Teil I.

osm2inter

1. osm2pgsql

1.1. Intro

Als Importierungskomponente kann osm2pgsql[1] sehr vielseitig eingesetzt werden. Innerhalb des Projektes löst osm2pgsql den OHDMConverter zum importieren von osm|osm.pbf|osm.bz2 in die intermediate Datenbank ab.

Den größten Vorteil von osm2pgsql bietet die Benutzung des „Flex Output“. Hierbei wird die Konvertierung mit einem lua Script angepasst.

1.2. osm2pgsql Flags

Für die fehlerfreie Importierung sind mehrere Flags notwendig. In der nachfolgenden Tabelle 1.1 sind diese vermerkt.

Tabelle 1.1.: flags

Flag	Beschreibung
-c	Spezifiziert die osm Datei
-d	Name der Datenbank
-U	Name des Datenbankbenutzers mit Rechten zur Erstellung (Standard: postgres)
-O	Spezifiziert den Output z.B.: flex, pgsql (Standard), gazetteer und null Für die Benutzung des lua Script basierten konvertierens, wird dieser Wert auf flex gesetzt
-x	Ermöglicht die Verwendung von user name, user id, changeset id, timestamp and version
-S	Dies gibt an, wie die Daten in die Datenbank importiert werden, ihr Format hängt von der Ausgabe ab in diesem Flag muss das lua Script angegeben werden

In Windows muss zusätzlich -W für eine Passworteingabe als Flag gesetzt werden

1.3. Lua Script

Für die Importierung in das bestehende intermediate Schema wird ein lua Script benötigt, welches die Importierung von osm Daten in die intermediate Datenbank spezifiziert.
Für eine bessere Erklärung möchte ich das lua Script in drei Abschnitte unterteilen:

1. Tabelleninizialisierung
2. Hilfsfunktionen und Variablen
3. Prozessfunktionen

Diese Beschreibung ist projektspezifisch, weitere Informationen zur Verwendung des lua Script können unter folgendem Link eingesehen werden:
https://osm2pgsql.org/doc/manual.html#the-flex-output

1.3.1. Tabelleninizialisierung

Im lua Script werden erforderliche Tabellen wie in Listing 1.1 angelegt. Dies wird benötigt um die Tabelle zu spezifizieren.

Listing 1.1: Initialisierung eine Tabelle für alle nodes

```
1  tables.nodes = osm2pgsql.define_table({
2    name = 'nodes',
3    ids = { type = 'node', id_column = 'osm_id' },
4    columns = {
5      { column = 'id', sql_type = 'bigserial', create_only = true },
6      { column = 'tstamp', sql_type = 'timestamp' },
7      { column = 'mapfeatures', type = 'hstore' },
8      { column = 'serializedtags', type = 'hstore' },
9      { column = 'geom', type = 'point', projection = 4326 },
10     { column = 'uid', type = 'text' },
11     { column = 'username', type = 'text' },
12     { column = 'name', type = 'text' },
13     { column = 'url', type = 'text' },
14     { column = 'geom_changed', type = 'bool' },
15     { column = 'object_changed', type = 'bool' },
16     { column = 'deleted', type = 'bool' },
17     { column = 'object_new', type = 'bool' },
18     { column = 'has_name', type = 'bool' },
19     { column = 'valid', type = 'bool' }
20   },
21   schema = SCHEMA_NAME
22 })
```

Listing 1.1 Zeile 4 spezifiziert den osm Typ für den die Tabelle angelegt werden soll (in dem Fall, für alle nodes) und die id der node wird in die Spalte „osm_id“ eingetragen.
Zeile 6 erzeugt eine Spalte mit einem uniquen Integer Wert, welcher mit einem weiteren SQL Script in einen „Primary Key“ verändert werden kann.
Die Einträge Zeile 7 - 20 definieren die weiteren Spalten der Tabelle „nodes“, welche mit einer Prozess Funktion beschrieben werden.
Des Weiteren kann wie in Zeile 22 ein Schema definiert werden, in die diese Tabelle geschrieben wird.

1.3.2. Hilfsfunktionen und Variablen

mapfeatures

Listing 1.2: Deklaration einer lua Tabelle für die mapfeatures

```
1 local map_features = {
2   'admin_level', 'aerialway', 'aeroway', 'amenity', 'barrier', 'boundary',
3   'building', 'craft', 'emergency', 'geological', 'healthcare', 'highway',
4   'historic', 'landuse', 'leisure', 'man_made', 'military', 'natural',
5   'office', 'place', 'power', 'public_transport', 'railway', 'route',
6   'shop', 'sport', 'telecom', 'tourism', 'water', 'waterway'
7 }
```

In Listing 1.2 werden die sogenannten „mapfeatures“[2] definiert.
OpenStreetMap stellt physische Merkmale am Boden (z. B. Straßen oder Gebäude) mithilfe von Tags dar, die an seine grundlegenden Datenstrukturen (nodes, ways und relations) angehängt sind. Jedes Tag beschreibt ein geografisches Attribut des Features, das von diesem bestimmten nodes, ways oder dieser relations angezeigt wird.

Hilfsfunktion für name, mapfeatures, serializedtags und url

Die Hilfsfunktion in Listing 1.3 (nächste Seite) analysiert das übergebene Objekt und gibt vier Werte zurück.

name Der primäre Name: im allgemeinen der prominenteste ausgeschilderte Name oder der gebräuchlichste Name in der/den Landessprache(n).

mapfeatures Eine lua Tabelle, welche ein key-value Paar enthält. Diese Tabelle enthält alle tags welche eine Übereinstimmung mit der map_features lua Tabelle (siehe Listing 1.2) haben.

serializedtags Eine lua Tabelle mit alle object.tags, welche nicht direkt für die weitere Konvertierung benötigt werden.

url Bei der Analyse der osm Dateien wurde festgestellt, dass eine url beziehungsweise Websitenreferenz auf verschiedene Arten eingetragen werden kann. Dieses Problem wurde ebenfalls mit der Hilfsfunktion get_tag_quadruple() realisiert.

Der *goto* Befehl entspricht in diesem Beispiel dem *break* in Java.
Zusätzlich dienen die Zeilen 23 - 42 der Vereinfachung der Tabelleneinträge. Alle Variablen die nil sind werden in PostgreSQL mit NULL eingetragen, somit entfällt eine String Vergleich um leere Einträge zu finden. Es kann direkt nach ISNULL oder NOTNULL im SQL Statement gefragt werden.

Listing 1.3: Hilfsfunktion zur osm object.tag Verarbeitung

```lua
local function get_tag_quadruple(object)
  local features = {}
  local ser = {}
  local url = nil
  local name = object:grab_tag('name')
  -- Iterate over each tag from the osm object
  for key, value in pairs(object.tags) do
    -- find url or website entry
    if key == 'url' then
      url = object:grab_tag(key)
      goto continue
    elseif key == 'website' then
      url = object:grab_tag(key)
      goto continue
    elseif osm2pgsql.has_suffix(key, ':url') then
      url = object:grab_tag(key)
      goto continue
    elseif osm2pgsql.has_suffix(key, ':website') then
      url = object:grab_tag(key)
      goto continue
  end

  if list_contains(map_features, key) then
    -- osm object.tag is definied as a mapfeature
    features[key] = value
    goto continue
  else
    -- osm object.tag is not very relevant,
    -- therefore it is stored in serializedtags table
    ser[key] = value
    goto continue
  end
  ::continue::
  end
  -- If the table is empty, an empty table should not be saved.
  -- Instead, the value is set to nil (PostgreSQL NULL)
  if next(features) == nil then
    features = nil
  end
  if next(ser) == nil then
    ser = nil
  end

  return name, features, url, ser
end
```

1.3.3. Prozessfunktionen

Damit die nodes, ways, relations spezifisch in die gewünschten Tabellen eingetragen werden, müssen die lua Funktionen osm2pgsql.process_node osm2pgsql.process_way osm2pgsql.process_relation entsprechend aufgerufen werden. Beispielhaft wurde die definierte Funktion osm2pgsql.process_relation siehe nächste Seite in Listing 1.4 aufgeführt. Diese enthält:

Tabelle 1.2.: Kurze Zeilen Beschreibung von Listing 1.4

Name	Zeilen-nummern	Beschreibung
get_tag_quadruple()	2	Erstellung von vier Variablen mit einer Hilfsfunktion (siehe Abschnitt 1.3.2)
tables . relations :add_row()	3-18	Fügt ein osm Objekt entsprechend der Tabelleninitialisierung (beispielhaft Unterabschnitt 1.3.1) in eine Tabelle ein. Hierbei wird die id automatisch vergeben und eine Geometrie anhand der Bestehenden Daten erzeugt.
Schleife über alle relationmembers	19-49	Alle nodes, ways, relations, die im tag *members* vermerkt sind werden in eine seperate Tabelle *relationmembers* eingetragen. Für die Eindeutigkeit wird der Typ des *members* abgefragt.

Die Prozessfunktionen für nodes und ways sieht ähnliches aus, nur das die osm2pgsql.process_node Funktion keine zusätzlichen *members* besitzt und in osm2pgsql.process_way die Einträge der ways nur aus nodes bestehen.

Listing 1.4: Hilfsfunktion zur osm object.tag Verarbeitung

```
function osm2pgsql.process_relation(object)
  local object_name, object_features, object_url, object_serializedtags =
    get_tag_quadruple(object)
  tables.relations:add_row({
    name = object_name,
    url = object_url,
    tstamp = reformat_date(object.timestamp),
    mapfeatures = object_features,
    serializedtags = object_serializedtags,
    geom = { create = 'area' },
    uid = object.uid,
    username = object.user,
    geom_changed = false,
    object_changed = false,
    deleted = false,
    object_new = false,
    has_name = false,
    valid = false
  })
  for _, member in ipairs(object.members) do
    -- if type is a node
    if member.type == "n" then
      tables.relationmembers:add_row({
        relation_id = object.id,
        node_id = member.ref,
        way_id = nil,
        member_rel_id = nil,
        role = member.role
      })
    end
    -- if type is a way
    if member.type == "w" then
      tables.relationmembers:add_row({
        relation_id = object.id,
        node_id = nil,
        way_id = member.ref,
        member_rel_id = nil,
        role = member.role
      })
    end
    -- if type is a relation
    if member.type == "r" then
      tables.relationmembers:add_row({
        relation_id = object.id,
        node_id = nil,
        way_id = nil,
        member_rel_id = member.ref,
        role = member.role
      })
    end
  end
end
```

1.4. Ausführung

Alle vorherigen Sektionen beschreiben die wichtigen Teile des Befehls zur Importierung von osm Daten in die intermediate Datenbank. Der Prozess der Importierung kann nun wie folgt ausgeführt werden.

```
1  osm2pgsql -d ohdm -U postgres -O flex -x \
2    -S /home/stesad/osm2inter/osm2inter.lua \
3    -c /home/stesad/osm2inter/berlin.osm
```

Hinweis: Die Dateien müssen mit absoluten Pfaden angegeben werden, oder im Verzeichnis des Datenbanknutzers gespeichert werden.
Powershell beziehungsweise Windows Command Ausführung in Appendix Abschnitt A.1

2. psql

psql ist ein terminalbasiertes Frontend für PostgreSQL. Es ermöglicht Ihnen, Abfragen interaktiv einzugeben, sie an PostgreSQL auszugeben und die Abfrageergebnisse anzuzeigen. Alternativ kann die Eingabe aus einer Datei erfolgen. Darüber hinaus bietet es eine Reihe von Meta-Befehlen und verschiedene shell-ähnliche Funktionen, um das Schreiben von Skripten und die Automatisierung einer Vielzahl von Aufgaben zu erleichtern.[3]

Für die Ausführung von psql werden in diesem Beispiel Flags benötigt die in Tabelle 2.1 aufgeführt sind.

Tabelle 2.1.: flags

Flag	Beschreibung
-d	Name der Datenbank
-c cammand	Spezifiziert ein Kommando das mit psql ausgeführt werden soll.
-f	Ermöglich die Verwendung von Dateien als Quelle für Befehle.

In Windows muss zusätzlich -W für eine Passworteingabe als Flag gesetzt werden

```
1  sudo -iu postgres psql -d ohdm \
2  -f /home/stesad/osm2inter/osm2inter_postprocess.sql
```

Hinweis: Die Dateien müssen mit absoluten Pfaden angegeben werden, oder im Verzeichnis des Datenbanknutzers gespeichert werden.
Powershell beziehungsweise Windows Command Ausführung in Appendix Abschnitt A.2

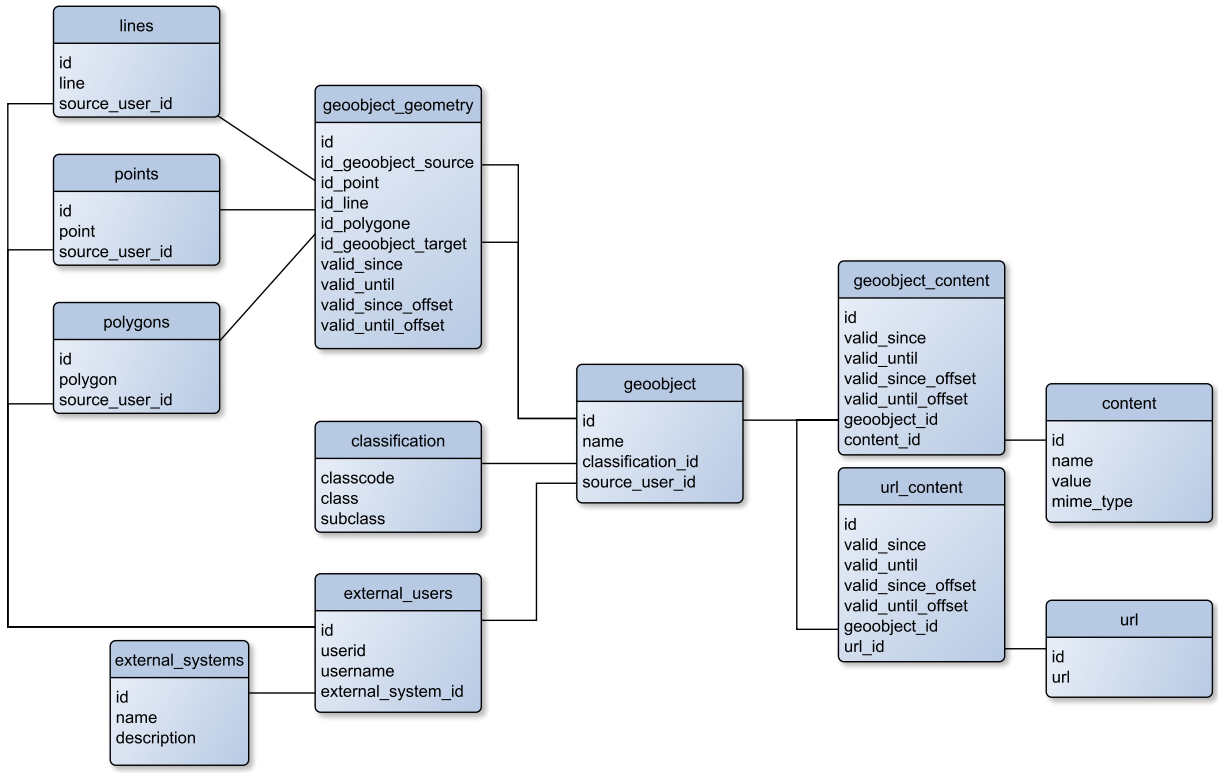
Teil II.

inter2ohdm

3. OHDM

3.1. Schema

Abbildung 3.1.: Schematischer Aufbau der OHDM Datenbank



Die Erstellung des Schemas wurde mithilfe der existierenden Beschreibung von https://github.com/OpenHistoricalDataMap/OHDM-Documentation/wiki/OHDM-Data-Model und den Konvertierten Einträgen des „alten“ OHDMConverters realisiert.

3.2. Theoretische Konvertierung

Abbildung 3.2.: Schematische Darstellung der einzelnen INSERT SQL Statemants

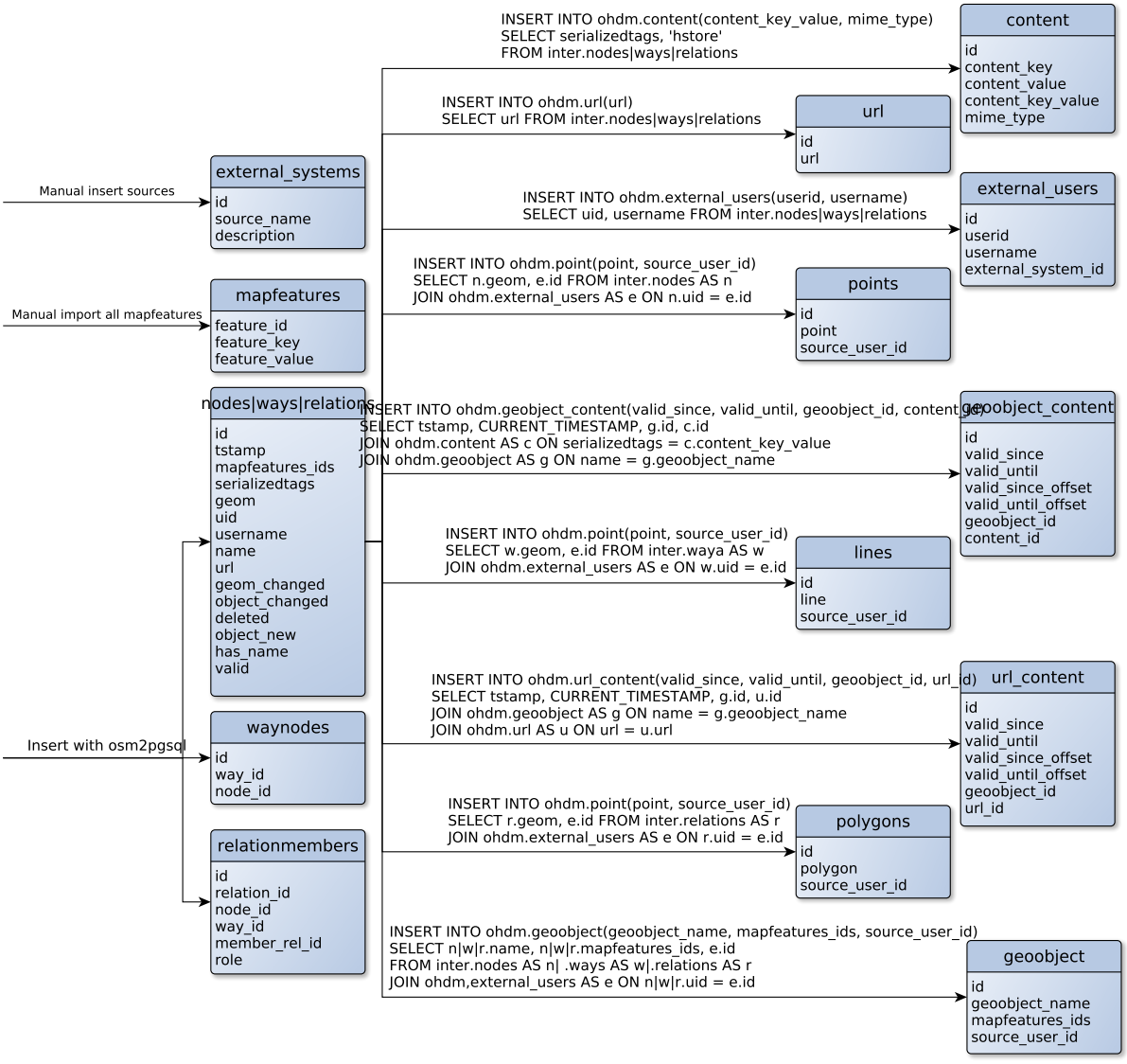
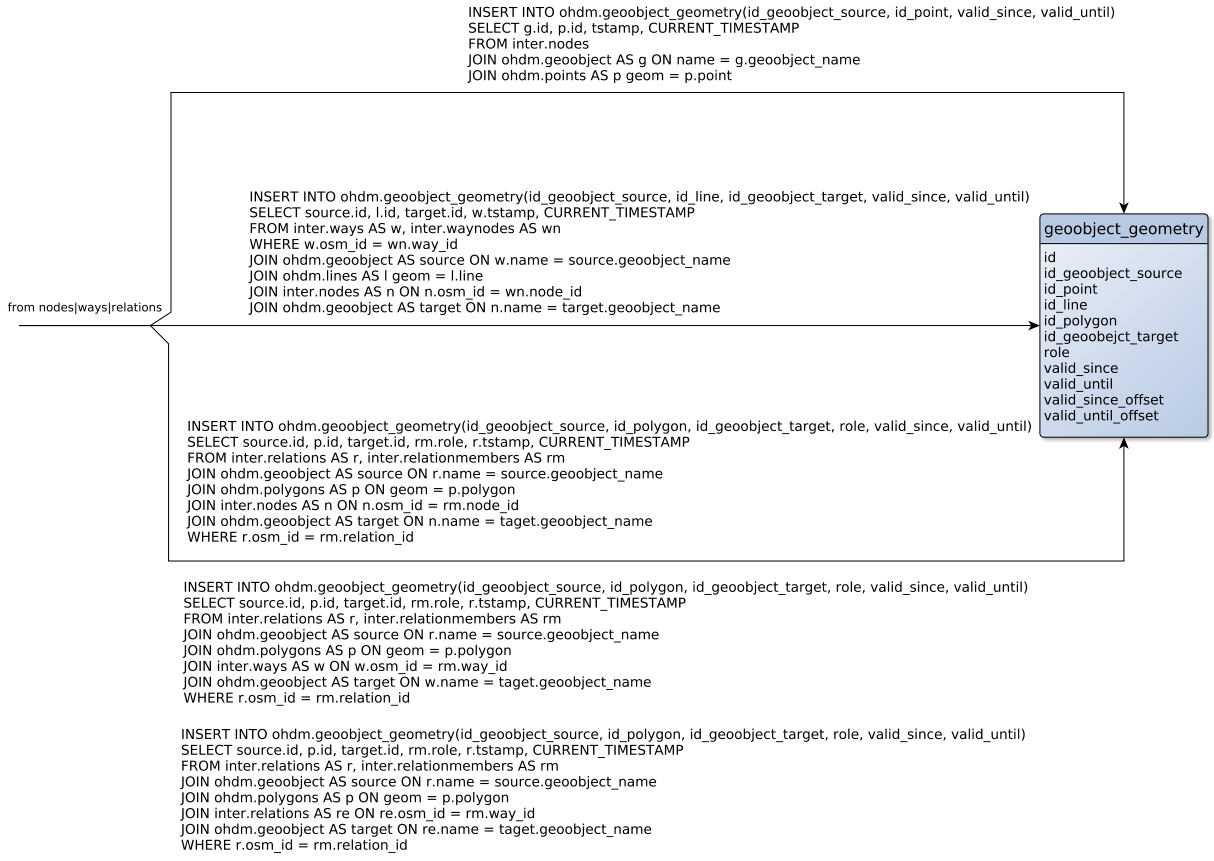


Abbildung 3.3.: Schematische Darstellung der einzelnen INSERT SQL Statemants




```
INSERT INTO ohdm.geoobject_geometry(id_geoobject_source, id_polygon, id_geoobject_target, role, valid_since, valid_until)
SELECT source.id, p.id, target.id, rm.role, r.tstamp, CURRENT_TIMESTAMP
FROM inter.relations AS r, inter.relationmembers AS rm
JOIN ohdm.geoobject AS source ON r.name = source.geoobject_name
JOIN ohdm.polygons AS p ON geom = p.polygon
JOIN inter.ways AS w ON w.osm_id = rm.way_id
JOIN ohdm.geoobject AS target ON w.name = taget.geoobject_name
WHERE r.osm_id = rm.relation_id
```

```
INSERT INTO ohdm.geoobject_geometry(id_geoobject_source, id_polygon, id_geoobject_target, role, valid_since, valid_until)
SELECT source.id, p.id, target.id, rm.role, r.tstamp, CURRENT_TIMESTAMP
FROM inter.relations AS r, inter.relationmembers AS rm
JOIN ohdm.geoobject AS source ON r.name = source.geoobject_name
JOIN ohdm.polygons AS p ON geom = p.polygon
JOIN inter.relations AS re ON re.osm_id = rm.way_id
JOIN ohdm.geoobject AS target ON re.name = taget.geoobject_name
WHERE r.osm_id = rm.relation_id
```

Quellenverzeichnis

[1] Osm2pgsql Manual - osm2pgsql. URL: https://osm2pgsql.org/doc/manual.html (besucht am 22. 04. 2022).

[2] OpenStreetMap - Map features. en. Nov. 2017. URL: https://wiki.openstreetmap.org/wiki/Map_features (besucht am 28. 04. 2022).

[3] psql. en. Nov. 2017. URL: https://www.postgresql.org/docs/9.2/app-psql.html (besucht am 28. 04. 2022).

A. Windows

A.1. osm2pgsql

change to windows variant

```
1 osm2pgsql -d ohdm -U postgres -O flex -x \
2 -S /home/stesad/osm2inter/osm2inter.lua \
3 -c /home/stesad/osm2inter/berlin.osm
```

A.2. psql

change to windows variant

```
1 sudo -iu postgres psql -d ohdm \
2 -f /home/stesad/osm2inter/osm2inter_postprocess.sql
```