OSM Data Processing with PostgreSQL / PostGIS

Jochen Topf
jochentopf.com
OpenStreetMap

PostgreSQL
osm2pgsql builds ways table but not line table

19 Jun, 08:48 SimonPoole ◆ 38.2k

how to get way OSM ID lat, lon from postgisql database?

21 Jan, 07:01 SimonPoole ◆ 38.2k

Update osm2pgsql database from apidb database

14 Aug, 11:02 YUNtee 16

Nominatim PgSQL database size growing gradually

29 Jul '16, 12:36 jot 496

osm2pgsql processing a 68MB planet diff for more than 19 hours

06 Jun, 10:36 SimonPoole ◆ 38.2k

From help.openstreetmap.org
What we will talk about...

- Background: Relational Databases, Geodata
- Converting OSM Data
- Use Cases
- Tools
- Tips & Tricks, Odds & Ends
Background:

Relational Databases
Databases

Databases store and manipulate data.

There are many different ways to organize data...
Relational Databases

All data is organized in Tables.
## Relational Databases

### Table Members:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Place</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>Sydney</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>Jenny</td>
<td>New York</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>Jeremy</td>
<td>Moskow</td>
<td>55</td>
</tr>
</tbody>
</table>
### Relational Databases

**Table Members:**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Place</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>Sydney</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>Jenny</td>
<td>New York</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>Jeremy</td>
<td>Moskow</td>
<td>55</td>
</tr>
</tbody>
</table>
Relational Databases

**Table Members:**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Place</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>Sydney</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>Jenny</td>
<td>New York</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>Jeremy</td>
<td>Moskow</td>
<td>55</td>
</tr>
</tbody>
</table>
## Relational Databases

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Place</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>Sydney</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>Jenny</td>
<td>New York</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>Jeremy</td>
<td>Moskow</td>
<td>55</td>
</tr>
</tbody>
</table>
## Relational Databases

### Table Members:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Place</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>Sydney</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>Jenny</td>
<td>New York</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>Jeremy</td>
<td>Moskow</td>
<td>55</td>
</tr>
</tbody>
</table>
### Relational Databases

**Table Members:**

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Place</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>Sydney</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>Jenny</td>
<td>New York</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>Jeremy</td>
<td>Moskow</td>
<td>55</td>
</tr>
</tbody>
</table>
Data Types

Fields have a type:

- Text
- Integer
- Numeric
- Date
- ...

## Relational Databases

### Table Members:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Place</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>Sydney</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>Jenny</td>
<td>New York</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>Jeremy</td>
<td>Moskow</td>
<td>55</td>
</tr>
</tbody>
</table>
Structured Query Language
Data Access using SQL

```sql
SELECT Name
FROM Members;
```

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
</tr>
<tr>
<td>Jenny</td>
</tr>
<tr>
<td>Jeremy</td>
</tr>
</tbody>
</table>
### Data Access using SQL

```sql
SELECT Name, Age
FROM Members;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jenny</td>
<td>42</td>
</tr>
<tr>
<td>Joe</td>
<td>34</td>
</tr>
<tr>
<td>Jeremy</td>
<td>55</td>
</tr>
</tbody>
</table>
SELECT Name, Age
FROM Members
ORDER BY Age;

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>34</td>
</tr>
<tr>
<td>Jenny</td>
<td>42</td>
</tr>
<tr>
<td>Jeremy</td>
<td>55</td>
</tr>
</tbody>
</table>
Data Access using SQL

```
SELECT Name, Age
FROM Members
WHERE Age > 40
ORDER BY Name;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jenny</td>
<td>42</td>
</tr>
<tr>
<td>Jeremy</td>
<td>55</td>
</tr>
</tbody>
</table>
INSERT INTO Members (Name, Place, Age) VALUES ('Julia', 'London', 27);

UPDATE Members SET Place = 'Helsinki' WHERE Id = 2;

DELETE FROM Members WHERE Name = 'Jeremy';
INSERT INTO Members (Name, Place, Age)
VALUES ('Julia', 'London', 27);

UPDATE Members SET Place = 'Helsinki'
WHERE Id = 2;

DELETE FROM Members
WHERE Name = 'Jeremy';
INSERT INTO Members (Name, Place, Age)
VALUES ('Julia', 'London', 27);

UPDATE Members SET Place = 'Helsinki'
WHERE Id = 2;

DELETE FROM Members
WHERE Name = 'Jeremy';
Advanced SQL: Aggregate Functions

SELECT Avg(Age) FROM Members;

<table>
<thead>
<tr>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
</tr>
</tbody>
</table>
Advanced SQL: JOIN
The Magic

You define a structure ("schema").
You add data.
You ask for data back.
The database software does everything else.
Not so magic...

Performance can depend on structure

You still need to know a bit…
Indexes

Indexes allow faster access for some queries

Tradeoff: Indexes need space and need to be updated vs. faster queries
Relational Databases

PostgreSQL
MySQL, MariaDB
SQLite
...
Relational Databases

PostgreSQL

MySQL, MariaDB

SQLite
PostgreSQL

Open Source

lots of features

good documentation, books, etc.

popular, great eco-system, well-supported

powerful plugin system
Background:

Geodata
Simple Feature Model

- Point
- MultiPoint
- LineString
- MultiLineString
- Polygon
- MultiPolygon
We want to store this in a database
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>151.22</td>
<td>-33.85</td>
</tr>
<tr>
<td>2</td>
<td>Jenny</td>
<td>-74.01</td>
<td>40.70</td>
</tr>
<tr>
<td>3</td>
<td>Jeremy</td>
<td>37.61</td>
<td>55.95</td>
</tr>
</tbody>
</table>
A Better Approach...

- Text
- Integer
- Numeric
- Date
- Geometry

...
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Geom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>POINT(151.22, -33.85)</td>
</tr>
<tr>
<td>2</td>
<td>Jenny</td>
<td>POINT(-74.01, 40.70)</td>
</tr>
<tr>
<td>3</td>
<td>Jeremy</td>
<td>POINT(37.61, 55.95)</td>
</tr>
</tbody>
</table>
PostGIS: Plugin

CREATE EXTENSION postgis;
PostGIS: Datatypes

GEOMETRY

GEOMETRY (POINT)
GEOMETRY (LINESTRING)
GEOMETRY (POLYGON)

(also: GEOGRAPHY)
PostGIS: CRS

PostGIS knowns > 5000 Coordinate Systems (CRS/SRS)

Each Geometry associated with SRID.

Allows Transformations

Mix and match data sources
Most important CRSes:

WGS84 – EPSG:4326

Web Mercator – EPSG:3857
PostGIS: Datatypes

GEOMETRY (POINT, 4326)
GEOMETRY (LINESTRING, 4326)
GEOMETRY (POLYGON, 4326)
Coordinates

Always first X axis, then Y axis (as in mathematics).

so: longitude first, then latitude.
Well Known Text (WKT)

POINT(4 3)

LINESTRING(12 4, 3 2, 7, 9)

POLYGON((0 0, 4 0, 4 4, 0 4, 0 0))

MULTIPOINT / -LINESTRING / -POLYGON
PostGIS: Indexes

Normal indexes are good for 1-dimensional data

Spatial indexes are good for 2/3-dimensional data (R-tree)
PostGIS: Operations

Huge number of operations on spatial data
PostGIS: ST_Contains
PostGIS: ST_Union
PostGIS: ST_Intersection
PostGIS: ST_Buffer

ST_Buffer(geom, 1) (by default join=round)

ST_Buffer(geom, 1, 'join=mitre')
ST_Buffer(geom, 1, 'join=bevel')
PostGIS: ST_ShortestLine

Image: CC-BY-NC-SA
www.h2gis.org
Converting OSM Data
OSM Data Model
Mismatch

OSM Data Model

Relational / Simple Feature Data Model
Mismatch

OSM Data Model

Conversion

Relational / Simple Feature Data Model
Conversion: Selection

What data do we actually need?

- nodes, ways, relations?
- user id, timestamp, version, …?

which tags?
Conversion: Data Types

tags in OSM:
key → value (both text)

Map to:
text, integer, boolean, enums, ...
Conversion: Tags → Attributes

tags in OSM are flexible, table columns are fixed

<table>
<thead>
<tr>
<th>Highway</th>
<th>Name</th>
<th>Oneway</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary</td>
<td>Main St</td>
<td>false</td>
</tr>
<tr>
<td>residential</td>
<td>Elm St</td>
<td>true</td>
</tr>
<tr>
<td>trunk</td>
<td></td>
<td>true</td>
</tr>
</tbody>
</table>
Conversion: Tags → Attributes

tags in OSM are flexible, table columns are fixed

<table>
<thead>
<tr>
<th>Highway</th>
<th>Name</th>
<th>Oneway</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary</td>
<td>Main St</td>
<td>false</td>
</tr>
<tr>
<td>residential</td>
<td>Elm St</td>
<td>true</td>
</tr>
<tr>
<td>trunk</td>
<td></td>
<td>true</td>
</tr>
</tbody>
</table>
Conversion: Tags $\rightarrow$ Attributes

tags in OSM are flexible, table columns are fixed

<table>
<thead>
<tr>
<th>Highway</th>
<th>Name</th>
<th>Oneway</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary</td>
<td>Main St</td>
<td>false</td>
</tr>
<tr>
<td>residential</td>
<td>Elm St</td>
<td>true</td>
</tr>
<tr>
<td>trunk</td>
<td></td>
<td>true</td>
</tr>
</tbody>
</table>
### Conversion: hstore and JSON

<table>
<thead>
<tr>
<th>Place</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>city</td>
<td>de: München, en: Munich</td>
</tr>
<tr>
<td>city</td>
<td>de: Aachen, fr: Aix-la-Chapelle</td>
</tr>
<tr>
<td>village</td>
<td>de: Lübben, hsb: Lubin</td>
</tr>
</tbody>
</table>
Conversion: Tables

split data into tables...

few tables vs. many tables

by geometry type and/or by subject type
Split by Geometry Type

Tables:

nodes
ways
areas
<table>
<thead>
<tr>
<th>Geometry/Feature Class</th>
<th>Tables:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>restaurants</td>
</tr>
<tr>
<td></td>
<td>bus_stops</td>
</tr>
<tr>
<td></td>
<td>addresses</td>
</tr>
<tr>
<td></td>
<td>places</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Conversion: Handling lists
	nodes in ways

members in relations

tags in nodes, ways, or relations
## Conversion: Way Nodes

<table>
<thead>
<tr>
<th>Ways</th>
<th>WayNodes</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>WayId</td>
<td>WayId</td>
<td>Nodeld</td>
</tr>
<tr>
<td>Version</td>
<td>NodelId</td>
<td>Version</td>
</tr>
<tr>
<td>UserId</td>
<td>SeqNo</td>
<td>UserId</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### Conversion: Way Nodes

#### Ways
- **WayId**
- **Version**
- **UserId**
- **NodeId**

#### Nodes
- **NodeId**
- **Version**
- **UserId**
- **...**

---

**Array of Ids**
Conversion: Relation Members

similar to way nodes

but

array of tuple (type, id, role)
Conversion: Geometry

Nodes → Points

Ways → LineStrings / Polygons

Multipolygon relations → Polygons

Route relations → MultiLineStrings

...
Conversion: Geometry

Generalized geometries

For lower zoom levels / small scales

Selection – Merging – Simplification
Conversion: Where?

Conversion can happen

1. before import in code

2. after import in the DB
Conversion: Where?

Conversion can happen

1. before import in code

2. after import in the DB
Conversion: Where?

Conversion can happen
1. before import in code
2. after import in the DB

fast
flexible
Conversion: Assemble Lines

Take Locations from Nodes

Assemble them into LineStrings
Conversion: Node locations

Where to store node locations?

1. in the database
2. in specialized index
Conversion: Polygons

Assemble (Multi)Polygons from Ways/Relations
One-off Import vs. Import + Updates

lots of trade-offs

not all software/schemas support updates
Complete data needed for updates
Complete data needed for updates
Complete data needed for updates

Two kinds of data:
1. The data you need for you application
2. The data needed to allow updating
Complete data needed for updates

Two kinds of data:
1. The data you need for your application
2. The data needed to allow updating

Where? Database? External Storage?
Snapshot vs. History

most use cases only need current OSM data

some need history of OSM data
Snapshot vs. History

most use cases only need 
current OSM data

some need 
history of OSM data

Much more effort needed!
Use Cases
Use Cases

API DB
Rendering
Geocoding
Routing
Analytics
API DB

Schema used in the main OSM database

PostgreSQL - No PostGIS!

Normal access via HTTP API

You can run your own
API DB

Needs all (also historical) data

Multiple writers, transactions

Allow bounding-box download

Allow read/write access

Create full dumps and replication diffs
Rendering

Turning data into maps

Render into bitmap, vector tiles, etc.
Rendering
Get all data for an area quickly
Multiple layers
Create generalized geometries
One writer, multiple reader
Regular updates
Geocoding

“Search“

Geocoding – Address to Location

Reverse Geocoding – Location to Address
Geocoding

Build address hierarchy

Quick “fuzzy“ search

One writer, multiple reader

Regular updates
Routing

Using PostgreSQL plugin PgRouting

Flexible, but slow
Routing

Build network of streets

Calculate weights

Find route through network
Analytics

Statistics

Comparing data

Conflating data

Many diverse needs
Example 1: Wind Power

Find a place that ...

... has lots of steady winds
... is near existing high voltage lines
... is far from residential areas
Example 2: Public Transport

How far is the nearest public transport stop?

How many people live where the nearest stop is more than x meters away?

Where should a new bus route go?
Example 3: OSM Contributors

Who are the most active OSM contributors?

What kinds of things do they map?

Where do they do their mapping?
Analytics

Flexible data model

Use of many geometric operations

Batch processing in multiple steps
Tools
psql

SELECT osm_id, osm_type, highway, substr(ST_AsText(geom), 1, 46) AS geom FROM highway_motorway LIMIT 10;

<table>
<thead>
<tr>
<th>osm_id</th>
<th>osm_type</th>
<th>highway</th>
<th>geom</th>
</tr>
</thead>
<tbody>
<tr>
<td>3700602</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.3957525 51.4045029, 12.3940079 51.3962457)</td>
</tr>
<tr>
<td>4068262</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.1950993 51.4221162, 12.195996 51.4221162)</td>
</tr>
<tr>
<td>4068272</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.1956903 51.4236168, 12.194645 51.4221162)</td>
</tr>
<tr>
<td>4068276</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.2079569 51.426431, 12.2111331 51.4221162)</td>
</tr>
<tr>
<td>4071131</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.375885 51.4084059, 12.3754508 51.4084059)</td>
</tr>
<tr>
<td>4071132</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.355407 51.4120578, 12.353102 51.4120578)</td>
</tr>
<tr>
<td>4071145</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.2989367 51.4155116, 12.2967172 51.4155116)</td>
</tr>
<tr>
<td>4071159</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.1805927 51.3958729, 12.1805298 51.3958729)</td>
</tr>
<tr>
<td>4100106</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.1999562 51.4310166, 12.1979877 51.4310166)</td>
</tr>
<tr>
<td>4259290</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.5483554 51.3650722, 12.542398 51.3650722)</td>
</tr>
</tbody>
</table>

(10 rows)
```sql
SELECT osm_id, osm_type, highway, substr(ST_AsText(geom), 1, 46) AS geom
FROM highway_motorway
LIMIT 10;
```

<table>
<thead>
<tr>
<th>osm_id</th>
<th>osm_type</th>
<th>highway</th>
<th>geom</th>
</tr>
</thead>
<tbody>
<tr>
<td>3700602</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.3957525 51.4045029, 12.3940679 51.4)</td>
</tr>
<tr>
<td>4068262</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.1950993 51.4221162, 12.195596 51.4)</td>
</tr>
<tr>
<td>4068272</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.1956983 51.4236168, 12.194645 51.4)</td>
</tr>
<tr>
<td>4068276</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.2079569 51.426431, 12.2111331 51.4)</td>
</tr>
<tr>
<td>4071131</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.375885 51.4084059, 12.3754508 51.4)</td>
</tr>
<tr>
<td>4071132</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.355407 51.4120578, 12.353102 51.4)</td>
</tr>
<tr>
<td>4071145</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.2989367 51.4155116, 12.2967172 51.4)</td>
</tr>
<tr>
<td>4071159</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.1899527 51.3958729, 12.1805298 51.4)</td>
</tr>
<tr>
<td>4100106</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.1999562 51.4310166, 12.1979877 51.4)</td>
</tr>
<tr>
<td>4259290</td>
<td>way</td>
<td>motorway</td>
<td>LINESTRING(12.5483554 51.3050722, 12.5492398 51.4)</td>
</tr>
</tbody>
</table>

(10 rows)
Osmosis

https://wiki.osm.org/wiki/Osmosis

Use case: API DB, Analytics
Updates: Yes
Schema: Several
Status: Not being maintained
Osmosis Schemas

API DB (version 0.6)

PostGIS Snapshot Schema (uses hstore)

PostGIS Simple Schema (no hstore)

(API DB MySQL <0.6)
osm2pgsql

Used in “standard“ OSM rendering toolchain

https://wiki.osm.org/wiki/Osm2pgsql

Use case: Rendering
Updates: Yes
Schema: Few tables (hstore optional)
Status: Maintained
Imposm3

Alternative rendering toolchain

https://imposm.org

Use case: Rendering
Updates: Yes
Schema: Many tables
Status: Actively maintained
Nominatim
Standard OSM search/geocoding

https://nominatim.org

Use case: (Reverse) Geocoding
Schema: Optimized for geocoding
Status: Actively maintained

Uses osm2pgsql (with special plugin)
Osmium

https://osmcode.org/osmium-tool/

Use case: Analytics, (Rendering)

Updates: No
Schema: Simple
Status: Actively maintained
Simple to run for ad-hoc use
osm-postgresql-experiments

Experimental, very flexible data import

https://github.com/osmcode/osm-postgresql-experiments

Use case: Rendering, Analytics
Updates: (Yes)
Schema: Flexible
Status: Experimental
osm2pgRouting

Importer für PgRouting

https://github.com/pgrouting/osm2pgRouting

Use case: Routing
Updates: No
Schema: PgRouting
Status: Maintained
Tips & Tricks

Odds & Ends
Quantity Structure

How much disk space do I need?
How much memory do I need?
How long will an import take?
Quantity Structure

How much disk space do I need?

How much memory do I need?

How long will an import take?

hundreds of Gbytes for full planet
Quantity Structure

How much disk space do I need?

How much memory do I need?

How long will an import take?

hundreds of Gbytes for full planet

More!
Quantity Structure

- How much disk space do I need?
- How much memory do I need?
- How long will an import take?

- hundreds of Gbytes for full planet
- many hours if not days for planet
Start small...

Do not try to import the whole planet at first!

Start small
(e.g. with data for a city)
and
work your way up
Minimize data

1. Filter data outside DB if you can
2. Import data into DB
Importing Data

1. load data
2. create indexes
3. ANALYZE
Performance Tuning

You will need to tune your PostgreSQL!

Settings in postgresql.conf:
shared_buffers, work_mem, maintenance_work_mem, fsync,
synchronous_commit, checkpoint_timeout,
checkpoint_completion_target, ...
Indexes

Learn how indexes work and when they are used

Also for spatial indexes!

Use EXPLAIN command
The COPY command

COPY instead of INSERT
more efficient, use it if possible
Learning Curve

PostgreSQL / PostGIS is an incredible powerful tool

“Magic“ working of PostgreSQL can be surprising

Configure the logs and look at them
Learning Curve

Spatial operations are extra magic!

Again: Start small
Always growing

Database will grow over time (not only because of more OSM data)

VACUUM

Still grow more
Other SQL Databases

MySql, MariaDB

Oracle Spatial

Sqlite (Library, not Server)
Other SQL Databases

MySql, MariaDB

Oracle Spatial

Sqlite (Library, not Server)

godeata support lacking
Other SQL Databases

- MySql, MariaDB
- Oracle Spatial
- Sqlite (Library, not Server)

- geodata support lacking
- proprietary
Other SQL Databases

- geodata support lacking
  - MySql, MariaDB
  - Oracle Spatial
- Sqlite (Library, not Server)
  - proprietary
  - not as powerful, problems with huge datasets, but can be useful for some applications