Nissos | POINT(11.6782292187646 48.0624766850937)**

**ottofonti | POINT(11.6788494156367 48.0625453064189)**

**Maharani | POINT(11.668440187282 48.0673745650675)**

**Trattoria Portofino | POINT(11.6770623072105 48.069298205928)**

**Balkan Grill | POINT(11.6895553575298 48.0660005090129)**

**da Bruno | POINT(11.6917742861131 48.0726922707044)**

Who has committed the most data?

SELECT user, COUNT(\*) FROM open\_street.nodes

GROUP BY user

;

|  |  |
| --- | --- |
| ToniE | 271359 |
| rolandg | 53268 |
| Rainero | 24353 |
| ludwich | 19676 |

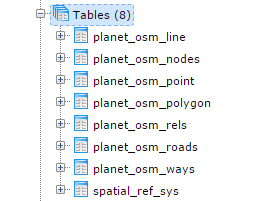
## Target 3: Explore your Database

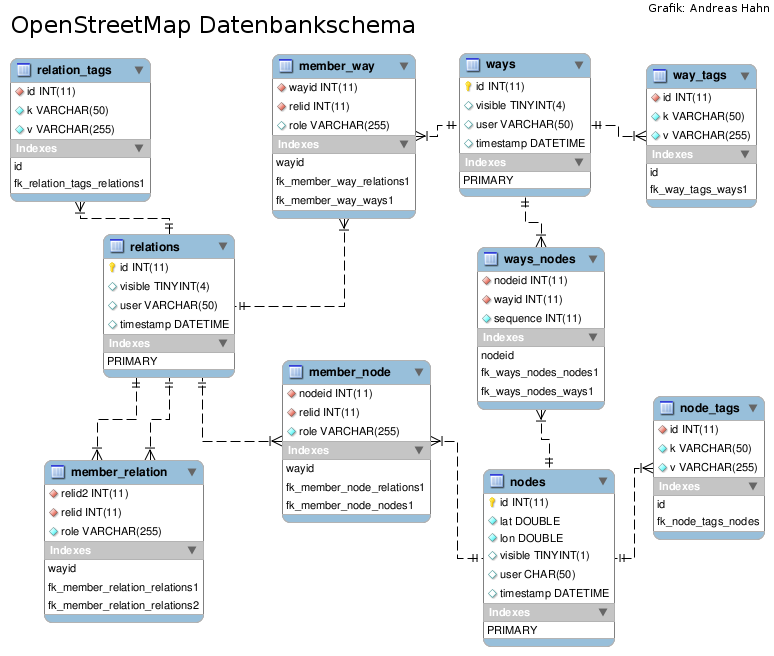
After building your local database you’ll explore your data by running queries. Make sure to document these queries and their results in the submission document described below. See the [**Project Rubric**](https://review.udacity.com/#!/projects/3168208620/rubric) for more information about query expectations.

Solution:

Below is the converted table structure of the PostgresDB:

PostgreSQL Tables:



The actual raw data of openstreet has the below structure/schema:  


## Target 4: Document your Work

Create a document (pdf, html) that directly addresses the following sections from the [**Project Rubric**](https://review.udacity.com/#!/projects/3168208620/rubric).

* Problems encountered in your map
* Overview of the Data
* Other ideas about the datasets

Try to include snippets of code and problematic tags (see [**MongoDB Sample Project**](https://docs.google.com/document/d/1F0Vs14oNEs2idFJR3C_OPxwS6L0HPliOii-QpbmrMo4/pub) or [**SQL Sample Project**](https://gist.github.com/carlward/54ec1c91b62a5f911c42#file-sample_project-md)) and visualizations in your report if they are applicable.

Use the following code to take a systematic sample of elements from your original OSM region. Try changing the value of k so that your resulting SAMPLE\_FILE ends up at different sizes. When starting out, try using a larger k, then move on to an intermediate k before processing your whole dataset.

## Solution:

This document is my documentation, the data is been held within this document. The source data was far too big for GitHUB.

Appendix:

**Information sources:**

http://craigthomas.ca/blog/category/openstreetmap/

http://www.mysqltutorial.org/mysql-subquery/

https://de.slideshare.net/jynus/query-optimization-with-mysql-57-and-mariadb-10-even-newer-tricks

http://wiki.openstreetmap.org/wiki/User:Tagtheworld

http://goblor.de/wp/2009/10/16/openstreetmap-projekt-teil-1-openstreetmap-daten-in-mysql-datenbank-einlesen/

mysqldump.exe --defaults-file="c:\users\mpauli~1.ccs\appdata\local\temp\tmpetkjmq.cnf" --user=root --host=127.0.0.1 --protocol=tcp --port=3306 --default-character-set=utf8 --routines --skip-triggers "open\_street"

<https://review.udacity.com/#!/rubrics/25/view>

Create DB for

postgres=# create extension hstore;

|  |
| --- |
| -- Database creation script for the snapshot PostgreSQL schema. |
|  |  |
|  | -- Drop all tables if they exist. |
|  | DROP TABLE IF EXISTS actions; |
|  | DROP TABLE IF EXISTS users; |
|  | DROP TABLE IF EXISTS nodes; |
|  | DROP TABLE IF EXISTS ways; |
|  | DROP TABLE IF EXISTS way\_nodes; |
|  | DROP TABLE IF EXISTS relations; |
|  | DROP TABLE IF EXISTS relation\_members; |
|  | DROP TABLE IF EXISTS schema\_info; |
|  |  |
|  | -- Drop all stored procedures if they exist. |
|  | DROP FUNCTION IF EXISTS osmosisUpdate(); |
|  |  |
|  |  |
|  | -- Create a table which will contain a single row defining the current schema version. |
|  | CREATE TABLE schema\_info ( |
|  | version integer NOT NULL |
|  | ); |
|  |  |
|  |  |
|  | -- Create a table for users. |
|  | CREATE TABLE users ( |
|  | id int NOT NULL, |
|  | name text NOT NULL |
|  | ); |
|  |  |
|  |  |
|  | -- Create a table for nodes. |
|  | CREATE TABLE nodes ( |
|  | id bigint NOT NULL, |
|  | version int NOT NULL, |
|  | user\_id int NOT NULL, |
|  | tstamp timestamp without time zone NOT NULL, |
|  | changeset\_id bigint NOT NULL, |
|  | tags hstore |
|  | ); |
|  | -- Add a postgis point column holding the location of the node. |
|  | SELECT AddGeometryColumn('nodes', 'geom', 4326, 'POINT', 2); |
|  |  |
|  |  |
|  | -- Create a table for ways. |
|  | CREATE TABLE ways ( |
|  | id bigint NOT NULL, |
|  | version int NOT NULL, |
|  | user\_id int NOT NULL, |
|  | tstamp timestamp without time zone NOT NULL, |
|  | changeset\_id bigint NOT NULL, |
|  | tags hstore, |
|  | nodes bigint[] |
|  | ); |
|  |  |
|  |  |
|  | -- Create a table for representing way to node relationships. |
|  | CREATE TABLE way\_nodes ( |
|  | way\_id bigint NOT NULL, |
|  | node\_id bigint NOT NULL, |
|  | sequence\_id int NOT NULL |
|  | ); |
|  |  |
|  |  |
|  | -- Create a table for relations. |
|  | CREATE TABLE relations ( |
|  | id bigint NOT NULL, |
|  | version int NOT NULL, |
|  | user\_id int NOT NULL, |
|  | tstamp timestamp without time zone NOT NULL, |
|  | changeset\_id bigint NOT NULL, |
|  | tags hstore |
|  | ); |
|  |  |
|  | -- Create a table for representing relation member relationships. |
|  | CREATE TABLE relation\_members ( |
|  | relation\_id bigint NOT NULL, |
|  | member\_id bigint NOT NULL, |
|  | member\_type character(1) NOT NULL, |
|  | member\_role text NOT NULL, |
|  | sequence\_id int NOT NULL |
|  | ); |
|  |  |
|  |  |
|  | -- Configure the schema version. |
|  | INSERT INTO schema\_info (version) VALUES (6); |
|  |  |
|  |  |
|  | -- Add primary keys to tables. |
|  | ALTER TABLE ONLY schema\_info ADD CONSTRAINT pk\_schema\_info PRIMARY KEY (version); |
|  |  |
|  | ALTER TABLE ONLY users ADD CONSTRAINT pk\_users PRIMARY KEY (id); |
|  |  |
|  | ALTER TABLE ONLY nodes ADD CONSTRAINT pk\_nodes PRIMARY KEY (id); |
|  |  |
|  | ALTER TABLE ONLY ways ADD CONSTRAINT pk\_ways PRIMARY KEY (id); |
|  |  |
|  | ALTER TABLE ONLY way\_nodes ADD CONSTRAINT pk\_way\_nodes PRIMARY KEY (way\_id, sequence\_id); |
|  |  |
|  | ALTER TABLE ONLY relations ADD CONSTRAINT pk\_relations PRIMARY KEY (id); |
|  |  |
|  | ALTER TABLE ONLY relation\_members ADD CONSTRAINT pk\_relation\_members PRIMARY KEY (relation\_id, sequence\_id); |
|  |  |
|  |  |
|  | -- Add indexes to tables. |
|  | CREATE INDEX idx\_nodes\_geom ON nodes USING gist (geom); |
|  |  |
|  | CREATE INDEX idx\_way\_nodes\_node\_id ON way\_nodes USING btree (node\_id); |
|  |  |
|  | CREATE INDEX idx\_relation\_members\_member\_id\_and\_type ON relation\_members USING btree (member\_id, member\_type); |
|  |  |
|  |  |
|  | -- Set to cluster nodes by geographical location. |
|  | ALTER TABLE ONLY nodes CLUSTER ON idx\_nodes\_geom; |
|  |  |
|  | -- Set to cluster the tables showing relationship by parent ID and sequence |
|  | ALTER TABLE ONLY way\_nodes CLUSTER ON pk\_way\_nodes; |
|  | ALTER TABLE ONLY relation\_members CLUSTER ON pk\_relation\_members; |
|  |  |
|  | -- There are no sensible CLUSTER orders for users or relations. |
|  | -- Depending on geometry columns different clustings of ways may be desired. |
|  |  |
|  | -- Create the function that provides "unnest" functionality while remaining compatible with 8.3. |
|  | CREATE OR REPLACE FUNCTION unnest\_bbox\_way\_nodes() RETURNS void AS $$ |
|  | DECLARE |
|  | previousId ways.id%TYPE; |
|  | currentId ways.id%TYPE; |
|  | result bigint[]; |
|  | wayNodeRow way\_nodes%ROWTYPE; |
|  | wayNodes ways.nodes%TYPE; |
|  | BEGIN |
|  | FOR wayNodes IN SELECT bw.nodes FROM bbox\_ways bw LOOP |
|  | FOR i IN 1 .. array\_upper(wayNodes, 1) LOOP |
|  | INSERT INTO bbox\_way\_nodes (id) VALUES (wayNodes[i]); |
|  | END LOOP; |
|  | END LOOP; |
|  | END; |
|  | $$ LANGUAGE plpgsql; |
|  |  |
|  |  |
|  | -- Create customisable hook function that is called within the replication update transaction. |
|  | CREATE FUNCTION osmosisUpdate() RETURNS void AS $$ |
|  | DECLARE |
|  | BEGIN |
|  | END; |
|  | $$ LANGUAGE plpgsql; |
|  |  |
|  | -- Manually set statistics for the way\_nodes and relation\_members table |
|  | -- Postgres gets horrible counts of distinct values by sampling random pages |
|  | -- and can be off by an 1-2 orders of magnitude |
|  |  |
|  | -- Size of the ways table / size of the way\_nodes table |
|  | ALTER TABLE way\_nodes ALTER COLUMN way\_id SET (n\_distinct = -0.08); |
|  |  |
|  | -- Size of the nodes table / size of the way\_nodes table \* 0.998 |
|  | -- 0.998 is a factor for nodes not in ways |
|  | ALTER TABLE way\_nodes ALTER COLUMN node\_id SET (n\_distinct = -0.83); |
|  |  |
|  | -- API allows a maximum of 2000 nodes/way. Unlikely to impact query plans. |
|  | ALTER TABLE way\_nodes ALTER COLUMN sequence\_id SET (n\_distinct = 2000); |
|  |  |
|  | -- Size of the relations table / size of the relation\_members table |
|  | ALTER TABLE relation\_members ALTER COLUMN relation\_id SET (n\_distinct = -0.09); |
|  |  |
|  | -- Based on June 2013 data |
|  | ALTER TABLE relation\_members ALTER COLUMN member\_id SET (n\_distinct = -0.62); |
|  |  |
|  | -- Based on June 2013 data. Unlikely to impact query plans. |
|  | ALTER TABLE relation\_members ALTER COLUMN member\_role SET (n\_distinct = 6500); |
|  |  |
|  | -- Based on June 2013 data. Unlikely to impact query plans. |
|  | ALTER TABLE relation\_members ALTER COLUMN sequence\_id SET (n\_distinct = 10000); |

Working MySQL Version f the schema:

SET SQL\_MODE="NO\_AUTO\_VALUE\_ON\_ZERO";

--

-- Datenbank: `osm`

--

-- --------------------------------------------------------

--

-- Tabellenstruktur für Tabelle `member\_node`

--

CREATE TABLE `member\_node` (

`nodeid` int(11) NOT NULL,

`relid` int(11) NOT NULL,

`role` varchar(255) NOT NULL,

KEY `wayid` (`nodeid`,`relid`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--

-- Tabellenstruktur für Tabelle `member\_relation`

--

CREATE TABLE `member\_relation` (

`relid2` int(11) NOT NULL,

`relid` int(11) NOT NULL,

`role` varchar(255) NOT NULL,

KEY `wayid` (`relid2`,`relid`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--

-- Tabellenstruktur für Tabelle `member\_way`

--

CREATE TABLE `member\_way` (

`wayid` int(11) NOT NULL,

`relid` int(11) NOT NULL,

`role` varchar(255) default NULL,

KEY `wayid` (`wayid`,`relid`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--

-- Tabellenstruktur für Tabelle `nodes`

--

CREATE TABLE `nodes` (

`id` int(11) NOT NULL,

`lat` double NOT NULL,

`lon` double NOT NULL,

`visible` tinyint(1) default NULL,

`user` char(50) default NULL,

`timestamp` datetime default NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--

-- Tabellenstruktur für Tabelle `node\_tags`

--

CREATE TABLE `node\_tags` (

`id` int(11) NOT NULL,

`k` varchar(50) NOT NULL,

`v` varchar(255) NOT NULL,

KEY `id` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--

-- Tabellenstruktur für Tabelle `relations`

--

CREATE TABLE `relations` (

`id` int(11) NOT NULL,

`visible` tinyint(4) default NULL,

`user` varchar(50) default NULL,

`timestamp` datetime default NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--

-- Tabellenstruktur für Tabelle `relation\_tags`

--

CREATE TABLE `relation\_tags` (

`id` int(11) NOT NULL,

`k` varchar(50) NOT NULL,

`v` varchar(255) NOT NULL,

KEY `id` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--

-- Tabellenstruktur für Tabelle `ways`

--

CREATE TABLE `ways` (

`id` int(11) NOT NULL,

`visible` tinyint(4) default NULL,

`user` varchar(50) default NULL,

`timestamp` datetime default NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--

-- Tabellenstruktur für Tabelle `ways\_nodes`

--

CREATE TABLE `ways\_nodes` (

`nodeid` int(11) NOT NULL,

`wayid` int(11) NOT NULL,

`sequence` int(11) NOT NULL,

KEY `nodeid` (`nodeid`,`wayid`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--

-- Tabellenstruktur für Tabelle `way\_tags`

--

CREATE TABLE `way\_tags` (

`id` int(11) NOT NULL,

`k` varchar(50) NOT NULL,

`v` varchar(255) NOT NULL,

KEY `id` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;