**QGIS Topology**

* **Einleitung**
* **Voraussetzungen**
* **Das Vorgehen**
* **Die Simulation**
* **Neuer Ansatz**

**Einleitung**

Um Topologie in QGIS nutzbar zu machen, muss man zunächst einmal die Datenbankerweiterung PostGIS in der PostgreSQL Datenbank installieren. Hier erstellt man nun eine neue Datenbank und installiert die Erweiterung postgis sowie postgis\_topology. Um eine neue Topologie anzulegen erstellt man anhand der von postgis\_topology bereitgestellten Funktionen eine Topologie:

SELECT topology.CreateTopology('conf',26986);

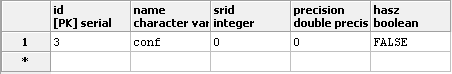
Als Übergabeparameter geben wir hier den Namen unserer neuen Topologie sowie die SRID des gewünschten Koordinatensystems an.

Dadurch entsteht ein neues Schema mit dem entsprechenden Namen. In diesem Schema befindet sich:

1. edge\_data: Alle Edges mit Start- und Endnode sowie dessen Orientierung
2. face
3. node
4. relation

Sobald man eine neue Topologie erstellt, wird diese in der Datenbank unter dem Schema topology, in der Database angelegt.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id  [PK]serial | name  character varying | srid  integer | precision  double precision | hasz  boolean |
| 3 | conf | 0 | 0 | FALSE |

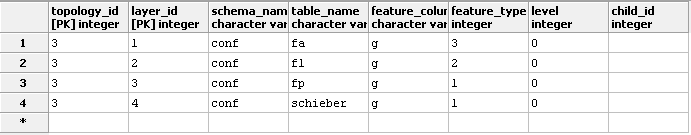


Um die Topologie für Betriebsmittel nutzbar zu machen müssen den Betriebsmitteln eine neue „TopoGeom“ Spalte hinzugefügt werden. Dies geschieht anhand von

**AddTopoGeometryColumn**(varchar topology\_name, varchar schema\_name, varchar table\_name, varchar column\_name, varchar feature\_type);

Die Layer, die eine TopoGeom Spalte besitzen, werden in der Datenbank topology.layer registriert:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| topology\_id  [PK]integer | layer\_id  [PK]integer | schema\_name  character varying | feature\_column  character varying | feature\_tye  integer | level  integer | child\_id  integer |
| 3 | 1 | conf | fa | g | 3 | 0 |
| 3 | 2 | conf | fl | g | 2 | 0 |
| 3 | 3 | conf | fp | g | 1 | 0 |
| 3 | 4 | conf | schieber | g | 1 | 0 |

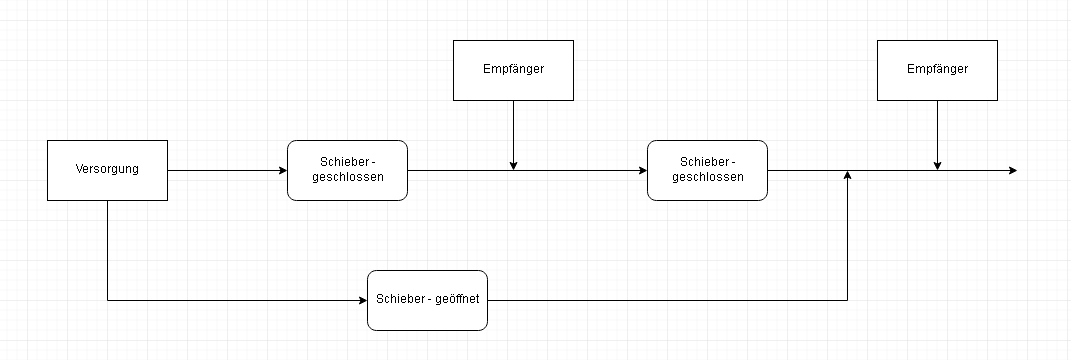


Nun kann ich QGIS öffnen und gewohnt neue Betriebsmittel erfassen. Diese werden dann automatisch in der TopoGeom und als Nodes, Edges etc. hinzugefügt.

Da QGIS und PostGIS keine weiteren Tools zur Verwendung der Topologie anbietet. Muss man eigene Funktionen schreiben um z.B. die Netzwerkverfolgung zu ermöglichen. Hier bietet es sich an SQL Funktionen zu formulieren, die über QGIS Plug-Ins genutzt werden können.

**Simulation der Schieber**

Oftmals kommt es in einer Gasfachschale vor, dass eine Simulation der Gasverteilung durchgeführt werden muss. Dadurch, dass sich Objekte öffnen und schließen, verändert sich auch der Gas Fluss. Natürlich muss stets sichergestellt werden, dass die Versorgung dadurch nicht an ungewollten stellen und an gewollten stellen unterbrochen ist. Beispielsweise bei Baumaßnahmen an dem Leitungsnetzwerk.



**Die Vorrausetzungen für solch eine Simulation sind:**

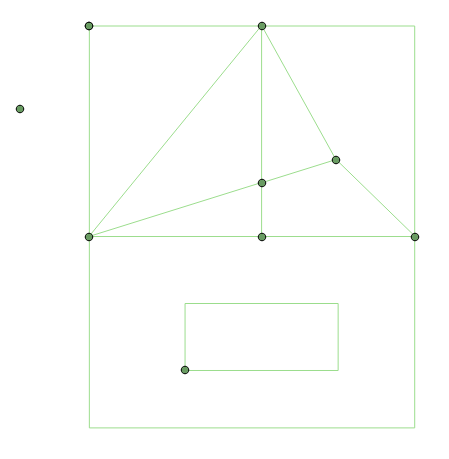
1. Netzwerkverfolgung: Es muss möglich sein alle verbundenen Nodes/Edges von einem Startpunkt aus zu erfassen.
2. Eingrenzung: Es muss möglich sein das Gebiet einzugrenzen, damit nicht ungewollt umliegende Netzwerke die Simulation beeinflussen.
3. Betriebsmittel: Es muss möglich sein von jedem dieser Nodes/Edges die zugehörigen Betriebsmittel und seine Attribute abzufragen.

**Das Vorgehen**

1. Eingrenzen des Bereichs
2. Ermitteln aller Topologie Objekte
3. Ermitteln aller Empfänger
4. Netzwerkverfolgung mit Startpunkt Versorgung und Ausgrenzung geschlossener Betriebsmittel
5. Ermitteln der neuen Anzahl an Empfänger und Vergleich

**Netzwerkverfolgung**

Ziel ist es eine räumliche Netzwerkverfolgung durch eine kostengünstigere topologische/Graphen gesteuerte Netzwerkverfolgung abzulösen. Hierfür wird eine SQL Funktion in dem Schema topology angelegt. Getestet wird dies hier anhand von:



**Rekursive Abfrage mit dem Startpunkt 1**

Kostenpunkt 124.12 .. 134.78

WITH RECURSIVE path AS (

SELECT edge\_id, start\_node, end\_node

FROM conf.edge\_data

WHERE start\_node = 1 OR end\_node = 1

UNION

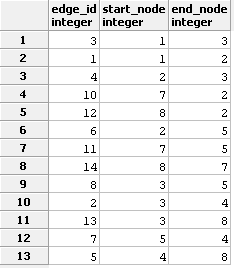
SELECT e.edge\_id, e.start\_node, e.end\_node

FROM conf.edge\_data e, path s

WHERE s.start\_node = e.start\_node OR s.start\_node = e.end\_node OR s.end\_node = e.end\_node OR s.end\_node = e.start\_node )

SELECT \* FROM path;

Diese Funktion ermöglicht es uns alle verbundenen Edges von einer Startedge heraus zu verfolgen.



|  |  |  |  |
| --- | --- | --- | --- |
|  | edge\_id  integer | start\_node  integer | end\_node  integer |
| 1 | 3 | 1 | 3 |
| 2 | 1 | 1 | 2 |
| 3 | 4 | 2 | 3 |
| 4 | 10 | 7 | 2 |
| 5 | 12 | 8 | 2 |
| 6 | 6 | 2 | 5 |
| 7 | 11 | 7 | 5 |
| 8 | 14 | 8 | 7 |
| 9 | 8 | 3 | 5 |
| 10 | 2 | 3 | 4 |
| 11 | 13 | 3 | 8 |
| 12 | 7 | 5 | 4 |
| 13 | 5 | 4 | 8 |

**Eingrenzung**

Die Eingrenzung kann durch leichte Selektion in QGIS umgesetzt werden. Dies wird später in die Grundfunktion „Gasnetzwerksimulation“ als Hilfsmittel mit eingebaut.

**Betriebsmittel**

Sobald wir nun einen Node, eine Edge oder Anderes haben, wollen wir alle zugehörigen Betriebsmittel ermitteln. Um dies zu erreichen, müssen wir zunächst den Feature\_type unseres Topologie Objektes suchen. Dadurch, dass die Betriebsmittellayer in topology.layer registriert sind kann man sich per Datenbankabfrage alle möglichen Betriebsmittellayer des gesuchten Typs zurückgeben lassen:

-- Function to grab all tables from a specific topology with type; represents (1:point,2:line,...)

CREATE OR REPLACE FUNCTION topology.getTypeLayer(topology text,type int)

RETURNS TABLE

(table\_name character varying,

feature\_column character varying,

feature\_type integer)

AS

$func$

DECLARE

SQL text;

BEGIN

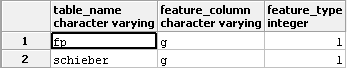
SQL = 'SELECT table\_name,feature\_column,feature\_type FROM topology.layer WHERE schema\_name = ''' || topology || ''' and feature\_type = ' || type;

RETURN QUERY Execute SQL;

END

$func$ LANGUAGE plpgsql;

SELECT \* FROM topology.getTypeLayer('conf',1)



|  |  |  |
| --- | --- | --- |
| table\_name  Character varying | feature\_column  character varying | feature\_type  integer |
| fp | g | 1 |
| schieber | g | 1 |

Nun kennen wir alle Layer auf denen wir nach möglichen Betriebsmitteln suchen müssen. Dies machen wir anhand von zwei weiteren Funktionen. Wobei die erste eine temporäre Tabelle erstellt und die zweite auf diese Zugreift:

-- Function to grab result

CREATE OR REPLACE FUNCTION topology.getBetriebsmittelFromTypeLayer(topology\_name character varying,table\_name character varying,feature\_column character varying,feature\_type integer,element\_id integer)

RETURNS VOID

AS

$func$

DECLARE

SQL text;

BEGIN

SQL = 'INSERT INTO topology.result (table\_name,feature\_column,feature\_type,feature\_id,record\_id) VALUES(''' || table\_name || ''',''' || feature\_column || ''',''' || feature\_type || ''',''' || element\_id || ''',topology.getBetriebsmittel(''' || topology\_name || '.' || table\_name || ''',''' || feature\_column || ''',''' || element\_id || ''',''' || feature\_type || '''));';

Execute SQL;

END

$func$ LANGUAGE plpgsql;

SELECT topology.getBetriebsmittelFromTypeLayer('conf','fp','g',1,2);

-- Loop trough tablename and column to get Betriebsmittel

CREATE OR REPLACE FUNCTION topology.getAllBetriebsmittel(topology\_name character varying,element\_type integer,element\_id integer)

RETURNS TABLE (

id bigint,

table\_name character varying,

feature\_column character varying,

feature\_type integer,

feature\_id integer,

record\_id integer

)

AS

$func$

DECLARE

my\_cursor CURSOR FOR SELECT \* FROM topology.getTypeLayer(topology\_name,element\_type);

my\_record RECORD;

BEGIN

OPEN my\_cursor;

TRUNCATE TABLE topology.result;

LOOP

FETCH my\_cursor INTO my\_record;

EXIT WHEN NOT FOUND;

PERFORM topology.getBetriebsmittelFromTypeLayer(topology\_name,my\_record.table\_name,my\_record.feature\_column,my\_record.feature\_type,element\_id);

END LOOP;

CLOSE my\_cursor;

RETURN QUERY

SELECT \* FROM topology.result;

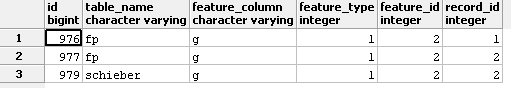
END

$func$ LANGUAGE plpgsql;

Anhand von:

SELECT \* FROM topology.getAllBetriebsmittel('conf',1,2),

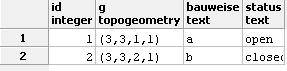
Ist es nun möglich mit dem Namen der Topologie, dem Feature Type und der ID des Topologie Objektes alle zugehörigen Betriebsmittel zu erfassen:



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Id  bigint | table\_name  character varying | feature\_column  character varying | feature\_type  integer | feature\_id  integer | record\_id  integer |
| 976 | fp | g | 1 | 2 | 1 |
| 977 | fp | g | 1 | 2 | 2 |
| 979 | schieber | g | 1 | 2 | 2 |

Anhand des table\_name und der record\_id lassen sich die Attribute aus der Betriebsmitteltabelle abfragen:

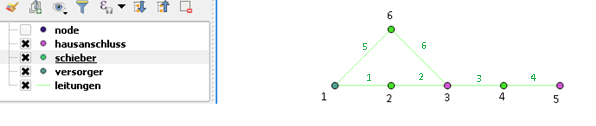
SELECT \* FROM conf.fp where id in (SELECT record\_id FROM topology.getAllBetriebsmittel('conf',1,2) where table\_name = 'fp');



|  |  |  |  |
| --- | --- | --- | --- |
| id  integer | g  topogeometry | bauweise  text | status  text |
| 1 | (3,3,1,1) | a | open |
| 2 | (3,3,2,1) | b | close |

**Die Simulation**

Zunächst einmal wurde ein Netzwerk mit 6 Nodes erstellt. Anhand dieses Netzwerkes wird nun die Simulation durchgeführt.



**Datenmodell**

**Hausanschluss:** Stellt den Anschluss zu einem Gebäude da. Falls dieser nicht Versorgt wird, wird natürlich auch das Gebäude nicht versorgt. Dies ist somit das entscheidende Kriterium der Simulation.

**Schieber:** Schieber haben zwei Status. Sie können offen oder zu sein und somit den Gas Fluss zulassen oder blockieren. Somit ist dies unser Ausschlusskriterium.

**Versorger:** Dies ist lediglich die abstrakte Darstellung einer Gasquelle.

**Leitungen:** Das Gas fließt hier in beide Richtungen. Wir sprechen also von einem nicht gerichteten Graphen.

In diesem Netzwerk sind Schieber 2 und 4 geschlossen. Wenn wir nun also die Simulation durchführen, sollte auffallen, dass der Hausanschluss 5 nicht versorgt wird.

Diesen Ausschluss machen wir anhand von:

WITH RECURSIVE path AS (

SELECT edge\_id, start\_node, end\_node

FROM simulation.edge\_data

WHERE start\_node = 1 OR end\_node = 1

UNION

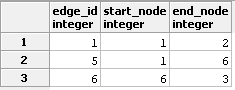
SELECT e.edge\_id, e.start\_node, e.end\_node

FROM simulation.edge\_data e, path s

WHERE ((SELECT status FROM simulation.schieber where id in (SELECT record\_id FROM topology.getAllBetriebsmittel('simulation',1,e.start\_node) where table\_name = 'schieber')) = 'offen' OR (SELECT status FROM simulation.schieber where id in (SELECT record\_id FROM topology.getAllBetriebsmittel('simulation',1,e.end\_node) where table\_name = 'schieber')) = 'offen') AND ( s.start\_node = e.start\_node OR s.start\_node = e.end\_node OR s.end\_node = e.end\_node OR s.end\_node = e.start\_node ) )

SELECT \* FROM path;

Und liefert das Ergebnis:



|  |  |  |
| --- | --- | --- |
| edge\_id  integer | start\_node  integer | end\_node  integer |
| 1 | 1 | 2 |
| 5 | 1 | 6 |
| 6 | 6 | 3 |

Wenn wir uns nun alle Betriebsmittelinformationen der Nodes holen bekommen wir:

SELECT \* FROM topology.getAllBetriebsmittel('simulation',1,1)

UNION

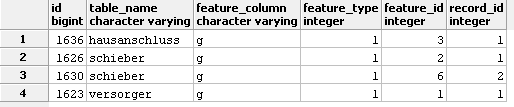
SELECT \* FROM topology.getAllBetriebsmittel('simulation',1,2)

UNION

SELECT \* FROM topology.getAllBetriebsmittel('simulation',1,6)

UNION

SELECT \* FROM topology.getAllBetriebsmittel('simulation',1,3)



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id  bigint | table\_name  character varying | feature\_column  character varying | feature\_type  integer | feature\_id  integer | record\_id  integer |
| 1636 | hausanschluss | g | 1 | 3 | 1 |
| 1626 | schieber | g | 1 | 2 | 1 |
| 1630 | schieber | g | 1 | 6 | 2 |
| 1623 | versorger | g | 1 | 1 | 1 |

**Fazit:**

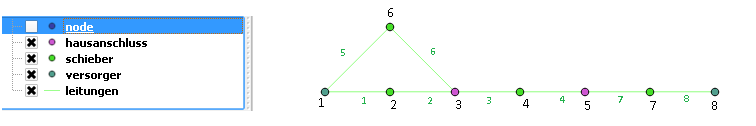
Die Simulation ist erfolgreich:

* Feststellung welche Hausanschlüsse mit Gas versorgt werden: Hausanschluss 5 wird nicht versorgt.

Die Simulation ist nicht erfolgreich, da:

* Theoretisch wird die Edge 2 und 3 auch mit Gas versorgt
* Theoretisch wird der Schieber 4 auch mit Gas versorgt
* Falls eine andere Quelle existiert wird diese nicht berücksichtigt -> Es muss eine Netzwerkverfolgung von jeder Quelle aus gestartet werden

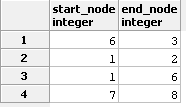
Das Modell wird nun um einen weiteren offenen Schieber und eine Quelle erweitert. Bei der erneuten Simulation fällt nun auf, dass Hausanschluss 5 eigentlich versorgt sein sollte. Doch dies ist nicht der Fall.



**Neuer Ansatz**

1. Eingrenzen des Bereichs
2. Ermitteln aller Topologie Objekte
3. Ermitteln aller Empfänger
4. Ermitteln aller Quellen
5. Netzwerkverfolgung für jede Quelle/mit Startpunkt aller Quellen und Ausgrenzung geschlossener Betriebsmittel
6. Ermitteln der neuen Anzahl an Empfänger und Vergleich

Nun wird der Schieber 7 geschlossen und das Ergebnis erneut verglichen.



|  |  |
| --- | --- |
| start\_node  integer | end\_node  integer |
| 6 | 3 |
| 1 | 2 |
| 1 | 6 |
| 7 | 7 |

WITH RECURSIVE path AS (

SELECT edge\_id, start\_node, end\_node

FROM simulation.edge\_data

WHERE start\_node = 1 OR end\_node = 1

UNION

SELECT e.edge\_id, e.start\_node, e.end\_node

FROM simulation.edge\_data e, path s

WHERE ((SELECT status FROM simulation.schieber where id in (SELECT record\_id FROM topology.getAllBetriebsmittel('simulation',1,e.start\_node) where table\_name = 'schieber')) = 'offen' OR (SELECT status FROM simulation.schieber where id in (SELECT record\_id FROM topology.getAllBetriebsmittel('simulation',1,e.end\_node) where table\_name = 'schieber')) = 'offen') AND ( s.start\_node = e.start\_node OR s.start\_node = e.end\_node OR s.end\_node = e.end\_node OR s.end\_node = e.start\_node ) )

,path2 AS (

SELECT edge\_id, start\_node, end\_node

FROM simulation.edge\_data

WHERE start\_node = 8 OR end\_node = 8

UNION

SELECT e.edge\_id, e.start\_node, e.end\_node

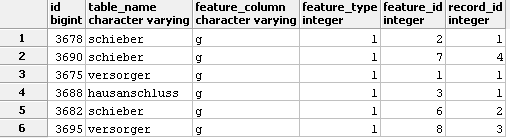
FROM simulation.edge\_data e, path2 s

WHERE ((SELECT status FROM simulation.schieber where id in (SELECT record\_id FROM topology.getAllBetriebsmittel('simulation',1,e.start\_node) where table\_name = 'schieber')) = 'offen' OR (SELECT status FROM simulation.schieber where id in (SELECT record\_id FROM topology.getAllBetriebsmittel('simulation',1,e.end\_node) where table\_name = 'schieber')) = 'offen') AND ( s.start\_node = e.start\_node OR s.start\_node = e.end\_node OR s.end\_node = e.end\_node OR s.end\_node = e.start\_node ) )

SELECT DISTINCT start\_node,end\_node FROM path UNION SELECT DISTINCT start\_node,end\_node FROM path2;

Wenn wir wieder die dazugehörigen Betriebsmittel abfragen erfahren wir, dass diese Simulation nun auch den Fall von mehreren Quellen abdeckt.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id  bigint | table\_name  character varying | feature\_column  character varying | feature\_type  integer | feature\_id  integer | record\_id  integer |
| 3678 | schieber | g | 1 | 2 | 1 |
| 3690 | schieber | g | 1 | 7 | 4 |
| 3675 | versorger | g | 1 | 1 | 1 |
| 3688 | hausanschluss | g | 1 | 3 | 1 |
| 3682 | schieber | g | 1 | 6 | 2 |
| 3695 | versorger | g | 1 | 8 | 3 |



**Überschneidungsproblem**

In postgis gibt es ein Grundlegendes Problem was die Topologie angeht, denn bei Leitungen ist es wichtig, dass sie Geographisch übereinander liegen können ohne sich zu berühren. Postgis erlaubt dies jedoch nicht. Sobald sich zwei Edges schneiden, blockiert die Datenbank die Erstellung oder erstellt einen Node am Schnittpunkt.

Wenn man sich den Code dafür anguckt, sieht man folgendes:

--

-- Check if the edge crosses an existing node

--

FOR rec IN EXECUTE 'SELECT node\_id FROM '

|| quote\_ident(atopology) || '.node '

|| 'WHERE ST\_Crosses('

|| quote\_literal(aline::text) || '::geometry, geom'

|| ')'

LOOP

RAISE EXCEPTION 'Edge crosses node %', rec.node\_id;

END LOOP;

--

-- Check if the edge intersects an existing edge

-- on anything but endpoints

--

-- Following DE-9 Intersection Matrix represent

-- the only relation we accept.

--

-- F F 1

-- F \* \*

-- 1 \* 2

--

-- Example1: linestrings touching at one endpoint

-- FF1 F00 102

-- FF1 F\*\* 1\*2 <-- our match

--

-- Example2: linestrings touching at both endpoints

-- FF1 F0F 1F2

-- FF1 F\*\* 1\*2 <-- our match

--

FOR rec IN EXECUTE 'SELECT edge\_id, geom, ST\_Relate('

|| quote\_literal(aline::text)

|| '::geometry, geom, 2) as im'

|| ' FROM '

|| quote\_ident(atopology) || '.edge '

|| 'WHERE '

|| quote\_literal(aline::text) || '::geometry && geom'

LOOP

IF ST\_RelateMatch(rec.im, 'FF1F\*\*1\*2') THEN

CONTINUE; -- no interior intersection

END IF;

--

-- Check if this geometry crosses any node

--

FOR rec IN EXECUTE

'SELECT node\_id, ST\_Relate(geom, '

|| quote\_literal(acurve::text) || '::geometry, 2) as relate FROM '

|| quote\_ident(atopology)

|| '.node WHERE geom && '

|| quote\_literal(acurve::text)

|| '::geometry'

LOOP

IF ST\_RelateMatch(rec.relate, 'T\*\*\*\*\*\*\*\*') THEN

RAISE EXCEPTION 'SQL/MM Spatial exception - geometry crosses a node';

END IF;

END LOOP;

--

-- Check if this geometry has any interaction with any existing edge

--

FOR rec IN EXECUTE 'SELECT edge\_id, ST\_Relate(geom,'

|| quote\_literal(acurve::text)

|| '::geometry, 2) as im FROM '

|| quote\_ident(atopology)

|| '.edge\_data WHERE geom && '

|| quote\_literal(acurve::text) || '::geometry'

LOOP

--RAISE DEBUG 'IM=%',rec.im;

IF ST\_RelateMatch(rec.im, 'F\*\*\*\*\*\*\*\*') THEN

CONTINUE; -- no interior intersection

END IF;

IF ST\_RelateMatch(rec.im, '1FFF\*FFF2') THEN

RAISE EXCEPTION

'SQL/MM Spatial exception - coincident edge %', rec.edge\_id;

END IF;

-- NOT IN THE SPECS: geometry touches an edge

IF ST\_RelateMatch(rec.im, '1\*\*\*\*\*\*\*\*') THEN

RAISE EXCEPTION

'Spatial exception - geometry intersects edge %', rec.edge\_id;

END IF;

IF ST\_RelateMatch(rec.im, 'T\*\*\*\*\*\*\*\*') THEN

RAISE EXCEPTION

'SQL/MM Spatial exception - geometry crosses edge %', rec.edge\_id;

END IF;

END LOOP;

**Topologischer Hintergrund**

Die Überschneidung von zwei Linien des gleichen Feature Typs zählt als topologischer Fehler. Es ist lediglich valide, das die Features gemeinsame Start oder Endpunkte haben. Diese Regel wird beispielsweise benutzt, um Straßensegmente oder Kreuzungen darzustellen.

**(**[**http://desktop.arcgis.com/de/arcmap/10.3/manage-data/editing-topology/geodatabase-topology-rules-and-topology-error-fixes.htm**](http://desktop.arcgis.com/de/arcmap/10.3/manage-data/editing-topology/geodatabase-topology-rules-and-topology-error-fixes.htm)**)**

Anhand des bereits vorinstallierten Plug-Ins „Topologie Prüfung“ in QGIS lassen sich beispielsweise diese Fehler darstellen.

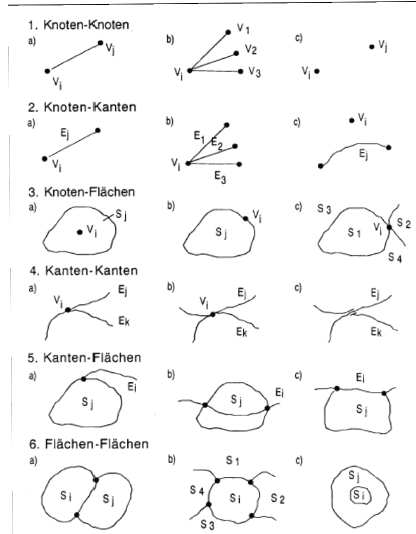
**Euler-Charakteristik**

Allgemein gilt für jeden planaren Graphen:

C = p – l + f = 2

Euler´sche Charakteristik = Anzahl Punkte – Anzahl Linien + Anzahl Flächen = 2 (<https://www.youtube.com/watch?v=-9OUyo8NFZg>)

Dies garantiert topologische Invarianz. Heißt also, dass die Nachbarschaftsverhältnisse stets erhalten bleiben. Dies kann so nicht garantiert werden, da wir eine „verbotene“ topologische Beziehung von Kante-Kante erstellen. Also einen **nicht** planaren Graphen erhalten.

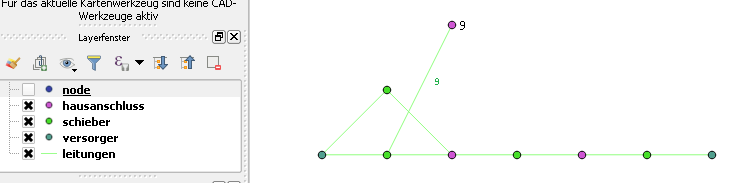


Somit können natürlich auch nur Algorithmen für nicht planare Graphen angewendet werden.

**Neues Vorgehen**

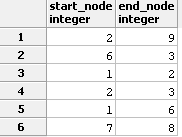
Nun wird unser Netzwerk erneut erweitert. Diese Mal wird von Schieber 2 ein weiterer Hausanschluss erzeugt. Besonders hierbei; Die Funktion zur Erstellung einer Edge wird modifiziert, sodass eine Überkreuzung möglich ist. Dann wird die Simulation erneut ausgeführt.

**Neue Funktion:** topology.st\_addedgemodface\_nosplit



**Neue Simulation**

Da der Schieber 2 geschlossen ist, wird Hausanschluss 9 erfolgreich nicht mit Gas versorgt. Nun wird Schieber 2 geöffnet. Das Ergebnis zeigt, dass der topologische Fehler keinen Einfluss auf die Simulation hat.



Ein Überprüfen anhand des Plug-Ins zeigt, dass die Edge 8 und 9 ein topologischer Fehler ist. Beides sind „Dangles“, also Edgdes dessen Node ein Endpunkt des Graphen darstellt. Dies wird jedoch z.B. bei Straßen erlaubt, wenn es sich um Sackgassen handelt. Außerdem bleibt der Graph trotzdem planar.

<https://www.google.de/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&ved=0ahUKEwiGm5XOreDVAhXREVAKHf70AiMQFgg5MAI&url=https%3A%2F%2Fi11www.iti.kit.edu%2F_media%2Finformation%2Fscripts%2Fscripte%2Falgo-plan.pdf&usg=AFQjCNGCtEkG0e6tayJx-10IKwhYg-YsUQ>

**Fazit**

Es muss also geprüft werden, ob wichtige Algorithmen für planare Graphen innerhalb der Leitungen benötigt werden. Weiterhin; ob es bereits bestehende Regeln zu einem Gasnetzwerk bezüglich der „Dangles“ gibt.

**Mengentest**

In Zusammenarbeit mit München wird ein Gasnetzwerk bereitgestellt. Im Gegensatz zu der Simulation handelt es sich um 674507 Nodes und 636456 Edges. Nun soll der Datenbestand in das Postgis-Topologiemodell umgewandelt werden. Vorher wurde hierfür SmallworldGIS benutzt und somit befinden sich die Node und Edge Daten auch dabei. Diese können also einfach durch kopieren übernommen werden.

SELECT DropTopology('gas\_topo');

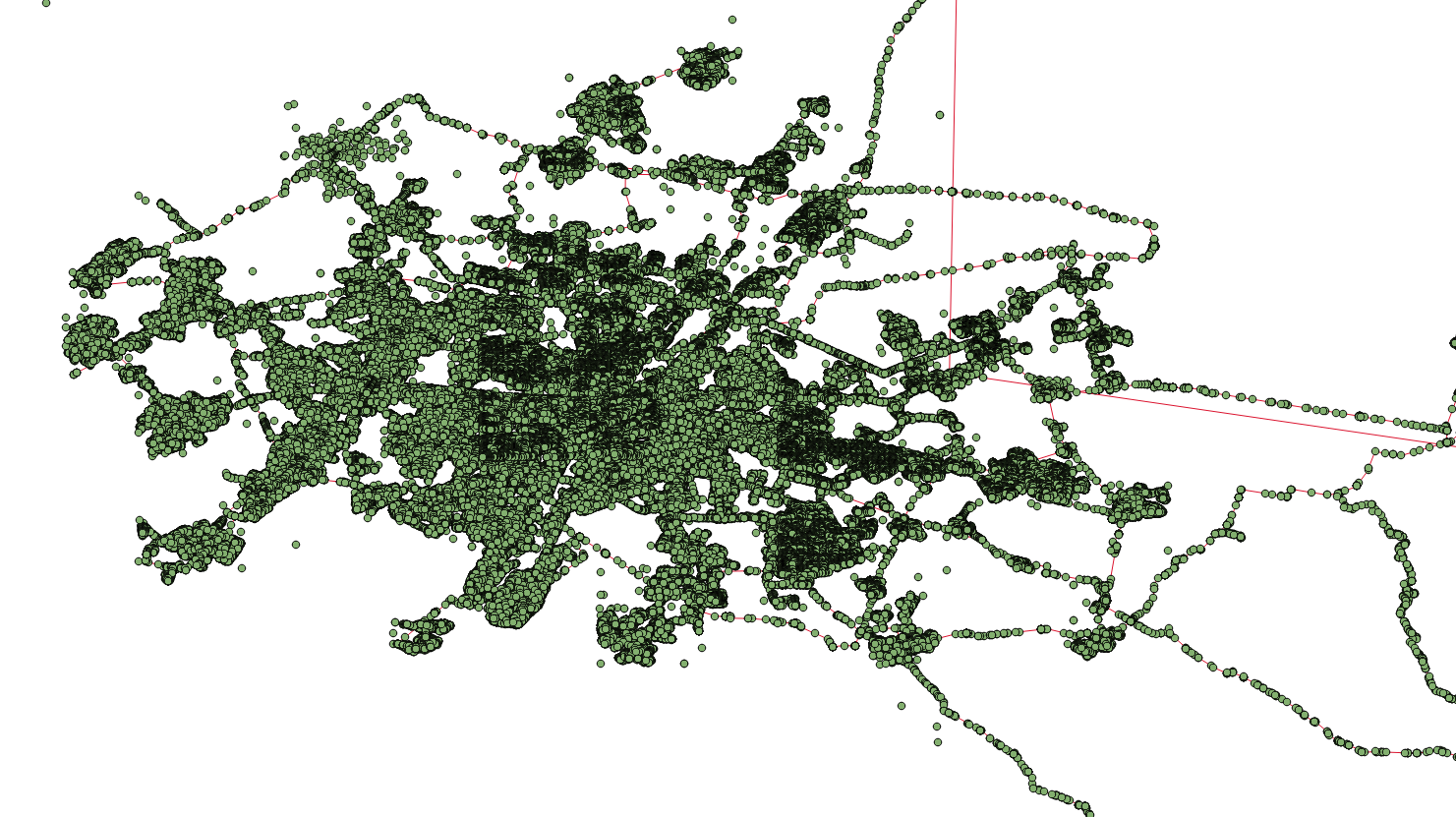
SELECT CreateTopology('gas\_topo',31468);

INSERT INTO gas\_topo.node SELECT node\_id[3], 0, ST\_SETSRID(ST\_MakePoint(coord[1] \* 0.001, coord[2] \* 0.001),31468) from ga.sw\_gis\_node;

INSERT INTO gas\_topo.edge\_data SELECT link\_id[3], first\_node\_id[3], last\_node\_id[3], link\_id[3],link\_id[3],link\_id[3],link\_id[3],0,0,

ST\_SETSRID(ST\_MakeLine(ST\_MakePoint(coord\_1[1] \* 0.001, coord\_1[2] \* 0.001), ST\_MakePoint(coord\_2[1] \* 0.001, coord\_2[2] \* 0.001)),31468) from ga.sw\_gis\_link

Nun wurde bereits das komplette Topologienetzwerk bereitgestellt:



Jetzt müssen aber noch die Betriebsmittel eingefügt werden.

**SmallworldGIS Topologie:**

Node -> point -> definition -> id

Über die node\_id gelangt man an die zugehörigen Points. Diese enthalten einen rwo\_code und eine rwo\_id. Anhand des rwo\_code können wir in der Tabelle „sw\_gis\_rwo\_definition“ den Tabellennamen der Betriebsmittel herausfinden. Zusammen mit der rwo\_id können wir nun das Betriebsmittel aus der Betriebsmitteltabelle identifizieren.

Postgis Topologie:

Node-> layer -> id

Über den feature\_type schauen wir in der layer Tabelle, welche Tabellen alles betroffen sind. Zusammen mit der node\_id und den Tabellennamen kriegen wir anhand einer einfachen Funktion(GetTopoElements) die Betriebsmittel.

Dies bedeutet also, dass wir zunächst einmal den Betriebsmittellayer komplett in die neue Topologie übernehmen können und ihn dann lediglich als Layer registrieren:

CREATE TABLE gas\_topo.absperrarmatur AS

TABLE ga.g\_absperrarmatur;

SELECT AddTopoGeometryColumn('gas\_topo', 'gas\_topo', 'absperrarmatur', 'g', 'POINT');

Nun befüllen wir die TopoGeom Spalte (g) mit der entsprechenden Umwandlung:

UPDATE gas\_topo.absperrarmatur SET g = toTopoGeom(position\_in\_betrieb,'gas\_topo',1,0);

Bei ausführen der Funktion getAllBetriebsmittel fällt nun auf, dass dies Aufgrund einer Fehlenden ID Spalte in dem Betriebsmittellayer fehlschlägt.

Das Betriebsmittel wird nun also mit einer autoinkrementellen id-spalte versehen:

ALTER TABLE gas\_topo.absperrarmatur ADD COLUMN id INTEGER;

CREATE SEQUENCE gas\_topo.test\_id\_seq OWNED by gas\_topo.absperrarmatur.id;

ALTER TABLE gas\_topo.absperrarmatur ALTER COLUMN id SET DEFAULT nextval('gas\_topo.test\_id\_seq');

UPDATE gas\_topo.absperrarmatur SET id = nextval('gas\_topo.test\_id\_seq');

Nun lässt sich die Funktion wie gewohnt nutzen. Und es kann eine Simulationsabfrage erstellt werden.

SELECT \* FROM topology.getAllBetriebsmittel('gas\_topo',1,65258240)



**Konvertierung 2**

In dem Smallworld Modell gibt es die Möglichkeit mehrere Betriebsmittel auf den gleichen Punkt zu legen, jedoch keine Verbindung herzustellen. Deshalb müssen die TopoGeometrien erneut per Hand erstellt werden:

SELECT (SELECT node\_id[3] FROM ga.sw\_gis\_point where rwo\_id[3] = m.rwo\_id[3] LIMIT 1) FROM ga.g\_absperrarmatur m LIMIT 100;

UPDATE gas\_topo.absperrarmatur\_test t SET g =

CreateTopoGeom('gas\_topo' -- Topo name

,1 -- Type

,3 -- Layer

,CAST('{{' || (SELECT node\_id[3] FROM ga.sw\_gis\_point where rwo\_id[3] = t.rwo\_id[3] LIMIT 1) || ',1}}' AS integer[]));

SELECT (SELECT link\_id[3] FROM ga.sw\_gis\_chain\_link where chain\_id[3] = (SELECT chain\_id[3] FROM ga.sw\_gis\_chain WHERE rwo\_id[3] = m.rwo\_id[3] LIMIT 1) LIMIT 1) FROM ga.g\_versorgungsltg\_abschnitt m LIMIT 100;

UPDATE gas\_topo.absperrarmatur\_test t SET g =

CreateTopoGeom('gas\_topo' -- Topo name

,1 -- Type

,3 -- Layer

,CAST('{{' || (SELECT node\_id[3] FROM ga.sw\_gis\_point where rwo\_id[3] = t.rwo\_id[3] LIMIT 1) || ',1}}' AS integer[]));

UPDATE gas\_topo.anschlussltg\_abschnitt\_test t SET g =

CreateTopoGeom(

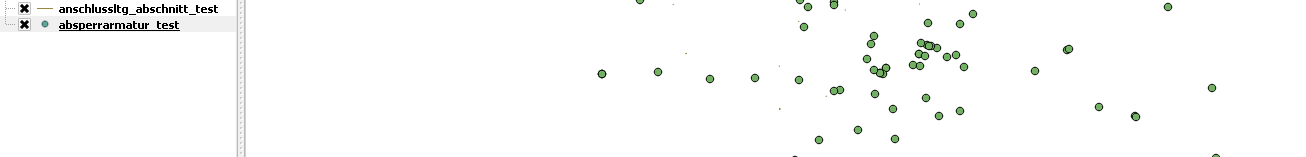
'gas\_topo', -- Topo name

2, -- type (lineal)

4, -- layer id

CAST('{{' || (SELECT link\_id[3] FROM ga.sw\_gis\_chain\_link where chain\_id[3] = (SELECT chain\_id[3] FROM ga.sw\_gis\_chain WHERE rwo\_id[3] = t.rwo\_id[3] LIMIT 1) LIMIT 1) || ',2}}' AS integer[]));

Dies erlaubt es, dass über die RWO\_ID und die node\_id oder die link\_id ein Bezug zu Node und Edge hergestellt werden kann. Nun sind die Smallworld Daten auch in QGIS sichtbar:



Geschwindigkeit mit Realdaten

* Tabelle Absperrarmatur (Node): 12443 Einträge - 489083ms > ca. 8M
* Tabelle Hauptltg\_abschnitt: 13886 Einträge – 902930ms > ca. 15M
* Tabelle Anschlussltg\_abschnitt: 184455 Einträge – 35622859ms 10h

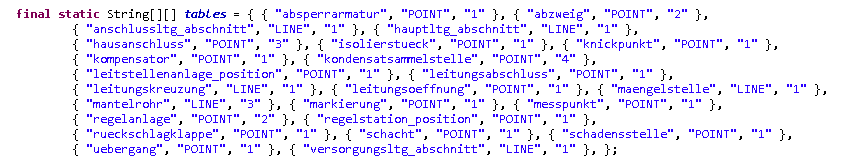
Es fällt auf das eine relativ „geringe“ Datenmenge eine Menge Zeit in Anspruch nimmt. Nach einer Analyse der Querry fällt auf das die Aufteilung eines Arrays erheblich Performance verbraucht. Durch also leichte Änderung von rwo\_id[3] = t.rwo\_id[3] zur rwo\_id = rwo\_id und chain\_id = chain\_id in der inneren Abfrage konnte die Geschwindigkeit um teilweise ein **1000faches** beschleunigt werden.

Durch die „Verbesserung“ konnte folgendes erreicht werden:

* Tabelle Absperrarmatur (Node): 12443 Einträge > ca. 10s
* Tabelle Hauptltg\_abschnitt: 13886 Einträge > ca. 20s
* Tabelle Anschlussltg\_abschnitt: 184455 Einträge > ca. 2m

**Konvertierung 3**

Es fällt auf das manche Tabellen mehrere geometrie-Spalten haben und somit auch mehrere topogeoms bekommen müssten.



Manche der Tabellen haben sogar Geometrien verschiedenen Typs. Die Kondensat Sammelstelle bspw. Hat LINE und POINT. Weiterhin ergeben manche dieser Zeilen keinen Eintrag auf eine RWO\_ID wodurch diese ausfallen würden.

Deshalb der Versuch mehrere Spalten zu erstellen und mit einem OFFSET Parameter die Ergebnisse einschränken.

UPDATE gas\_topo.absperrarmatur t SET g = CreateTopoGeom('gas\_topo',1,1,CAST('{{' || (SELECT node\_id[3] FROM ga.sw\_gis\_point where rwo\_id = t.rwo\_id OFFSET 0) || ',1}}' AS integer[]));

Obwohl man somit nur noch einen Rückgabewert hat, meldet Postgres:

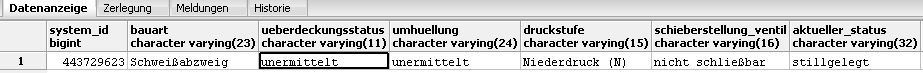
FEHLER: als Ausdruck verwendete Unteranfrage ergab mehr als eine Zeile

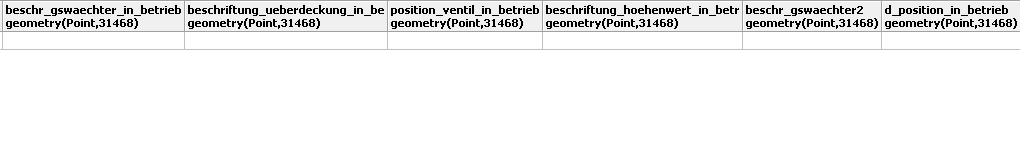
Gelöst wird dies durch die Erweiterung OFFSET 0 LIMIT 1. Ein weiteres Problem das entstanden ist, ist das man nicht unterscheiden kann zu welcher topogeom der Node nun zugeteilt werden soll. Selbst wenn ich durch OFFSET und LIMIT begrenze, ordne ich die Ergebnisse nicht. Deshalb wird die Reihenfolge ignoriert (Sollte keine Rolle spielen). Weitere Fehlermeldung:

FEHLER: Wert für Domäne topoelementarray verletzt Check-Constraint „dimensions“

Dies soll bedeuten, dass keine node für diese rwo\_id gefunden wurde. Mit einer leichten Abfrage von: SELECT \* FROM gas\_topo.abzweig t WHERE (SELECT node\_id[3] FROM ga.sw\_gis\_point where rwo\_id = t.rwo\_id OFFSET 0 LIMIT 1) IS NOT NULL;

Stellt sich somit herraus das von 181044 Zeilen lediglich eine Fehlerhaft ist. Diese wird nun entfehrnt.



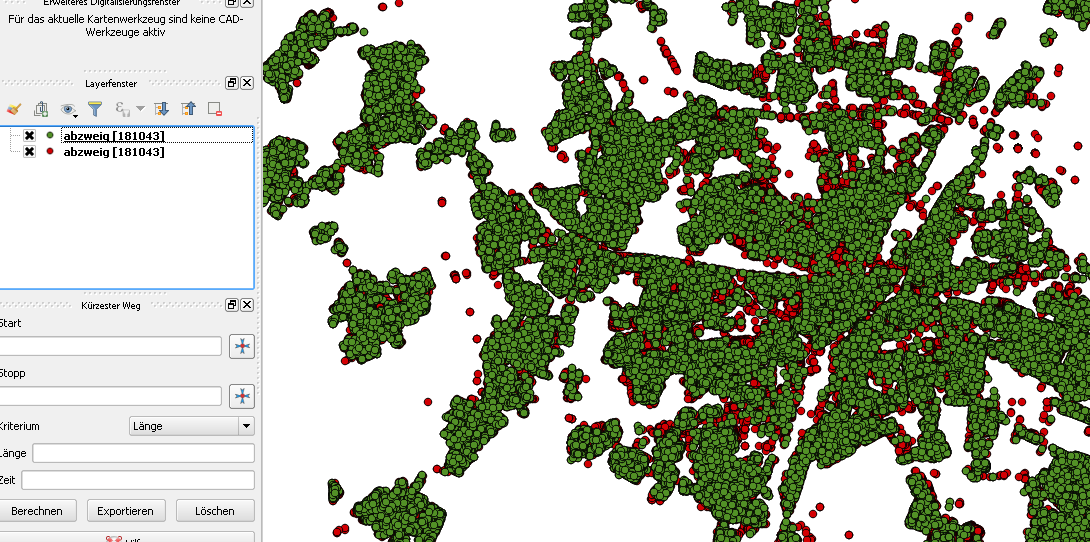


Auffällig ist auch das hier einfach keine Geometrie eingetragen ist und das dieser Abzweig stillgelegt ist.

DELETE FROM gas\_topo.abzweig where system\_id = 443729623;

Um nun nicht nochmal die gleichen nodes im Abzweig zuzuordnen schreiben wir die Querry um:

UPDATE gas\_topo.abzweig t SET h = CreateTopoGeom('gas\_topo',1,9,CAST('{{' || (SELECT node\_id[3] FROM ga.sw\_gis\_point where rwo\_id = t.rwo\_id OFFSET 1 LIMIT 1) || ',1}}' AS integer[])) WHERE system\_id IN (SELECT system\_id FROM gas\_topo.abzweig t WHERE (SELECT node\_id[3] FROM ga.sw\_gis\_point where rwo\_id = t.rwo\_id OFFSET 1 LIMIT 1) IS NOT NULL) ;



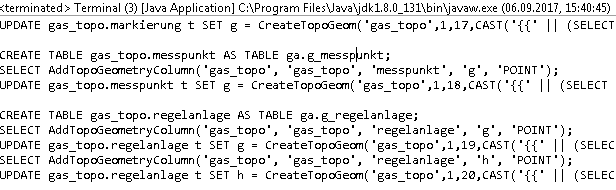
Leitungsoeffnungen hat keine systemid. Diese fehlt leider für unser aktuelles Vorgehen. Betrachtung später.

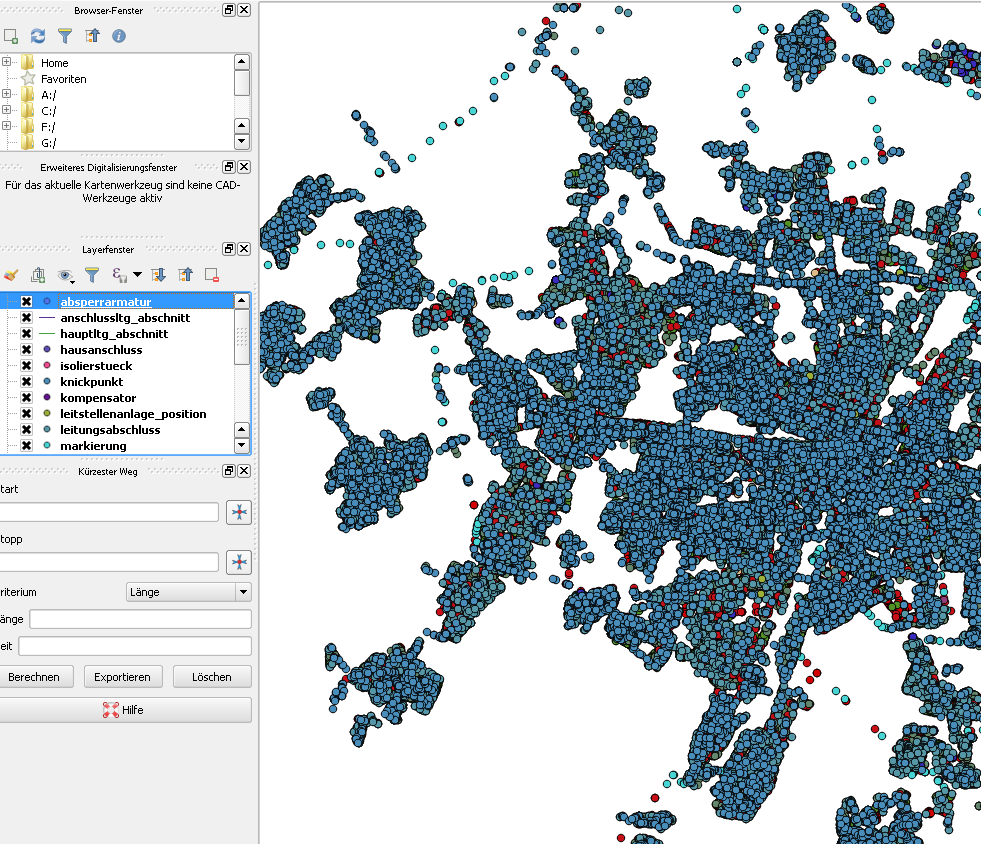
Helfer Programm

Ich habe ein Programm in JAVA geschrieben das mir die Arbeit abnimmt. Es nimmt lediglich ein String[] Array entgegen mit den Informationen: {TABLE\_NAME, FEATURE\_TYPE, N\_GEOMETRY\_COLUMNS}

Dies muss aber noch Angespasst werden:

* Verschiedene Geometrietypen in einer Tabelle





**SW Modell Link <> Chain**

Rwo\_id – chain\_id 1:1 Beziehung

Chain\_id – link\_id 1:n Beziehung

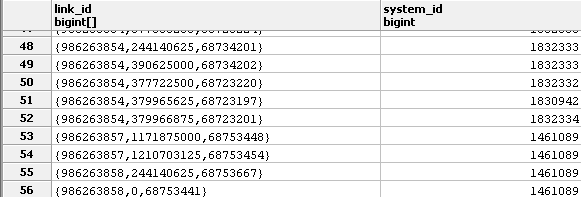
Rwo\_id – link\_id 1:n Beziehung

* Ein Betriebsmittel kann auf mehreren Edges liegen
* Für jede 1:n müsste eine weitere Topogeom angelegt werden

Über einen Join kann die Verteilung überprüft werden:

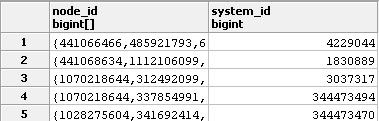
SELECT link\_id,ga.g\_hauptltg\_abschnitt.system\_id FROM ga.sw\_gis\_chain\_link JOIN ga.sw\_gis\_chain ON ga.sw\_gis\_chain.chain\_id = ga.sw\_gis\_chain\_link.chain\_id

JOIN ga.g\_hauptltg\_abschnitt ON ga.g\_hauptltg\_abschnitt.rwo\_id = ga.sw\_gis\_chain.rwo\_id;



Für nodes könnte man das Gleiche per JOIN abfragen:

SELECT node\_id,system\_id FROM ga.sw\_gis\_point JOIN ga.g\_abzweig ON ga.sw\_gis\_point.rwo\_id = ga.g\_abzweig.rwo\_id;



**Konvertierung Funktion**

Nun da wir die ganzen Probleme der einfachen Konvertierung dargestellt haben, kann man eine Funktion bauen, die automatisch die Tabelle mit den Topogeoms erstellt und diese somit auch als Layer registiert.

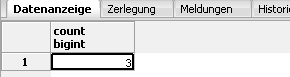
**Vorgang**

1. Bestimmen der maximalen Anzahl an zugehörigen Topogeoms
2. Identifizieren der Topogeom Typen
3. Erstellen der Topogeom Spalten
4. Generieren der Topogeoms

**Maximale Topogeoms**

Um die maximale Anzahl an Topogeoms zu Erfahren (Also wie vielen Edges/Nodes können einem Betriebsmittel zugeordnet werden [Höchste 1:n Beziehung]). Dies kann leicht durch eine Erweiterung der Abfrage durchgeführt werden:

SELECT count(node\_id) FROM ga.sw\_gis\_point JOIN ga.g\_abzweig ON ga.sw\_gis\_point.rwo\_id = ga.g\_abzweig.rwo\_id GROUP BY system\_id ORDER BY count DESC LIMIT 1;



Dies würde also bedeuten, dass wir drei Spalten für die Topogeoms anlegen müssen.

**Identifizieren der Topogeom Typen**

In den Tabellen an sich gibt es oftmals mehrere Geometriespalten, die nur Informationen über Beschriftungen enthalten. Deshalb ist es nicht möglich die Spalten an sich durchzugehen und den Typen abzufragen.

Ein anderer Weg ist, die Maximalen Topogeoms für beide Typen abzufragen:

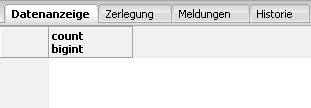
Die Tabelle Abzweig hat maximal 3 node\_ids(Siehe „Maximale Topogeoms“)

* Führe das selbe für Links(Edges) aus

SELECT count(link\_id) FROM ga.sw\_gis\_chain\_link JOIN ga.sw\_gis\_chain ON ga.sw\_gis\_chain.chain\_id = ga.sw\_gis\_chain\_link.chain\_id

JOIN ga.g\_abzweig ON ga.g\_abzweig.rwo\_id = ga.sw\_gis\_chain.rwo\_id GROUP BY system\_id ORDER BY count DESC LIMIT 1;

**Wir erhalten KEIN Ergebnis, also wissen wir, dass wir nur POINT Topogeoms anlegen müssen.**



**Anders ist dies beispielsweise bei dem Hauptltg\_Abschnitt**



CREATE OR REPLACE FUNCTION topology.getMaxTopoNodes(layer text)

RETURNS setof bigint AS

$func$

DECLARE

SQL text;

BEGIN

SQL = 'SELECT count(node\_id) FROM ga.sw\_gis\_point JOIN ' || layer || ' ON ga.sw\_gis\_point.rwo\_id = ' || layer || '.rwo\_id GROUP BY system\_id ORDER BY count DESC LIMIT 1';

RETURN QUERY Execute SQL;

END

$func$ LANGUAGE plpgsql;

SELECT topology.getMaxTopoNodes('ga.g\_abzweig');

CREATE OR REPLACE FUNCTION topology.getMaxTopoEdges(layer text)

RETURNS setof bigint AS

$func$

DECLARE

SQL text;

BEGIN

SQL = 'SELECT count(link\_id) FROM ga.sw\_gis\_chain\_link JOIN ga.sw\_gis\_chain ON ga.sw\_gis\_chain.chain\_id = ga.sw\_gis\_chain\_link.chain\_id

JOIN ' || layer || ' ON ' || layer || '.rwo\_id = ga.sw\_gis\_chain.rwo\_id GROUP BY system\_id ORDER BY count DESC LIMIT 1;';

RETURN QUERY Execute SQL;

END

$func$ LANGUAGE plpgsql;

SELECT topology.getMaxTopoEdges('ga.g\_hauptltg\_abschnitt');

**Erstellen der Topogeom Spalten**

Der Code ist der gleiche wie auch bereits vorher. (Siehe oben)

**Generieren der Topogeoms**

Die Topogeoms werden wie auch bevor anhand der entsprechenden ID erstellt und mit dem Betriebsmittel verlinkt.

**Netzwerkverfolgung München**

Verbesserte Funktion zur Abfrage der Betriebsmittel. Lediglich alle Einträge für die ID müssten je nach Datenmodell ausgetauscht werden.

DROP FUNCTION simulation.getAllRecords(topo\_id int,elementid int,type int);

CREATE OR REPLACE FUNCTION simulation.getAllRecords(topo\_id int,elementid int,type int)

RETURNS TABLE(

id int,

schem\_name character varying,

tabl\_name character varying) AS

$func$

DECLARE

my\_cursor CURSOR FOR SELECT schema\_name,table\_name,feature\_column FROM topology.layer where topology\_id = topo\_id and feature\_type = type;

my\_record RECORD;

SQL text;

BEGIN

BEGIN

DROP TABLE tmpr;

CREATE TEMPORARY TABLE tmpr (

id int,

schema\_name character varying,

table\_name character varying

);

END;

OPEN my\_cursor;

IF EXISTS (SELECT 1 FROM tmpr) THEN

DELETE FROM tmpr;

END IF;

LOOP

SQL = 'INSERT INTO tmpr SELECT DISTINCT id,schema\_name,table\_name FROM topology.layer';

FETCH my\_cursor INTO my\_record;

EXIT WHEN NOT FOUND;

SQL = SQL || ' JOIN ' || my\_record.schema\_name || '.' || my\_record.table\_name || ' ON (SELECT GetTopoGeomElements(' || my\_record.feature\_column || ')' || ' = ARRAY['|| elementid || ',' || type || ']) = true AND table\_name = ''' || my\_record.table\_name || ''' AND topology.layer.topology\_id = ' || topo\_id || ' AND topology.layer.feature\_type = ' || type;

EXECUTE SQL;

END LOOP;

RETURN QUERY SELECT \* FROM tmpr;

END

$func$ LANGUAGE plpgsql;

SELECT \* FROM simulation.getAllRecords(4,2,1);

-- Equivalent zu

SELECT DISTINCT id,schema\_name,table\_name FROM topology.layer JOIN simulation.schieber ON (SELECT GetTopoGeomElements(g) = ARRAY[2,1]) = true AND table\_name = 'schieber' AND topology.layer.topology\_id = 4 AND topology.layer.feature\_type = 1



Eine Abfrage an die Tabelle München hat eine Dauer von 2 Minuten ergeben. Deshalb wird nun die Abfrage umgestaltet und auf die Relation Tabelle bezogen. Dies sollte den Prozess ungemein beschleunigen.

Anhand der Relation dauer von 45 Sekunden. Diese Abfrage wird erneut beschleunigt. Deutlich zeigt das die Arrayauflösung viel Performance beansprucht. Dies wird nun durch die Funktion topology.id(feature\_column) ersetzt:

DROP FUNCTION topology.getBetriebsmittel(topology text,elementid int,typeid int);

CREATE OR REPLACE FUNCTION topology.getBetriebsmittel(topology text,elementid int,typeid int)

RETURNS TABLE (

id int,

s\_name character varying,

t\_name character varying

) AS

$func$

DECLARE

SQL text;

SQL\_DYNAMIC text = 'SELECT r.topogeo\_id,l.table\_name,l.feature\_column FROM '|| topology||'.relation r,topology.layer l WHERE r.element\_id = '|| elementid ||' and r.element\_type = '|| typeid ||' and l.layer\_id = r.layer\_id and l.schema\_name = '''|| topology ||'''';

my\_record RECORD;

BEGIN

TRUNCATE topology.result;

FOR my\_record IN EXECUTE SQL\_DYNAMIC

LOOP

SQL = 'INSERT INTO topology.result VALUES((SELECT system\_id FROM ' || topology || '.' || my\_record.table\_name || ' WHERE id(' || my\_record.feature\_column ||' ) = ' || my\_record.topogeo\_id || '),''' || topology || ''',''' || my\_record.table\_name || ''')';

EXECUTE SQL;

END LOOP;

RETURN QUERY SELECT \* FROM topology.result;

END

$func$ LANGUAGE plpgsql;

SELECT \* FROM topology.getBetriebsmittel('gas\_test',3860174,1);

Die Abfrage dauerte ca. 280 ms.

Simulation

Nun wird anhand des ganzen Topologienetzwerkes die Simulationsabfrage gemacht. Dies geschieht anhand von:

WITH RECURSIVE path AS (

SELECT edge\_id, start\_node, end\_node

FROM gas\_test.edge\_data

WHERE start\_node = 3860174 OR end\_node = 3860174

UNION

SELECT e.edge\_id, e.start\_node, e.end\_node

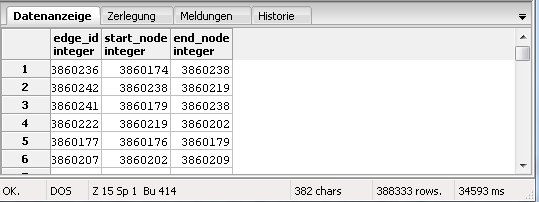
FROM gas\_test.edge\_data e, path s

WHERE ((SELECT id FROM topology.getBetriebsmittel('gas\_test',s.start\_node,1)LIMIT 1) <> 0) AND s.start\_node = e.start\_node OR s.start\_node = e.end\_node OR s.end\_node = e.end\_node OR s.end\_node = e.start\_node )

SELECT \* FROM path;

Dauer: 269646ms -> 269s -> ca. 4-5 Minuten

Die Abfrage geht aber auch davon aus, dass es nur einen Hausanschluss zu jedem Node gibt. Ob dies ein Problem ist müsste besprochen werden.



11 Records / MS

**Simples Traceout ohne Stopattribut**

WITH RECURSIVE path AS (

SELECT edge\_id, start\_node, end\_node

FROM gas\_test.edge\_data

WHERE start\_node = 3860174 OR end\_node = 3860174

UNION

SELECT e.edge\_id, e.start\_node, e.end\_node

FROM gas\_test.edge\_data e, path s

WHERE s.start\_node = e.start\_node OR s.start\_node = e.end\_node OR s.end\_node = e.end\_node OR s.end\_node = e.start\_node )

SELECT \* FROM path;



51 Records / MS

**Traceout with validate on absperrarmatur**

WITH RECURSIVE path AS (

SELECT edge\_id, start\_node, end\_node

FROM gas\_test.edge\_data

WHERE start\_node = 3860174 OR end\_node = 3860174

UNION

SELECT e.edge\_id, e.start\_node, e.end\_node

FROM gas\_test.edge\_data e, path s

WHERE ((SELECT \* FROM topology.validateAttribute('gas\_test','absperrarmatur','armaturenstellung','geschlossen',e.start\_node,1)) = false) AND s.start\_node = e.start\_node OR s.start\_node = e.end\_node OR s.end\_node = e.end\_node OR s.end\_node = e.start\_node )

SELECT \* FROM path;



**Fazit**

Eins der Betriebsmittel abzufragen dauert ca. 50 ms.

Eins der Attribute zu überprüfen dauert ca. 150ms.

**Weitere Notizen**

* Wie kann ich überhaupt beweisen, dass es in einem großen System richtig ist?
* In einem Netzwerk sind Flächen/ Überschneidungen unwichtig. Wurde damals benutzt mit der Euler Charakteristik zur Überprüfung bei der Digitalisierung.

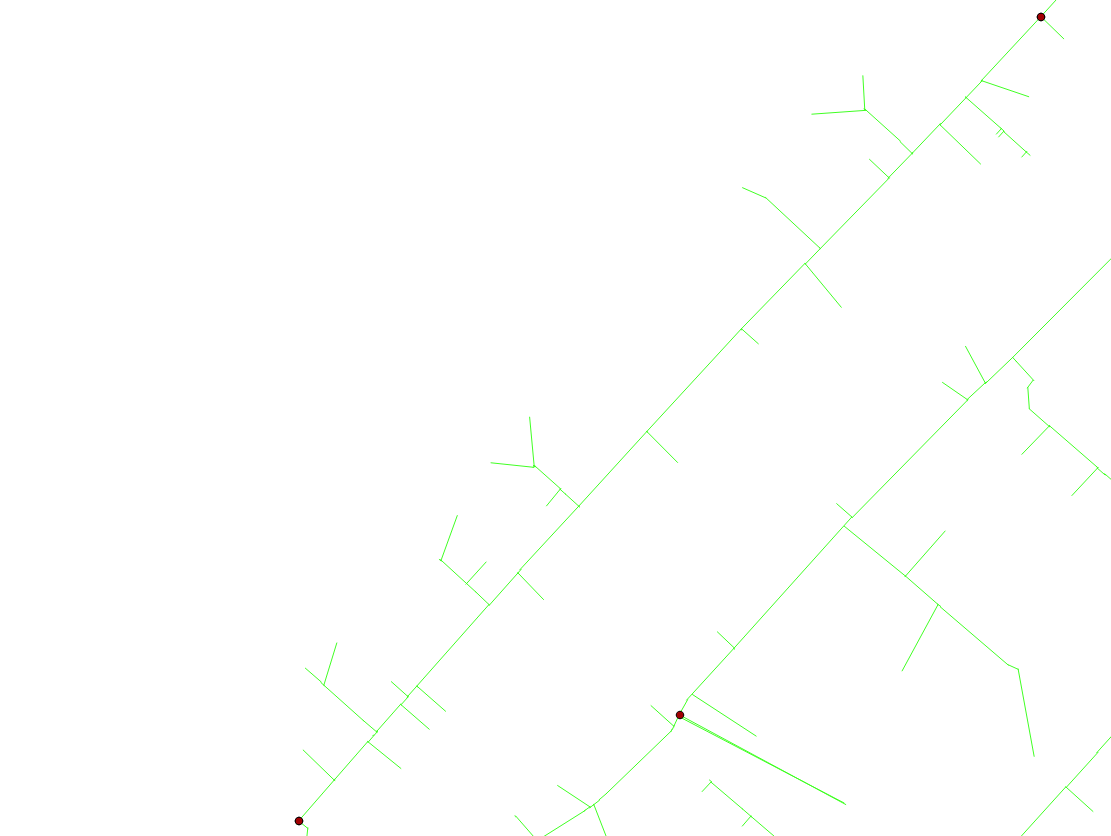
Überprüfung

Nehme eine Abteilung in dem großen Netzwerk und Teile dies anhand von zwei Erfundenen Absperrarmaturen:

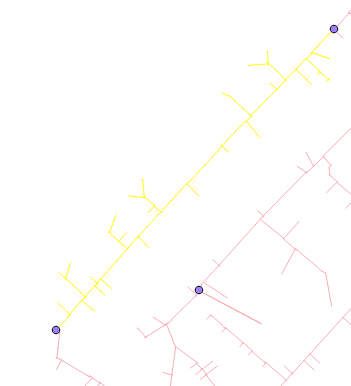
1. Node: 437082429
2. Node: 11865607

INSERT INTO gas\_test.absperrarmatur (g,armaturenstellung) VALUES(CreateTopoGeom('gas\_test',1,7,'{{437082429,1}}'),'geschlossen');

INSERT INTO gas\_test.absperrarmatur (g,armaturenstellung) VALUES(CreateTopoGeom('gas\_test',1,7,'{{11865607,1}}'),'geschlossen');



Als Testpunkt zur Simulation wird dann 1000338 benutzt.



Die Geschwindigkeit durch Index verbessern. Das spart Zeit kostet jedoch Speicher. Dieser ist Dank der heutigen Technologie sehr billig.

select topology.validateAttributeBool('gas\_test','absperrarmatur','armaturenstellung','geschlossen',node\_id,1) from gas\_test.node limit 100

Geschwindigkeit vor Index: 5400ms

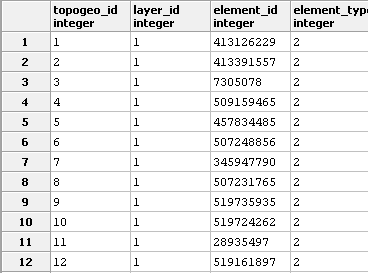
Index element\_id: 1780ms > 3.03x Geschwindigkeit

+ topogeo\_id: 1610ms

+ layer\_id: 1500-1800ms

+ element\_type: 1500-1800ms

Index von relations Tabelle



Die Simulation des obrigen Abbildes hat sich damit von 24000ms / 24s auf 6432ms / 6s verbessert. Eine vierfache steigerung.

Die Relation Tabelle des Datenbestandes hat eine Größenordnung von 18mb mit 421762 Einträgen.

Nachdem die Indexes erstellt wurden:

Index element\_id: 18mb

+ topogeo\_id: 18mb

+ layer\_id: 18mb

+ element\_type: 18mb

Man sieht also, dass selbst die Speicherkosten extreme niedrig sind.

SELECT nspname || '.' || relname AS "relation",

pg\_size\_pretty(pg\_total\_relation\_size(C.oid)) AS "total\_size"

FROM pg\_class C

LEFT JOIN pg\_namespace N ON (N.oid = C.relnamespace)

WHERE nspname NOT IN ('pg\_catalog', 'information\_schema')

AND C.relkind <> 'i'

AND nspname !~ '^pg\_toast'

ORDER BY pg\_total\_relation\_size(C.oid) DESC;

Um nach dem Selektieren direkt einen neuen Layer mit dem Ergebnis anzufertigen wird ein Temporärer Layer mit dem Typen memory erstellt:

temp = QgsVectorLayer(*"LineString?crs=epsg:31468"*,*”name”*, *"memory"*)

Weiterhin wird dem neuen Layer alle Attribute aus dem Inputlayer hinzugefügt:

temp\_data = temp.dataProvider()

attr = layer.dataProvider().fields().toList()

temp\_data.addAttributes(attr)

temp.updateFields()

Dem temporären Layer werden nun alle selektierten Features hinzugefügt:

temp\_data.addFeatures(layer.selectedFeatures())

Und letzendlich wird der Layer registriert:

QgsMapLayerRegistry.instance().addMapLayer(temp)