Inhaltsverzeichnis

Verbindungen schaffen mit PostgreSQL Foreign Data Wrappers

Über Foreign Data Wrapper (FDW) kann aus der Datenbank heraus eine Verbindung zu anderen Datenquellen aufgebaut werden. Dadurch ist es nicht mehr notwendig, dass alle Daten, die in einem Projekt innerhalb der PostgreSQL-Datenbank verwendet werden sollen, sich auch in der Datenbank befinden müssen.

Das Konzept des datenbankübergreifenden Zugriffs ist nicht PostgreSQL spezifisch, sondern wurde im SQL/MED Standard (MED – Management of External Data) bereits 2008 definiert (https://wiki.postgresql.org/wiki/SQL/MED). Foreign Data Wrappers wurden bisher nur in wenigen Softwareprodukten implementiert, darunter PostgreSQL, MariaDB und IBM/DB2.

Dieser Beitrag versucht mit einfachen Beispielen das Thema zu erklären. Die Beispiele beziehen sich auf OSGeoLIive 13.0 und können einfach ausprobiert werden (https://live.osgeo.org).

Zugriff auf Daten über dblink

Bevor es FDW gab war der Zugriff aus einer Datenbank auf eine andere PostgreSQL Datenbank ebenfalls möglich. Hierzu konnte und kann die Extension dblink verwendet werden. Dblink hat aber die Einschränkung, dass nur auf Daten aus PostgreSQL-Datenbanken zugegriffen werden kann.

```
CREATE EXTENSION dblink;
SELECT *
FROM dblink('dbname=osm_local',
            'SELECT osm_id, name, way as geom
            FROM public.planet_osm_point
            WHERE amenity = ''cafe''')
      AS foo
      ( osm_id int8, name text, geom
geometry(point,4326));
```

Beispiel 1: Zugriff über dblink auf die Tabelle planet_osm_point in der Datenbank osm_local (OSGeoLive)

Foreign Data Wrapper in PostgreSQL

Ein Zugriff auf weitere externe Datenquellen war dann 2001 mit der Einführung von Foreign Data Wrapper mit der PostgreSQL Version 9.1 möglich. In dieser Version bestand allerdings lediglich lesender Zugriff auf externe Daten. Es gab aber schon diverse FDW für unterschiedlichste externe Datenquellen.

Ab PostgreSQL 9.3 (2013) kann nun auch schreibend auf Daten zugegriffen werden.

Neben dem Zugriff auf andere PostgreSQL-Datenbanken werden zahlreiche andere Datenquellen unterstützt (https://wiki.postgresql.org/wiki/Foreign_data_wrappers).

Hier sind beispielsweise Implementationen für ODBC sowie SQL-

Datenbanken wie ORACLE, MS SQL, Informix, MySQL, SQLite oder auch NoSQL-Datenbanken wie Kafka oder CouchDB sowie Dateischnittstellen zu CSV, JSON, pg_dump-Dateien oder auch auf komprimierte Dateien zu nennen.

Besonders interessant und sehr mächtig ist der OGR-Wrapper, der den Zugriff auf zahlreiche Geodatenquellen ermöglicht.

Die diversen Erweiterungen wurden von unterschiedlichsten Personen entwickelt und müssen separat installiert werden.

Nun soll anhand von einfachen Beispielen die Verwendung von Foreign Data Wrappern veranschaulicht werden.

Das Konzept

Der Zugriff auf externe Datenquelle erfolgt immer nach dem gleichen Konzept.

- Laden der benötigten Erweiterung z.B. postgres_fdw
- Erzeugen eines entfernten Servers (Foreign Server)
- Definition des User-Mappings (Foreign User). Mit welchem Benutzer soll auf die entfernte Datenquelle zugegriffen werden (wird nicht immer benötigt).
- Definition der entfernten Tabelle (Foreign Table). Tabellendefinition passend zu den entfernten Daten. Kann über IMPORT FOREIGN SCHEMA angelegt werden.

Anschließend kann auf die Foreign Table über SQL zugegriffen werden.

postgresql_fdw für den Zugriff auf PostgreSQL-Datenbanken

Aus der Datenbank natural_earth2 soll auf die Datenbank osm_local zugegriffen werden.

1. Laden der Erweiterung postgres_fdw

Diese Erweiterung liegt in einer Standardinstallation vor und muss nicht separat installiert werden. Die Erweiterung muss lediglich geladen werden.

```
CREATE EXTENSION postgres_fdw;
```

2. Erzeugen des Fremdservers

Beim Erzeugen des Fremdservers muss definiert werden, welcher FDW verwendet werden soll (hier postgresql_fdw). Außerdem müssen Optionen für die Verbindung angegeben werden. Hier sind es die Verbindungsparameter, um die entfernte Datenbank zu erreichen.

```
CREATE SERVER fdw_pg_server_osm_local
  FOREIGN DATA WRAPPER postgres_fdw
  OPTIONS
  (host '127.0.0.1', port '5432', dbname 'osm_local');
```

3. Definiton des User Mappings

Definition des Benutzers, mit dem auf die entfernte Quelle zugegriffen werden soll.

```
CREATE USER MAPPING FOR user
  SERVER fdw_pg_server_osm_local
 OPTIONS (user 'user', password 'user');
```

4. Aufbau der Fremdtabellen mit Hilfe von IMPORT FOREIGN SCHEMA

```
--Definition eines Schemas, in dem die Fremdtabellen
angelegt werden sollen
CREATE SCHEMA osm_fdw;
```

IIMPORT FOREIGN SCHEMA importiert alle Tabellen und auch Sichten des entfernten Schemas in das Ziel-Schema. Über LIMIT oder EXCEPT

kann eine Auswahl erfolgen.

```
IMPORT FOREIGN SCHEMA public
  LIMIT TO (planet_osm_polygon, planet_osm_point)
  FROM SERVER fdw_pg_server_osm_local
  INTO osm_fdw;
```

```
-- Nur die aufgeführten Tabellen werden als Fremdtabellen
angelegt:
LIMIT TO (table1, table2)

-- Es werden alle Tabellen als Fremdtabellen angelegt,
mit Ausnahme der aufgeführten Tabellen :
EXCEPT (table1, table2)
```

5. Zugrff auf Fremdtabellen

Nun können die Daten verwendet werden. Die folgende Abfrage nutzt die Natural Earth Provinzen und die OSM-Punkte der Fremdtabelle planet_osm_point und ermittelt, wie viele Punkte mit amenity = 'cafe' sich in der Provinz mit Namen 'Bucharest' befinden.

```
SELECT count(*)
  FROM
  ne_10m_admin_1_states_provinces_shp p,
  osm_fdw.planet_osm_point o
  WHERE
    ST_Distance(o.way, p.the_geom) = 0
    AND amenity = 'cafe'
    AND p.name = 'Bucharest';

    count
    ---------
    174
```

Zugriff auf einfache Textdateien

Dieser Zugriff kann auf unterschiedlichen Wegen erfolgen. Eine Möglichkeit ist über File-FDW (file_fdw).

Informationen zur Installation finden Sie unter https://www.postgresql.org/docs/curren/file-fdw.html

Der Zugriff soll anhand der OSGeoLive Passwort-Datei (/home/user/Desktop/passwort.txt) veranschaulicht werden.

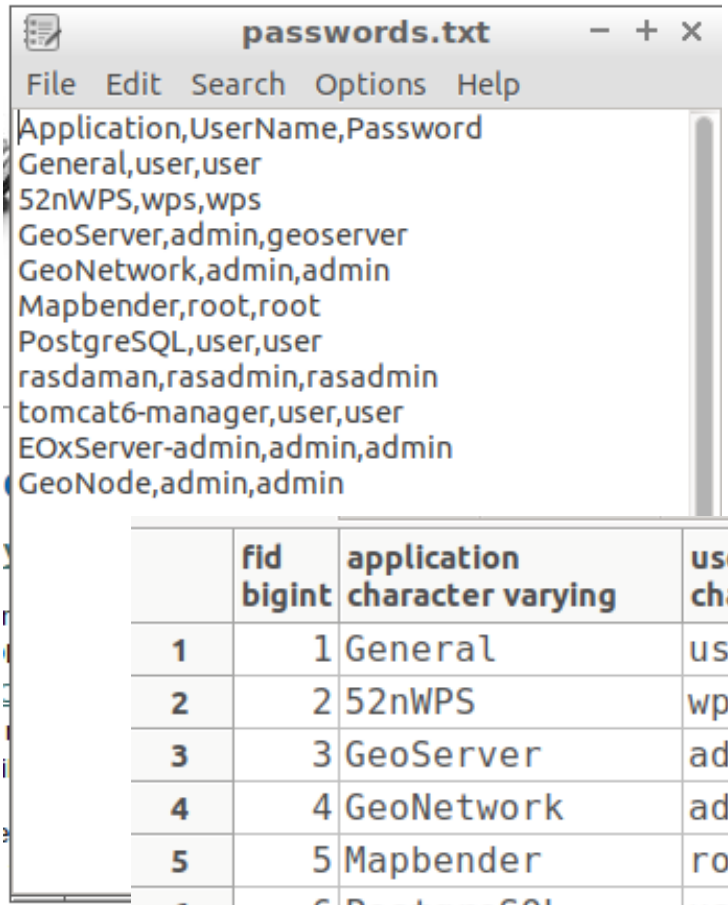
```
CREATE EXTENSION file_fdw;


CREATE SERVER fdw_server_file
FOREIGN DATA WRAPPER file_fdw;


CREATE FOREIGN TABLE passwords (
```

```
  application varchar,
  username varchar,
  password varchar
) SERVER fdw_server_file
OPTIONS (
  filename '/home/user/Desktop/passwords.txt', format
  'csv',
  header 'true'
);

SELECT * FROM passwords;
```



	fid bigint	application character varying	username character varying	password character varying
1	1	General	user	user
2	2	52nWPS	wps	wps
3	3	GeoServer	admin	geoserver
4	4	GeoNetwork	admin	admin
5	5	Mapbender	root	root
6	6	PostgreSQL	user	user
7	7	rasdaman	rasadmin	rasadmin
8	8	tomcat6-manager	user	user
9	9	EOxServer-admin	admin	admin
10	10	GeoNode	admin	admin

OGR-Foreign Data Wrapper (ogr_fdw)

Der Zugriff auf zahlreiche Vektor-Geodatenformate kann über den FDW ogr_fdw erfolgen, der von Paul Ramsey (PostGIS Chair) entwickelt wurde.

Über ogr_fdw werden zahlreiche Formate wie Geopackage, OGC WFS, OSM, ESRI Shape, KML unterstützt. Es besteht sogar schon Unterstützung für WFS 3.0.

Installation

ogr_fdw kann aus den Quellen heraus installiert werden. Der Code liegt auf GitHub.

https://github.com/pramsey/pgsql-ogr-fdw

Ubuntu/Debian-Pakete für ogr_fdw

Für Debian und Ubuntu liegen Pakete vor, die eine Installation sehr einfach machen.

https://packages.ubuntu.com/source/bionic/pgsql-ogr-fdw

https://packages.debian.org/sid/postgresql-10-ogr-fdw

sudo apt-get install postgresql-10-ogr-fdw

Anzeige der verfügbaren Formate

ogr_fdw kann unterschiedlich viele Formate unterstützen. Eine Liste der unterstützten Formate erhalten Sie über (Ausgabe im Terminal):

```
/usr/lib/postgresql/10/bin/ogr_fdw_info -f
Supported Formats:
-> "OGR_GRASS" (readonly)
-> "PCIDSK" (read/write)
-> "netCDF" (read/write)
```

```
-> "JP2OpenJPEG" (readonly)
-> "PDF" (read/write)
-> "MBTiles" (read/write)
-> "EEDA" (readonly)
-> "ESRI Shapefile" (read/write)
-> "MapInfo File" (read/write)
-> "UK .NTF" (readonly)
-> "OGR_SDTS" (readonly)
-> "S57" (read/write)
-> "DGN" (read/write)
-> "OGR_VRT" (readonly)
-> "REC" (readonly)
-> "Memory" (read/write)
-> "BNA" (read/write)
-> "CSV" (read/write)
-> "NAS" (readonly)
-> "GML" (read/write)
-> "GPX" (read/write)
-> "LIBKML" (read/write)
-> "KML" (read/write)
-> "GeoJSON" (read/write)
...
```

Aufbau einer Verbindung zu einer ESRI Shape-Datei

Über ogr_fdw_info mit der Option -s kann eine Liste der Datenquellen (Layers) – hier in einem Verzeichnis - ausgegeben werden.

```
cd /usr/lib/postgresql/10/bin/
./ogr_fdw_info -s /home/user/data/natural_earth2/

        Layers:
        ne_10m_geography_marine_polys
```

```
        ne_10m_geography_regions_points
        ne_10m_urban_areas
        ne_10m_populated_places
        ne_10m_admin_0_countries
        ne_10m_geography_regions_polys
        ne_10m_admin_1_states_provinces_shp
        ne_10m_geography_regions_elevation_points
        ne_10m_lakes
        ne_10m_ocean
        ne_10m_rivers_lake_centerlines
        ne_10m_land
```

Für den Zugriff auf Shapes wird lediglich ein Foreign Server und eine Foreign Table benötigt. User Mapping ist nicht notwendig.

Der Foreign Server verweist auf das Verzeichnis, in dem sich die Shapedateien befinden.

```
CREATE SERVER myserver
  FOREIGN DATA WRAPPER ogr_fdw
  OPTIONS (
    datasource '/home/user/data/natural_earth2/',
    format 'ESRI Shapefile' );
```

Die Fremdtabelle greift dann auf eine Shape-Datei zu.

```
CREATE FOREIGN TABLE ne_10m_populated_places (
  fid bigint,
  geom Geometry(Point,4326),
  scalerank integer,
  natscale integer,
  labelrank integer,
  featurecla varchar,
```

```
  name varchar,
…...
  pop2050 real,
  cityalt varchar
) SERVER myserver
OPTIONS (layer 'ne_10m_populated_places');
```

Beim Zugriff kann es zu Problemen mit der Zeichenkodierung (Encoding) kommen.

```
'utf-8' codec can't decode byte 0xed in position 1:
invalid continuation byte
```

ogr_fdw unterstützt Konfigurationsoptionen, über die beispielsweise das Encoding der Datenquelle angegeben werden kann.

```
CREATE SERVER myserver_latin1
  FOREIGN DATA WRAPPER ogr_fdw
  OPTIONS (
    datasource '/home/user/data/natural_earth2/',
    format 'ESRI Shapefile',
    config_options 'SHAPE_ENCODING=LATIN1'
);
```

OGR-FDW und OSM-Daten

ogr_fdw unterstützt auch OpenStreetMap-Daten als Datenquelle.

```
/usr/lib/postgresql/10/bin/ogr_fdw_info -s
/home/user/feature_city.osm
Layers:
  points
  lines
  multilinestrings
  multipolygons
```

```
  other_relations
```

ogr_fdw_info zur Ausgabe der SQL-Definition

ogr_fdw_info macht das Leben sogar noch einfacher und schreibt die SQL-Statements für den Aufbau der Foreign Table für Sie.

Dies soll hier am Beispiel von OSM als Datenquelle veranschaulicht werden. Es soll der Layer points eingebunden werden.

Extrahieren Sie dazu die OSM-Daten /home/user/data/osm/feature_city.osm.bz2 nach /*home*/user.

Der folgende Befehl auf der Kommandozeile gibt das benötigte SQL aus.

```
 /usr/lib/postgresql/10/bin/ogr_fdw_info -s
/home/user/feature_city.osm -l points


CREATE SERVER myserver_osm
  FOREIGN DATA WRAPPER ogr_fdw
  OPTIONS (
    datasource '/home/user/feature_city.osm',
    format 'OSM' );


CREATE FOREIGN TABLE points (
  fid bigint,
  geom Geometry(Point,4326),
  osm_id varchar, name varchar,
  barrier varchar,  highway varchar,
  ref varchar,  address varchar,
  is_in varchar,  place varchar,
  man_made varchar,
  other_tags varchar
) SERVER myserver_osm
```

```
OPTIONS (layer 'points');
```

OGR FDW und WFS

ogr_fdw kann auch auf einzelne Feature-Types eines WFS zugreifen. Der Feature-Type kann so einfach als Datenquelle angesprochen werden.

Dies ermöglicht beispielsweise einen bequemen Zugriff auf die Flurstücke von Rheinland-Pfalz, die über einen WFS bereitgestellt werden. Die Foreign Table kann dann in Mapbender im SearchRouter für eine Suche verwendet werden.

Das Beispiel zeigt den Zugriff auf einen QGIS Server WFS. Erzeugen Sie dafür über das QGIS Server Quickstart einen WFS, der die Länder der NaturalEarth-Daten einbindet (siehe http://live.osgeo.org/en/quickstart/qgis_mapserver_quickstart.html).

```
CREATE SERVER myserver_wfs_qgis_server
  FOREIGN DATA WRAPPER ogr_fdw
  OPTIONS (
    datasource
'WFS:http://localhost/cgi-bin/qgis_mapserv.fcgi?map=/home
/user/world.qgz',
    format 'WFS',
    config_options 'CPL_DEBUG=ON');
```

```
CREATE SCHEMA fdw_wfs;


IMPORT FOREIGN SCHEMA ogr_all
FROM server myserver_wfs_qgis_server
INTO fdw_wfs;
```

Über die Option CPL_DEBUG=ON kann das Debugging aktiviert werden, so dass ersichtlich ist, welche Anfragen geschickt werden und ggf. Fehlermeldungen besser interpretiert werden können. Zur Ausgabe muss client_min_messages auf debug2 gesetzt werden.

```
Show client_min_messages;
SET client_min_messages=debug2;
```

Die Anfrage liefert die Länder des WFS featureTypes zurück:

```
Select * from fdw_wfs.ne_10m_admin_0_countries;
```

Fazit

Foreign Data Wrapper bringen mehr Flexibilität durch den einfachen Zugriff auf verschiedenste Datenquellen. Die Möglichkeit, auch schreibend auf externe Quellen zuzugreifen, macht das Ganze noch attraktiver.

Präsentation FOSS4G 2019

Folien Präsentation Fun With Foreign Data Wrapper (FOSS4G 2019)

https://github.com/astroidex/presentations/raw/master/FOSS4G2019/FOSS4G_2019_PostgreSQL_FDW_Emde.pdf

Viideo zur Präsentation Fun with Forein Data Wrappers (FOSS4G 2019)

https://av.tib.eu/media/43423