OVERPASS TURBO GOES POSTGIS

Frederik Ramm

<frederik@remote.org>

State of the Map EU 2025, Dundee

OPENSTREETMAP DATABASES

- Standard API ("editing API")
- 2007: "XAPI" (2011: jXAPI)
- 2011: Overpass API

Overpass API Query Form

Query

extract POIs for a region

1

I am currently working on a list of available hotels in the Republic of Georgia. As the OSM-Database has lots of POIs, is it possible to access this data for that purpose? I would like to run a query to retrieve hotels with addresses, phone numbers and other details that are entered. The coordinates would not matter for my purpose.

The result should be an Excel-file or similar table format.

How can I do this? All help appreciated!

query | pois | lists | database

asked **08 May '12, 12:26**

moszkva ter
2.1k●22●39●60
accept rate: 17%

4

You can get the data in JSON format in two steps from Overpass API:

First, obtain the area by pasting the following request

```
<osm-script output="json">

<coord-query lat="42.2" lon="43.3"/>
<print/>

</osm-script>
```

into the Overpass API query form.

From the result, you can read off the area with name "Georgia" that the relevant area id is 3600028699.

Now you can query for the really wanted data, again on Overpass API query form:

```
<osm-script output="json">

<query type="node">
  <area-query ref="3600028699"/>
  <has-kv k="tourism" v="hotel"/>
</query>
<print/>
<area-query ref="3600028699"/>
<query type="way">
  <recurse type="node-way"/>
  <has-kv k="tourism" v="hotel"/>
</query>
<print/>

</osm-script>
```

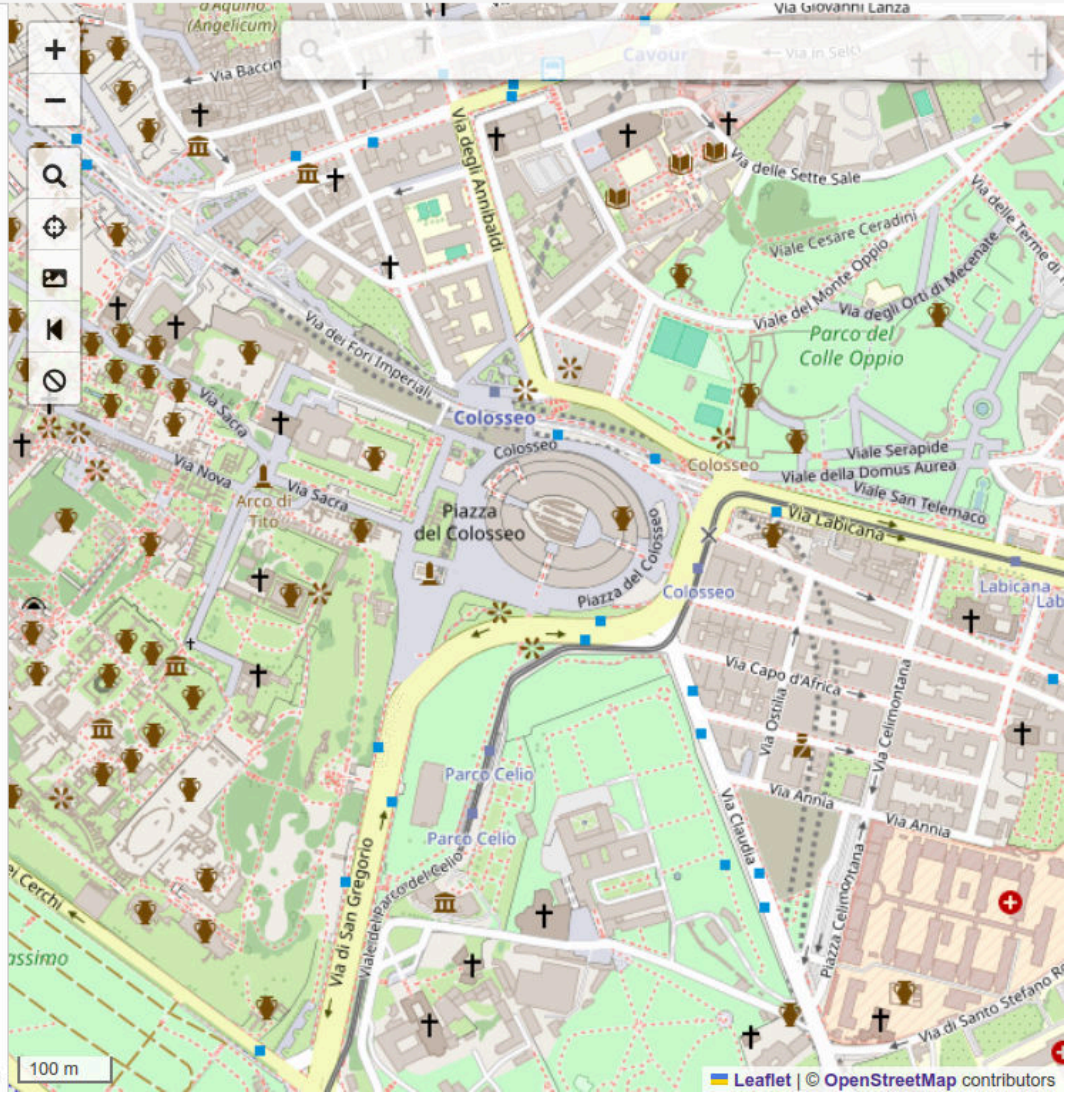
This results in a 92 KB big JSON file containing all information about nodes or ways in Georgia tagged with `tourism=hotel`. If you want additionally hostels, repeat the second step with `tourism=hostel` and so on.

OPENSTREETMAP DATABASES

- Standard API ("editing API")
- 2007: "XAPI" (2011: jXAPI)
- 2011: Overpass API
- 2013: Overpass Turbo

Run | Share | Export | Wizard | Save | Load | Settings | Help | overpass turbo | Map | Data

```
/*
This is an example Overpass query.
Try it out by pressing the Run button above!
You can find more examples with the Load tool.
*/
node
  [amenity=drinking_water]
  ({{bbox}});
out;
```



Download amenities kmz file

0

Hello, I would like to download amenities of a certain type and area, preferably in kmz format. For example I would like to have a kmz file of amenity=drinking_water in Berlin (more or less). How to do it?

download | export | amenities | kmz

asked **24 Aug '21, 12:35**

Leszek
11●1●1●2
accept rate: 0%

One Answer:

active answers | oldest answers | newest answers | popular answers

1

You can use overpass turbo to retrieve specific POIs from OSM.

The following query will search for all drinking water POIs in the currently visible area:

```
[out:json][timeout:25];
// gather results
(
  nwr["amenity"="drinking_water"]({{bbox}});
  nwr["drinking_water"="yes"]({{bbox}});
);
// print results
out body;
>;
out skel qt;
```

You can run it here. Hit run. Then go to Export -> Save as KML.

permanent link

answered **28 Oct '21, 10:11**

scai ♦

33.3k ● 21 ● 309 ● 459

accept rate: 23%

Thank you very much!

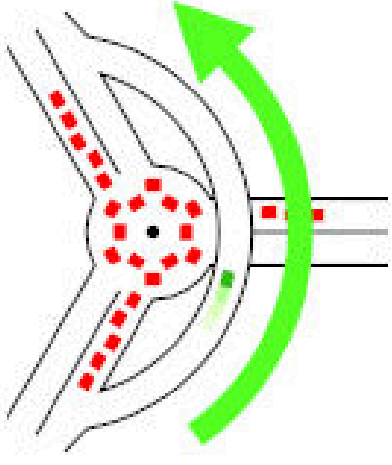
Leszek (28 Oct '21, 10:16)

PROBLEM:

- nobody has a PostGIS database with OSM at hand "just like that"
- many won't even be running Linux
- nobody is stupid enough to run a PostGIS open to everyone

NOBODY?

INTRODUCING POSTPASS



Overpass
API

PostGIS

INTRODUCING POSTPASS

- on the server side, a small wrapper around PostGIS: github.com/woodpeck/postpass
- currently runs on Geofabrik servers: github.com/woodpeck/postpass-ops

IT GOES LIKE THIS:

```
curl -g https://postpass.geofabrik.de/api/0.2/interpreter \
   --data-urlencode "data=
     SELECT tags->>'name' as name, geom
     FROM postpass_point
     WHERE tags->>'amenity'='fast_food'
       AND geom && st_makeenvelope(
          8.34, 48.97, 8.46,49.03, 4326)"
```

→ returns a GeoJSON FeatureCollection

SAFETY MEASURES

Against "drop table X" and so on:

- use read-only PostgreSQL account

Against "select * from everything":

- first, wrap query in an EXPLAIN() (side effect: syntax check)
- then choose slow, medium, or fast queue – with faster queues having more workers

EXAMPLES

FAST FOOD IN KARLSRUHE

Overpass

```
[out:json];
node["amenity"="fast_food"]
   (48.97,8.34,49.03,8.46);
out;
```

approx. 350ms

Postpass

```
SELECT geom, tags
FROM postpass_point
WHERE tags->>'amenity'='fast_food'
   AND way && st_makeenvelope(
      8.34,48.97,8.46,49.03,4326)
```

approx. 180ms

FAST FOOD IN KARLSRUHE

Overpass

```
data=[out:json];
nwr["amenity"="fast_food"]
   (48.97,8.34,49.03,8.46);
out geom;
```

approx. 650ms

Postpass

```
SELECT geom, tags
FROM postpass_pointpolygon
WHERE tags->>'amenity'='fast_food'
   AND geom && st_makeenvelope(
      8.34,48.97,8.46,49.03,4326)
```

approx. 250ms

DATABASE SCHEMA

DATABASE SCHEMA IS CLOSE TO "STANDARD" OSM2PGSQL:

- basic tables postpass_point, _line, _polygon
- columns osm_type, osm_id, tags, geom (EPSG:4326)
- "tags" is a JSON (not hstore) column
- convenience views postpass_pointline, _pointpolygon, _linepolygon, _pointlinepolygon
- "middle" tables

details: https://github.com/woodpeck/postpass-ops/blob/main/SCHEMA.md

MORE EXAMPLES

TRAM LINE 3 IN KARLSRUHE

Overpass

```
rel[ref=3][route=tram]
   (48.97,8.34,49.03,8.46);>;
out;
```

approx. 600ms

approx. 850ms for all lines

Postpass

```
SELECT geom, tags
FROM postpass_line
WHERE tags->>'route'='tram'
   AND tags->>'ref'='3'
   AND geom && st_makeenvelope(
      8.34,48.97,8.46,49.03,4326)
```

approx. 250ms

approx. 650ms for all lines

ADDR:STREET WITHOUT ADDR:POSTCODE

Overpass

```
[out:json];
  nw["addr:street"]
     [!"addr:postcode"]
        (48.97,8.34,49.03,8.46);
>;
out skel;
```

approx. 600ms

Postpass

```
SELECT osm_type,osm_id,geom
FROM postpass_pointpolygon
WHERE tags?'addr:street'
   AND NOT tags?'addr:postcode'
   AND geom && st_makeenvelope(
      8.34,48.97,8.46,49.03,4326)
```

approx. 300ms

ALL THINGS WITH REF=15398 ON THE PLANET

Overpass

```
[out:json];
nwr[ref="15398"];
>;
out;
```

approx. 300ms

Postpass

```
SELECT osm_type, osm_id, geom
FROM postpass_pointlinepolygon
WHERE tags->>'ref'='15398'
```

approx. 300s (!)

ALL THINGS WITH REF=15398 ON THE PLANET

Overpass

```
[out:json];
nwr[ref="15398"];
>;
out;
```

approx. 300ms

Postpass

```
SELECT osm_type, osm_id, geom
FROM postpass_pointlinepolygon
WHERE tags @> '{ "ref" : "15398" }'
```

approx. 30ms (!)

ADMIN BOUNDARIES NAMED "DUNDEE"

Overpass

```
[out:json];
rel[name="Dundee"]
 [boundary="administrative"];
>;
out;
```

approx. 400ms

Postpass

```
SELECT osm_id, geom
FROM postpass_polygon
WHERE tags->>'name'='Dundee'
AND tags @> '{ "boundary" :
   "administrative",
   "name" : "Dundee" }'
```

approx. 300ms

POSTPASS IS GEOMETRY BASED

Sucks at problems like:

- find ways tagged X that are in a relation tagged Y
- do stuff with un-tagged notes or ways

You *can* do some things with the so-called "middle" tables", but you're better off with Overpass for these.

OVERPASS TURBO INTEGRATION

- Overpass Turbo has always had a data source selector: {{data:...}}
- Overpass Turbo converts Overpass data to GeoJSON – not needed here
- only minimal changes were required to support Postpass
- live since March 2025 (thanks Martin)

OVERPASS TURBO INTEGRATION

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT geom,tags
FROM postpass_point
WHERE tags->>'amenity'='fast_food'
   AND geom && {{bbox}}
```

(link)

NON-GEOMETRY QUERIES

How many "amenities" are there of each type in the bounding box?

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/,geojson=false

SELECT count(*), tags->>'amenity' as amenity
FROM postpass_point
    WHERE tags?'amenity'
    AND geom && {{bbox}}
GROUP BY amenity ORDER BY count DESC
```

(link)

(with curl, use --data-urlencode "options[geojson]=false")

NON-GEOMETRY QUERIES

Sum of highway lengths by type

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/,geojson=false

SELECT sum(st_length(geom::geography)), tags->>'highway' as highway
FROM postpass_line
    WHERE tags?'highway'
    AND geom && {{bbox}}
GROUP BY highway ORDER BY sum DESC
```

(link)

AGGREGATION

Sum up bike parking capacity near stations

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT SUM((bikeparking.tags->>'capacity')::integer) AS cap,
   station.tags->>'name' AS station, station.geom
FROM postpass_pointpolygon bikeparking,
   postpass_pointpolygon station
WHERE station.tags->>'railway' in ('station','halt')
AND bikeparking.tags->>'amenity'='bicycle_parking'
AND station.geom && {{bbox}}
AND st_dwithin(station.geom,bikeparking.geom,.001)
GROUP BY station.tags->>'name', station.geom
```

(link)

AGGREGATION

Sum up bike parking capacity near stations

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT SUM((bikeparking.tags->>'capacity')::integer) AS cap,
   station.tags->>'name' AS station, station.geom
FROM postpass_pointpolygon bikeparking,
   postpass_pointpolygon station
WHERE station.tags->>'railway' in ('station','halt')
AND bikeparking.tags->>'amenity'='bicycle_parking'
AND station.geom && {{bbox}}
AND st_dwithin(station.geom,bikeparking.geom,.001)
GROUP BY station.tags->>'name', station.geom
```

AGGREGATION

Sum up bike parking capacity near stations

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT SUM((bikeparking.tags->>'capacity')::integer) AS cap,
   station.tags->>'name' AS station, station.geom
FROM postpass_pointpolygon bikeparking,
   postpass_pointpolygon station
WHERE station.tags->>'railway' in ('station','halt')
AND bikeparking.tags->>'amenity'='bicycle_parking'
AND station.geom && {{bbox}}
AND st_dwithin(station.geom,bikeparking.geom,.001)
GROUP BY station.tags->>'name', station.geom
```

AGGREGATION

Sum up bike parking capacity near stations

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT SUM((bikeparking.tags->>'capacity')::integer) AS cap,
   station.tags->>'name' AS station, station.geom
FROM postpass_pointpolygon bikeparking,
   postpass_pointpolygon station
WHERE station.tags->>'railway' in ('station','halt')
AND bikeparking.tags->>'amenity'='bicycle_parking'
AND station.geom && {{bbox}}
AND st_dwithin(station.geom,bikeparking.geom,.001)
GROUP BY station.tags->>'name', station.geom
```

GEOMETRY OPERATIONS

find overlapping landuses in area

```
{{data:sql,server=https://postpass.geofabrik.de/api/0.2/}}

SELECT inter, landuse1, landuse2 FROM (
    SELECT
      st_intersection(a.geom, b.geom) as inter,
      a.tags->>'landuse' as landuse1,
      b.tags->>'landuse' as landuse2
    FROM postpass_polygon a, postpass_polygon b
    WHERE a.osm_id < b.osm_id
    AND a.geom &&{{bbox}}
    AND b.geom &&{{bbox}}
    AND a.tags?'landuse' AND b.tags?'landuse'
    AND st_overlaps(a.geom, b.geom)
) WHERE st_area(inter)>0.000001
```

(link)

SOMEWHERE ON THE FEDIVERSE

"I dare you to find a town in England without a Wetherspoons"

https://en.osm.town/@amapanda/115536969723066452

OVERPASS TURBO INTEGRATION

N.B. Postpass is also supported by **Ultra,** an Overpass-like thing with vector tiles!

EXPLAIN ENDPOINT

(thanks to Daniel Schep)

Postpass lets you run an EXPLAIN if you use the endpoint "explain" instead of "interpreter":

```
curl -g https://postpass.geofabrik.de/api/0.2/explain \
   --data-urlencode "data=
     SELECT tags->>'name' as name, geom
     FROM postpass_point
     WHERE tags->>'amenity'='fast_food'
       AND geom && st_makeenvelope(
          8.34, 48.97, 8.46,49.03, 4326)"
```

```
{
  "plan": [
    {
      "Plan": {
        "Alias": "postpass_point",
        "Async Capable": false,
        "Filter": "((tags ->> 'amenity'::text)='fast_food'::text)",
        "Index Cond": "(geom && '010300..'::geometry)",
        "Index Name": "postpass_point_geom_idx",
        "Node Type": "Index Scan",
        "Parallel Aware": false,
        "Plan Rows": 1,
        "Plan Width": 64,
        "Relation Name": "postpass_point",
        "Scan Direction": "Forward",
        "Startup Cost": 0.55,
        "Total Cost": 185.57
      }
    }
  ],
  "queue": "quick"
}
```

Feature Comparison

	Overpass	Postpass
request with bbox	good	often faster
request without bbox	good	good (use @>)
raw OSM data	good	difficult
aggregations	limited	good
geometry operations	limited	good

THANK YOU –

PATCHES WELCOME!

Frederik Ramm <frederik@remote.org>