PostGIS in Action

Onlinekarten mit PostGIS auf Basis von amtlichen ALKIS Daten und OpenStreetMap

FOSSGIS 2017, Passau

Oliver Tonnhofer <olt@omniscale.de | @oltonn>

URL der Folien auf der letzten Seite

Omniscale

Über mich

Oliver Tonnhofer

- Omniscale GmbH & Co. KG, Oldenburg
  - Open Source Entwicklung (Client/Server)
  - MapProxy und Imposm Entwicklung und Support
  - OpenStreetMap Hosting
  - ALKIS+OSM Kartendienste

Omniscale

Inhalt

- Daten importieren
- Strukturieren
- Homogenisieren
- Optimieren
- Aktualisieren

Achtung!

Einige Punkte dieser Präsentation sind nur für größere Datenmengen relevant.

Erst messen, dann handeln.

Importieren

Tools

- OSM:
  - Imposm 3
- ALKIS
  - PostNAS/OGR
- Andere Daten:
  - OGR (`ogr2ogr`)
  - Eigene Skripte

Importieren

- Dokumentation lesen
  - Optionen verstehen
- Software aktuell halten
  - Neue Optionen testen

Beispiel für "versteckte" Funktion

PG_USE_COPY

- `ogr2ogr --config PG_USE_COPY YES`
- Importiert Datenstrom statt einzelner INSERT Statements
- Rund 4x schneller

Beispiel für "versteckte" Funktion

PG_USE_COPY

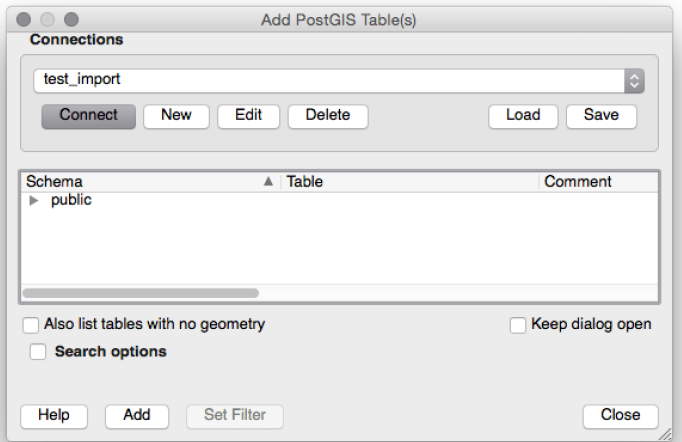
- `ogr2ogr --config PG_USE_COPY YES`
- Importiert Datenstrom statt einzelner INSERT Statements
- Rund 4x schneller
- Seit GDAL 2.0 Standardmäßig aktiv

Strukturieren

Tabellen strukturieren mit Postgres-Schemas

- Postgres-Schema != Datenbank-Schema
- Postgres-Schema == Namespace

Tabellen strukturieren mit Postgres-Schemas



Omniscale

Postgres-Schema

Strukturieren unterschiedlicher Daten

```
CREATE SCHEMA osm;
CREATE SCHEMA alkis;
```

Postgres-Schema

Strukturieren unterschiedlicher Daten

```
CREATE SCHEMA osm;
CREATE SCHEMA alkis;
```

Gleiche Tabellen in unterschiedlichen Schema:

```
CREATE TABLE   osm.gebaeude ...;
CREATE TABLE alkis.gebaeude ...;
```

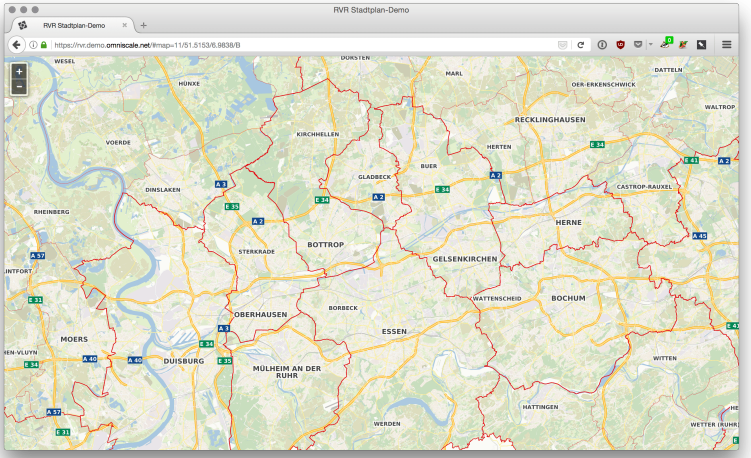
Zugriff auf Postgres-Schema

Über `schema.tabelle`:

```
SELECT * FROM   osm.gebaeude;
SELECT * FROM alkis.gebaeude;
```

Stadplanwerk Regionalverband Ruhr

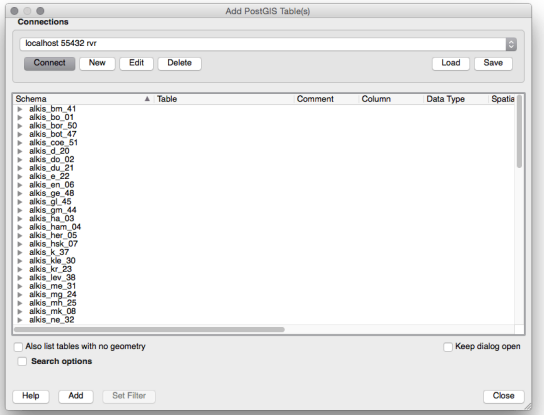
38 Kreise + OSM



Omniscale

Stadplanwerk Regionalverband Ruhr

38 Kreise + OSM



Omniscale

Daten in Tabellen strukturieren

Landnutzung

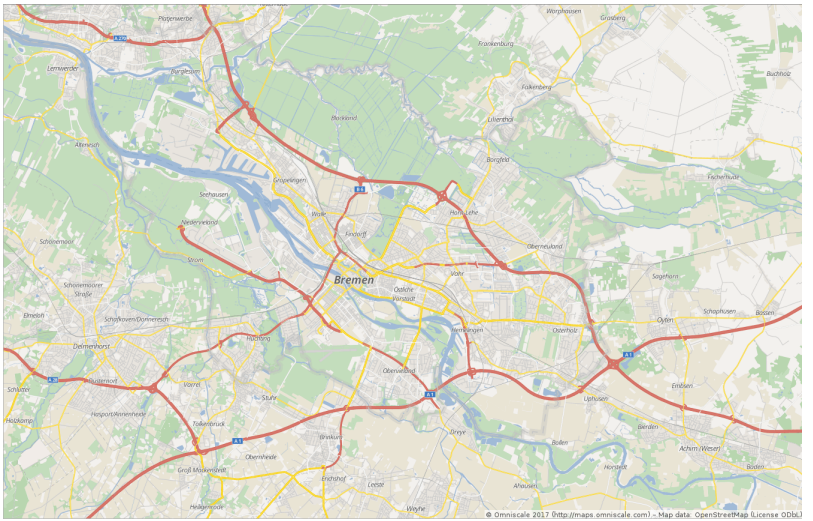
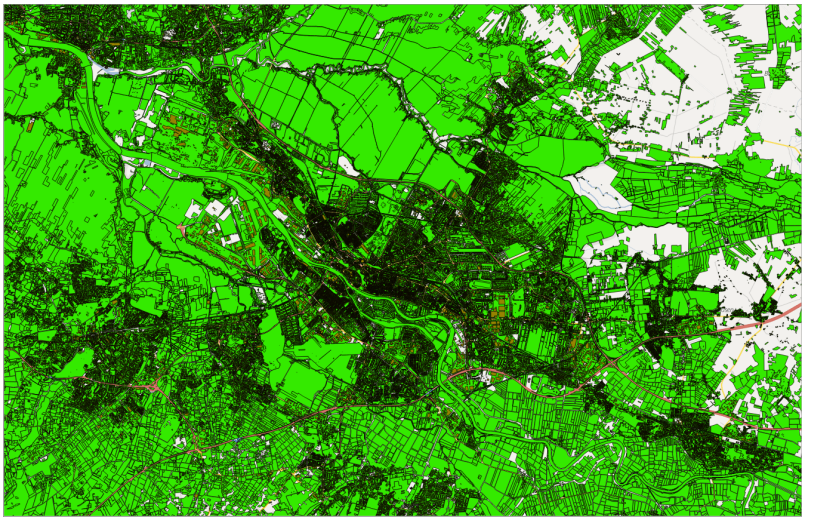


Bild: © 2017 Omniscale • OpenStreetMap - contributors

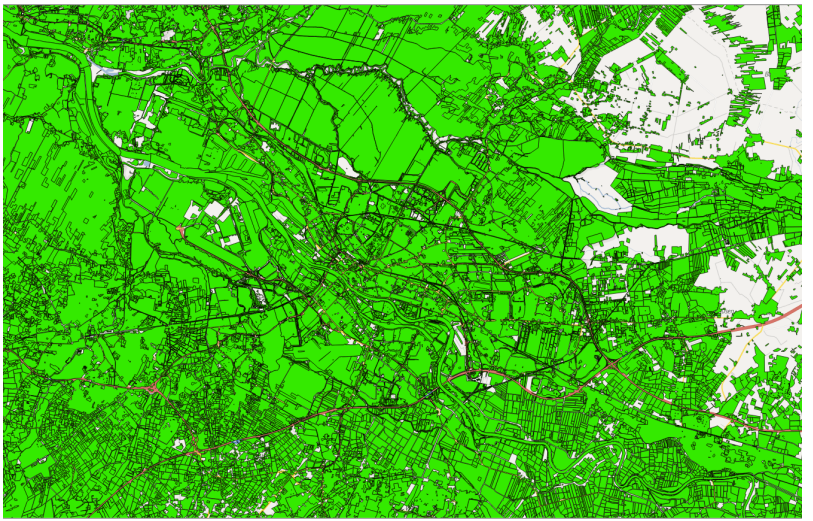
Omniscale

Landnutzung: Alle Polygone aus OSM



Omniscale

Landnutzung: Nur relevante Polygone



Omniscale

Optimal: Eine Tabelle pro Kartenlayer

- Eine Tabelle für Landnutzung
  - Wälder, Parks, bebaute Flächen, Friedhöfe
- Eine Tabelle für Wasserflächen

Optimal: Eine Tabelle pro Kartenlayer

- Eine Tabelle für Landnutzung
  - Wälder, Parks, bebaute Flächen, Friedhöfe
- Eine Tabelle für Wasserflächen
- Eine Tabelle für Gebäude
- …

Optimal: Eine Tabelle pro Kartenlayer

- Eine Tabelle für Landnutzung
  - Wälder, Parks, bebaute Flächen, Friedhöfe
- Eine Tabelle für Wasserflächen
- Eine Tabelle für Gebäude
- …

Für OSM Daten mit Imposm möglich.

Optimal: Eine Tabelle pro Kartenlayer

- Eine Tabelle für Landnutzung
  - Wälder, Parks, bebaute Flächen, Friedhöfe
- Eine Tabelle für Wasserflächen
- Eine Tabelle für Gebäude
- …

Für OSM Daten mit Imposm möglich.

ALKIS?

ALKIS Daten (PostNAS)

ca. 30 Tabellen für Landnutzung

```
ax_bahnverkehr ax_bahnverkehrsanlage ax_bauwerkimgewaesserbereich
ax_bauwerkimverkehrsbereich ax_bauwerkoderanlagefuersportfreizeitunderholung
ax_dammwalldeich ax_einrichtungenfuerdenschiffsverkehr
ax_flaechebesondererfunktionalerpraegung ax_flaechegemischternutzung
ax_fliessgewaesser ax_flugverkehr ax_flugverkehrsanlage ax_friedhof ax_gehoelz
ax_hafenbecken ax_halde ax_heide ax_industrieundgewerbeflaeche
ax_klassifizierungnachwasserrecht ax_landwirtschaft ax_meer ax_platz
ax_schiffsverkehr ax_sportfreizeitunderholungsflaeche ax_stehendesgewaesser
ax_strassenverkehr ax_tagebaugrubesteinbruch ax_untergeordnetesgewaesser
ax_vegetationsmerkmal ax_wald ax_wohnbauflaeche
```

Homogenisieren

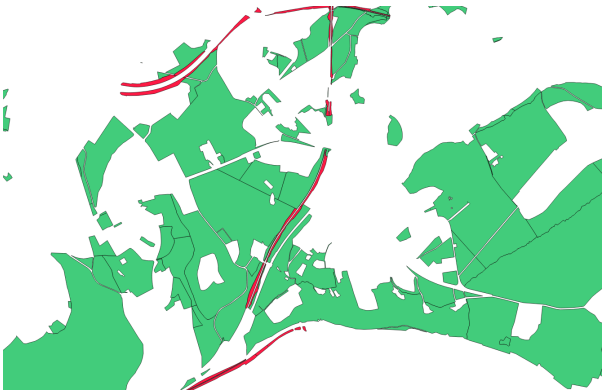
Wälder

Aufgabe: Alle Flächen mit Laub- und Nadelholz sollen als Wald dargestellt werden

Wälder

Aufgabe: Alle Flächen mit Laub- und Nadelholz sollen als Wald dargestellt werden

Datenquellen: ax_wald, ax_gehoelz, ax_vegetationsmerkmal



Wälder

Lösung: Abfragen mit `UNION ALL` kombinieren

```
SELECT wkb_geometry FROM ax_wald
UNION ALL
SELECT wkb_geometry FROM ax_gehoelz
UNION ALL
SELECT wkb_geometry FROM ax_vegetationsmerkmal
  WHERE bewuchs = ANY (ARRAY[1011, 1012, 1021, 1022, 1023, 1250])
```

Vorteil: Eine SQL Abfrage und ein Kartenlayer

Layer Landnutzung

- Weitere Kombination mit anderen Landnutzungen
- Unterscheidung durch `typ`-Spalte

Layer Landnutzung

- Weitere Kombination mit anderen Landnutzungen
- Unterscheidung durch `typ`-Spalte

```
SELECT wkb_geometry, 'wald' AS typ FROM ax_wald
UNION ALL
SELECT wkb_geometry, 'friedhof' AS typ FROM ax_friedhof
...
```

Layer Landnutzung

- Weitere Kombination mit anderen Landnutzungen
- Unterscheidung durch `typ`-Spalte

```
SELECT wkb_geometry, 'wald' AS typ FROM ax_wald
UNION ALL
SELECT wkb_geometry, 'friedhof' AS typ FROM ax_friedhof
...
```

```
...
UNION ALL
  SELECT wkb_geometry,
  CASE
    WHEN bewuchs = ANY (ARRAY[1011, 1012, 1021, 1022, 1023, 1250]) THEN 'wald'
    WHEN bewuchs = 1500 THEN 'gras'
    ELSE NULL::text
  END AS typ
  FROM ax_vegetationsmerkmal
UNION ALL
...
```

Layer Landnutzung

Einfaches Kartenstyling über `typ`

```
CLASS
  EXPRESSION ('[typ]' = 'wald')
  STYLE
    COLOR "#d0f0c0"
  END
END
CLASS
  EXPRESSION ('[typ]' = 'gras')
  STYLE
    COLOR "#c0f0d0"
  END
END
...
```

Layer Landnutzung

- `UNION ALL` mit ~30 AX-Tabellen

Layer Landnutzung

- `UNION ALL` mit ~30 AX-Tabellen
- Statt komplexer Abfrage im Kartenstyle, Nutzung von Views:

```
CREATE VIEW landnutzung AS SELECT
    wkb_geometry::geometry(Geometry, 25832) AS "geometry",
    typ::VARCHAS as "typ"
  FROM (
    SELECT ...
    UNION ALL
    SELECT ...
    UNION ALL
    ...
  )
```

Layer Landnutzung

Deutlich vereinfachtes Kartenstyling

```
 LAYER
    NAME landnutzung
    MAXSCALEDENOM 50000
    STATUS ON
    TYPE POLYGON
    DATA "geometry FROM landnutzung"
    CONNECTION "host=..."
    CONNECTIONTYPE postgis
    PROCESSING "CLOSE_CONNECTION=DEFER"
    PROJECTION
      "init=epsg:25832"
    END
...
```

Optimieren

Erst messen, dann handeln.

PostgreSQL Tuning

- `shared_buffer`
- `work_mem`
- `maintenance_work_mem`
- ...

Logging Mapserver

```
MAP
  ...
  CONFIG "MS_ERRORFILE" "/tmp/mapserver.log"
  DEBUG 2
  ...
END
```

Logging Mapserver

```
MAP
  ...
  CONFIG "MS_ERRORFILE" "/tmp/mapserver.log"
  DEBUG 2
  ...
END
```

```
[Mon Mar  6 14:39:17 2017].697586 msDrawMap(): Layer 0 (hintergrund), 0.001s
[Mon Mar  6 14:39:17 2017].698592 msDrawMap(): Layer 2 (landnutzung), 1.832s
```

Logging PostgreSQL

```
logging_collector = on
log_directory = 'pg_log'
log_min_duration_statement = 50
```

Logging PostgreSQL

```
logging_collector = on
log_directory = 'pg_log'
log_min_duration_statement = 50
```

```
LOG:  duration: 1264.708 ms  execute <unnamed>: SELECT ...
```

Explain Analyze

```
SELECT geometry
FROM osm_roads where geometry && ST_GeomFromText('POLYGON((...))',3857);
```

Explain Analyze

```
SELECT geometry
FROM osm_roads where geometry && ST_GeomFromText('POLYGON((...))',3857);
```

```
EXPLAIN ANALYZE
SELECT geometry
FROM osm_roads where geometry && ST_GeomFromText('POLYGON((...))',3857);
```

Explain Analyze

```
SELECT geometry
FROM osm_roads where geometry && ST_GeomFromText('POLYGON((...))',3857);
```

```
EXPLAIN ANALYZE
SELECT geometry
FROM osm_roads where geometry && ST_GeomFromText('POLYGON((...))',3857);
```

```
Seq Scan on roads  (cost=0.00..39552.96 rows=27707 width=128)
                   (actual time=1.085..2566.371 rows=30604 loops=1)
   Filter: (geometry && '0103000020110F...'::geometry)
   Rows Removed by Filter: 976273
 Planning time: 0.288 ms
Execution time: 2568.893 ms
```

Mit Geometry-Index

```
 Bitmap Heap Scan on roads  (cost=1115.01..29437.21 rows=27707 width=128)
                            (actual time=8.410..26.016 rows=30604 loops=1)
   Recheck Cond: (geometry && '0103000020110F...'::geometry)
   Heap Blocks: exact=4646
   ->  Bitmap Index Scan on roads_geom  (cost=0.00..1108.09 rows=27707 width=0)
                                        (actual time=7.509..7.509 rows=30604 loops=1)
         Index Cond: (geometry && '0103000020110F...'::geometry)
 Planning time: 0.378 ms
Execution time: 27.624 ms
```

BBOX Filter

Normale Abfrage

Aus:

```
SELECT geometry FROM gebauede
```

BBOX Filter

Normale Abfrage

Aus:

```
SELECT geometry FROM gebauede
```

wird:

```
SELECT * FROM
(SELECT geometry FROM gebauede)
AS x WHERE x.geometry && ST_Envelope....
```

BBOX Filter

Normale Abfrage

Aus:

```
SELECT geometry FROM gebauede
```

wird:

```
SELECT * FROM
(SELECT geometry FROM gebauede)
AS x WHERE x.geometry && ST_Envelope....
```

PostgreSQL kann trotzdem den Geometrie-Index verwenden.

BBOX Filter mit Geometrie-Operatoren in Abfragen

ST_Centroid Abfrage

Aus:

```
(SELECT ST_Centroid(geometry) AS geometry FROM gebauede)
```

BBOX Filter mit Geometrie-Operatoren in Abfragen

ST_Centroid Abfrage

Aus:

```
(SELECT ST_Centroid(geometry) AS geometry FROM gebauede)
```

wird:

```
SELECT * FROM
(SELECT ST_Centroid(geometry) AS geometry FROM gebauede)
AS x WHERE x.geometry && ST_Envelope....
```

BBOX Filter mit Geometrie-Operatoren in Abfragen

Lösung:

- MapServer: `!BOX!`
- Mapnik: `!BBOX!`

Aus:

```
SELECT ST_Centroid(geometry) AS geometry FROM gebauede
WHERE geometry && !BOX!
```

BBOX Filter mit Geometrie-Operatoren in Abfragen

Lösung:

- MapServer: `!BOX!`
- Mapnik: `!BBOX!`

Aus:

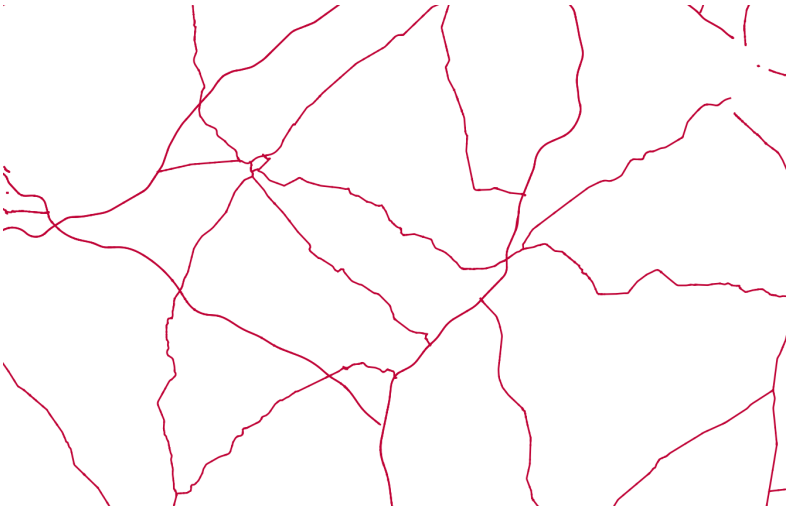
```
SELECT ST_Centroid(geometry) AS geometry FROM gebauede
WHERE geometry && !BOX!
```

wird:

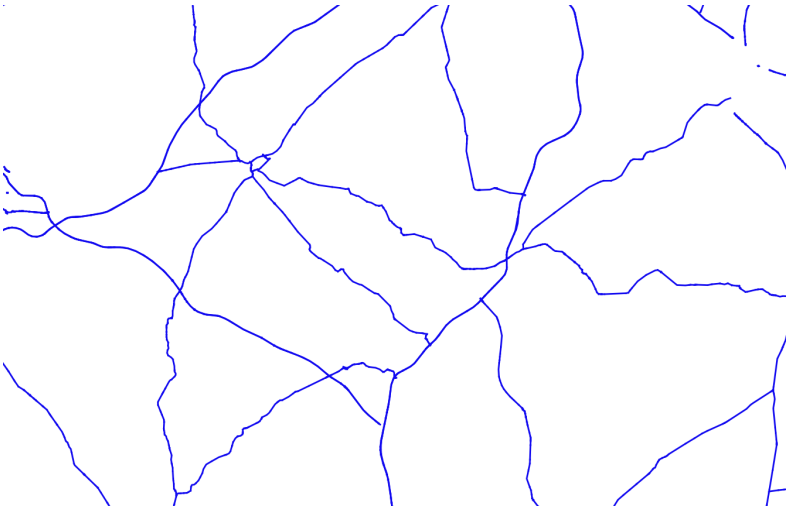
```
SELECT * FROM
(SELECT ST_Centroid(geometry) AS geometry FROM gebauede
  WHERE geometry && ST_Envelope(...))
AS x WHERE x.geometry && ST_Envelope....
```

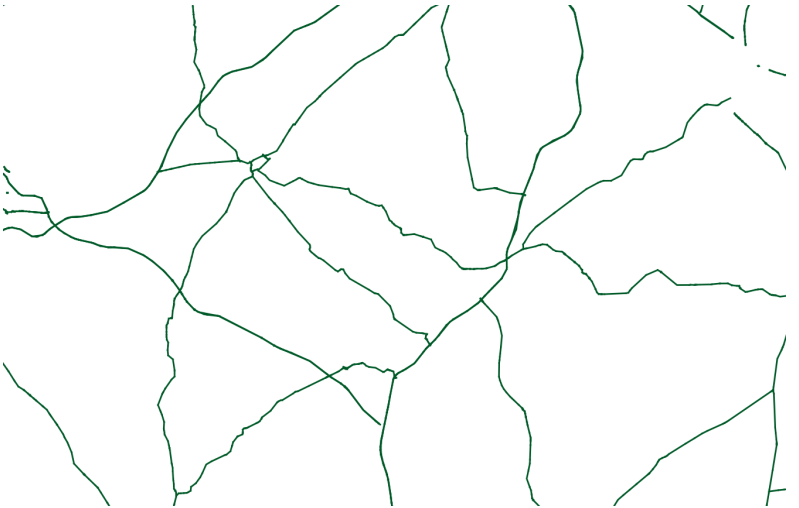
Materialized Views

siehe letzte Folien



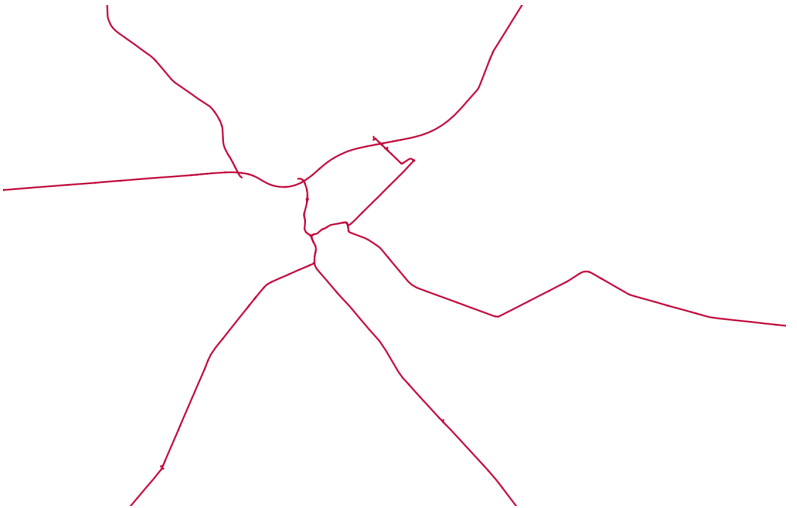
Omniscale





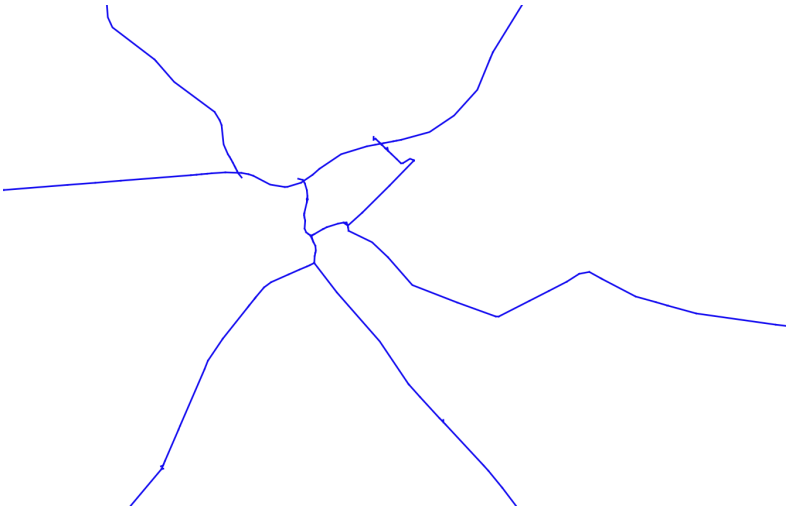
Omniscale

geometry



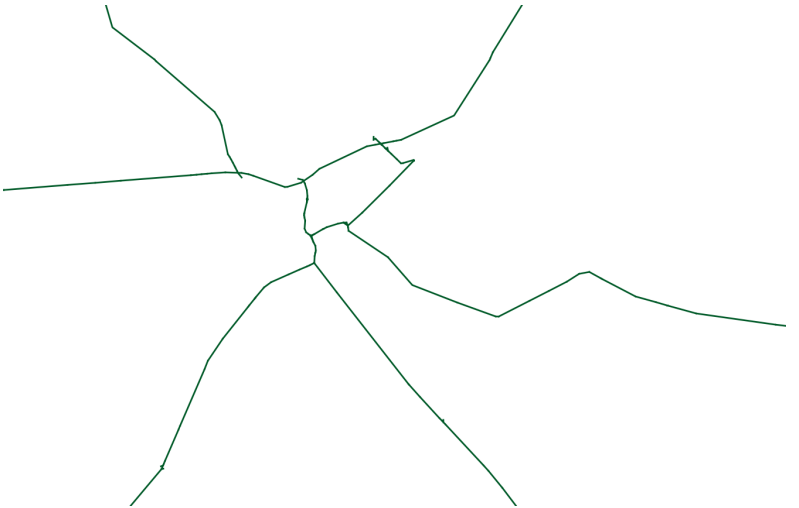
Omniscale

ST_Simplify(geometry, 75)



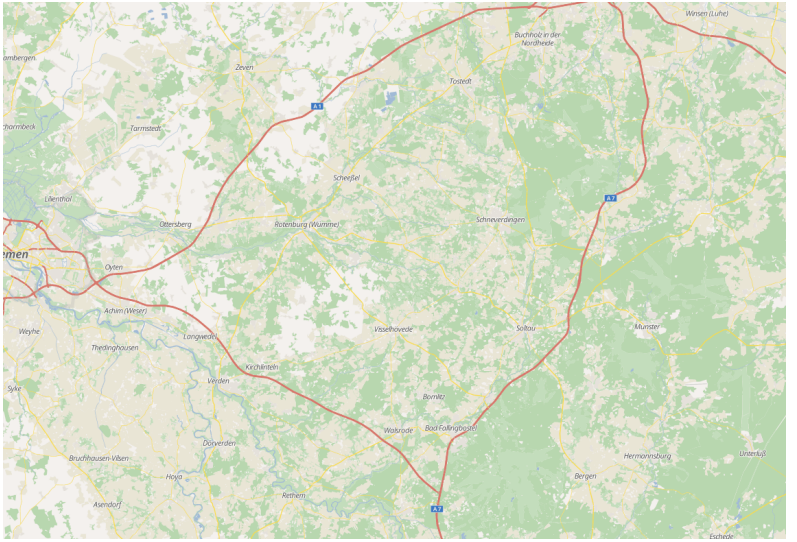
Omniscale

ST_Simplify(geometry, 300)



Omniscale

ST_Simplify(geometry, 300)



Omniscale

Generalisieren

Vereinfachen der Geometrien im Query:

```
SELECT ST_Simplify(geometry, 75) AS geometry, type FROM roads;
```

Generalisieren

Vereinfachen der Geometrien im Query:

```
  SELECT ST_Simplify(geometry, 75) AS geometry, type FROM roads;
```

Zusätzliche Tabelle mit vereinfachten Geometrien:

```
CREATE TABLE roads_gen AS
  SELECT ST_Simplify(geometry, 75) AS geometry, type FROM roads;
```

Generalisieren

Vereinfachen der Geometrien im Query:

```
  SELECT ST_Simplify(geometry, 75) AS geometry, type FROM roads;
```

Zusätzliche Tabelle mit vereinfachten Geometrien:

```
CREATE TABLE roads_gen AS
  SELECT ST_Simplify(geometry, 75) AS geometry, type FROM roads;
```

Zusätzlich filtern:

```
CREATE TABLE roads_gen AS
  SELECT ST_Simplify(geometry, 75) AS geometry, type FROM roads
  WHERE type IN ('motorway', 'trunk', 'motorway_link', 'trunk_link', 'primary');
```

Generalisieren

Vereinfachen der Geometrien im Query:

```
  SELECT ST_Simplify(geometry, 75) AS geometry, type FROM roads;
```

Zusätzliche Tabelle mit vereinfachten Geometrien:

```
CREATE TABLE roads_gen AS
  SELECT ST_Simplify(geometry, 75) AS geometry, type FROM roads;
```

Zusätzlich filtern:

```
CREATE TABLE roads_gen AS
  SELECT ST_Simplify(geometry, 75) AS geometry, type FROM roads
  WHERE type IN ('motorway', 'trunk', 'motorway_link', 'trunk_link', 'primary');
```

In Imposm mit `generalized_tables` möglich.

Landnutzung nach Typ

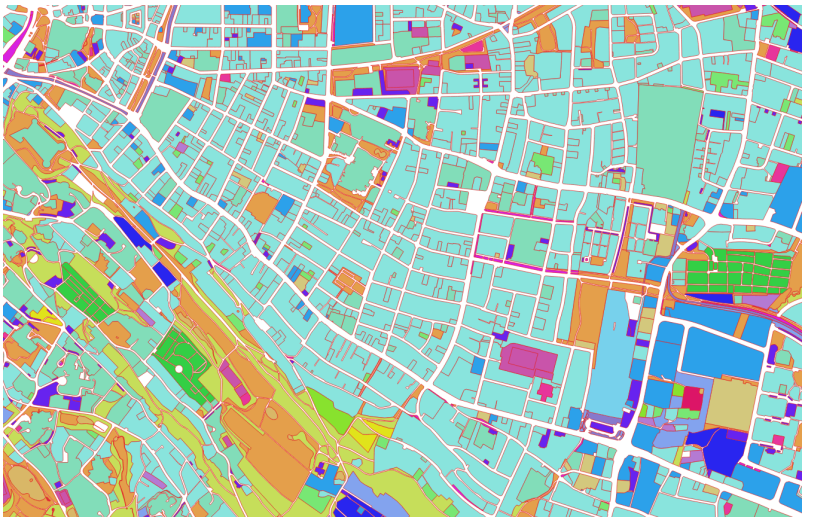


Bild: © Kartendaten Stadt Bielefeld

Omniscale

Landnutzung nach Typ vereinen

```
CREATE TABLE landnutzung_union AS
  SELECT ST_Dump(ST_Union(geometry)), typ FROM landnutzung GROUP BY typ;
```

Landnutzung nach Typ vereint

Bild: © Kartendaten Stadt Bielefeld

Omniscale

Landnutzung nach Typ

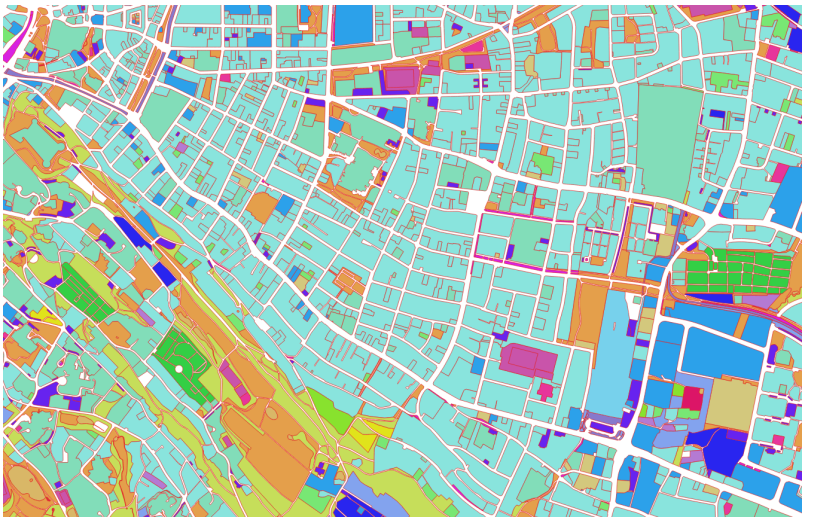


Bild: © Kartendaten Stadt Bielefeld

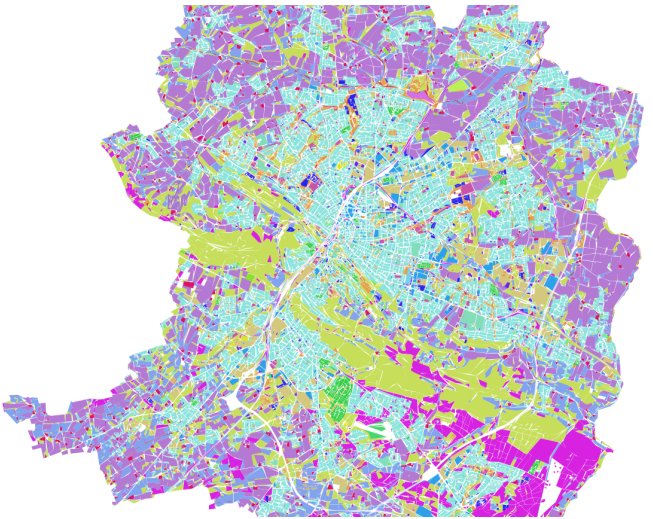
Omniscale

Vereint und generalisiert

Bild: © Kartendaten Stadt Bielefeld

Omniscale

Vereint und generalisiert



ORKa-MV

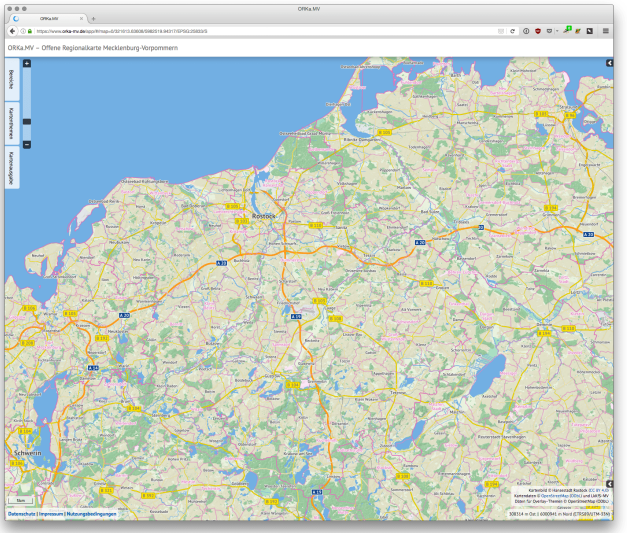
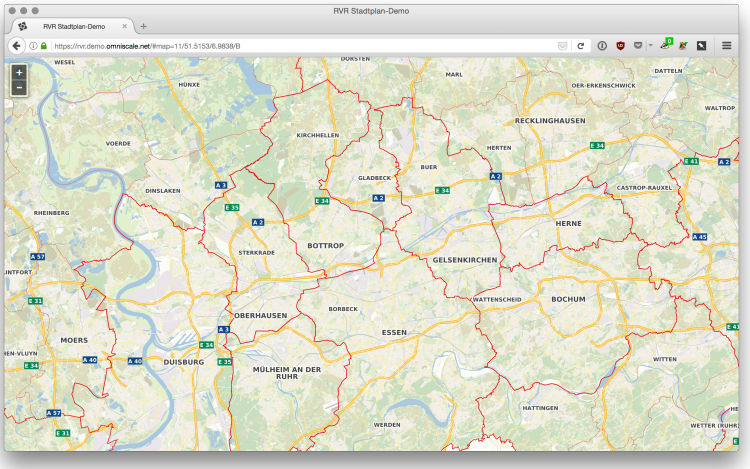


Bild: © Hansestadt Rostock (CC-By 4.0) • OpenStreetMap (ODbL) • LkkfS-MV

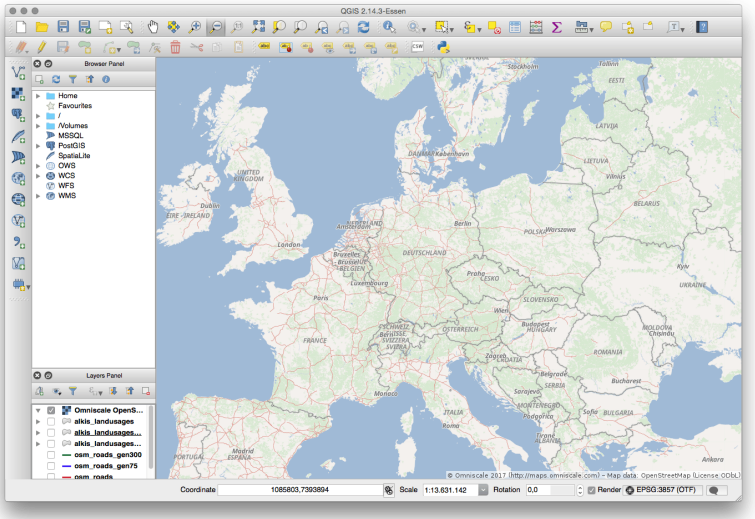
Omniscale

Stadplanwerk Regionalverband Ruhr



Omniscale

maps.omniscale.com



WMS < 1 Sekunde

Omniscale

Aktualisieren

Aktualisieren bestehender Daten

- Transaktionen nutzen
  - *Ganz oder gar nicht*
  - Lesender Zugriff weiter möglich

Aktualisieren bestehender Daten

- Transaktionen nutzen
  - *Ganz oder gar nicht*
  - Lesender Zugriff weiter möglich

Unterstützt von:

- Imposm 3
- `ogr2ogr -ds_transaction` (ab 2.0)

Neuimport aktueller Daten

- Transaktionen nutzen
  - Lesender Zugriff auf alten Datenbestand weiter möglich

Neuimport aktueller Daten

- Transaktionen nutzen
  - Lesender Zugriff auf alten Datenbestand weiter möglich

Aber Achtung:

`DROP TABLE`, `CREATE TABLE`, `TRUNCATE`, etc. blockieren lesende Operationen

Neuimport aktueller Daten

Alternative: Import in Schema

```
CREATE SCHEMA import;
CREATE TABLE import.tabelle;
INSERT INTO import.tabelle....
```

Neuimport aktueller Daten

Alternative: Import in Schema

```
CREATE SCHEMA import;
CREATE TABLE import.tabelle;
INSERT INTO import.tabelle....
```

Austausch

```
ALTER TABLE public.tabelle SET SCHEMA backup;
ALTER TABLE import.tabelle SET SCHEMA public;
```

Neuimport aktueller Daten

Alternative: Import in Schema

```
CREATE SCHEMA import;
CREATE TABLE import.tabelle;
INSERT INTO import.tabelle....
```

Austausch

```
ALTER TABLE public.tabelle SET SCHEMA backup;
ALTER TABLE import.tabelle SET SCHEMA public;
```

```
DROP TABLE backup.tabelle;
```

Neuimport aktueller Daten

Alternative: Import in Schema

Achtung: Views werden automatisch geändert

- public.tabelle → backup.tabelle

Neuimport aktueller Daten

Alternative: Import in Schema

Achtung: Views werden automatisch geändert

- public.tabelle → backup.tabelle

Neuimport mit OGR:

- `ALTER SCHEMA alkis RENAME TO alkis_backup;`
- Neuimport
- Views aktualisieren
- `DROP SCHEMA alkis_backup;`

Neuimport aktueller Daten

Alternative: Import in Schema

Neuimport mit Imposm 3:

- `imposm3 import -dbschema-import import`
- `imposm3 import -deployproduction.`

Zusammenfassung

- Daten importieren
- Strukturieren
- Homogenisieren
- Optimieren
- Aktualisieren

Fin

Danke! Fragen?

Folien:

- https://talks.omniscale.de/2017/fossgis/postgis/

Me:

- Oliver Tonnhofer, Omniscale
- tonnhofer@omniscale.de
- @oltonn

Materialized Views

Wenn Views zu langsam werden

- zu viele Tabellen
- aufwändige Operatoren

```
CREATE MATERIALIZED VIEW landnutzung AS
SELECT ST_Centroid(geometry) FROM (
  SELECT geometry FROM x
  UNION ALL
  SELECT geometry FROM x
  UNION ALL
  ...
  )
```

Materialized Views

Erstellt eine Kopie der Daten!

Aktualisierung:

```
REFRESH MATERIALIZED VIEW landnutzung;
```

Materialized Views

Fallstrick

Neuimport der Quelltabellen über `DROP TABLE` und `CREATE TABLE` nicht möglich.

Je nach Datenvolumen:

```
CREATE TABLE landnutzung AS
SELECT ST_Centroid(geometry) FROM (
    SELECT geometry FROM x
    UNION ALL
    SELECT geometry FROM x
    UNION ALL
    ...
    )
```