Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Bericht über die Praktikumstätigkeit

Implementierung einen neuen Prototyps des OHDMConverters

an der HTW Berlin

Fachbereich 4 - Informatik, Kommunikation und Wirtschaft,
Studiengang Angewandte Informatik

Zeitraum des Fachpraktikums
14.02.2022 - 14.05.2022

Vorname, Name	Stefan Sadewasser
Matrikelnummer	568158
Studiengang, Fachbereich	Angewandte Informatik, Fachbereich 4
E-Mail	stefan.sadewasser@student.HTW-Berlin.de

Inhaltsverzeichnis

Abkürzungsverzeichnis

GUI Graphical User Interface 3

OHDM Open Historical Data Map 4

DBMS Database Management System 4

OSM OpenStreetMap 2

PBF Protocolbuffer Binary Format

Teil I.

Praktikumsplan, Datenbank, javabasierte Konvertierung

1. Praktikum

1.1. Arbeitsauftrag

Im Rahmen des Praktikums sollte ein Linux-Server vor ein Kartenprojekt neu aufgesetzt werden. In Zusammenarbeit mit den Administratoren vor Ort musste ein PostGIS-Datenbankserver aufgesetzt werden. Danach sollte eine DBs aufgesetzt werden, die in Summe ca. 1 TByte Daten verwalten kann. Das Füllen der DBs würde über vorhandene Tools erfolgen. Die zusätzliche Aufgabe war es Studierende zu unterstützt, die an der Weiterentwicklung dieser Tools arbeiten.
Die Arbeit sollte wie folgt aufgeteilt werden:

1. Dokumentation und Aufsetzen der Testumgebung
   a) Dokumentation einer Installationsanleitung für die Benutzung des OHDMConverters zur Importierung von OpenStreetMap (OSM) Daten
   b) Dokumentation der Aufsetzung einer Testumgebung
   c) Dokumentation wie in 1a für Windows und MacOS Systeme
2. Wiederaufsetzen des physischen OHM Servers
   a) Aufsetzen 2 separater Datenbankserver auf dem physischen OHM Server
   b) Importierung des planet.osm Datensatzes im Produktivdatenbankserver
   c) Formulierung eines Cron-Jobs zur Automatisierung
   d) Importierung von OSM Daten von Deutschland im Integrationsdatenbankserver

1.2. Planung

Die Aufteilung wurde in separierten Arbeitspaketen festgehalten, welche dann im Planungskalender für ein oder zwei Kalenderwochen Bearbeitungszeit eingetragen wurden.

KW	7	8	9	10	11	12	13	14	15	16	17	18	19	20
AP 1a														
	AP 1b													
			AP 2a											
				AP 2b										
						AP 1c								
								AP 2c						
									AP 2d					

2. Datenbankserver

Der Datenbankserver sollte mit 2 separaten PostGIS Datenbanken erstellt werden. Die Erstellung und Einrichtung des Servers selbst geschah in enger Zusammenarbeit mit dem Laboringenieur Axel Wagner.
Der OHM Server wurde mit einem Ubuntu 20.04 in englischer Sprache und ohne Graphical User Interface (GUI) eingerichtet.
Des Weiteren wurden auf dem Server mehrere administrative Tools installiert, welche den Mitarbeitern beziehungsweise Laboringenieuren die Arbeit mit den Servern erleichtern.

1. **aptitude**
   ist eine Erweiterung der Paketverwaltung APT, aber im Gegensatz zu apt-get führt aptitude über Änderungen der installierten Pakete „genauer“ Buch, so dass nicht mehr benötigte Pakete automatisch erkannt und deinstalliert werden. Die Installationsgeschichte wird in ein Log geschrieben, wodurch später angezeigt werden kann, wann oder warum ein Paket installiert wurde.

2. **openssh-server**
   Die OpenSSH-Serverkomponente sshd wartet ständig auf Client-Verbindungen von einem der Client-Tools. Wenn eine Verbindungsanforderung auftritt, baut sshd die richtige Verbindung auf, je nachdem, welches Client-Tool die Verbindung herstellt. Wenn sich der entfernte Computer beispielsweise mit der ssh-Client-Anwendung verbindet, baut der OpenSSH-Server nach der Authentifizierung eine Fernsteuerungssitzung auf. Wenn ein entfernter Benutzer eine Verbindung zu einem OpenSSH-Server mit scp herstellt, initiiert der OpenSSH-Server-Daemon nach der Authentifizierung eine sichere Kopie von Dateien zwischen dem Server und dem Client.

3. **net-tools**
   Eine Sammlung von Programmen, die den Basissatz der NET-3-Netzwerkdistribution für das Linux-Betriebssystem bilden. Dieses Paket enthält arp, hostname, ifconfig, ipmaddr, iptunnel, mii-tool, nameif, netstat, plipconfig, rarp, route und slattach.

4. **git**
   ist ein dezentrales Versionsverwaltungssystem.

5. **nullmailer**
   ist ein reiner Weiterleitungs-MTA (Mail Transfer Agent). Das bedeutet, dass alle auf einem System eingehenden E-Mails an einen konfigurierten externen Mailserver weitergeleitet werden. Dies kann nützlich sein, wenn die Installation eines lokalen E-Mail-Servers nicht erwünscht oder nicht wirklich sinnvoll ist, aber zumindest die System-E-Mails müssen irgendwo hin weitergeleitet werden.

6. **logwatch**
   ist ein in Perl geschriebenes Tool zur Analyse von Logdateien. Es soll Systemadministratoren helfen, die Übersicht über alle Vorgänge auf einem Serversystem zu behalten. Logwatch durchsucht die Logdateien des Systems und generiert eine Kurzfassung daraus, deren Gestaltung individuell konfiguriert werden kann. Diese kann dann entweder als Datei weiterverarbeitet oder zum Versenden an einen Mailserver weitergereicht werden.

7. **apticron**
   ist ein kleines Shellskript zur automatischen Benachrichtigung über Paket-Updates per E-Mail.

8. **fail2ban**
   ist ein Set aus Client, Server und Konfigurationsdateien, welches Logdateien überwacht, dort nach vordefinierten Mustern sucht und nach diesen temporär IP-Adressen sperrt.

Der OHM Server ist wie auch das Open Historical Data Map (OHDM) Projekt eine HTW Berlin Projekt, somit musst auch die Erreichbarkeit des Server aus dem HTW Netz gewährleistet werden. Dies wurde mit einem bash Script realisiert, das *iptables* Einträge zur Port/IP Freigabe enthält.

Das OHDM Projekt benutzt als Database Management System (DBMS) PostgreSQL mit PostGIS als Erweiterung.
Der erste Schritt, ist die Installierung von PostgreSQL. Dieser lässt sich auf Ubuntu 20.04 wie in Listing 2.1 installieren.

Listing 2.1.: Installation PostgreSQL

```
1 sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -
    cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
2 wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo
    apt-key add -
3 sudo apt-get update
4 sudo apt-get -y install postgresql-14
5 # from 2022-02-24
```

Im Anschluss kann PostGIS als eine räumliche Datenbankerweiterung für PostgreSQL (siehe Listing 2.2) installiert werden.

Listing 2.2.: Installation PostGIS

```
1 sudo apt-get install postgresql-14-postgis-3
2 sudo apt-get install postgresql-14-postgis-3-scripts
```

Während der Installationen sollte standardmäßig ein PostgreSQL Cluster[1] erstellt werden, welches folgende Daten besitzt:

Servername:	localhost
Port:	5432
Clustername:	main
Datenbankbenutzer	postgres
Standarddatenbank	postgres
Datenbasis	/var/lib/postgresql/14/main
Datenbank log Datei	/var/log/postgresql/postgresql-14-main.log
Pfad zu den Konfigurationen	/etc/postgresql/14/main/

2.1. Multiple Datenbanken in PostgreSQL

Mit PostgreSQL ist es möglich mehrere Datenbanken parallel laufen zu lassen.
Anhand des Arbeitspaketes sollten 2 zusätzliche Cluster auf dem OHM Server entstehen.
Hierfür wurden bestehende beziehungsweise mitinstallierte Tools verwendet.

2.1.1. Cluster erzeugen

Zur Erzeugung eines neuen PostgreSQL Clusters wird der Befehl in Listing 2.3 verwendet.
Für eine detaillierte Beschreibung der Clusterdaten dient Listing 2.3 Zeile 3 als Vorlage.
Weitere Überlegungen dazu in Anhang A.

Listing 2.3.: Erzeugung eines PostgreSQL Clusters

```
1 sudo pg_createcluster [postgresql_version_number] [clustername] -p [port]
2 # Beispiel mit integration
3 sudo pg_createcluster 14 integration -p 5433\
```

Durch die Verwendung des Befehls aus Listing 2.3 werden die Clusterdaten wie folgt erzeugt:

Servername:	localhost
Port:	5432
Clustername:	integration
Datenbankbenutzer	postgres
Standarddatenbank	postgres
Datenbasis	/var/lib/postgresql/14/integration
Datenbank log Datei	/var/log/postgresql/postgresql-14-integration.log
Pfad zu den Konfigurationen	/etc/postgresql/14/integration

Weitere Informationen unter:
http://manpages.ubuntu.com/manpages/trusty/man8/pg_createcluster.8.html

2.1.2. Cluster steuern

Ein start/stop oder restart lässt sich nun mit dem Befehl in Listing 2.4 realisieren.

Listing 2.4.: Steuerung des Clusters

```
1 sudo pg_ctlcluster [postgresql_version_number] [clustername] [start|stop|
    restart]
2 # Beispiel mit integration
3 sudo pg_ctlcluster 14 integration [start|stop|restart]
```

Auch hierzu gibt es weitere Informationen, die unter:
http://manpages.ubuntu.com/manpages/trusty/man8/pg_ctlcluster.8.html
eingesehen werden können.

2.1.3. Cluster als Service registrieren

Eine Integration des erstellten Clusters als Service ist in Listing 2.5 einzusehen. Dies wird benötigt um auch nach einem Server Neustart gewährleisten zu können das der Datenbank Server dieses Clusters erreichbar ist.

Listing 2.5.: Registierung des Clusters als System Service

```
1 sudo systemctl enable postgresql@[postgresql_version_number]-[clustername].
    service
2 # Beispiel mit integration
3 sudo systemctl enable postgresql@14-integration.service
```

2.2. Datenbank Fernzugriff

Für eine Freigabe des Zugriffs auf die Datenbank von anderen Adressen als den localhost muss nicht nur der physische Server mit entsprechenden Freigaben eingestellt werden, sondern auch die Datenbank selbst.
Hierfür müssen die Konfigurationsdateien *postgresql.conf* und *pg_hba.conf* der Datenbank angepasst werden. Diese befinden sich im Pfad: /etc/postgresql/[version_number]/[clustername]/ beziehungsweise am Beispiel integration: /etc/postgresql/14/integration (vgl. Unterabschnitt 2.1.1)

2.2.1. postgresql.conf

In der Datenbankkonfigurationsdatei können einige Anpassungen vorgenommen werden. In diesem Beispiel wird sich auf die Anpassung der Abgehörten IP Adressen beschränkt.
in den Grundeinstellungen ist ein PostgreSQL Datenbank Server so eingestellt das ein Zugriff auf die Daten nur Lokal möglich ist. Für die Möglichkeit eines Fernzugriffs auf die Daten muss der Eintrag (siehe Listing 2.6) geändert werden. Wie in den Kommentaren ersichtlich kann dabei auf 2 Varianten zurückgegriffen werden.

1. Freigabe einer Liste von IP Adressen die durch Komma separiert sind oder
2. Freigabe aller IP Adressen durch Benutzung „*“

Die 2. Variante sollte die fehlerfreie Freigabe von IP Adressen in den Firewall Einstellungen enthalten. Damit keine Sicherheitskonzepte verletzt werden.

Listing 2.6.: postgresql.conf Ausschnitt

```
1 # - Connection Settings -
2
3 #listen_addresses = 'localhost'   # what IP address(es) to listen on;
4 # comma-separated list of addresses;
5 # defaults to 'localhost'; use '*' for all
6 # (change requires restart)
```

2.2.2. pg_hba.conf

Die Einstellungen der Authentifizierungsmethode werden in der Datenbankkonfigurationsdatei *pg_hba.conf* definiert. In Listing 2.7 ist ein Ausschnitt einer solchen Datei auf einem Ubuntu 20.04 zu sehen. Diese definiert zum Beispiel in Zeile 2 eine Peer Authentifikation des Systembenutzers *postgres*.
Bei der Peer-Authentifizierungsmethode wird der Betriebssystem-Benutzername des Clients vom Kernel abgerufen und als zulässiger Datenbank-Benutzername verwendet (mit optionaler Zuordnung der Benutzernamen). Diese Methode wird nur bei lokalen Verbindungen unterstützt.(vgl. [2])

Listing 2.7.: pg_hba.conf Ausschnitt

```
1  # Database administrative login by Unix domain socket
2  local   all             postgres                                peer\
3
4  # TYPE  DATABASE        USER            ADDRESS                 METHOD
5
6  # "local" is for Unix domain socket connections only
7  local   all             all                                     peer
8  # IPv4 local connections:
9  host    all             all             127.0.0.1/32            scram-sha-256
10 # IPv6 local connections:
11 host    all             all             ::1/128                 scram-sha-256
12 # Allow replication connections from localhost, by a user with the
13 # replication privilege.
14 local   replication     all                                     peer
15 host    replication     all             127.0.0.1/32            scram-sha-256
16 host    replication     all             ::1/128                 scram-sha-256
```

3. Konvertierung

3.1. OHDMConverter Java

Der javabasierte OHDMConverter diente als Projekthilfsprogramm um alle Importierungen oder Konvertierungen einer Datei/Schema in ein anderes zu realisieren.
Annahme: Da der OHDMConverter bis zum Beginn des Praktikums „nur“ mit kleineren oder Testdatensätzen getestet und validiert wurde, konnten einige Fehler nicht gefunden werden.

3.1.1. Importierung der planet.osm Datei

Die Hauptaufgabe war es das planet.osm File in die PostgreSQL Datenbank zu importieren. Die Vorbereitung dieser Importierung musste mit größter Sorgfalt bearbeitet werden, um:

- den fehlerfreien Ablauf zu gewährleisten,
- die Auslastung des physischen Servers so minimal wie möglich zu beschränken und
- Sicherheitskonzepte des physischen und des Datenbankservers einzuhalten.

Darüber hinaus musste auf die Größe der Datei berücksichtigt werden. Das heißt alle Tests fanden mit kleineren Dateien statt.

Tabelle 3.1.: Daten der planet.osm Datei

Eigenschaft	Wert
Größe OSM Datei	115 GB
Größe der PBF[3] Datei	63 GB
Anzahl Nodes in der Datenbank[4]	7 663 759 219
Anzahl Ways in der Datenbank[4]	856 401 369
Anzahl Relations in der Datenbank[4]	9 879 181

Daten vom: 2022-05-03 23:59 UTC

3.1.2. Fehlverhalten bei der Importierung

admin_level

Der Schlüssel admin_level=* beschreibt die Verwaltungsebene eines Merkmals innerhalb einer Regierungshierarchie. Er wird hauptsächlich für die Grenzen territorialer politischer Einheiten (z. B. Land, Staat, Gemeinde) zusammen mit boundary=administrative verwendet. Aufgrund kultureller und politischer Unterschiede entsprechen die Verwaltungsebenen verschiedener Länder nur annähernd einander.(vgl. [5])
Von OSM ist der Wert dieses Schlüssels als numerischer Wert zu speichern. Im OHDMConverter wurde auch von dieser Aussage ausgegangen, sodass für den Wert in Java ein Integer Wert angelegt wird und der Bedingung: Sollte der Wert nicht gelesen werden können wird das OSM Objekt verworfen.
Der Folgefehler daraus ist, dass alle OSM Objekte, die mit einem nicht numerischen Wert für den Schlüssel: *admin_level* eingetragen sind, nicht gelesen werden. Leider gibt es sehr viele dieser Objekte die für andere OSM Objekte wichtig sind, sodass im Umkehrschluss Objekt zu dem das Objekt gehört nicht mehr darstellbar sind beziehungsweise einen fehlerhaften Querverweis haben.

Abbruch

Die Importierung der OSM Datei in die PostgreSQL Datenbank wurde nicht vollständig ausgeführt. Anhand der log Dateien konnte ein Aussage zur Menge der nicht betrachteten OSM Objekte und einem weiteren Fehler getroffen werden. Die beiden letzten Zeilen des logs von osm2inter, allerdings für die Dokumentation aufbereitet, kann in Listing 3.1 eingesehen werden.

Listing 3.1.: Letzte zwei Zeilen des logs der Importierung von osm2inter

```
1 nodes: 7,505,830,920
2 ways: 836,886,605
3 relations: 2,382,475
4 elapsed time: 9 : 5 : 58 : 30
5 throwable caught in startElement:
6 java.lang.StringIndexOutOfBoundsException:
7 begin 1, end 0, length 1
```

Zum Zeitpunkt der Importierung waren laut:
https://taginfo.openstreetmap.org/reports/database_statistics
7 565 505 545 Nodes, 844 325 562 Ways und 9 742 774 Relations in der planet.osm Datei enthalten. Somit fehlten der Datenbank:

60 000 000 kurz	60 Mio, das entspricht	0,8%	Nodes
8 000 000 kurz	8 Mio, das entspricht	0,9%	Ways
7 000 000 kurz	7 Mio, das entspricht	75,5%	Relations

3.2. Erkenntnis

Aufgrund der gravierenden Fehler (vgl. Unterabschnitt 3.1.2) im javabasierten OHDMConverter wurde die Lösung als Java Applikation verworfen. Dies geschah in Rücksprache mit dem Projektleiter Prof. Dr.-Ing. Thomas Schwotzer, welcher die Grundlegende Idee OSM Daten in die OHDM Datenbank zu importieren, als reines PsotgreSQL Projekt umsetzten wollte.
Das OHDM Projekt beziehungsweise der OHDMConverter als Java Applikation läuft bereits seit einigen Jahren kann aber mit den aktuellen Mitteln nicht stabil genug implementiert werden (auch in absehbarer Zeit nicht).

Somit wurde die Projektidee neu definiert.

Teil II.

OHDMConverter als PostgreSQL Projekt

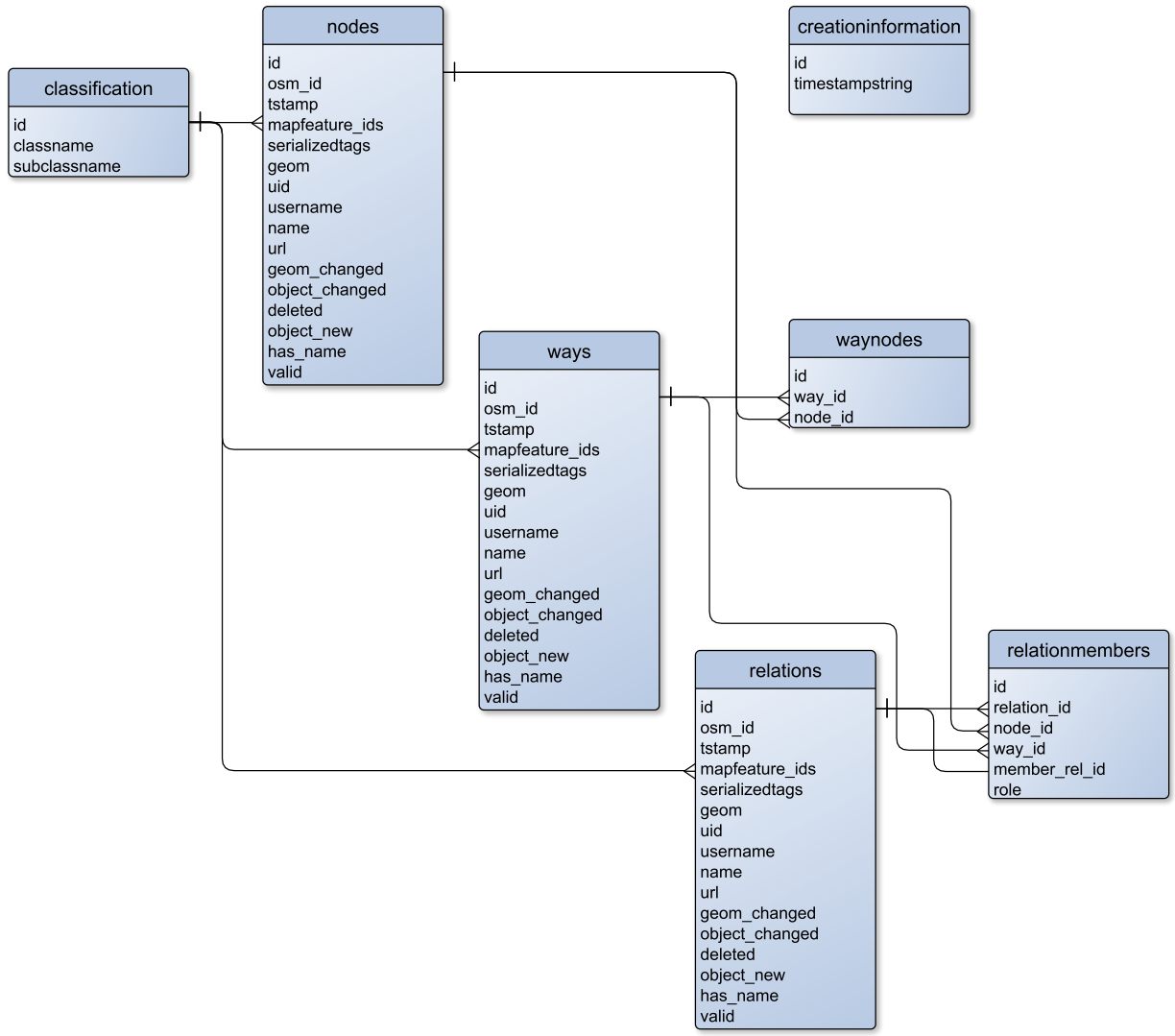
4. Projektidee

Die Grundidee von OHDM ist immer noch die selbe, kurz es sollen OSM Daten um die Dimension Zeit erweitert werden. Auch bleiben die bestehenden Datenbankschemata gleich. Die größte Änderung ist die Implementierung.

4.1. osm2inter

Für die Importierung von OSM Dateien soll ein bestehendes Tool „osm2pgsql“[6] verwendet werden. osm2pgsql ist dafür ausgelegt OSM Daten in eine PostGIS Datenbank zu importieren.

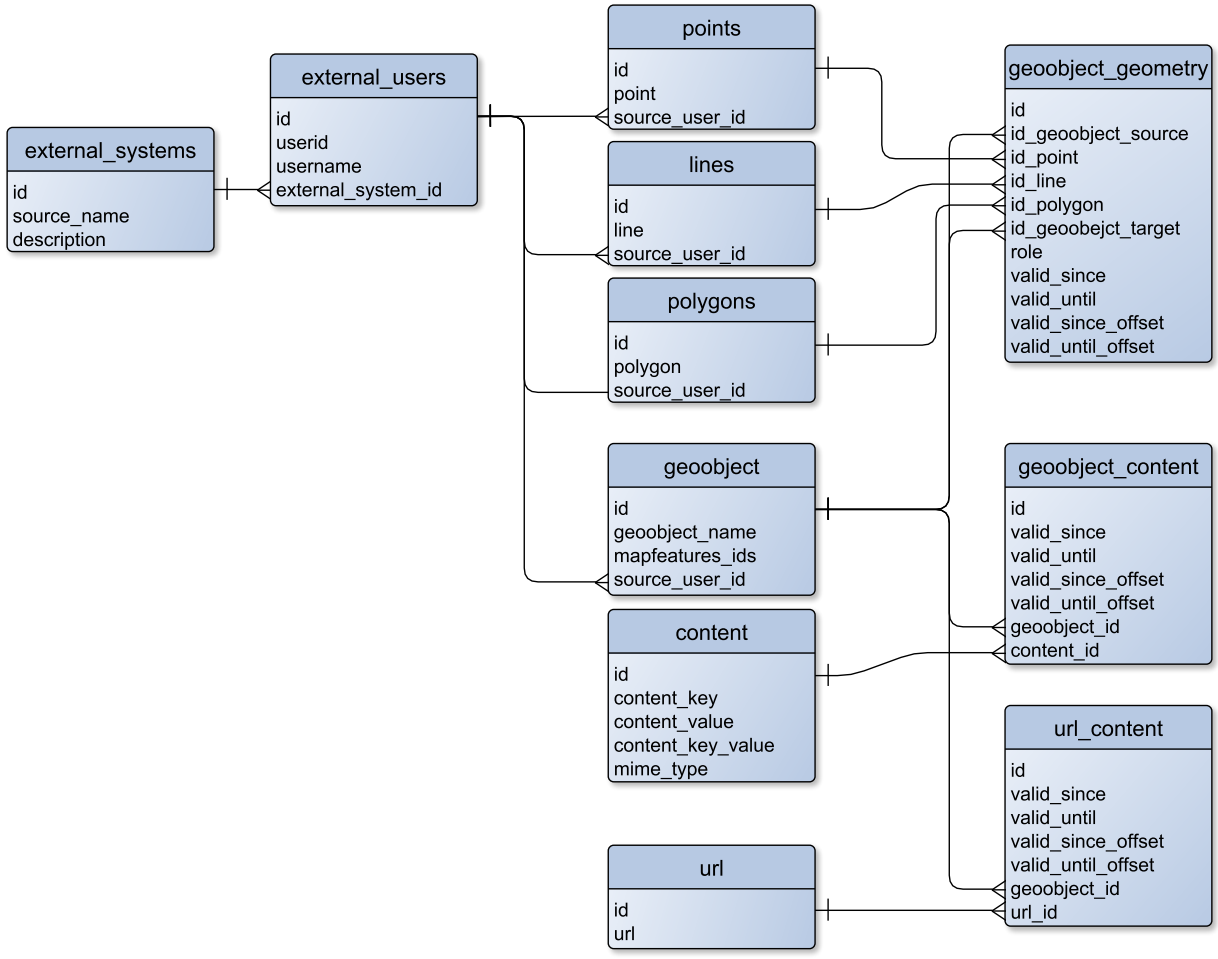
Abbildung 4.1.: Entity Relationship Diagramm der intermediate Datenbank



4.2. inter2ohdm

Die Konvertierung der intermediate Datenbank soll mithilfe von SQL Scripts realisiert werden. Hierbei soll, wenn möglich das Tool „psql“ eingesetzt werden.

Abbildung 4.2.: Entity Realtionship Diagramm der OHDM Datenbank



4.3. Zusammenfassung

Die beiden Datenbankschemata intermediate (vgl. Abbildung 4.1) und OHDM (vgl. Abbildung 4.2) sollen beibehalten werden. Allerdings soll jetzt mehr mit den Bordmittel von osm und PostgreSQL gearbeitet werden.

Die neue Projektidee musste mit bestehenden Datenstrukturen arbeiten, da sonst auch alle anderen Teilprojekte einer Anpassung bedurften.

5. osm2pgsql

5.1. Intro

Als Importierungskomponente kann osm2pgsql[7] sehr vielseitig eingesetzt werden. Innerhalb des Projektes löst osm2pgsql den OHDMConverter zum importieren von OSM in die intermediate Datenbank ab. Zusaätzlich lassen sich mit dem osm2pgsql nicht nur xml basierte OSM Dateien importieren sondern auch pbf und bz2 Kontainer. Allgemein sind PBF Datei zu bevorzugen, da sie nur ungefähr halb so groß, wie die xml basierenden OSM Dateien sind.[8]

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

Die Installation kann auf dem GitHub Repository über die Readme.md eingesehen werden

5.2. osm2pgsql Flags

Für die fehlerfreie Importierung sind mehrere Flags notwendig. In der nachfolgenden Tabelle 5.1 sind diese einzeln aufgelistet mit einer entsprechenden Erklärung.

Tabelle 5.1.: 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.
(mehr Ideen: Anhang B)

5.3. 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 wird 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

5.3.1. Tabelleninizialisierung

Im lua Script werden erforderliche Tabellen wie in Listing 5.1 angelegt. Dies wird benötigt um die Tabelle zu spezifizieren.

Listing 5.1.: 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 = 'mapfeature_ids', type = 'text' },
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 5.1 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.

5.3.2. Hilfsfunktionen und Variablen

mapfeatures

Listing 5.2.: 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 5.2 werden die sogenannten „mapfeatures“[9] 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.

Der aktuelle Stand beschriebt zusätzlich im lua Script eine csv import alles händisch inserierten Map Features.
(mehr Ideen: Anhang C)

Hilfsfunktion für name, mapfeatures, serializedtags und url

Die Hilfsfunktion in Listing 5.3 (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 5.2) haben. Der Rückgabetype ist ein String, der alle Classification Ids, Semikolon separiert, enthält.

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 *continue* 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 5.3.: Hilfsfunktion zur osm object.tag Verarbeitung

```lua
  local function get_tag_quadruple(object)
  -- table with classcodes from the mapfeatures
  local features = {}
  -- declation with one entry '-1' when the osm object has not a mapfeature
  table.insert(features, '-1')
  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 = get_classcode(key, value, features)
      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(ser) == nil then
      ser = nil
    end
  -- to save the lua table as text in PostgreSQL Database,
  -- the table entries concat with ';' as delimiter
  return name, table.concat(features, ';'), url, ser
end
```

5.3.3. 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 5.4 aufgeführt. Diese enthält:

Tabelle 5.2.: Kurze Zeilen Beschreibung von Listing 5.4

Name	Zeilen-nummern	Beschreibung
get_tag_quadruple()	2	Erstellung von vier Variablen mit einer Hilfsfunktion (siehe Unterunterabschnitt 5.3.2)
tables . relations :add_row()	3-18	Fügt ein osm Objekt entsprechend der Tabelleninitialisierung (beispielhaft Unterabschnitt 5.3.1) 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, mit dem Unterschied, dass die osm2pgsql.process_node Funktion keine zusätzlichen *members* besitzt und in osm2pgsql.process_way die Einträge der ways nur aus nodes bestehen.

Listing 5.4.: 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
```

5.4. 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.

6. 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.[10]
Für die Ausführung von psql werden in diesem Beispiel Flags benötigt die in Tabelle 6.1 aufgeführt sind.

Tabelle 6.1.: flags

Flag	Beschreibung
-d	Name der Datenbank
-c command	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
(mehr Ideen: Anhang B)

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

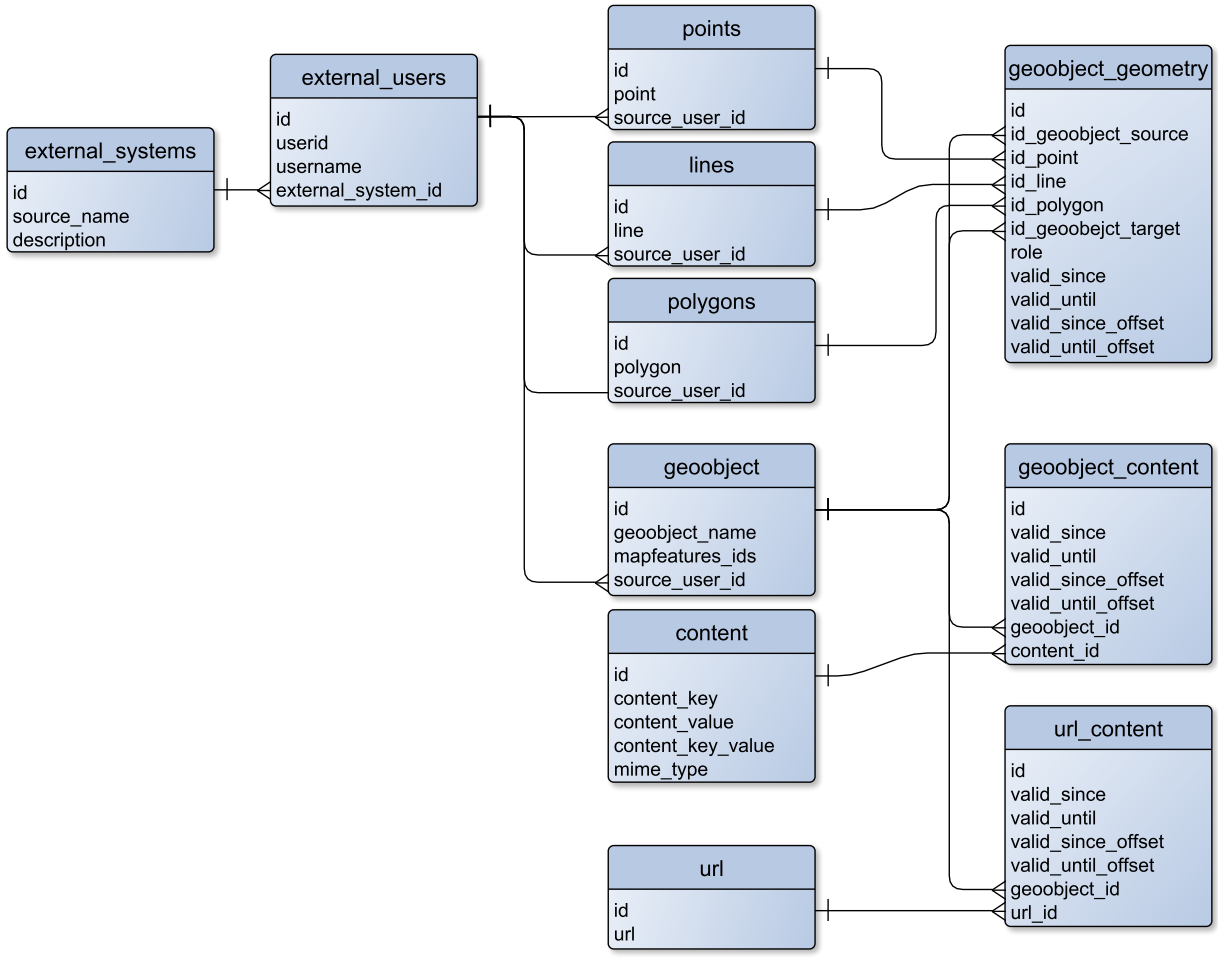
Hinweis: Die Dateien müssen mit absoluten Pfaden angegeben werden, oder im Verzeichnis des Datenbanknutzers gespeichert werden.

7. OHDM

Die Konvertierung wird ausschließlich mit einem PostgreSQL Scripts realisiert[1], das mit psql[10] gelesen wird. Nachfolgend wird diese Script in drei Teile gesplittet.

7.1. Tabellen

Abbildung 7.1.: OHDM Entity Relationship Modell



gleich der Abbildung 4.2

Im ersten Teil des PostgreSQL Scriptes werden alle Tabellen (siehe Abbildung 7.1) erzeugt, die Spaltennamen gesetzt mit deren Datenbankvariablentypen und die Primärschlüssel Eigenschaften eingetragen. Beispielhaft für diesen Vorgang ist in Listing 7.1 zu sehen, wie die Tabelle „geoobject_geometry“ initialisiert wird.

[1]Link zum PostgreSQL Script im OHDMConverter Reporitory
https://github.com/OpenHistoricalDataMap/OHDMConverter/blob/SteSad/inter2ohdm/inter2ohdm.sql

Listing 7.1.: Erzeugung der „geoobject_geometry“ Tabelle

```
1  CREATE TABLE IF NOT EXISTS ohdm.geoobject_geometry
2  (
3  id                    BIGSERIAL NOT NULL,
4  id_geoobject_source   BIGINT,
5  id_point              BIGINT,
6  id_line               BIGINT,
7  id_polygon            BIGINT,
8  id_geoobject_target   BIGINT,
9  role                  VARCHAR,
10 valid_since           DATE,
11 valid_until           DATE,
12 valid_since_offset    BIGINT,
13 valid_until_offset    BIGINT,
14 CONSTRAINT geoobject_geometry_pkey PRIMARY KEY (id)
15 );
```

7.2. Einfacher Insert

Mehrere Tabellen werden mit simplen SQL Queries initialisiert. Im Beipsiel der Tabelle „external_users“ werden lediglich die Tabelleneinträge von allen *nodes*, *ways* und *relations* vereinigt. Diese Vereinigungsmenge enthält alle Einträge, die zur Initialisierung der „external_users“ Tabelle benötigt werden. (vgl. Listing 7.2)

Listing 7.2.: Insert Statement für die „external_users“ Tabelle

```
1  INSERT INTO ohdm.external_users(userid, username)
2  (
3  SELECT uid::BIGINT, username FROM inter.nodes
4  UNION
5  SELECT uid::BIGINT, username FROM inter.ways
6  UNION
7  SELECT uid::BIGINT, username FROM inter.relations
8  );
```

7.3. Join Insert

Die meisten weiteren Initialisierungen werden mithilfe eines JOIN[2] Statements realisiert. Hierbei wird eine Vereinigungsmenge gebildet, die abhänig von einer Spalte der beider zu vereinenden Tabellen ist. In Listing 7.3 wird eine spezielle Initialsiserung aufgezeigt, welche alle Relationen der intermediate Datenbank in Verbindung mit der „relationmembers“ Tabelle in die „geoobject_geometry“ Tabelle der OHDM Datenbank einfügt.

[2] vgl. https://www.postgresql.org/docs/current/tutorial-join.html

Listing 7.3.: Insert Statement für die „geoobject_geometry“ Tabelle auf Basis von allen Realationen der intermediate Datenbank

```
1  INSERT INTO ohdm.geoobject_geometry(
2  id_geoobject_source,
3  id_polygon,
4  id_geoobject_target,
5  role,
6  valid_since,
7  valid_until
8  )
9  (
10 SELECT source.id, p.id, target.id, rm.role, r.tstamp, CURRENT_TIMESTAMP
11 FROM inter.relations AS r
12 JOIN inter.relationmembers AS rm ON r.osm_id = rm.relation_id
13 JOIN ohdm.geoobject AS source ON r.name = source.geoobject_name
14 JOIN ohdm.polygons AS p ON geom = p.polygon
15 JOIN inter.relations AS re ON re.osm_id = rm.way_id
16 JOIN ohdm.geoobject AS target ON re.name = target.geoobject_name
17 );
```

Teil III.

Schlussbetrachtung

8. Fazit

Die Implementation des Prototypes ist soweit abgeschlossen, das auch bestehende Tools des OHDMConverters weiterhin das OHDM Datenbankschema verwenden können.

9. Zusätzliche Ergebnisse

9.1. Laufzeitanalyse

Tabelle 9.1.: Laufzeitanalyse

Laufzeit	Max	Min	Durchschnitt	Median
Sekunden	138	108	128.16	128
Minuten:Sekunden	02:17	01:48	02:07	02:07

9.2. Backgroundprozess

sudo: a terminal is required to read the password; either use the -S option to read from standard input or configure an askpass helper

10. Weitere Überlegungen

Quellenverzeichnis

[1] *Creating a Database Cluster*; en; Feb. 2021;
https://www.postgresql.org/docs/9.5/creating-cluster.html
(besucht am 03. 05. 2022).

[2] *21.9. Peer Authentication*; en; Feb. 2022;
https://www.postgresql.org/docs/14/auth-peer.html
(besucht am 04. 05. 2022).

[3] *DE:PBF Format – OpenStreetMap Wiki*;
https://wiki.openstreetmap.org/wiki/DE:PBF_Format
(besucht am 04. 05. 2022).

[4] *OpenStreetMap Taginfo*;
https://taginfo.openstreetmap.org/reports/database_statistics
(besucht am 04. 05. 2022).

[5] *Key:admin_level – OpenStreetMap Wiki*;
https://wiki.openstreetmap.org/wiki/Key:admin_level
(besucht am 04. 05. 2022).

[6] *Home - osm2pgsql*;
https://osm2pgsql.org/
(besucht am 04. 05. 2022).

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

[8] *PBF Format – OpenStreetMap Wiki*;
https://wiki.openstreetmap.org/wiki/PBF_Format
(besucht am 06. 05. 2022).

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

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

Teil IV.

Anhang

A. Multiple Datenbanken in PostgreSQL

Die Alternative zur Erstellung eines PostgreSQL Clusters ist die Verwendung von initdb[1], allerdings gibt mit dieser Variante einige Herausforderungen die mit pg_createcluster leichter beziehungsweise überhaupt zu bewältigen waren.

1. Steuerung des Clusters für die Serververwaltung
2. Cluster als Service auch nach einem Neustart des Server starten

B. Windows Flag W

Theoretisch sollte es möglich sein auch in Windows eine Peer Authentifizierung zuzulassen. Dafür müsste die Einstellung in der pg_hba.conf geändert werden und dann validiert ob und wie mit der Datenbank interagiert werden kann.

C. Curl Map Features

Um die Arbeit mit den Map Features[9] zu erleichtern, müsste man ein curl Script implemntieren, dass die Tabelleneinträge auf der Map Features Webseite ausliest und in eine csv Datei oder ähliches schreibt.

Damit wäre es im Anschluss möglich die csv Datei als Insert Grundlage der classification Tabelle zu verweden.