HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

COMPUTER SCIENCE

IFS
INSTITUTE FOR
SOFTWARE

OSMNames
Local Open Street Name Database

Term Project

Department of Computer Science
University of Applied Science Rapperswil

Spring Term 2016

Author: Andreas Egloff
Advisor: Prof. Stefan Keller

ABSTRACT

There is a need for a data set consisting of street names (geo names) of the world. Such gazetteer data however is either not available for every country (openaddresses.io) or is not in a suitable format. Furthermore, if such data is found, it is often not for free.

Another problem we face is formatting postal addresses. Each country has its own set of formatting rules and in order to have a database of addresses one needs to bring these different formats to a common format and vice versa.

This thesis includes work in the following areas: First and foremost, the main objective was creating a global and structured data set with street names with the help of OSM data. Additionally, similar products, mostly Nominatim had been analyzed in order to get familiar with the topic. Second, the workflow had to be dockerized so that it can be easily deployed on any system. Finally, an export in a suitable format had to be made publicly available so it can be used in conjunction with a geocoder.

A data set containing planet data could be provided. A secondary target, importing additional house numbers for each street, could not be met in scope of this project. The workflow could be setup in such a way, that it can easily be setup anywhere (via Docker). The simplicity of the installation and, most of all, the clear arrangement of code is a big advantage over other products like Nominatim. After all, the processing run times are way faster than the latter, which takes up to several days for a global data set.

The resulting and easy accessible data export has been successfully integrated by Klokan Technologies with a SphinxSearch powered geocoder.

Further information is available on https://osmnames.org.

KEYWORDS: OPENSTREETMAP, DOCKER, POSTGIS, ADDRESSES, GAZETTEER, GEOCODER, GEO NAMES

MANAGEMENT SUMMARY

Problem Statement

There is a need for a data set consisting of street names (geo names) of the world. Such gazetteer data however is either not available for every country (openaddresses.io) or is not in a suitable format. Furthermore, if such data is found, it is often not for free.

Another problem we face is formatting postal addresses. Each country has its own set of formatting rules and in order to have a database of addresses one needs to bring these different formats to a common format and vice versa.

Finally, such a dataset could be used in conjunction with a geocoder in various applications.

Approach

The main objective is creating a global and structured data set with street names with the help of OpenStreetMap (OSM) data. There are two ways to look at this assignment as depicted in the following L shaped figure.

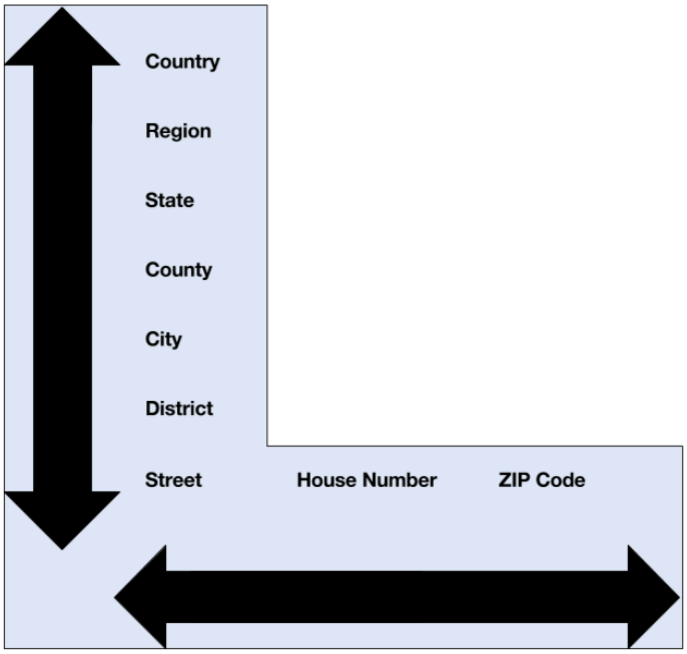

FIGURE A – POSTAL ADDRESS REPRESENTATION (C)SK

One way is to look at the vertical axis representing the hierarchy and the street names, the other being the horizontal, contextual axis representing a postal address. The focus of this work will be on creating a global data set representing the vertical axis.

The tasks can be summarized as follows:

- Analyze similar products, mostly Nominatim in order to get familiar with the topic.
- Create a Docker workflow that imports OSM data and does the ranking of the features.
- Refine data quality especially when it comes to street segments that belong together.
- With the help of the ranking from above the features should get hierarchized.
- An export should be provided so it can be used in conjunction with a geocoder.
- The data should be made available to download in a usable format.

The whole Docker workflow can be run regularly at a later date.

Achievements

First of all, a data set containing planet data could be provided. A secondary target, importing additional house numbers for each street, could not be met in scope of this project.

The workflow could be setup in such a way, that it can easily be setup anywhere (via Docker). The simplicity of the installation and, most of all, the clear arrangement of code is a big advantage over other products like Nominatim. After all, the processing run times are way faster than the latter, which takes up to several days for a global data set.

The TSV file from the planet export includes 21'055'840 entries. The current data export can be downloaded at https://osmnames.org .

The resulting data have been successfully integrated by Klokan Technologies with a SphinxSearch powered geocoder. The result includes osm2vectortiles-generated vector tiles and can be seen here: https://osmnames.klokantech.com/ .

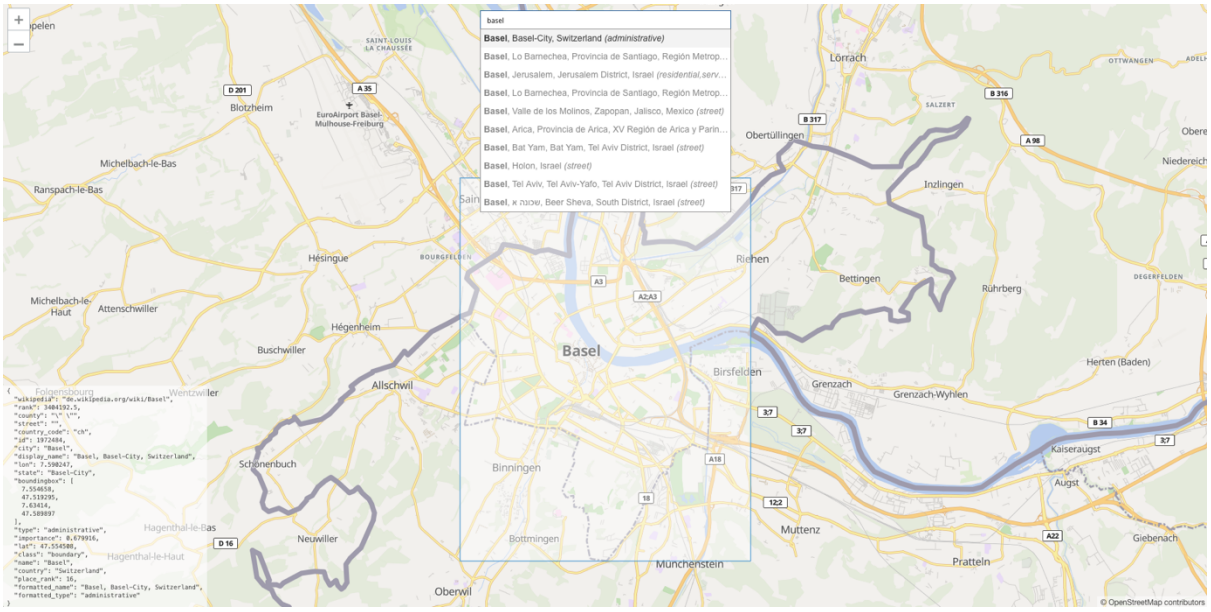

FIGURE B – OSMNAMES USED AS A GEOCODER

The implementation makes use of the calculated importance in order to sort the results. The bounding box is also visualized. The meta-data can be seen in the lower-left corner.

Outlook

The current version 1.1 is quite satisfactory and could already be implemented in a real life application as seen above. However, there are still a few improvements that can be developed.

First and foremost, the database and Docker settings need to be tweaked, so the processing times go down even more. An enhancement that can benefit from these improvements is differential updates. With diff updates enabled one can think of automated regular updates of the name database. Such updates take a really short time compared to a full planet import and consequently improve the product overall.

Secondly, it would be great to have house numbers in the database as well. This extension however brings a further challenge when it comes to the export. Further work has to be done in designing the export format, unless we want to have an entry in the export file for every house number, resulting in a humongous file size.

DECLARATION OF ORIGINALITY

I hereby declare that

- this thesis and the work reported herein was composed by and originated entirely from me unless stated otherwise in the assignment of tasks or agreed otherwise in writing with the supervisor.
- all information derived from the published and unpublished work of others has been acknowledged in the text and references are correctly given in the bibliography.
- no copyrighted material (e.g. images) has been used illicitly in this work.

PLACE, DATE

SIGNATURE

TABLE OF CONTENTS

1 Introduction

This work consists of five main parts. The first chapter defines the problem statement as well as what goals need to be achieved. A short overview over the state of the art concludes this chapter.

The second chapter pinpoints the single parts of a typical requirements engineering process covering use cases, sequence and activity diagrams, as well as non-functional requirements.

The third chapter sketches an architectural view on the parts to be implemented.

The fourth chapter delves into the implementation itself. It consists of the documentation of the source code.

The fifth chapter ultimately presents the results that were achieved in scope of this work and gives a possible outlook for further work. It summarizes with a personal reflection on the project.

1.1 Problem Statement

There is a need for a data set consisting of street names (geo names) of the world. Such gazetteer data however is either not available for every country (openaddresses.io) or is not in a suitable format. Furthermore, if such data is found, it is often not for free.

Another problem we face is formatting postal addresses. Each country has its own set of formatting rules and in order to have a database of addresses one needs to bring these different formats to a common format and vice versa.

Finally, such a dataset could be used in conjunction with a geocoder in various applications.

1.2 PROJECT AIM

The main objective is creating a global and structured data set with street names with the help of OpenStreetMap (OSM) data. There are two ways to look at this assignment as depicted in the following L shaped figure.

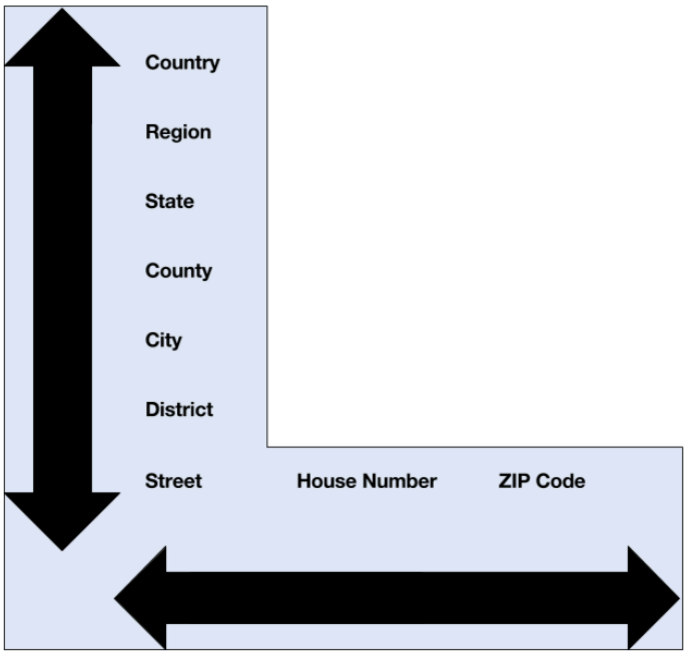

FIGURE 1-1 POSTAL ADDRESS REPRESENTATION (C)SK

One way is to look at the vertical axis representing the hierarchy and the street names, the other being the horizontal, contextual axis representing a postal address. The focus of this work will be on creating a global data set representing the vertical axis.

The tasks can be summarized as follows:

- Analyze similar products, mostly Nominatim in order to get familiar with the topic.
- Create a Docker workflow that imports OSM data and does the ranking of the features.
- Refine data quality especially when it comes to street segments that belong together.
- With the help of the ranking from above the features should get hierarchized.
- An export should be provided so it can be used in conjunction with a geocoder.
- The data should be made available to download in a usable format.

The whole Docker workflow can be run regularly at a later date.

The whole process planning as well as the target-performance comparison can be found in Appendix C - Project Management.

1.3 State Of The Art

The most prominent candidate providing similar functionalities as stipulated is Nominatim. It also makes use of OSM input data and as a (reverse) geocoder even powers the OpenStreetMap website itself.

Other products worth mentioning include Mapzen's 'Who's on First' along with Pelias, Geonames, Quattroshapes, OpenAdresses and OpenCageData some of which are described in more detail in Appendix B - Documentation.

In terms of address normalization and parsing there is libpostal which does a wonderful job in language classification, transliteration among other things as well as separating an address string into its components. It uses OpenCage's address formatting rules.

2 REQUIREMENTS ENGINEERING

2.1 USE CASES

The following diagram presents the use cases and how they interoperate with each other. UC_3 is not part of this work but is specified altogether for the sake of completeness.

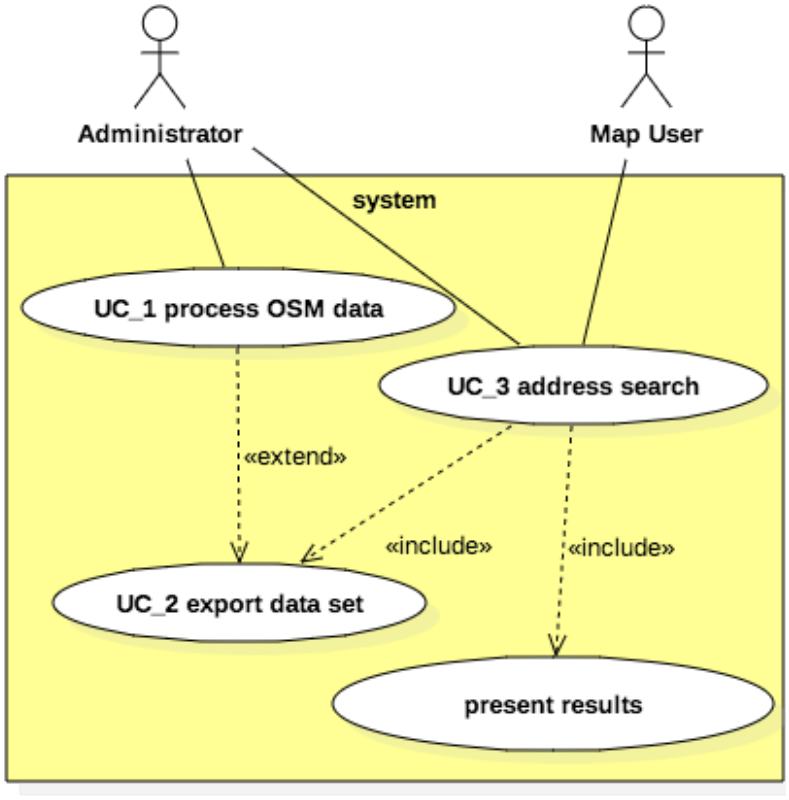

FIGURE 2-1 USE CASES

Use Case	UC_01
Use Case Name	Process OSM data
Use Case Description	The administrator wants some specified OSM data processed and as a result being able to export a hierarchized and structured data set.
Actor	Administrator
Pre-condition	• Sources cloned from GitHub • Docker installed • PBF file downloaded.
Basic Flow	The following steps represent Docker workflows. 1. The database gets initialized 2. Wikipedia data (for defining importance of places) gets downloaded and imported 3. The schema gets initialized 4. Now the actual OSM import happens with the help of imposm3. Furthermore, data refinements, ranking, hierarchy creation and street segments merging is done.
Post-condition	The administrator has processed OSM data in the database.
Alternative Flows	In order to get the PBF file there exists an optional Docker workflow. One needs to edit the URL in the source code in case a specific country extract is required.

TABLE 2-1 UC_1

Use Case	UC_02
Use Case Name	Export data set
Use Case Description	The administrator wants to export the processed data.
Actor	Administrator
Pre-condition	OSM data processed -> UC_1
Basic Flow	The data gets exported in a suitable TSV format and gzipped in the end.
Post-condition	The administrator gets an export of the database reflecting all the imported and processed attributes containing hierarchy and merged streets.
Alternative Flows	None

TABLE 2-2 UC_2

2.2 Sequence Diagrams

The following sequence diagram emphasizes the flow of control and data in the system when processing the OSM data.

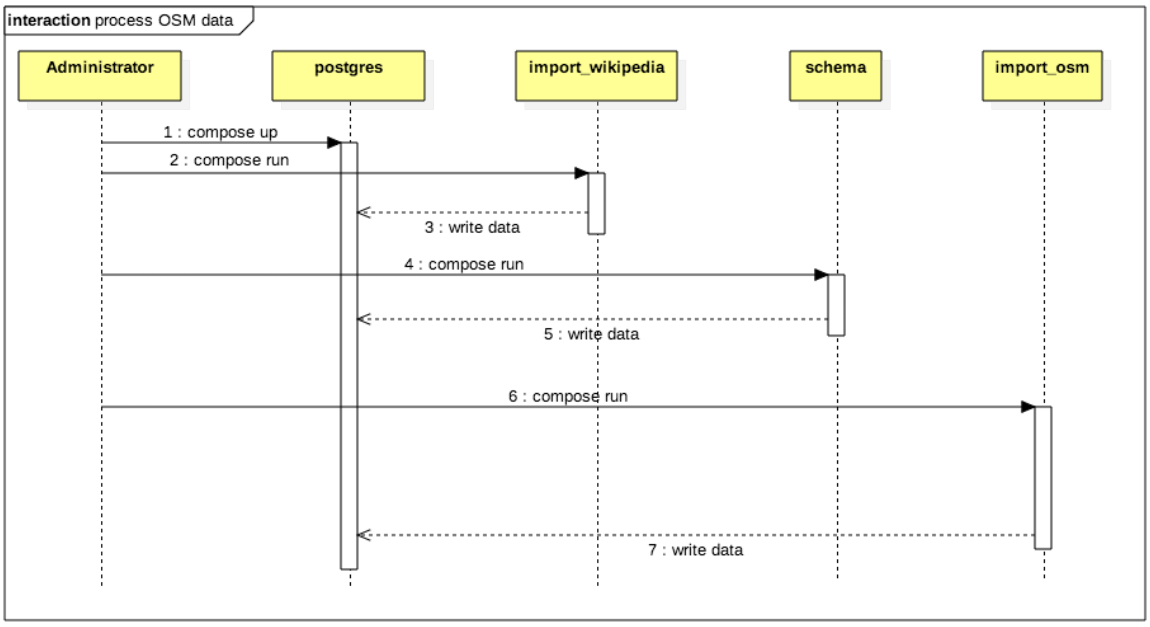

FIGURE 2-2 PROCESS OSM DATA

Finally, the administrator can export the data set.

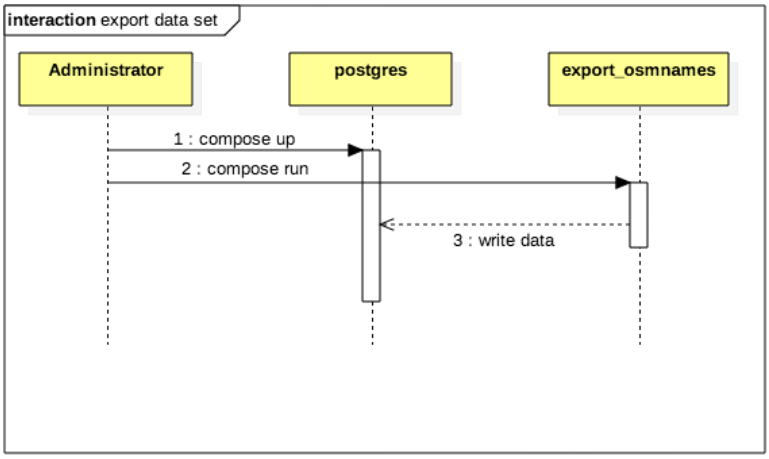

FIGURE 2-3 EXPORT DATA SET

2.3 Activity Diagrams

The following activity diagram shows the importing workflow as a whole.

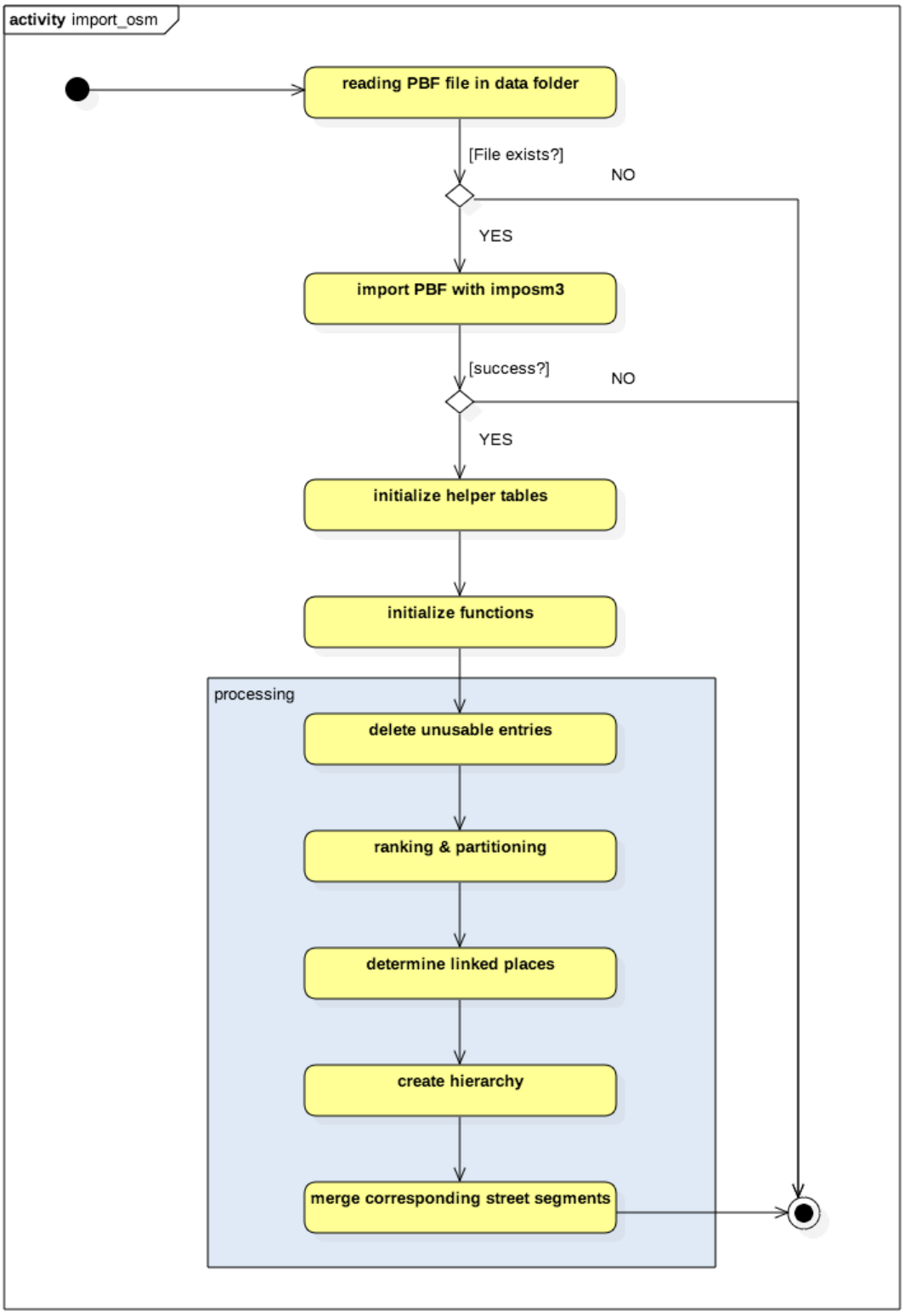

TABLE 2-3 IMPORT OSM DATA

2.4 Non-Functional Requirements

The following quality attributes should be taken into account:

ID	Description
NFR_01	The whole importing and processing workflow should not take more than 30 hours for a planet dump to process.
NFR_02	The resulting TSV must be correctly formatted. In order to achieve this requirement, possible tabs in the imported data need to be eliminated prior to the export.
NFR_03	Features with faulty geometries and/or empty names in the import data should be ignored.

TABLE 2-4 NON-FUNCTIONAL REQUIREMENTS

3 Architecture & Design

3.1 Overview

OSMNames is built with Docker and is therefore shipped in containers. This allows to have an extra layer of abstraction and avoids overhead of a real virtual machine. Specifically, it is built with Docker compose thus allowing to define a multi-container architecture defined in a single file. The Docker compose YAML file looks as follows:

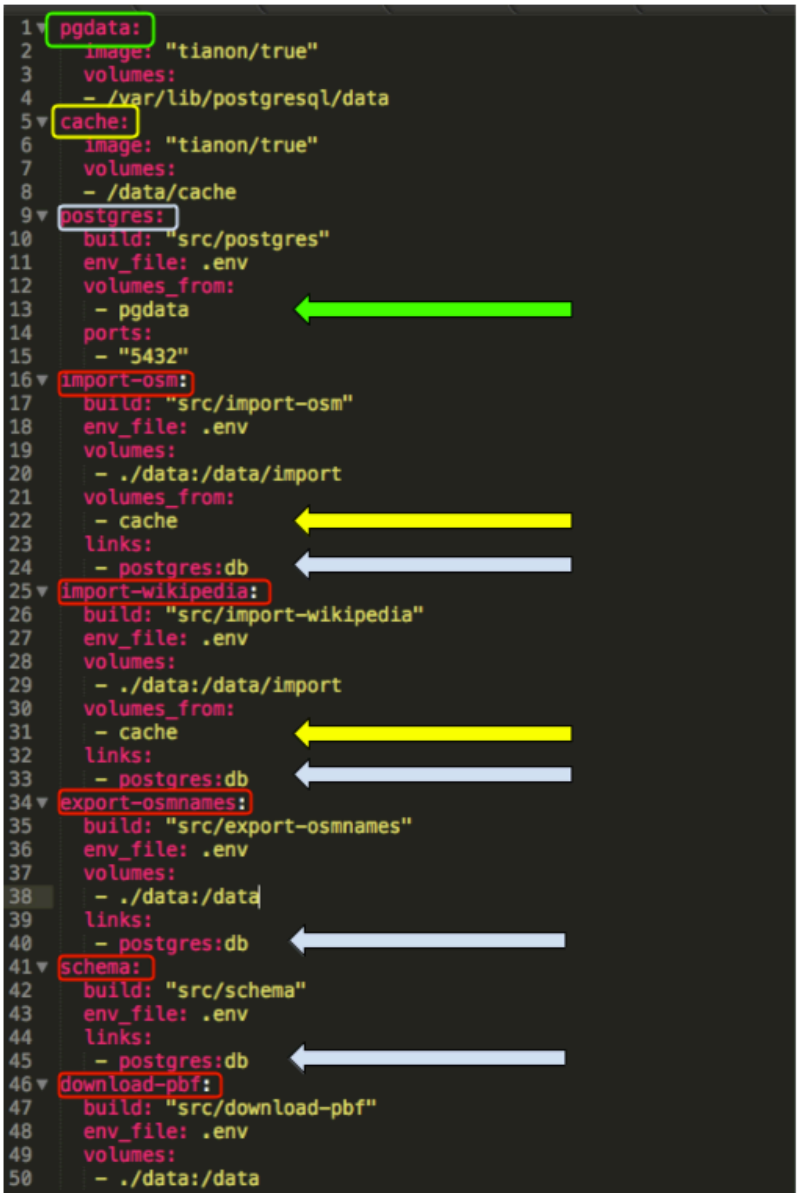

FIGURE 3-1 DOCKER COMPOSE FILE

The data-only container pattern is used in *cache* and *pgdata*. These containers are mounted from *import-osm* and *import-wikipedia* for storing cache data and in *postgres* for storing the database files.

The single containers can be started started separately but the whole environment can be easily brought up with a single *docker compose up* command.

Dockerfiles meet best practices followed defined by Docker [1].

3.1.1 DOCKER ARCHITECTURE

Docker uses a client-server architecture [2]. The client connects to the Docker daemon, which ultimately does the building, running and finally distributing the Docker containers. Client and daemon do not necessarily have to run on the same machine.

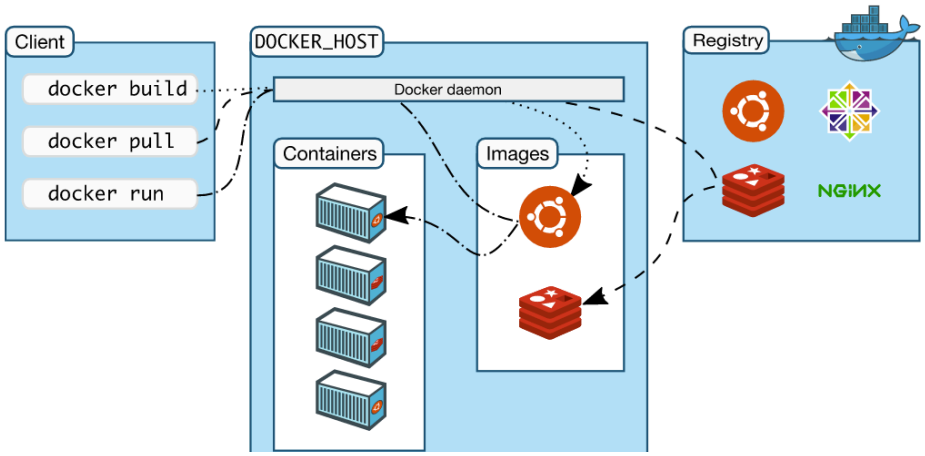

FIGURE 3-2 DOCKER ARCHITECTURE

Figure 3-2 illustrates the architecture.

Other Docker essentials include images, registries and containers.

Docker images are read-only templates and can for instance contain already installed software such as an Ubuntu Linux and an nginx webserver. They are used for creating Docker containers. Docker images are built with the *docker build* command.

Docker registries are used for distributing existing Docker images. They are either private or public (the Docker Hub). Docker images are downloaded from Docker registries with the *docker pull* command.

Docker containers are built from Docker images. They contain everything needed for an application to run and have their own file system and networking. Each container is an isolated application platform and can be run, started, stopped, moved and deleted. Docker containers are run with the *docker run* command.

3.2 DATA MODEL

The tables in the data model can be categorized in two categories. Helper tables and OSM data tables. The data model looks as follows:

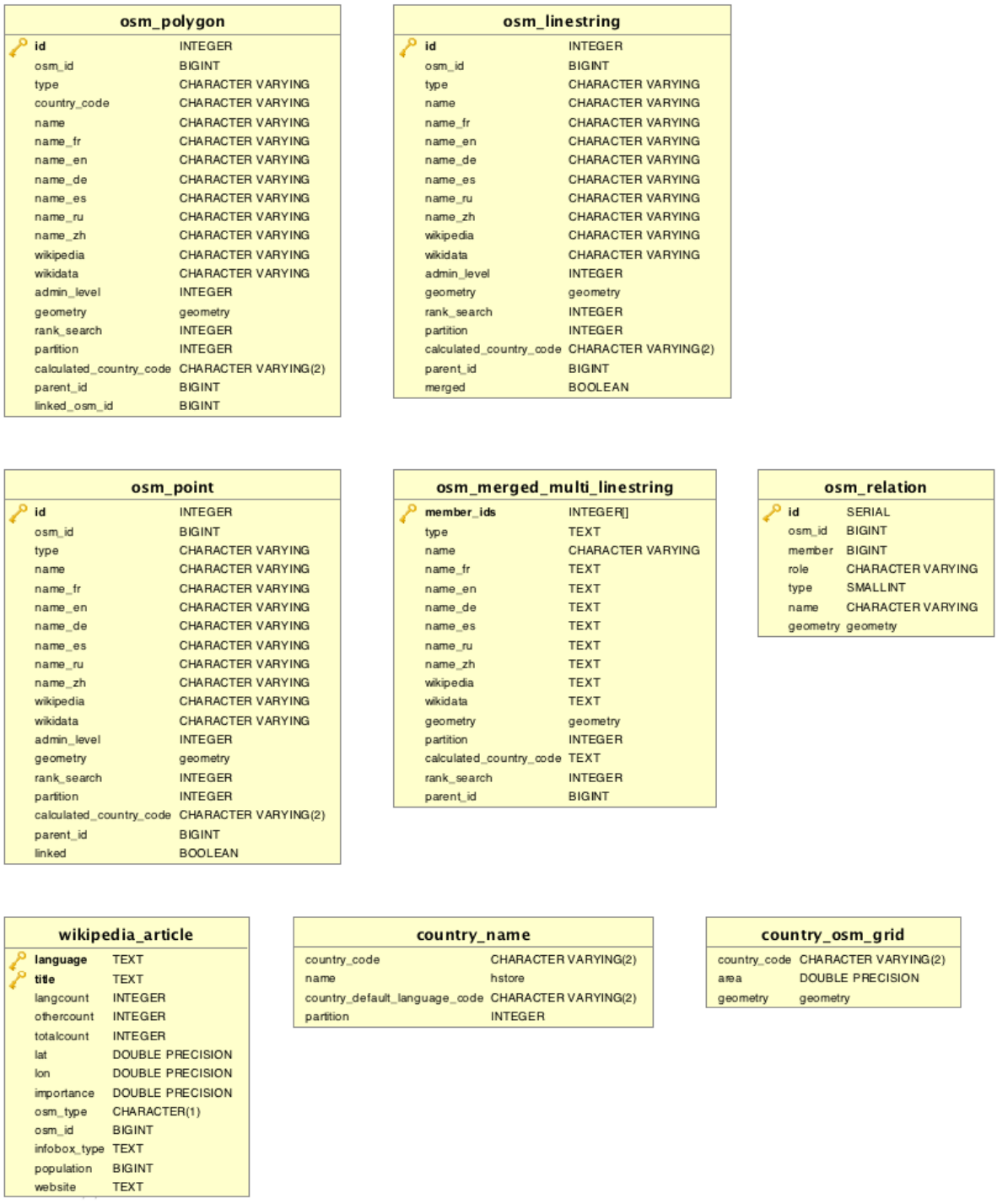

FIGURE 3-3 OSMNAMES DATA MODEL

Note that the bottom tables are helper tables and initialized before importing OSM data. The OSM tables are being constructed during the import by imposm3. Note that

the osm_id in the OSM tables is not necessarily the original *osm_id* from OpenStreetMap, but rather a special kind where *osm_ids* of relations are negated in order to prevent collisions with way IDs.

3.3 Export: Data Format

The requirement of the export data format is being simple to use. The decision led to using an UTF-8 encoded TSV like Geonames, where the first line contains the column names. Compared to a CSV, names are now allowed to have usual delimiters such as commas or semicolons [3]. The definition looks as follows:

Column Name	Description
name	The name of the feature (default language is en, others available are de, es, fr, ru, zh)
alternative_names	All other available and distinct names separated by commas
osm_type	The OSM type of the feature (node, way, relation)
osm_id	The unique osm_id for debug purposes
class	The class of the feature e.g. *boundary*
type	The type of the feature e.g. *administrative*
lon	The decimal degrees (WGS84) longitude of the centroid of the feature
lat	The decimal degrees (WGS84) latitude of the centroid of the feature
place_rank	Rank from 1-30 ascending, 1 being the highest. Calculated with the *type* and *class* of the feature.
importance	Importance of the feature, ranging [0.0-1.0], 1.0 being the most important. Calculated with wikipedia information or the *place_rank*.
street	The name of the street if the feature is some kind of street
city	The name of the city of the feature, if it has one
county	The name of the county of the feature, if it has one
state	The name of the state of the feature, it it has one
country	The name of the country of the feature
country_code	The ISO-3166 2-letter country code of the feature
display_name	The display name of the feature representing the hierarchy, if available in English
west	The western decimal degrees (WGS84) longitude of the bounding box of the feature
south	The southern decimal degrees (WGS84) latitude of the bounding box of the feature

east	The eastern decimal degrees (WGS84) longitude of the bounding box of the feature
north	The northern decimal degrees (WGS84) latitude of the bounding box of the feature
wikidata	The wikidata associated with the feature
wikipedia	The wikipedia URL associated with the feature

TABLE 3-1 OSMNAMES DATA EXPORT FORMAT

4 IMPLEMENTATION

4.1 OVERVIEW

This chapter shows in-depth some of the more interesting parts of the implementation. First of all, it addresses how and what kind of OSM data is imported. Second of all, it highlights how the data is processed and refined in the end. Third of all, it shows how the export of the data is done. Finally, it pinpoints some technical details in terms of DB performance and presents some functions used in OSMNames.

4.2 OSM IMPORT

OSM Data comes in different file formats. The whole planet is as big as 666GB uncompressed XML data [4]. This is also why initially it made sense to start with a smaller extract such as Switzerland. OSMNames uses PBF file formats which is way smaller (31GB) and therefore faster to download and much faster to process.

4.2.1 OSM DATA IN GENERAL

There are three data models in the OSM data model:

- Nodes
- Ways
- Relations

All of these components can have associated tags [5].

Nodes represent specific point on earth's surface with its latitude and longitude. They can describe standalone features such as bus stops. They are also used as points along ways. Nodes can also be included as members of relations.

Ways are ordered list of 2-2000 nodes and represent as polylines streets or rivers. As closed ways they can also represent traffic circles. Even more, as boundaries of areas they describe buildings or forests. In this special case the last node must be the same as the first one in the list. They are not suitable to represent multipolygon data.

Relations are multi-purpose data structures defining a relationship between two or more OSM data models. One worth mentioning is as a multipolygon describing an area with holes. Basically, relations are ordered lists of nodes, ways or other relations. Each *member* can have a *role* describing its purpose. Typically, the relation has at least a *type* tag defining its meaning.

Tags are used with every of the three data models. They are basic *key-value* fields with no fixed dictionary but rather following OSM conventions. The purpose is adding meaning to geographic data.

4.2.2 IMPOSM MAPPING

For importing OSM data into Postgres, imposm3 by Omniscale is used in favor of osm2pgsql mainly because of its superior speed results. It makes heavy use of parallel processing favoring multicore systems. Explicit tag filters are set in order to have only the relevant data imported. Due to the fact that imposm3 cannot import multiple geometry types into a single table, separate tables are created for points, linestrings as well as polygons.

This is an excerpt of how the mapping YAML could look like [6]:

```
tables:
   landusages:
     type: polygon
     mapping:
       natural: [wood, land]
       tourism: [zoo]
     …
```

LISTING 4-1

The features matching the following tags are imported:

Key	Values
place	• city • borough • suburb • quarter • neighbourhood • town • village • hamlet
landuse	• residential
boundary	• administrative
highway	• motorway

	• motorway_link • trunk • trunk_link • primary • primary_link • secondary • secondary_link • tertiary • tertiary_link • unclassified • residential • road • living_street • raceway • construction • track • service • path • cycleway • steps • bridleway • footway • corridor • crossing

TABLE 4-1 IMPOSM MATCHING

The following fields are then incorporated:

Key	Type	Description
id	Integer	the osm id (negative for relations)
geometry	Geometry	polygon, point or linestring
type	String	the mapping value from the table above
name	String	the name used locally
name_en	String	English (if available)
name_de	String	German (if available)
name_fr	String	French (if available)
name_es	String	Spanish (if available)
name_ru	String	Russian (if available)
name_zh	String	Chinese (if available)
wikipedia	String	wikipedia link
wikidata	String	wikidata Hash

admin_level	Integer	originally used for differentiate border rendering, now used for ranking
ISO3166-1:alpha2	String	the ISO 3166 2-letter country code
member_id	Integer	the id of the member
member_role	String	the role of the member
member_type	String	the type of the member

TABLE 4-2 IMPOSM IMPORTED TAGS

The complete mapping can be found in Documentation.

4.3 PROCESSING

After importing the OSM data with imposm3 the real processing begins. Each of the steps taken is described in this sub-chapter.

4.3.1 DELETE UNUSABLE ENTRIES

Since the goal is to have names in the data set, each entry with an empty name in all imported languages is useless and therefore deleted. Instead of NULL values, imposm3 writes empty strings which has to be accounted for.

Additionally, since the export should be in TSV format, any entries containing tabs are deleted as well.

```
DELETE FROM osm_polygon_tmp WHERE (name <> '' OR name_fr <> '' OR name_en
<> '' OR name_de <> '' OR name_es <> '' OR name_ru <> '' OR name_zh <> '')
IS FALSE;

--remove tabs, so the export in tsv is valid
UPDATE osm_polygon_tmp SET name =  regexp_replace(name,'\t', ' ') WHERE
name LIKE '%'||chr(9)||'%';
UPDATE osm_polygon_tmp SET name_fr =  regexp_replace(name_fr,'\t', ' ')
WHERE name_fr LIKE '%'||chr(9)||'%';
UPDATE osm_polygon_tmp SET name_en =  regexp_replace(name_en,'\t', ' ')
WHERE name_en LIKE '%'||chr(9)||'%';
UPDATE osm_polygon_tmp SET name_de =  regexp_replace(name_de,'\t', ' ')
WHERE name_de LIKE '%'||chr(9)||'%';
UPDATE osm_polygon_tmp SET name_es =  regexp_replace(name_es,'\t', ' ')
WHERE name_es LIKE '%'||chr(9)||'%';
UPDATE osm_polygon_tmp SET name_ru =  regexp_replace(name_ru,'\t', ' ')
WHERE name_ru LIKE '%'||chr(9)||'%';
UPDATE osm_polygon_tmp SET name_zh =  regexp_replace(name_zh,'\t', ' ')
WHERE name_zh LIKE '%'||chr(9)||'%';
```

LISTING 4-2

Note that this extract shows the empty name and tab removal only for one table.

4.3.2 *Ranking & Partitioning*

For every geometry type a new table is created since this is far more effective than altering the old tables and updating every single row (see chapter 4.6). Additionally, the according rank and partition are calculated.

```
 CREATE TABLE osm_polygon AS
(SELECT
    id,
    osm_id,
    type,
    country_code,
    name,
    name_fr,
    name_en,
    name_de,
    name_es,
    name_ru,
    name_zh,
    wikipedia,
    wikidata,
    admin_level,
    geometry,
    rpc.rank_search AS rank_search,
    rpc.partition AS partition,
    rpc.calculated_country_code AS calculated_country_code,
    NULL::bigint AS parent_id,
    NULL::bigint AS linked_osm_id
FROM
    osm_polygon_tmp p,
    determineRankPartitionCode(type, geometry, osm_id, country_code) AS rpc
);
```

LISTING 4-3

Pivotal to this process is the ranking for places and addresses as follows:

```
CREATE OR REPLACE FUNCTION rank_place(type TEXT, osmID bigint)
RETURNS int AS $$
BEGIN
      RETURN CASE
            WHEN type IN ('administrative') THEN 2*(SELECT
COALESCE(admin_level,15) FROM osm_polygon_tmp o WHERE osm_id = osmID)
            WHEN type IN ('continent', 'sea') THEN 2
            WHEN type IN ('country') THEN 4
            WHEN type IN ('state') THEN 8
            WHEN type IN ('county') THEN 12
            WHEN type IN ('city') THEN 16
            WHEN type IN ('island') THEN 17
            WHEN type IN ('region') THEN 18 -- dropped from previous value
of 10
            WHEN type IN ('town') THEN 18
            WHEN type IN
('village','hamlet','municipality','district','unincorporated_area','boroug
h') THEN 19
            WHEN type IN
('suburb','croft','subdivision','isolated_dwelling','farm','locality','isle
t','mountain_pass') THEN 20
            WHEN type IN ('neighbourhood', 'residential') THEN 22
            WHEN type IN ('houses') THEN 28
```

```
            WHEN type IN ('house','building') THEN 30
            WHEN type IN ('quarter') THEN 30
      END;
END;
$$ LANGUAGE plpgsql IMMUTABLE;


CREATE OR REPLACE FUNCTION rank_address(type TEXT)
RETURNS int AS $$
BEGIN
      RETURN CASE
            WHEN type IN
('service','cycleway','path','footway','steps','bridleway','motorway_link',
'primary_link','trunk_link','secondary_link','tertiary_link') THEN 27
            ELSE 26
      END;
END;
$$ LANGUAGE plpgsql IMMUTABLE;
```

LISTING 4-4

Note that these value mappings are the same as in Nominatim. If not available, the country code is calculated along with its partition code (unique integer value for each country) with the help of the pre-initialized table *country_osm_grid*.

```
CREATE OR REPLACE FUNCTION get_country_code(place geometry) RETURNS TEXT
  AS $$
DECLARE
  place_centre GEOMETRY;
  nearcountry RECORD;
BEGIN
  place_centre := ST_PointOnSurface(place);

  FOR nearcountry IN select country_code from country_osm_grid where
st_covers(geometry, place_centre) order by area asc limit 1
  LOOP
    RETURN nearcountry.country_code;
  END LOOP;

  FOR nearcountry IN select country_code from country_osm_grid where
st_dwithin(geometry, place_centre, 0.5) order by st_distance(geometry,
place_centre) asc, area asc limit 1
  LOOP
    RETURN nearcountry.country_code;
  END LOOP;

  RETURN NULL;
END;
$$
LANGUAGE plpgsql IMMUTABLE;
```

LISTING 4-5

4.3.3 *Determine linked places*

In order to determine linked places (points linked with polygons) additional tags about the relations are imported. Specifically, the *role* values *admin_centre* and *label* are of interest.

```
-- places with admin_centre tag
      UPDATE osm_polygon p
      SET linked_osm_id = r.member
      FROM osm_relation r
      WHERE
      r.type = 0 AND (r.role = 'admin_centre' OR r.role = 'admin_center')
      AND p.name = r.name
      AND p.osm_id = r.osm_id
      AND p.linked_osm_id IS NULL;
```

LISTING 4-6

This information is later on used in the export mainly to rule out point features linked to their polygon features as well as determining city types instead of administrative types.

4.3.4 *Create Hierarchy*

In order to create the *display_name,* the parent feature of every feature is determined with the following function:

```
CREATE OR REPLACE FUNCTION determineParentPlace(id_value BIGINT,
partition_value INT, rank_search_value INT, geometry_value GEOMETRY)
RETURNS BIGINT AS $$
DECLARE
  retVal BIGINT;
BEGIN
  FOR current_rank  IN REVERSE rank_search_value..1 LOOP
     SELECT id FROM osm_polygon WHERE partition=partition_value AND
rank_search = current_rank AND NOT id=id_value AND ST_Contains(geometry,
geometry_value) AND NOT ST_Equals(geometry, geometry_value) INTO retVal;
     IF retVal IS NOT NULL THEN
      return retVal;
    END IF;
  END LOOP;
RETURN retVal;
END;
$$ LANGUAGE plpgsql;
```

LISTING 4-7

With the reverse loop it is ensured to match only features with the same or a lower rank. Also, by checking geometry equality it is ensured that no infinite loop emerge (parent of feature A is feature B whose parent is feature A). This phenomenon was identified with European OSM data where geometry duplicates with different ids exist. Finally, only features with the same partition are considered.

For road features a different function is used (see chapter 4.7.1).

4.3.5 *Merge corresponding street segments*

In order to merge streets segments that belong together, a new table *osm_merged_multi_linestring* is created. The ids are being aggregated into an array, the type into a comma separated string. Linestrings are merged to a multi-linestring when they have at least one point in common.

```
CREATE TABLE osm_merged_multi_linestring AS
      SELECT array_agg(DISTINCT a.id) AS member_ids,
      string_agg(DISTINCT a.type,',') AS type,
      a.name, max(a.name_fr) AS name_fr,
      max(a.name_en) AS name_en,
      max(a.name_de) AS name_de,
      max(a.name_es) AS name_es,
      max(a.name_ru) AS name_ru,
      max(a.name_zh) AS name_zh,
      max(a.wikipedia) AS wikipedia,
      max(a.wikidata) AS wikidata,
      ST_UNION(array_agg(ST_MakeValid(a.geometry))) AS geometry,
      bit_and(a.partition) AS partition,
      max(a.calculated_country_code) AS calculated_country_code,
      min(a.rank_search) AS rank_search,
      a.parent_id
      FROM
            osm_linestring AS a,
            osm_linestring AS b
      WHERE
            ST_Touches(ST_MakeValid(a.geometry), ST_MakeValid(b.geometry))
AND
            a.parent_id = b.parent_id AND
            a.parent_id IS  NOT NULL AND
            a.name = b.name AND
            a.id!=b.id
      GROUP BY
            a.parent_id,
            a.name;
```

LISTING 4-8

Note that before merging, invalid geometries are attempted to be made valid without loosing vertices.

4.4 Wikipedia Import & Importance

In order to have an importance value for each feature, a wikipedia helper table is being downloaded from a Nominatim server. This is the same information Nominatim uses to determine the importance. It was decided to take this pre-calculated data instead of calculating it itself due to longer processing times (up to several days!). Also, the same calculations are applied, in order to achieve the same results.

If a feature has a wikipedia URL a matching entry in the wikipedia helper table is taken for calculating the importance with the following formula:

$$importance = \frac{\log(totalcount)}{\log\left(\max(totalcount)\right)}$$

EQUATION 4-1

Where *totalcount* is the number of references to the article from other wikipedia articles. In case there is no wikipedia information or no match was found, the following formula is applied:

$$importance = 0.75 - (rank/40)$$

EQUATION 4-2

Since every feature has a rank, it is ensured that every feature also has an importance.

4.5 EXPORT

The data for the TSV is extracted with the help of the pgclimb tool which takes an SQL file as an argument [7]. The results of the SELECT statements for each geometry table are then combined with UNION ALL. The resulting TSV is then being gzipped. The hierarchy for each feature is extracted with the following custom type and function:

```
CREATE TYPE parentInfo AS (
    state           TEXT,
    county          TEXT,
    city                TEXT,
    displayName         TEXT
);

CREATE OR REPLACE FUNCTION getParentInfo(name_value TEXT, id_value BIGINT,
from_rank INTEGER, delimiter character varying(2)) RETURNS parentInfo AS $$
DECLARE
  retVal parentInfo;
  current_rank INTEGER;
  current_id BIGINT;
  currentName TEXT;
BEGIN
  current_rank := from_rank;
  retVal.displayName := name_value;
  current_id := id_value;

  IF current_rank = 16 THEN
    retVal.city := retVal.displayName;
  ELSE
    retVal.city := '';
  END IF;
  IF current_rank = 12 THEN
    retVal.county := retVal.displayName;
  ELSE
```

```
    retVal.county := '';
  END IF;
  IF current_rank = 8 THEN
    retVal.state := retVal.displayName;
  ELSE
    retVal.state := '';
  END IF;

  --RAISE NOTICE 'finding parent for % with rank %', name_value, from_rank;

  WHILE current_rank >= 8 LOOP
    SELECT getLanguageName(name, name_fr, name_en, name_de, name_es,
name_ru, name_zh), rank_search, parent_id FROM osm_polygon  WHERE id =
current_id INTO currentName, current_rank, current_id;
    IF currentName IS NOT NULL THEN
      retVal.displayName := retVal.displayName || delimiter || ' ' ||
currentName;
    END IF;

    IF current_rank = 16 THEN
      retVal.city := currentName;
    END IF;
    IF current_rank = 12 THEN
      retVal.county := currentName;
    END IF;
    IF current_rank = 8 THEN
      retVal.state := currentName;
    END IF;
  END LOOP;
RETURN retVal;
END;
$$ LANGUAGE plpgsql;
```

LISTING 4-9

First, it checks if the feature itself has a rank of 16,12 or 8 (city, county, state). Then it determines the name of the parent, appends it to the display_name and checks if the parent itself is a city, county or state and so on. The *parent_ids* of the countries are always NULL and therefore the loop always terminates.

4.6 PostgreSQL Performance

As already mentioned the tables created by imposm3 are not being updated with the ranking & partitioning but rather copied to new tables since this is the faster way [8]. Of course the indices then have to be recreated.

Additional VACUUM ANALYZE commands are applied to enhance the efficiency of the script executions.

In addition, CLUSTER commands on the geometry indices help to speed up geometry queries further by reordering the table accordingly.

4.6.1 TUNING

For calculating bigger metro extracts or even planet files, a blazing fast SSD-based server has been provided by HSR. With the help of PgTune calculator [9], it has been tried to find optimal settings for Postgres for the task at hand, but eventually has been proven a time consuming task with a lot of trial and error. The following settings have been applied (machine with 50GB of RAM):

```
max_connections = 20
shared_buffers = 12800MB
effective_cache_size = 38400MB
work_mem = 320MB
maintenance_work_mem = 2GB
min_wal_size = 4GB
max_wal_size = 8GB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 500
```

LISTING 4-10

4.6.2 INDICES

The following indices have been applied to speed up the queries:

```
--create indexes
CREATE INDEX IF NOT EXISTS idx_osm_polgyon_geom ON osm_polygon USING gist
(geometry);
CREATE INDEX IF NOT EXISTS idx_osm_point_geom ON osm_point USING gist
(geometry);
CREATE INDEX IF NOT EXISTS idx_osm_linestring_geom ON osm_linestring USING
gist (geometry);

CREATE INDEX IF NOT EXISTS idx_osm_polygon_partition_rank ON osm_polygon
(partition,rank_search);
CREATE INDEX IF NOT EXISTS idx_osm_polygon_id ON osm_polygon (id);

CREATE INDEX IF NOT EXISTS idx_osm_point_osm_id ON osm_point (osm_id);

CREATE INDEX IF NOT EXISTS idx_osm_linestring_merged_false ON
osm_linestring (merged) WHERE merged IS FALSE;
```

LISTING 4-11

Most noteworthy is the creation of geometry GIST indices for the geometry tables. This speeds up spatial queries tremendously.

4.7 FUNCTIONS

At this point, additional important functions are described which are used for importing or exporting the data.

4.7.1 Finding Parent of Street segments

```
CREATE OR REPLACE FUNCTION findRoadsWithinGeometry(id_value
BIGINT,partition_value INT, geometry_value GEOMETRY) RETURNS VOID AS $$
BEGIN
      UPDATE osm_linestring SET parent_id = id_value WHERE parent_id IS
NULL AND ST_Contains(geometry_value,geometry);
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION determineRoadHierarchyForEachCountry() RETURNS
void AS $$
DECLARE
  retVal BIGINT;
BEGIN
  FOR current_partition  IN 1..255 LOOP
    FOR current_rank  IN REVERSE 22..4 LOOP
       PERFORM findRoadsWithinGeometry(id, current_partition, geometry)
FROM osm_polygon WHERE partition = current_partition AND rank_search =
current_rank;
    END LOOP;
  END LOOP;
END;
$$ LANGUAGE plpgsql;
```

LISTING 4-12

For every partition (country), all street segments that are contained in features having a rank of 22 or lower are determined and updated accordingly. 22 (neighborhood, residential) is the highest rank of features that can contain street segments. This way it is ensured, that the parent has the highest rank possible when a feature is contained in two parent features with different ranks.

4.7.2 Determining Partition

As already described in chapter 4.3.2 the pre-initialized table *country_osm_grid* is used to determine the partition of a feature. However, as there are quite some features that could not be classified, a different method has been developed. The key is to work with the now imported countries (having a rank of 4).

```
CREATE OR REPLACE FUNCTION determinePartitionFromImportedData(geom
geometry)
RETURNS INTEGER AS $$
DECLARE
  result INTEGER;
BEGIN
  SELECT partition, calculated_country_code from osm_polygon where
ST_Within(ST_PointOnSurface(geom), geometry) AND rank_search = 4 AND NOT
partition = 0 INTO result;
    RETURN result;
END;
$$ LANGUAGE plpgsql;
```

LISTING 4-13

4.7.3 *Language Precedence*

Because the names are imported in seven different languages, there needs to be a unique way of weighing which language is more relevant in the exported data. This happens in the following function with the precedence [English -> native name -> French -> German -> Spanish -> Russian -> Chinese]:

```
CREATE OR REPLACE FUNCTION getLanguageName(default_lang TEXT, fr TEXT, en
TEXT, de TEXT, es TEXT, ru TEXT, zh TEXT)
RETURNS TEXT AS $$
BEGIN
  RETURN CASE
    WHEN en NOT IN ('') THEN en
    WHEN default_lang NOT IN ('') THEN default_lang
    WHEN fr NOT IN ('') THEN fr
    WHEN de NOT IN ('') THEN de
    WHEN es NOT IN ('') THEN es
    WHEN ru NOT IN ('') THEN ru
    WHEN zh NOT IN ('') THEN zh
    ELSE ''
  END;
END;
$$ LANGUAGE plpgsql IMMUTABLE;
```

LISTING 4-14

Of course, this behavior can be interchanged.

4.7.4 *Alternative Names*

It is a requirement to have also the names in the export that weren't used in the name field in the export. This way a geocoder can index these fields as well and find for instance native names as well.

```
CREATE OR REPLACE FUNCTION getAlternativesNames(default_lang TEXT, fr TEXT,
en TEXT, de TEXT, es TEXT, ru TEXT, zh TEXT, name TEXT, delimiter character
varying)
RETURNS TEXT AS $$
DECLARE
  alternativeNames TEXT[];
BEGIN
  alternativeNames := array_distinct(ARRAY[default_lang, en, fr, de, es,
ru, zh]);
  alternativeNames := array_remove(alternativeNames, '');
  alternativeNames := array_remove(alternativeNames, name);
RETURN array_to_string(alternativeNames,delimiter);
END;
$$ LANGUAGE plpgsql IMMUTABLE;
```

LISTING 4-15

The name parameter is the value used in the name field, so it is excluded as well as empty name fields. Also, it is ensured that the names in the result are distinct.

4.7.5 *Country Names*

Country names are exported from the pre-initialized helper table *country_name*. This happens with the same precedence as used in described in chapter 4.7.3.

```
CREATE OR REPLACE FUNCTION countryName(partition_id int) returns TEXT as $$
  SELECT COALESCE(name -> 'name:en',name -> 'name',name -> 'name:fr',name -
> 'name:de',name -> 'name:es',name -> 'name:ru',name -> 'name:zh') FROM
country_name WHERE partition = partition_id;
$$ language 'sql';
```

LISTING 4-16

5 RESULTS

5.1 PRODUCT RESULTS

First of all, a data set containing planet data could be provided. A secondary target, importing additional house numbers for each street, could not be met in scope of this project.

The workflow could be setup in such a way, that it can easily be setup anywhere (via Docker). The simplicity of the installation and, most of all, the clear arrangement of code is a big advantage over other products like Nominatim. After all, the processing run times are way faster than the latter, which takes up to several days for a global data set.

Finally, the quality of the data proves to be quite satisfactory as seen in the real-life example in 0.

5.1.1 DATA

The TSV file from the planet export includes 21'055'840 entries. The current data export can be downloaded at https://osmnames.org.

If one is only interested in a specific country, he or she can download the file and easily extract the information with the following command:

```
awk -F $'\t' 'BEGIN {OFS = FS}{if (NR!=1) {  if ($16 =="[country_code]")  {
print}    } else {print}}' planet-latest.tsv > countryExtract.tsv
```

LISTING 5-1

Where [country_code] needs to be replaced with the ISO-3166 2-letter country code.

In terms of quality control, only manual checks have been executed. Since the data export now also contains a unique *osm_id*, it is possible to match the results with the results from a Nominatim search service. This automated data control, however, has not been developed in scope of this project.

5.1.2 Use Case: Search on Map

The resulting data export has been successfully integrated by Klokan Technologies with a SphinxSearch powered geocoder. The result includes osm2vectortiles-generated vector tiles and can be seen here: https://osmnames.klokantech.com/

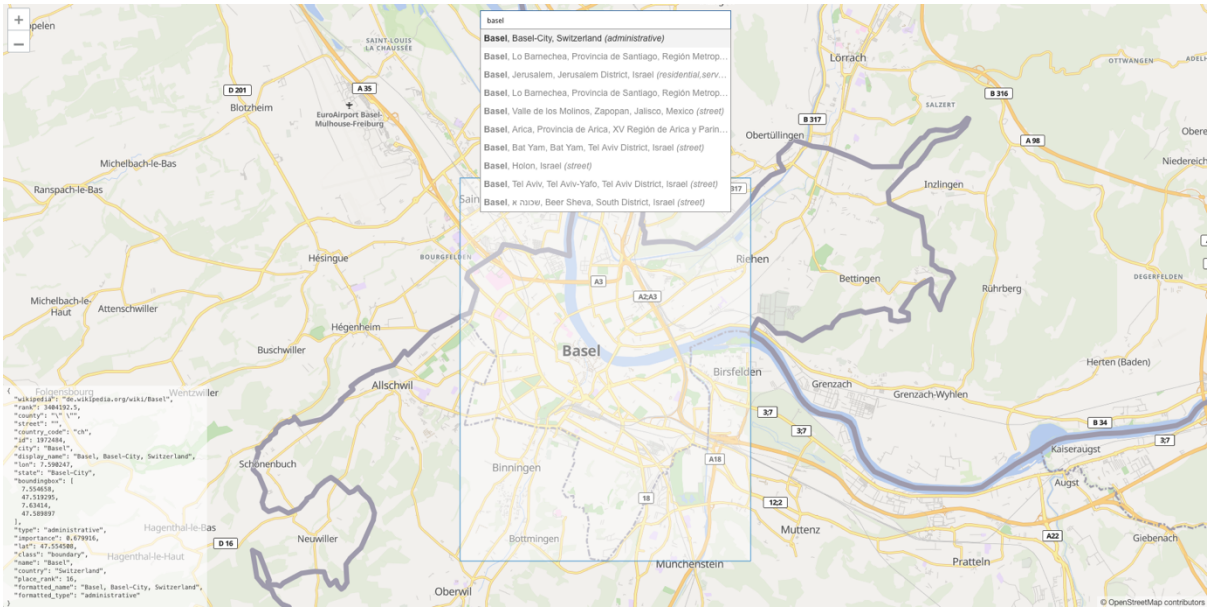

FIGURE 5-1 OSMNAMES USED AS A GEOCODER

The implementation makes use of the calculated importance in order to sort the results. The bounding box is also visualized. The meta-data can be seen in the lower-left corner.

5.1.3 Performance Measurements

The run times for a planet dump file on the test server provided are the following for each step:

Task	Time
Import Wikipedia	00:19 h
Import OSM data	03:01 h
Delete unusable entries	00:12 h
Ranking & Partitioning	01:59 h
Determine Linked Places	00:10 h
Create Hierarchy	11:40 h
Merging Corresponding Street Segments	01:46 h
Export to TSV	07:22 h
TOTAL	26:29 h

TABLE 5-1 PERFORMANCE TIMES

These results meet the non-functional requirement NFR_01 stated in chapter 2.4.

Surely, these times can still be reduced by tweaking the database as well as the Docker settings

5.2 Outlook

The current version 1.1 is quite satisfactory and could already be implemented in a real life application as seen above. However, there are still a few improvements that can be developed.

First and foremost, the database and Docker settings need to be tweaked, so the processing times go down even more. An enhancement that can benefit from these improvements is differential updates. With diff updates enabled one can think of automated regular updates of the name database. Such updates take a really short time compared to a full planet import and consequently improve the product overall.

Secondly, it would be great to have house numbers in the database as well. This extension however brings a further challenge when it comes to the export. Further work has to be done in designing the export format, unless we want to have an entry in the export file for every house number, resulting in a humongous file size.

5.3 Reflection

Working with Docker was a new experience. However, it showed me different way of abstracting, packaging and distributing applications which is quite useful. There are many scenarios to think of where Docker can be applied.

Furthermore, the project gave me insight into OpenStreetMap and the meaning of quite a few tags. Code analysis of Nominatim was quite cumbersome, since there does not exist any documentation whatsoever. Still, I was able to extract the information necessary to build a product that meets the requirements.

Working with PostGIS was really interesting as one works with real world data which is easier to grasp. However, the function names are not always self-evident and need to be properly read up on in the documentation and tested with small test data.
Speaking of test data: It proved wise to work with a small dataset in the beginning and expand it over and over again (CH->DACH->Europe->Planet). However, bigger data set can also mean different kinds of data and therefore a possible need for data integrity

checks. For instance, checking for geometry equality as there are such duplicates in Great Britain. Also, in the planet dataset there exist non-valid geometries which are needed to be intercepted and treated accordingly.

6 Acknowledgment

I express my sense of gratitude to my supervisor Prof. Stefan Keller. Thanks to his strong support I was able to work on this interesting topic. His invaluable insight into the matter showed me some of the quirks in working with geo data.

I would also like to take this opportunity to thank Dr. Petr Pridal of Klokan Technologies GmbH. With his support and regular meetings, I was able to keep this project on track. It has been a pleasure working with him.

I am very much thankful to Marcel Huber for giving me access to test infrastructure at HSR where I could run the heavy workload. His knowledge with Docker tuning helped me in optimizing database parameters.

I acknowledge with thanks the help of Lukas Martinelli and Manuel Roth for setting me up with a Docker template to get started with. At this point, I also want to thank Lukas Toggenburger for introducing me to the topic of OSM addresses.

And last, but not least, I am extremely thankful to Fränzi Zahner for her constant encouragement throughout this year.

7 Glossary

Geo Name	Name of areas, regions, localities, cities, suburbs, towns or settlements, or any geographical or topographical feature of public or historical interest. A feature can have many names in different languages [10].
Gazetteer	Geographical dictionary or directory used in conjunction with a map or atlas. They associate geo names with their associative features.
Geocoder	A (web) service that delivers a spatial representation in numerical coordinates for the given geo name as input.
Address	An address is a collection of information describing the location of a building, apartment or other structure using street names as references along with identifiers such as house numbers and zip codes.
Docker	Docker is a software containerization platform, which enables developers to easily build, ship and run applications.
PBF	Protocollbuffer Binary Format, file format for OpenStreetMap data which allows faster processing than XML
TSV	Tab-separated value, file format with a tab delimiter between values
YAML	Alternative format to JSON and XML

8 BIBLIOGRAPHY

[1] Docker. (2016, Aug.) Best practices for writing Dockerfiles. [Online]. https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/

[2] Docker. (2016, Aug.) Docker Overview. [Online]. https://docs.docker.com/engine/understanding-docker/

[3] GisGraphy. (2016, Aug.) About the street names CSV / TSV format. [Online]. https://download.gisgraphy.com/format.txt

[4] OpenSteetMap. (2016, Aug.) Planet.osm. [Online]. https://wiki.openstreetmap.org/wiki/Planet.osm

[5] OpenStreetMap. (2016, Aug.) Elements. [Online]. https://wiki.openstreetmap.org/wiki/Elements

[6] OmniScale Imposm 3.0.0a Documentation. (2016, Aug.) Data Mapping. [Online]. https://imposm.org/docs/imposm3/latest/mapping.html

[7] Lukas Martinelli. (2016, Aug.) GitHub pgclimb. [Online]. https://github.com/lukasmartinelli/pgclimb

[8] Nuno Teixeira. (2015, May) How to update large tables in PostgreSQL. [Online]. http://blog.codacy.com/2015/05/14/how-to-update-large-tables-in-postgresql/

[9] Alexey Vasiliev. (2016, Aug.) PGTune - Configuration calculator for PostgreSQL. [Online]. http://pgtune.leopard.in.ua/

[10] European Commission. (2016, Aug.) INSPIRE Registry. [Online]. http://inspire.ec.europa.eu/theme/gn

9 FIGURES

10 TABLES

Appendix A Installation

A.1 Installing OSMNames

With the following set of commands one can easily setup OSMNames in a matter of minutes. Prerequisites are a working installation of Docker https://www.docker.com/ along with Docker compose.

1. Checkout source from GitHub

   ```
   git clone https://github.com/geometalab/OSMNames.git
   ```

2. Either download specific PBF manually or the planet dump with the following Docker task

   ```
   docker-compose run download-pbf
   wget --directory-prefix=./data
   http://download.geofabrik.de/europe/switzerland-latest.osm.pbf
   ```

3. Setup the database

   ```
   docker-compose up -d postgres
   ```

4. Import wikipedia data

   ```
   docker-compose run import-wikipedia
   ```

5. Create the schema

   ```
   docker-compose run schema
   ```

6. Import the OSM data from the PBF file

   ```
   docker-compose run import-osm
   ```

7. Export the data to a TSV file

   ```
   docker-compose run export-osmnames
   ```

That's it. The export file can be found in the data folder.

Appendix B Documentation

B.1 Reviewed Products

One task of the project was reviewing some of the products on the market that have similar functionalities. They are presented at this point.

B.1.1 Nominatim

Nominatim is a geocoder and reverse geocoder used to search OSM data on openstreetmap.org. It requires quite a bunch of prerequisites such as a GCC compiler, a PostgreSQL installation with PostGIS, Proj4j, GEOS, PHP5, PHP-pgsql, PEAR::DB, wget, boost and for importing OSM data osmosis. It can be downloaded from www.nominatim.org .

It is easier to run it in a Docker container which was used in scope of this project:

https://hub.docker.com/r/nicopace/nominatim-docker/

Since the planet dump takes several days to import, only a small extract was imported (Switzerland). The goal was to understand the following activities of Nominatim:

- Ranking
- Hierarchy
- Importance

The ranking is basically a mapping of types to values between 1 and 30 and the exact matching was reproduced and described in 4.3.2.

The hierarchy is done on query time. In table *placex* every feature has a *parent_id* referencing another feature in the same table. The *display_name* is then constructed by following the path upwards. A similar concept has been followed with OSMNames.

The importance is determined with the help of the *wikipedia_article* table. Whenever a query has been run the matching importance is updated in the according *placex* entry. In order to get the same results, the wikipedia information has been imported in OSMNames and in case there is no match the same calculations based on the ranks are used.

B.1.2 Mapzen Pelias

Mapzen's Pelias is an open-source geocoder built on top of ElasticSearch. It also supports multiple data sources such as OSM, OpenAdresses, Geonames and Mapzen's own Who's on First.

In order to test the product without having to install all the required software components, the following vagrant development environment has been used:

https://github.com/pelias/vagrant

After getting it work, it was not possible to do a proper analysis of how the processing is done in detail, due to other more important requirements in the project.

B.1.3 Libpostal

As already mentioned, there is a problem when it comes to formatting postal addresses. Each country has its own set of formatting rules and in order to have a database of addresses one needs to bring these different formats to a common format and vice versa. This is where libpostal comes in handy. It is a c library with bindings for Python, Ruby, Go, Java, PHP and NodeJS and does address normalization and parsing by using statistical Natural Language Processing (NLP).

For testing purposes the Perl extension has been installed.

Example: Input string: `Häringstrasse 2, Altstadt, 8001 Zurich, Switzerland`

Parsing separates address string into components and uses address templates from OpenCage and uses them appropriately (https://github.com/OpenCageData/address-formatting).

Output: `[('haeringstrasse', 'road'), ('2', 'house_number'), ('altstadt', 'suburb'), ('8001', 'postcode'), ('zurich', 'city'), ('switzerland', 'country')]`

The following data is considered to the training of the NLP:

- OSM: training examples of parsed addresses and language classifications
- Geonames: for place names and postal code gazetteer
- Quattroshapes/Zetashapes: for administrative/local boundaries

Finally, libpostal also does address normalization, specifically:

- language classification

- abbreviations extended, e.g. Str->Strasse, Rd->Road, W -> West
- numeric expression translated to numbers
- transliteration, e.g. Cyrillic -> Latin

There also exists a Postgres extension which could be quite an interesting extension to the current OSMNames.

The current version can be found here: https://github.com/openvenues/libpostal

B.2 IMPOSM3 YAML MAPPING

The complete YAML mapping used in imposm3 is the following:

```
tables:
  linestring_tmp:
    type: linestring
    fields:
    - name: osm_id
      type: id
    - name: geometry
      type: geometry
    - name: type
      type: mapping_value
    - key: name
      name: name
      type: string
    - name: name_fr
      key: name:fr
      type: string
    - name: name_en
      key: name:en
      type: string
    - name: name_de
      key: name:de
      type: string
    - name: name_es
      key: name:es
      type: string
    - name: name_ru
      key: name:ru
      type: string
    - name: name_zh
      key: name:zh
      type: string
    - key: wikipedia
      name: wikipedia
      type: string
    - key: wikidata
      name: wikidata
      type: string
    - key: admin_level
      name: admin_level
      type: integer
    mapping:
      highway:
```

```
      - motorway
      - motorway_link
      - trunk
      - trunk_link
      - primary
      - primary_link
      - secondary
      - secondary_link
      - tertiary
      - tertiary_link
      - unclassified
      - residential
      - road
      - living_street
      - raceway
      - construction
      - track
      - service
      - path
      - cycleway
      - steps
      - bridleway
      - footway
      - corridor
      - crossing
  polygon_tmp:
    type: polygon
    fields:
    - name: osm_id
      type: id
    - name: geometry
      type: geometry
    - name: type
      type: mapping_value
    - name: country_code
      type: string
      key: ISO3166-1:alpha2
    - key: name
      name: name
      type: string
    - name: name_fr
      key: name:fr
      type: string
    - name: name_en
      key: name:en
      type: string
    - name: name_de
      key: name:de
      type: string
    - name: name_es
      key: name:es
      type: string
    - name: name_ru
      key: name:ru
      type: string
    - name: name_zh
      key: name:zh
      type: string
    - key: wikipedia
      name: wikipedia
      type: string
    - key: wikidata
      name: wikidata
```

```
    type: string
  - key: admin_level
    name: admin_level
    type: integer
  mapping:
    place:
    - city
    - borough
    - suburb
    - quarter
    - neighbourhood
    - town
    - village
    - hamlet
    landuse:
    - residential
    boundary:
    - administrative
point_tmp:
  type: point
  fields:
  - name: osm_id
    type: id
  - name: geometry
    type: geometry
  - name: type
    type: mapping_value
  - key: name
    name: name
    type: string
  - name: name_fr
    key: name:fr
    type: string
  - name: name_en
    key: name:en
    type: string
  - name: name_de
    key: name:de
    type: string
  - name: name_es
    key: name:es
    type: string
  - name: name_ru
    key: name:ru
    type: string
  - name: name_zh
    key: name:zh
    type: string
  - key: wikipedia
    name: wikipedia
    type: string
  - key: wikidata
    name: wikidata
    type: string
  - key: admin_level
    name: admin_level
    type: integer
  mapping:
    place:
    - city
    - borough
    - suburb
    - quarter
```

```
      - neighbourhood
      - town
      - village
      - hamlet
  relation:
    type: relation_member
    columns:
    - name: osm_id
      type: id
    - name: member
      type: member_id
    - name: role
      type: member_role
    - name: type
      type: member_type
    - name: geometry
      type: geometry
    - name: name
      key: name
      type: string
      from_member: true
    mapping:
      landuse:
      - residential
      boundary:
      - administrative
```

B.3 Source Documentation

The additional documentation, which is greatly based on this documentation and specially provided for Klokan Technologies GmbH, is being exported as a PDF an appended to this document.

Appendix C Project Management

C.1 Organization

This work has been done as a one-man operation with the help of Dr. Petr Pridal of Klokan Technologies GmbH and supervised by Prof. Stefan Keller. The source code is open source on GitHub (https://github.com/geometalab/OSMNames). Additionally, the source code is provided on a separate disc.

C.2 Planning & Coordination

Coordination meetings have been held every two weeks either via Skype or Google Hangouts. An agile software development process with adaptive planning as well as continuous improvement was followed.

After an initial phase of analysis, a 2-week release cycle was followed. The milestones for each release were the following:

Milestone	Date	Release	Description
M_01	March 14		Sync Meeting
M_02	April 25		Hierarchy/Relations Analysis Nominatim
M_03	May 9	v0.2	Towns in extracts
M_04	May 25	v0.3	Place, State, Country
M_05	June 16	v0.4	Structure finished, wikipedia import
M_06	July 9	v0.6	skipped
M_07	July 20	V1.0	World extract done
M_08	August 1	V1.1	Multiple improvements, data refinement

C.3 Workflow

The scripts were coded with Sublime text editor (https://www.sublimetext.com/).

GitHub was used as a code revision control and source code management tool. With the help of issues problems and requirements were documented and discussed.

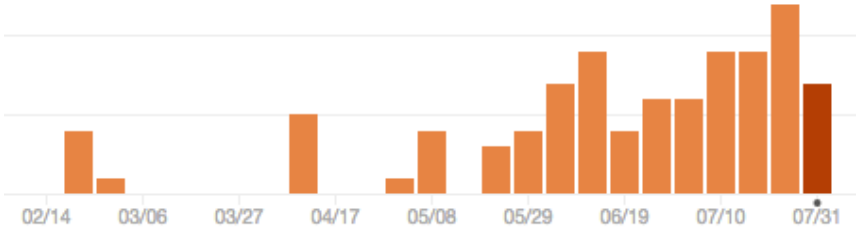

FIGURE C.1 COMMITS ON GITHUB

C.4 Target-performance Comparison & Monitoring

Each week all the tasks were analyzed in a target-performance comparison sheet. The pre-defined milestones were always met.

Projekt PA2 Semesterplan

Woche	Datum	Tasks	Probleme / Notizen	Milestones	Zeit IS	Zeit SOLL
9	22/02/16	Kickoff Meeting, Definition der Problemstellung			5	17
10	29/02/16				4	17
11	07/03/16				34	17
12	14/03/16			Nominatim installiert // Analyse Code wie Hierarchie berechnet wird	17	17
13	21/03/16				0	17
14	28/03/16	FERIEN			0	0
15	04/04/16	FERIEN			0	0
16	11/04/16				2	17
17	18/04/16				8	17
18	25/04/16				5	17
19	02/05/16				24	17
20	09/05/16			V 0.2 town extracts	9	17
21	16/05/16				0	17
22	23/05/16			V 0.3 place, state, country extracts	28	17
23	30/05/16				27	17
24	06/06/16			V 0.4 structure finished, wikipedida import	34	17
25	13/06/16			V 0.5 either updates from diff OR calculation of house numbers	15	17
26	20/06/16				5	17
27	27/06/16				19	17
28	04/07/16			V 0.6	20	17
29	11/07/16				21	17
30	18/07/16			V 1.0 World extract done	10	17
31	25/07/16			Schriftlicher Bericht anfangen	42	17
32	01/08/16			V 1.1 deployment	8	3
33	08/08/16				0	0
34	15/08/16				38	0
35	22/08/16				30	0
36	29/08/16			Abgabe Arbeit PA_2		0
					405	360

FIGURE C.2 TARGET-PERFORMANCE COMPARISON AND MONITORING SHEET