OSMNames Documentation

Release 1.1

Klokan Technologies GmbH

Aug 27, 2016

CONTENTS

Contents:

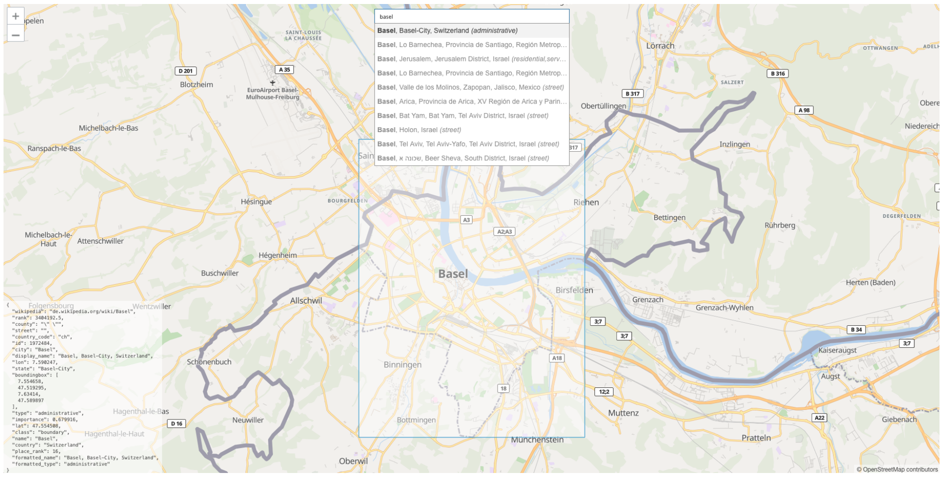
CHAPTER ONE

OSMNAMES INTRODUCTION

OSMNames is an open source tool that allows creating geographical gazetter data out of OpenStreetMap OSM files.

There is a need for a data set consisting of street names of the world. Such gazetteer data however is either not available for every country (openaddresses.io) or is not in a compact format. Furthermore, if such data is found, it is often not for free. A global data set can be downloaded at https://osmnames.org.

A current implementation on how the data looks like in a geocoder can be seen at https://osmnames.klokantech.com



basel

Basel, Basel-City, Switzerland *(administrative)*
Basel, Lo Barnechea, Provincia de Santiago, Región Metrop…
Basel, Jerusalem, Jerusalem District, Israel *(residential,serv…*
Basel, Lo Barnechea, Provincia de Santiago, Región Metrop…
Basel, Valle de los Molinos, Zapopan, Jalisco, Mexico *(street)*
Basel, Arica, Provincia de Arica, XV Región de Arica y Parin…
Basel, Bat Yam, Bat Yam, Tel Aviv District, Israel *(street)*
Basel, Holon, Israel *(street)*
Basel, Tel Aviv, Tel Aviv-Yafo, Tel Aviv District, Israel *(street)*
Basel, שכונה א, Beer Sheva, South District, Israel *(street)*

```
{
  "wikipedia": "de.wikipedia.org/wiki/Basel",
  "rank": 3404192.5,
  "county": "\" \"",
  "street": "",
  "country_code": "ch",
  "id": 1972484,
  "city": "Basel",
  "display_name": "Basel, Basel-City, Switzerland",
  "lon": 7.590247,
  "state": "Basel-City",
  "boundingbox": [
    7.554658,
    47.519295,
    7.63414,
    47.589897
  ],
  "type": "administrative",
  "importance": 0.679916,
  "lat": 47.554508,
  "class": "boundary",
  "name": "Basel",
  "country": "Switzerland",
  "place_rank": 16,
  "formatted_name": "Basel, Basel-City, Switzerland",
  "formatted_type": "administrative"
}
```

1.1 What can I do with OSMNames?

With OSMNames, you can create your own geocoder data set based on OpenStreetMap. It currently includes all addresses available without house numbers and zip codes. For each feature a hierarchy as well as an Wikipedia-based importance is calculated.

CHAPTER

TWO

COMPONENTS

OSMNames consists of the following components:

2.1 Docker

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.

2.2 Imposm3

Imposm3 by Omniscale is a data importer for OpenStreetMap data. It reads PBF files and writes the data into the PostgreSQL database. In OSMNames it 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.

2.3 PostgreSQL

PostgreSQL is the open source database powering OMSNames.

OSMNames uses PostgreSQL for the following tasks:

- Storing OSM data read from PBF file.
- OSM data processing
- data export to TSV file

At this moment OSMNames runs PostgreSQL 9.5.x version.

2.4 PostGIS

PostGIS is the extension which adds spatial capabilities to PostgreSQL. It allows working with geospatial types or running geospatial functions in PostgreSQL.

At this moment OSMNames runs PostGIS 2.2 version.

2.5 pgclimb

pgclimb is used by OSMNames for extracting the data from the PostgreSQL database into a TSV file.

The tool supports many formats such as JSON, CSV, XLSX, XML which can be enabled in OSMNames as well.

> The tool can be found on GitHub https://github.com/lukasmartinelli/pgclimb

CHAPTER

THREE

INSTALLATION

Note: In order to increase the speed of the processing, an SSD disk is recommended. It is also recommended to tweak the database settings to match the specficiations of your system.

A 100 GB disk should be sufficient to run the data.

3.1 System requirements

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.

3.2 Involved steps

Installations assume you use UTF8. You can set the locale by doing this:

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.

3.3 Extracting countries

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
```

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

CHAPTER

FOUR

ARCHITECTURE

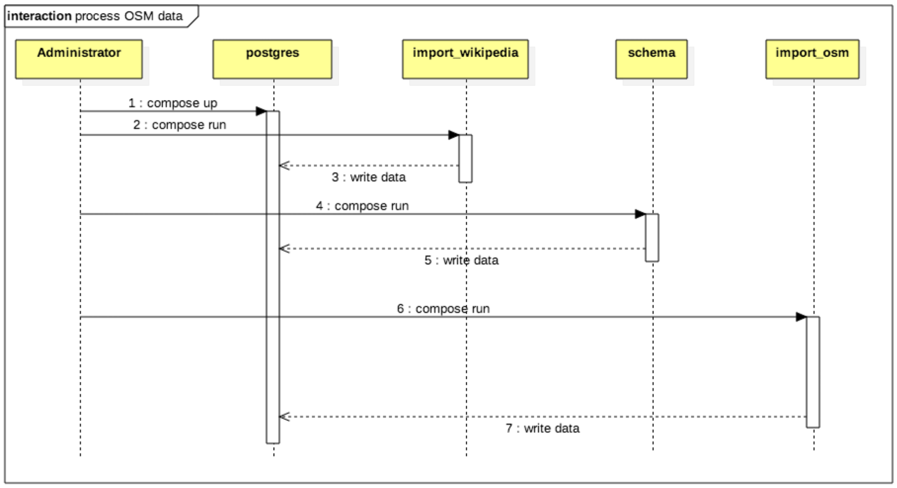
This describes the different architectural aspects of OSMNames:

4.1 Process of OSMNames

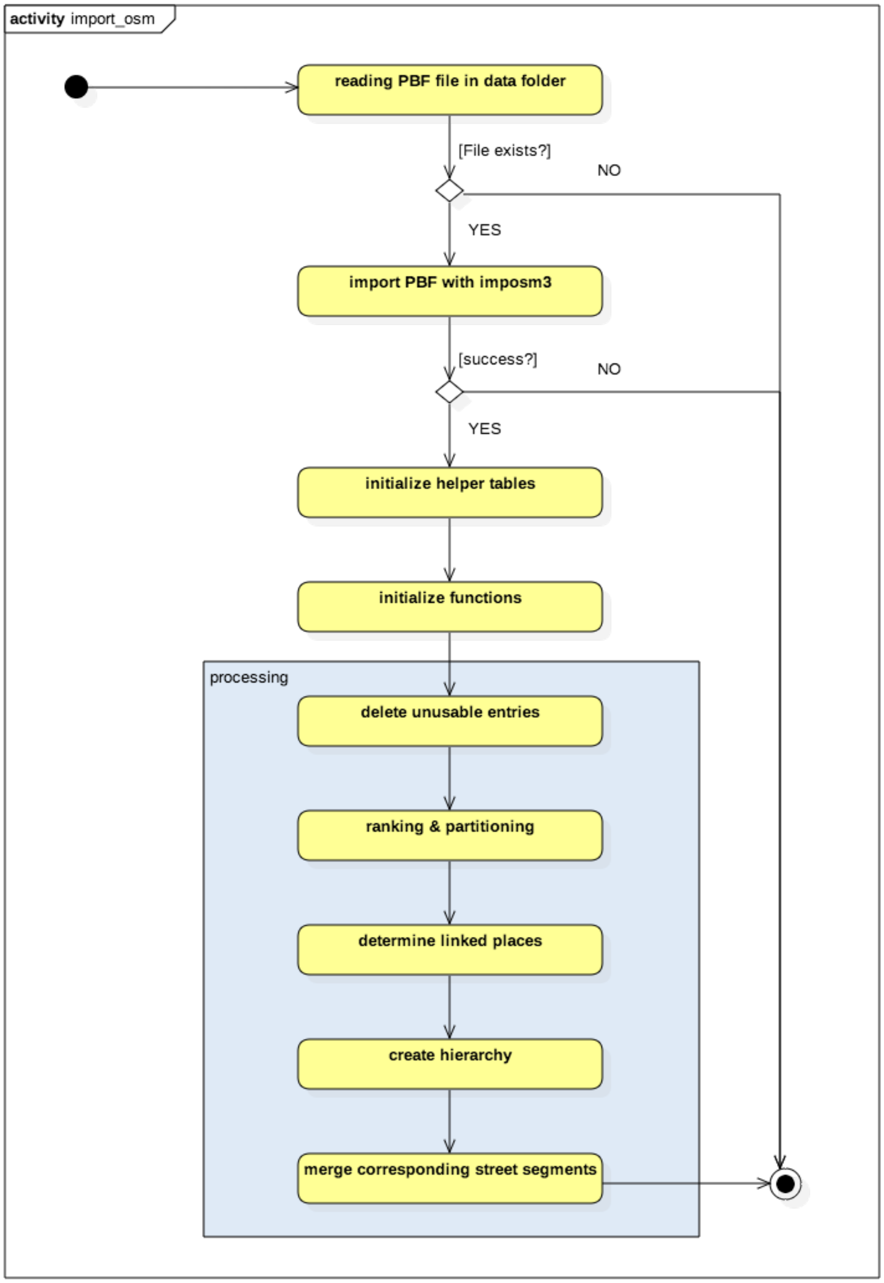
The following descibes the steps involved in the processing of data in OSMNames.

4.1.1 Data Import

In order to import OSM data the administrator downloads either downloads the pbf file or uses the corresponding docker compose task. The process looks like this:

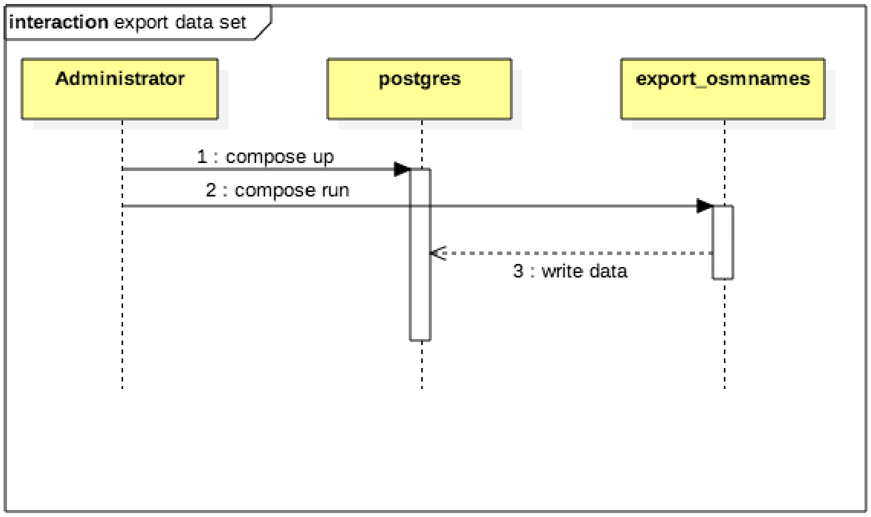


The essential important import process looks like this:



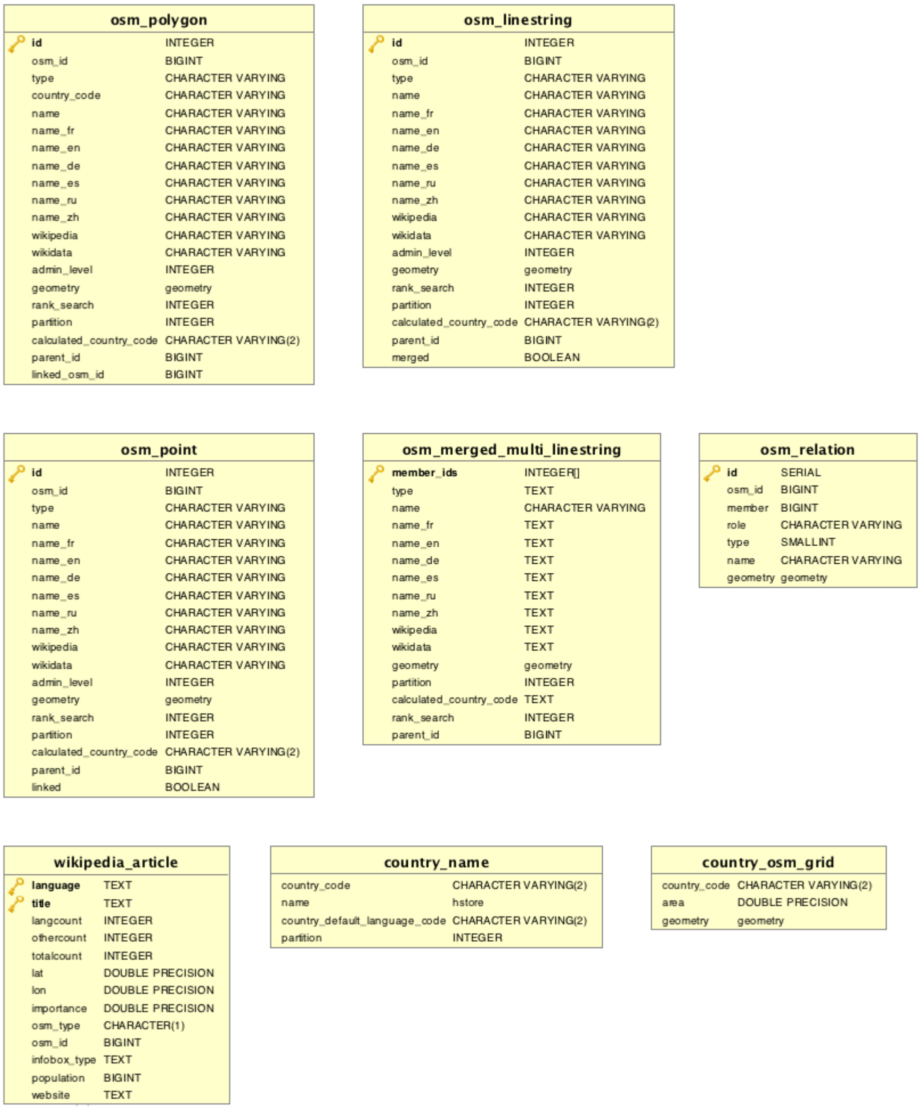
4.1.2 Data Export

The data export looks like this:



4.2 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:



osm_polygon	
id	INTEGER
osm_id	BIGINT
type	CHARACTER VARYING
country_code	CHARACTER VARYING
name	CHARACTER VARYING
name_fr	CHARACTER VARYING
name_en	CHARACTER VARYING
name_de	CHARACTER VARYING
name_es	CHARACTER VARYING
name_ru	CHARACTER VARYING
name_zh	CHARACTER VARYING
wikipedia	CHARACTER VARYING
wikidata	CHARACTER VARYING
admin_level	INTEGER
geometry	geometry
rank_search	INTEGER
partition	INTEGER
calculated_country_code	CHARACTER VARYING(2)
parent_id	BIGINT
linked_osm_id	BIGINT

osm_linestring	
id	INTEGER
osm_id	BIGINT
type	CHARACTER VARYING
name	CHARACTER VARYING
name_fr	CHARACTER VARYING
name_en	CHARACTER VARYING
name_de	CHARACTER VARYING
name_es	CHARACTER VARYING
name_ru	CHARACTER VARYING
name_zh	CHARACTER VARYING
wikipedia	CHARACTER VARYING
wikidata	CHARACTER VARYING
admin_level	INTEGER
geometry	geometry
rank_search	INTEGER
partition	INTEGER
calculated_country_code	CHARACTER VARYING(2)
parent_id	BIGINT
merged	BOOLEAN

osm_point	
id	INTEGER
osm_id	BIGINT
type	CHARACTER VARYING
name	CHARACTER VARYING
name_fr	CHARACTER VARYING
name_en	CHARACTER VARYING
name_de	CHARACTER VARYING
name_es	CHARACTER VARYING
name_ru	CHARACTER VARYING
name_zh	CHARACTER VARYING
wikipedia	CHARACTER VARYING
wikidata	CHARACTER VARYING
admin_level	INTEGER
geometry	geometry
rank_search	INTEGER
partition	INTEGER
calculated_country_code	CHARACTER VARYING(2)
parent_id	BIGINT
linked	BOOLEAN

osm_merged_multi_linestring	
member_ids	INTEGER[]
type	TEXT
name	CHARACTER VARYING
name_fr	TEXT
name_en	TEXT
name_de	TEXT
name_es	TEXT
name_ru	TEXT
name_zh	TEXT
wikipedia	TEXT
wikidata	TEXT
geometry	geometry
partition	INTEGER
calculated_country_code	TEXT
rank_search	INTEGER
parent_id	BIGINT

osm_relation	
id	SERIAL
osm_id	BIGINT
member	BIGINT
role	CHARACTER VARYING
type	SMALLINT
name	CHARACTER VARYING
geometry	geometry

wikipedia_article	
language	TEXT
title	TEXT
langcount	INTEGER
othercount	INTEGER
totalcount	INTEGER
lat	DOUBLE PRECISION
lon	DOUBLE PRECISION
importance	DOUBLE PRECISION
osm_type	CHARACTER(1)
osm_id	BIGINT
infobox_type	TEXT
population	BIGINT
website	TEXT

country_name	
country_code	CHARACTER VARYING(2)
name	hstore
country_default_language_code	CHARACTER VARYING(2)
partition	INTEGER

country_osm_grid	
country_code	CHARACTER VARYING(2)
area	DOUBLE PRECISION
geometry	geometry

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 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.

4.3 Export Data Format

The requirement to the export data format is being simple to use. The decision led to using an UTF-8 encoded TSV like with Geonames, where the first line contains the column names. Compared to a CSV name are now allowed to have usual delimiters such as commas or semicolons. Best practices from GISGraphy have been adapted. 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

CHAPTER

FIVE

IMPLEMENTATION

This describes the different implementational aspects of OSMNames:

5.1 Imposm Mapping

Currently the features matching following key/values are imported by imposm3:

- **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

The following fiels are then incorporated into OSMNames:

Column name	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

5.2 Processing

After importing the OSM data with imposm3 the real processing begins. Each of the steps taken is described at this point

5.2.1 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.

```
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;
```

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

```
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)||'%';

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

5.2.2 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. 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
 );
```

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 $$
BEGIN
      RETURN CASE
              WHEN type IN ('administrative') THEN 2*(SELECT COALESCE(admin_level,15)␣
↪FROM osm_polygon_tmp o WHERE osm_id = osmID)
              WHEN type IN ('continent', 'sea') THEN 2
              WHEN type IN ('country') THEN 4
              WHEN type IN ('state') THEN 8
              WHEN type IN ('county') THEN 12
              WHEN type IN ('city') THEN 16
              WHEN type IN ('island') THEN 17
              WHEN type IN ('region') THEN 18 -- dropped from previous value of 10
              WHEN type IN ('town') THEN 18
              WHEN type IN ('village','hamlet','municipality','district',
↪'unincorporated_area','borough') THEN 19
              WHEN type IN ('suburb','croft','subdivision','isolated_dwelling','farm',
↪'locality','islet','mountain_pass') THEN 20
              WHEN type IN ('neighbourhood', 'residential') THEN 22
              WHEN type IN ('houses') THEN 28
              WHEN type IN ('house','building') THEN 30
              WHEN type IN ('quarter') THEN 30
      END;
END;
$$ 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;
```

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;
```

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;
```

5.2.3 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.

```
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;
```

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.

5.2.4 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;
```

With the reverse loop it is ensured to match only features with the same or a lower rank. Also, with 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.

Finding Parent of Street segments

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 if a features is contained in two parent features with different ranks.

```
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;
```

5.2.5 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;
```

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

5.3 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;
```

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.

5.3.1 Language Precedence

Because the names are imported in seven different languages, there needs to be a unique of of weighing which language is more relevant for 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;
```

Of course, this behavior can be interchanged.

5.3.2 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;
```

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.

5.3.3 Country Names

Country names are exported from the pre-initialized helper table country_name. This happens with the same language precedence as defined in *getLanguageName*.

```
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';
```

5.3.4 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 = log (totalcount) / log( max(totalcount))
```

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)
```

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

The function *getImportance* for calculating the importance is called during the export.

5.3.5 Type of relations

In order to tackle the problem of relations often being administrative although being linked to 'city' nodes the following function has been developed:

```
CREATE OR REPLACE FUNCTION getTypeForRelations(linked_osm_id BIGINT, type_value TEXT,␣
↪rank_search INTEGER) returns TEXT as $$
DECLARE
  retVal TEXT;
BEGIN
IF linked_osm_id IS NOT NULL AND type_value = 'administrative' AND (rank_search = 16␣
↪OR rank_search = 12) THEN
  SELECT type FROM osm_point WHERE osm_id = linked_osm_id INTO retVal;
  IF retVal = 'city' THEN
  RETURN retVal;
  ELSE
  RETURN type_value;
  END IF;
ELSE
  return type_value;
 END IF;
END;
$$ LANGUAGE plpgsql IMMUTABLE;
```

5.4 Indices

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

```
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;
```

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