

Dynamic GIS Layers with QGIS and Spatial Databases



by Karsten Vennemann



Overview

- My Background
- Inspiration for the talk
 - blog post/video from 2013 by Tim Sutton - dynamic Queries in QGIS
 - python code from 2014 PostGIS Cookbook by Paolo Corti
- Examples of dynamic layers in QGIS
 - remote GeoJSON, QGIS Geometry Generator
 - DB Manager connections and database views
 - File based spatialite and geopackage
 - PostGIS
 - Database views including psql / plpython functions



Adding Remote Data- GeoJSON Example

Q Data Source Manager | Browser | Vector

Source Type

File Directory Database Protocol: HTTP(S), cloud, etc.

Encoding: Automatic

Protocol

Type: GeoJSON
URI: https://d2ad6b4ur7yvpq.cloudfront.net/naturalearth-3.3.0/ne_50m_populated_places.geojson

Authentication

Configurations Basic

User name:
Password: Optional

Warning: credentials stored as plain text in project file.
[Convert to configuration](#)

Types

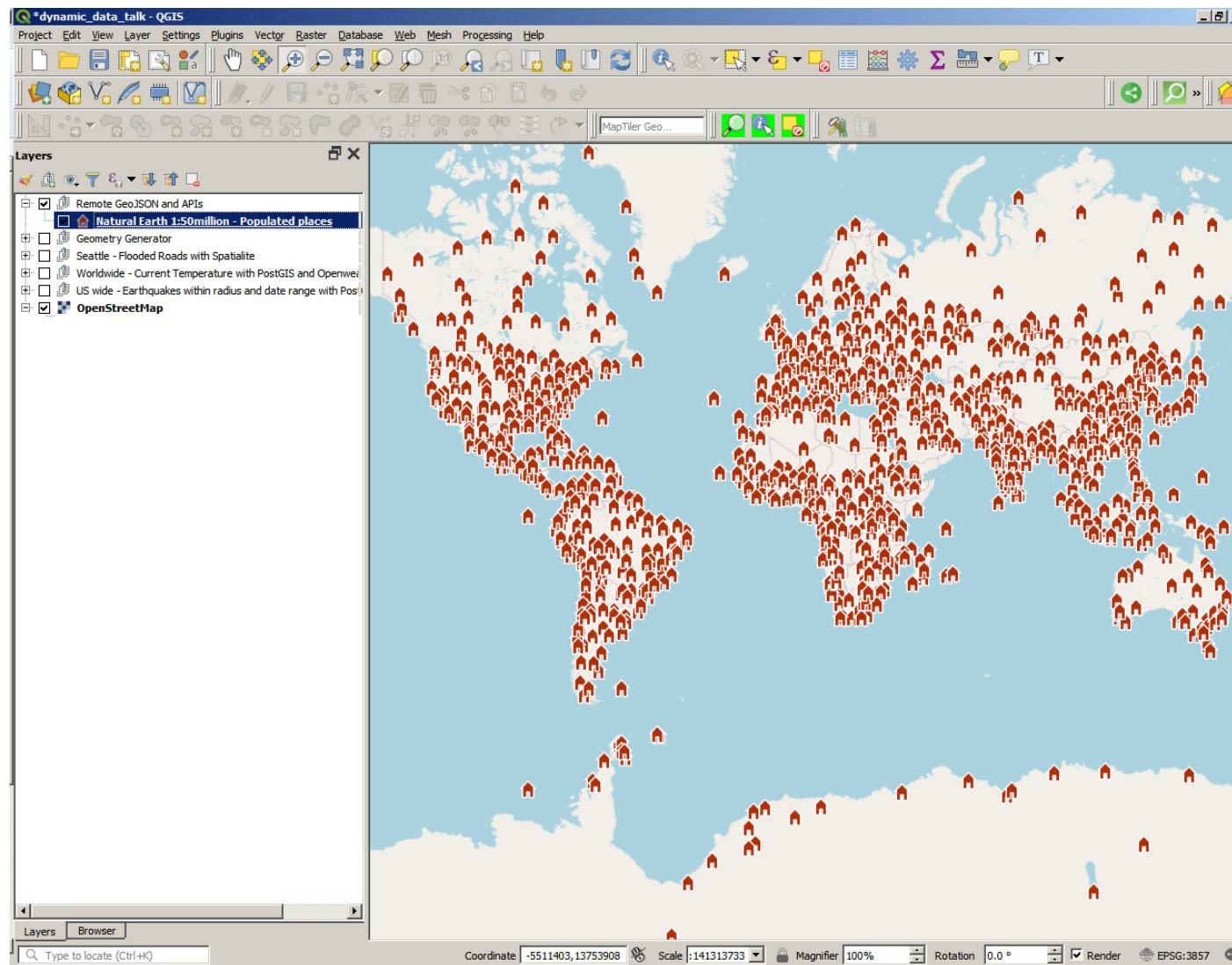
- GeoJSON
- HTTP/HTTPS/FTP
- AWS S3
- Google Cloud Storage
- Microsoft Azure Blob
- Alibaba Cloud OSS
- OpenStack Swift Object Storage
- WFS3 (experimental)
- GeoJSON
- GeoJSON - Newline Delimited
- CouchDB

GeoJSON File example

https://theurl.net/ne_50m_populated_places.geojson



Adding Remote Data- GeoJSON Example



Dynamic GIS Layers
with QGIS and Spatial Databases

 **TERRA GIS**
TERRESTRIAL ENVIRONMENT REGIONAL ANALYSIS



Buffer Example – Geometry Generator

A QGIS symbol style that can change display geometries

1 set symbol layer type
2 set correct output (result) geometry type
3 functions - such as buffer - can be used



Line generation Example – Geometry Generator

A QGIS symbol style that can change display geometries

The screenshot shows the 'Layer Properties' dialog for a layer named 'GeomGenerator - Census Tract Centroids'. The 'Symbology' tab is selected. In the 'Single symbol' section, the 'Symbol layer type' is set to 'Geometry generator'. The 'Geometry type' dropdown shows 'LineString / MultiLineString'. Below it, a code editor displays the following QGIS expression:

```
- difference(  
-   make_line(  
-     centroid($geometry),  
-     @map_extent_center  
-   ),  
-   make_circle(  
-     @map_extent_center,  
-     800  
-   )  
)
```

Below the code editor, there are checkboxes for 'Enable symbol layer' and 'Draw effects'. A yellow callout box on the right contains the following text:

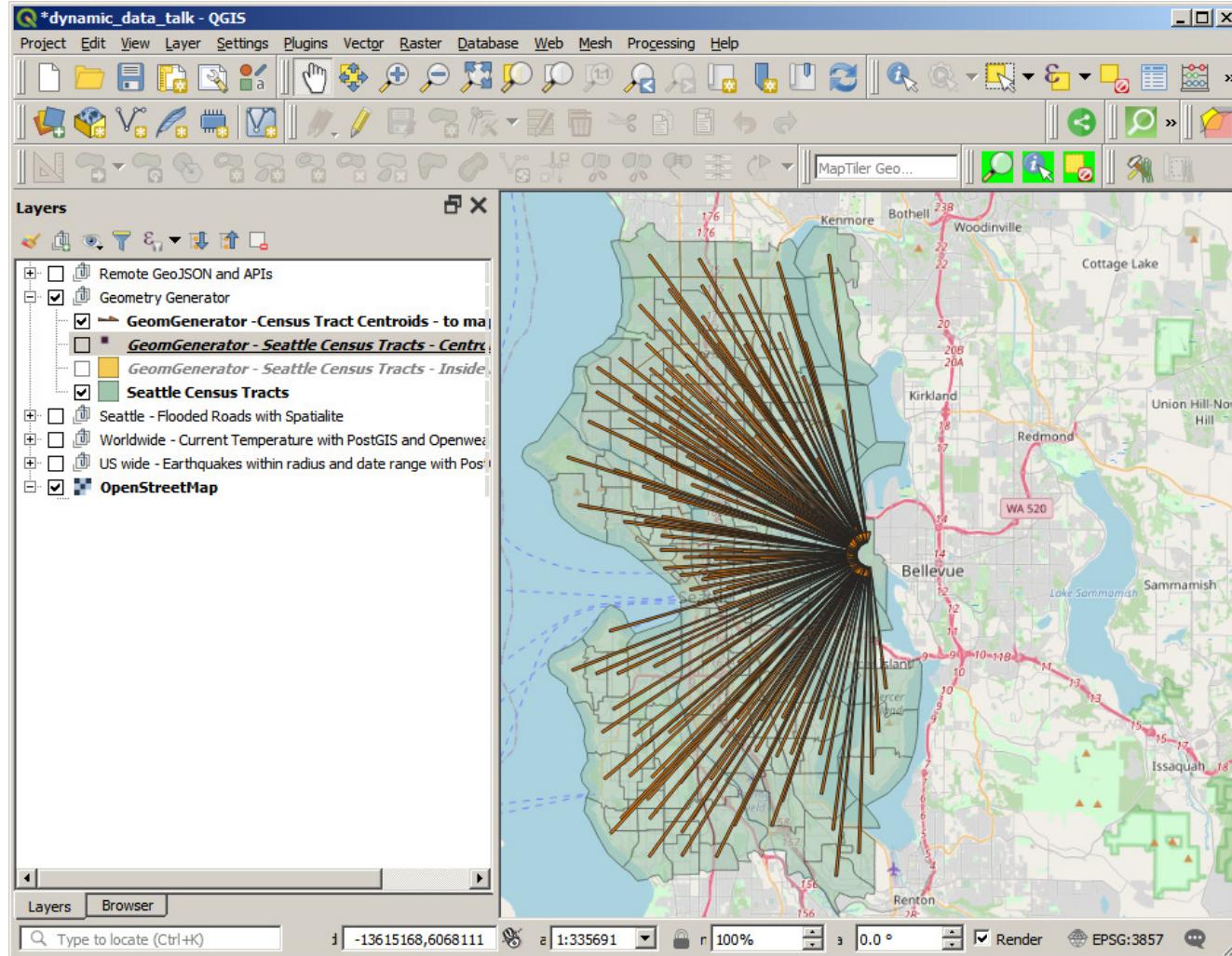
Create a line from each census tract centroid to the current map canvas center dynamically ... and omit map center area

At the bottom of the dialog are buttons for 'OK', 'Cancel', 'Apply', and 'Help'.



Dynamic Arrows display

census tract centroid **to the** current map canvas center **dynamically**





Dynamic Data in QGIS with Spatial Databases

- Dynamic query layers via DB manager
- PostGIS /Spatialite Database Views
- PostGIS Database Views
 - + custom plpython functions
 - + external API requests



DB Manager - Connecting to Spatial Databases

The screenshot shows the QGIS DB Manager window with the following structure:

- Providers:**
 - GeoPackage:** Contains a folder named `wodata.gpkg` which includes layers `nb`, `poi`, and `trees`. This item is highlighted with a red border and has a red number **1** next to it.
 - Oracle Spatial**
 - PostGIS:** Contains connections to various databases including `cmein`, `gisme_test`, `localhost`, `oda`, `onlineimage lokal`, `practicalqgis`, `steg`, `terrax - osgis`, `terray osgis`, and `tiger`. This item is highlighted with an orange border and has an orange number **2** next to it.
 - SpatialLite:** Contains databases `bg_seattle_all.sqlite` and `tracts_seattle.sqlite`, each with its own sub-layers. This item is highlighted with a green border and has a green number **3** next to it.
 - Virtual Layers**
- Toolbar:** Includes icons for Database, Table, Import Layer/File..., and Export to File... along with standard window controls.
- Tab Bar:** Shows tabs for Info, Table, and Preview.

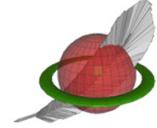
A QGIS core plug-in
Import / export data
Create 'dynamic' query layers

Connection Providers

1 Geopackage - wrapper around SQLite

2 PostGIS (create connection in browser panel)

3 Spatialite



Spatialite – file based Spatial Database

SpatiaLite is a spatial DBMS built on top of **SQLite**. Both formats are file based and thus are light weight and portable. The spatial components depend on the PROJ and GEOS libraries.

The screenshot shows the spatialite_gui interface. On the left, a tree view displays a project structure under 'User Data', including 'tracts_seattle' and other metadata layers. The main area features a table viewer with columns: onhopi, popoth, poptwo, pophis, popwhite2, popblack2, popaian2, popasian2, and pop. Below the table is a message bar indicating 'current block: 1 / 151 [151 rows] [fetched in 00:00:00.296]'. At the bottom, a SQL query window shows the following code:

```
SELECT ROWID, "OGC_FID", "GEOMETRY", "statefp10", "countyfp10", "tractsf10", "popwhite2", "popblack2", "popaian2", "popasian2", "pop, "onhopi", "popoth", "poptwo", "pophis, "popwhite2", "popblack2", "popaian2", "popasian2", "pop FROM "tracts_seattle"  
ORDER BY ROWID
```

The status bar at the bottom indicates the current SQLite DB is located at H:\projects\gis_classes\2020\qgis\workshop\data\seattle\census\tracts_seattle.sqlite.

Spatialite GUI

- PostGIS is an extension for PostgreSQL
- adds support for geographic objects to PostgreSQL
- enables PostgreSQL server to be used as a backend spatial database for GIS
- Spatial operations and analysis simply mean running a (spatial) SQL query in the database

1. Dynamic query layers via DB manager



DB Manager

Database Table

Import Layer/File... Export to File...

Providers

- GeoPackage
- Oracle Spatial
- PostGIS
- Spatialite
 - bg_seattle_all.sqlite
 - flood.sqlite
 - tracts_seattle.sqlite
 - data_licen...
 - tracts_sea...
- Virtual Layers

Info Table Preview Query (bg_seattle_all.sqlite) Query (tracts_seattle.sqlite)

Saved query Name Save Delete Load File Save As File

```
1 select *, st_buffer(GEOMETRY,-900) as buffergeom from  
2 tracts_seattle where pop10 > 7700
```

Execute 1 rows, 0.004 seconds Create a view Clear Query History

OGC_FID	GEOMETRY	statefp10	countyfp10	tractce10	ge
1 41	b'\x00\x01\x00\...	53	033	004100	5303300

Load as new layer

Column with unique values geoid10

Geometry column buffergeom

Layer name (prefix) buffered_tract_

Example of a query layer
Select one Census Tract with population > 7700
Buffer GEOMETRY inside (-900 feet)



2. Database Views PostGIS or Spatialite

- Database views can also be added to QGIS just like 'regular' tables
(via QGIS browser or DB manager)
- A unique ID column is needed to make this work correctly. If none is available we can use

```
SELECT row_number() OVER () AS gid, ...
```

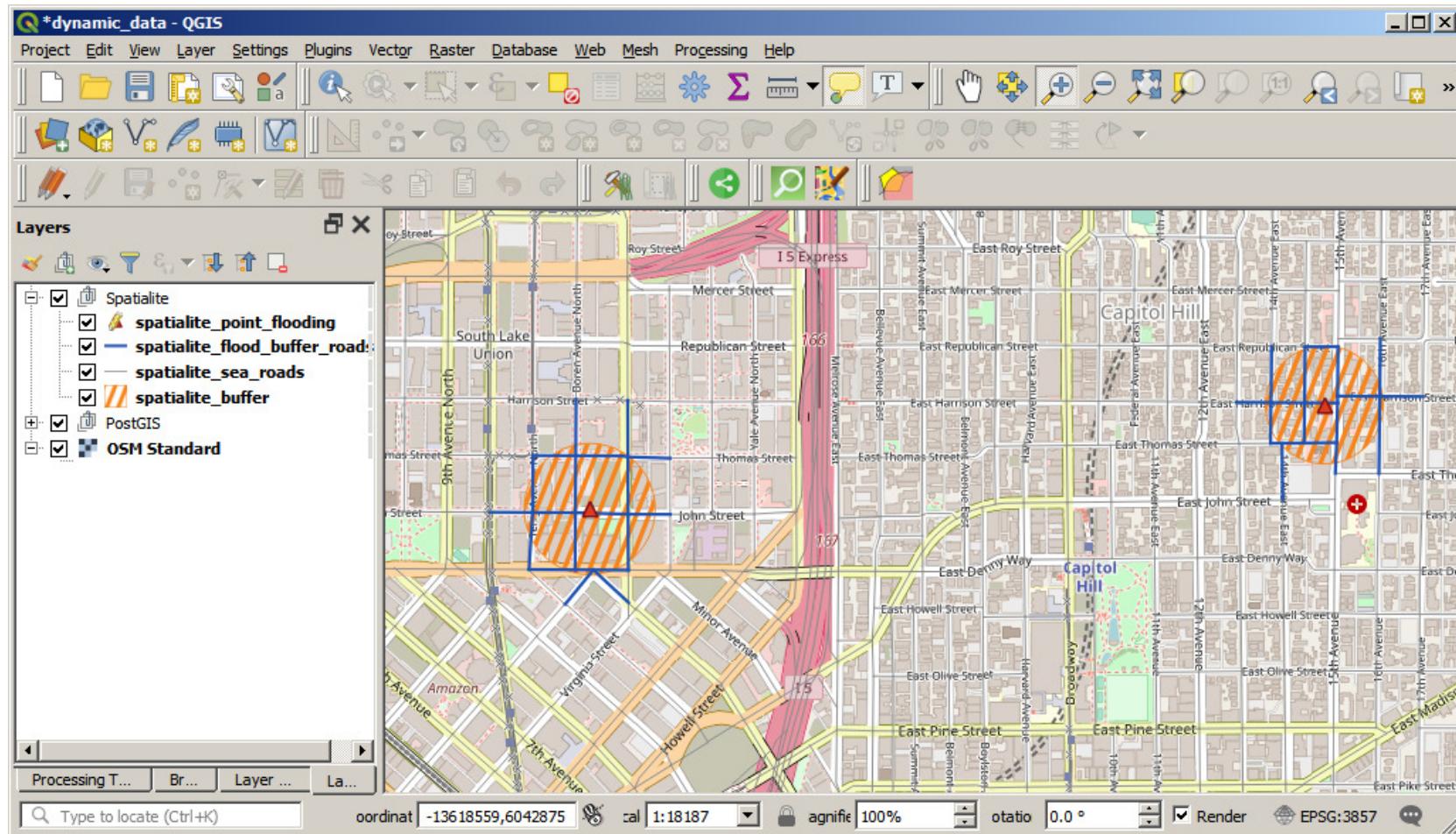
in the query to create one on the fly

- Views and query layer can include references to multiple tables and include PostgreSQL and PostGIS functions. *This can be used to change geometries on the fly ...*



Example - Dynamic Layers with Spatialite

What will happen: buffer point to create flooded area using attribute columns, and display only affected (intersecting) roads





Example - Dynamic Layers with Spatialite

How was this done ?

The screenshot shows the spatialite_gui interface. On the left is a tree view of the database structure under 'User Data'. It includes a folder named 'flood_point_buffer' which contains three files: 'flood_point_buffer', 'point_flooding', and 'sea_roads'. Below this are other categories like ISO / INSPIRE Metadata, Styling (SLD/SE), Topology-Geometry, Topology-Network, Raster Coverages, Vector Coverages, Registered WMS layers, PostgreSQL, Metadata, Internal Data, and Spatial Index. The main window displays a SQL script:

```
CREATE VIEW "flood_point_buffer" AS SELECT point_flooding.id,
st_buffer(point_flooding.geom, point_flooding.radius_m) AS geom,
point_flooding.radius_m FROM
point_flooding
```

A callout box highlights the following steps:

- Add two layers **flood points** (empty) and **seattle roads** to spatialite database
- Create a view in Spatialite GUI with buffer function (using radius column) and add to QGIS

At the bottom, the status bar shows: Current SQLite DB: H:\projects\gis_classes\2020\qgis\workshop\data\collection\dynamic\flood.sqlite



Example - Dynamic Layers with Spatialite

How was this done ?

View added to QGIS as a Query Layer via DB Manager
Needs a unique ID column !

The screenshot shows the QGIS DB Manager interface. On the left, the 'Providers' tree view shows several databases: GeoPackage, Oracle Spatial, PostGIS, SpatiaLite, bg_seattle_all.sqlite, flood.sqlite (selected), tracts_seattle.sqlite, and Virtual Layers. Under flood.sqlite, there are three tables: flood_point_buffer, point_flooding, and sea_roads. The main window displays a query editor titled 'Query (flood.sqlite)'. The query is:

```
1 select * from flood_point_buffer
```

Below the query, the results table shows one row:

	id	geom	radius_m
1	27	b'\x00\x01\x11\x0f\x00\x00\xe3\x86...' (hex representation of a geometry)	234

At the bottom of the query editor, there are options to 'Load as new layer'. The 'Column with unique values' dropdown is set to 'id'. The 'Geometry column' dropdown is set to 'geom'. The 'Layer name (prefix)' field contains 'q_buffer'. The 'Load' button is highlighted.



Example - Dynamic Layers with Spatialite

How was this done ?

added a second query Layer: seattle roads intersecting the flood buffer

The screenshot shows the DB Manager interface. On the left, the Providers tree includes GeoPackage, Oracle Spatial, PostGIS, and Spatialite. Under Spatialite, there is a folder 'bg_seattle_all.sqlite' and a file 'flood.sqlite'. Inside 'flood.sqlite', there are three tables: 'flood_point_buffer', 'point_flooding', and 'sea_roads'. A query window titled 'Query (flood.sqlite)' contains the following SQL code:

```
1 SELECT DISTINCT r.id, r.geom, r.stname FROM sea_roads r, flood_point_buffer b
2 WHERE st_intersects(r.geom, b.geom)
```

The results table shows 22 rows returned in 1.812 seconds. The columns are id, geom, and stname. The data is as follows:

	id	geom	stname
1	1172	b'\x00\x01\x11\...	E YESLER WAY
2	3534	b'\x00\x01\x11\...	E YESLER WAY
3	5075	b'\x00\x01\x11\...	18TH AVE
4	5263	b'\x00\x01\x11\...	17TH AVE S
5	6390	b'\x00\x01\x11\...	CENTRAL PARK TRL
6	6581	b'\x00\x01\x11\...	E SPRUCE ST

Below the results, there are options to 'Load as new layer' (checkbox checked), 'Column with unique values' (checkbox unchecked), 'Layer name (prefix)' (text input 'road_selection_'), 'Geometry column' (dropdown 'geom'), 'Retrieve columns' (button), 'Set filter' (button), 'Load' (button), and 'Cancel' (button).

3. Combining PostGIS Database Views and dynamic queries



plus we will use

- custom plpython functions
- external API requests

prerequisites

- Postgresql version 9.3+ *using LATERAL join type*
- PostGIS extension
- Python 2.7+ or 3+
- plpython language

3. Combining PostGIS Database Views and dynamic queries

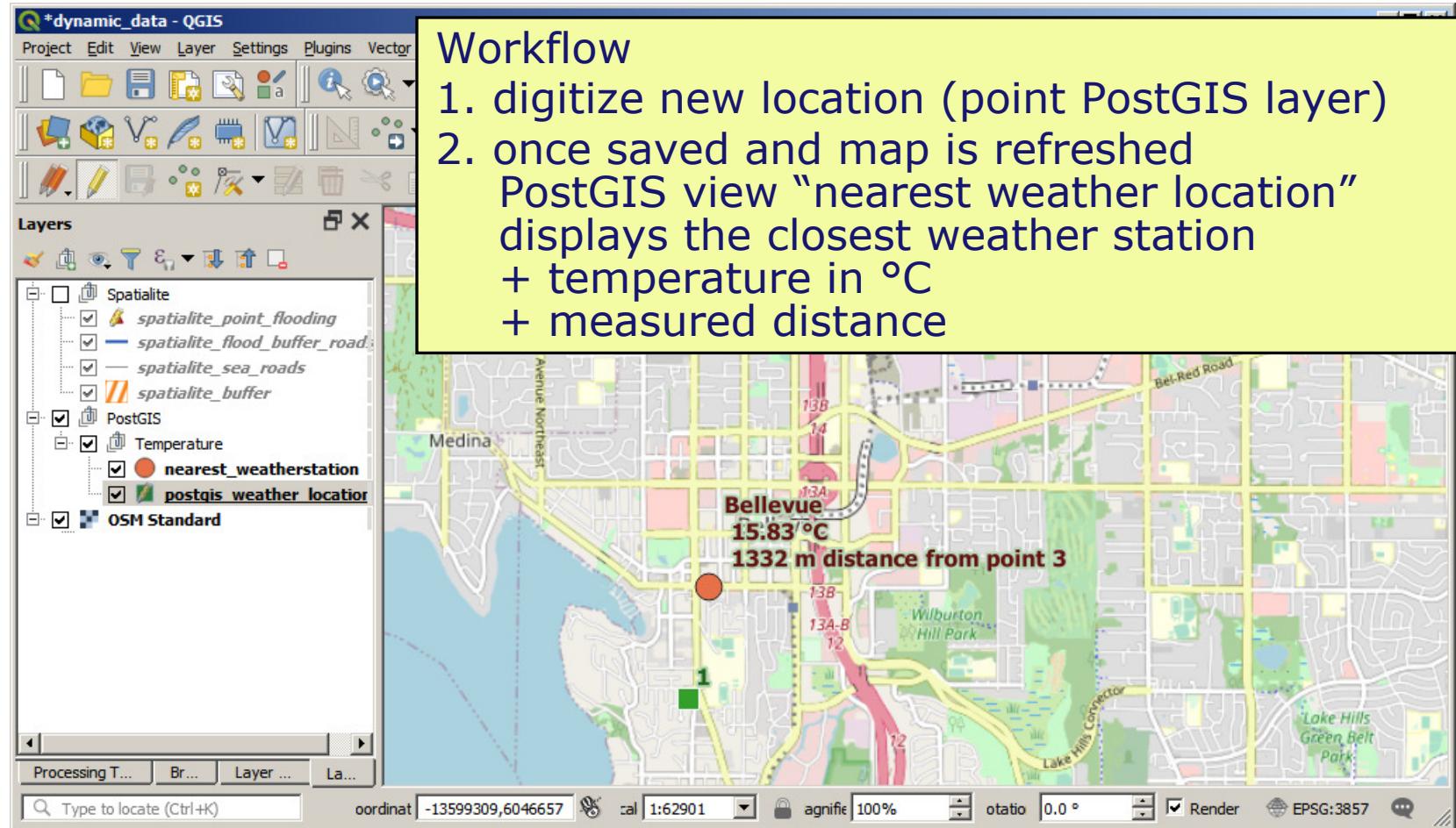


Example 1: Openweather map API

- custom plpython function `GetWeatherAll(lon, lat)`
 - ➡ takes a coordinate pair lon/lat and retrieves current temperature at closest weather stations plus location coordinates of the closest weather station
 - A dynamic display of those data in QGIS is enabled via a PostGIS Database view joining a PostGIS point layer and the `GetWeatherAll` function
- ➡ *Each time a new location is added to the PostGIS point layer the view will be updated, current weather data dynamically retrieved from Open weather map and displayed in QGIS***

Result

PostGIS Database View with current weather data via dynamic Weather API query in plpython function





PostGIS Functions - Spatial SQL

pgAdmin File ▾ Object ▾ Tools ▾ Help ▾

Browser

- ociia
- osgis
 - Casts
 - Catalogs
 - Event Triggers
 - Extensions (5)
 - file_fdw
 - ogr_fdw
 - plpgsql
 - postgis**
 - postgres_fdw
- Foreign Data Wrappers (3)
 - Languages (3)
 - plpgsql
 - plpython2u
 - plpython**
- Schemas (2)
 - osgis
 - public
 - Collations
 - Domains
 - FTS Configurations
 - FTS Dictionaries
 - FTS Parsers
 - FTS Templates
 - Foreign Tables
- Functions (1183)
 - _st_countagg_transfn(agg agg_count, rast raster, nband integer, exclude_nodata_value boolean, s
 - _add_overview_constraint(ovschema name, ovtable name, ovcolumn name, refschema name, refat
 - _add_raster_constraint(cn name, sql text)
 - _add_raster_constraint_alignment(rastschema name, rasttable name, rastcolumn name)
 - _add_raster_constraint_blocksize(rastschema name, rasttable name, rastcolumn name, axis text)
 - _add_raster_constraint_coverage_tile(rastschema name, rasttable name, rastcolumn name)
 - _add_raster_constraint_extent(rastschema name, rasttable name, rastcolumn name)
 - add raster constraint nodata values(rastschema name, rasttable name, rastcolumn name)

Custom Functions - plpython

Code

```

1 import urllib2
2 import json as json
3 data = urllib2.urlopen('http://api.openweathermap.org/dat
4 js_data = json.load(data)
5 if int(js_data['id'])!=0: # check if we have at least a w
6     station = js_data
7     stationname = station['name']
8     stationname.encode("utf8")
9     print 'Data from weather station %s' % station['name']
10    if 'main' in station:
11        if 'temp' in station['main']:
12            temperature = station['main']['temp'] - 273.15 # we
13            cityid = station['id']
14        else:
15            temperature = -9999.9
16    else:
17        temperature = -9999.7
18 else:
19    temperature = -9999.6
20 data_city = urllib2.urlopen('http://api.openweathermap.or
21 js_data_city = json.load(data_city)
22 city = js_data_city
23 clong = city['coord']['lon']
24 clat = city['coord']['lat']
25 cname = city['name'].encode("utf8")

```

Weather data from OpenWeathermap API



Q Weather in your city

Get Started API Pricing Maps FAQ Partners Blog Marketplace

Sign in Support

Current weather data

Home / API / Current weather

Access current weather data for any location on Earth including over 200,000 cities!

Current weather is frequently updated based on global models and data from more than 40,000 weather stations. Data is available in JSON, XML, or HTML format.

We provide 40-year Historical weather data for ANY location. The price is highly competitive - only 10\$ per location! Learn more

Call current weather data for one location

Please remember that all Examples of API calls that listed on this page are just samples and do not have any connection to the real API service!

By city name

Description:

You can call by city name or city name, state code and country code. API responds with a list of weather parameters that match a search request.

API call:

```
api.openweathermap.org/data/2.5/weather?q={city name}&appid={your api key}
```

- Call current weather data for one location
 - By city name
 - By city ID
 - By geographic coordinates
 - By ZIP code
- Call current weather data for several cities
 - Cities within a rectangle zone
 - Cities in cycle
 - Call for several city IDs
- Bulk downloading
- Parameters of API response
 - JSON
 - XML
 - List of condition codes
 - Min/max temperature in current weather API and forecast API
- Other features
 - Format
 - Units format
 - Multilingual support
 - Call back function for JavaScript code

OpenWeathermap

API HTTP request examples

```
# city and country
```

```
http://api.openweathermap.org/data/2.5/weather?  
q=Berlin,DE  
&appid=de429a
```

```
# latitude and longitude
```

```
http://api.openweathermap.org/data/2.5/weather?  
lat=13.74&  
lon=100.49&  
appid=de429a
```

returns JSON



```
JSON Raw Data Headers  
Save Copy Collapse All Expand All | Filter JSON  
▼ coord:  
  lon: 13.41  
  lat: 52.52
```

```
JSON Raw Data Headers  
Save Copy Collapse All Expand All | Filter JSON  
▼ coord:  
  lon: 100.49  
  lat: 13.74  
▼ weather:  
  ▼ 0:  
    id: 803  
    main: "Clouds"  
    description: "broken clouds"  
    icon: "04d"  
    base: "stations"  
  ▼ main:  
    temp: 303.58  
    feels_like: 305.43  
    temp_min: 302.59  
    temp_max: 305.37  
    pressure: 1010  
    humidity: 54  
    visibility: 10000  
  ▼ wind:  
    speed: 2.67  
    deg: 243  
  ▼ clouds:  
    all: 67  
  dt: 1596072876  
  sys:  
    type: 3  
    id: 2033363  
    country: "TH"  
    sunrise: 1596063721  
    sunset: 1596109604  
    timezone: 25200  
    id: 1608132  
    name: "Nonthaburi"
```



Custom plpython (python2) Function in PostgreSQL

GetWeatherAll(lon float, lat float)

```

CREATE OR REPLACE FUNCTION GetWeatherAll(lon float, lat float) RETURNS weather_values AS $$
import urllib2
import json as json
data = urllib2.urlopen('http://api.openweathermap.org/data/2.5/weather?lat=%s&lon=%s&appid=de429a' % (lat, lon))
js_data = json.load(data)
if int(js_data['id'])!=0: # check if we have at least a weather station
    station = js_data
    stationname = station['name']
    stationname.encode("utf8")
    print 'Data from weather station %s' % station['name'].encode("utf8")
    if 'main' in station:
        if 'temp' in station['main']:
            temperature = station['main']['temp'] - 273.15 # we want the temperature in Celsius
            cityid = station['id']
        else:
            temperature = -9999.9
    else:
        temperature = -9999.7
else:
    temperature = -9999.6
data_city = urllib2.urlopen('http://api.openweathermap.org/data/2.5/weather?id=%s&appid=de429a' % (cityid))
js_data_city = json.load(data_city)
city = js_data_city
clong = city['coord']['lon']
clat = city['coord']['lat']
cname = city['name'].encode("utf8")
return {"temperature": temperature, "clong": clong, "clat": clat, "cname": cname }

$$ LANGUAGE plpythonu;

# create custom data type that is returned form weatehr data query to openweathermap
CREATE TYPE weather_values AS (
    temperature float,
    clong float,
    clat float,
    cname text
);

```



OpenWeathermap API HTTP request

`http://api.openweathermap.org/data/2.5/weather?`

`lat=13.74&`

`lon=100.49&`

`appid=de4da`

returns JSON



JSON	Raw Data	Headers
<code>Save</code>	<code>Copy</code>	<code>Collapse All</code>
<code>Expand All</code>		<code>Filter JSON</code>
<code>coord:</code>		
<code>lon:</code>	100.49	
<code>lat:</code>	13.74	
<code>weather:</code>		
<code>0:</code>		
<code>id:</code>	803	
<code>main:</code>	"Clouds"	
<code>description:</code>	"broken clouds"	
<code>icon:</code>	"04d"	
<code>base:</code>	"stations"	
<code>main:</code>		
<code>temp:</code>	303.58	
<code>feels_like:</code>	305.43	
<code>temp_min:</code>	302.59	
<code>596072876</code>		05.37
<code>010</code>		0000
<code>4</code>		.67
<code>0000</code>		43
<code>.67</code>		7
<code>43</code>		596072876
<code>7</code>		033363
<code>596072876</code>		TH"
<code>033363</code>		596063721
<code>TH"</code>		596109604
<code>596063721</code>		5200
<code>596109604</code>		608132
<code>5200</code>		Nonthaburi"
<code>608132</code>		
<code>Nonthaburi"</code>		

SQL Query with GetWeatherAll Function - output

Query Editor
Query History

```
1  SELECT GetWeatherALL ('100.49'::real, '13.74'::real);
```

Explain
Notifications
Data Output

getweatherall
weather_values

(31.15,100.5,13.75,Nonthaburi)

Read-only column



Database view with **LATERAL** join type

location (points)

LATERAL joined with

GetWeatherAll() (function results)

```
CREATE OR REPLACE VIEW osgis.nearest_weatherstation
AS
SELECT row_number() OVER () AS gid,
       st_x(st_transform(w.geom, 4326)) AS loc_x,
       st_y(st_transform(w.geom, 4326)) AS loc_y,
       round(st_distance(st_transform(w.geom, 3857),
                         st_transform(st_setsrid(st_makepoint(f.clong, f.clat), 4326), 3857))) ::numeric, 0)
AS dist_m,
       f.temperature,
       f.clong,
       f.clat,
       st_setsrid(st_makepoint(f.clong, f.clat), 4326) AS geom,
       f.cname
FROM osgis.location w,
     LATERAL getweatherall(st_x(st_transform(w.geom, 4326)), st_y(st_transform(w.geom, 4326))) f(temperature, clong, clat, cname);

ALTER TABLE osgis.nearest_weatherstation
OWNER TO postgres;
```

Database view with LATERAL join type

a closer look at the LATERAL join of

location **w** and

GetWeatherAll() **f**

```
SELECT row_number() OVER () AS gid,  
...  
FROM  
    osgis.location w, # location points  
    LATERAL  
    getweatherall(          # custom function  
        st_x(st_transform(w.geom, 4326)),   # location x  
        st_y(st_transform(w.geom, 4326)))   # location y  
        f(temperature, clong, clat, cname);  
    # returns these 4 params temperature, x,y, station name
```



Database view

	Explain	Data Output	Geometry Viewer									
	gid	loc_x	loc_y	dist_m	temperature	clong	clat	geom	cname			
1	1	4.37059297493893	13.8390708719451	2686	21.58	4.35	43.83	0101000020E610...	Nîmes			
2	2	5.54314720055178	43.217257305345	1346	23.73	5.55	43.21	0101000020E610...	Cassis			
3	3	5.4753469120692	13.2528992258199	6471	23.84	5.52	43.28	0101000020E610...	La Penne-sur-Huveaune			
4	4	9.03262314990176	18.6905440474365	2266	13.75	9.02	48.68	0101000020E610...	Böblingen			
5	5	8.73183280033311	19.4875126335156	3618	16.98	8.75	49.47	0101000020E610...	Wilhelmsfeld			
6	6	-122.308166699283	17.6380686773299	5234	19.89	-122.33	47.61	0101000020E610...	Seattle			



- digitized x, y coordinates
- data from openweather API joined values via plypython function
- row number, distance and geometry -> Postgresql/PostGIS

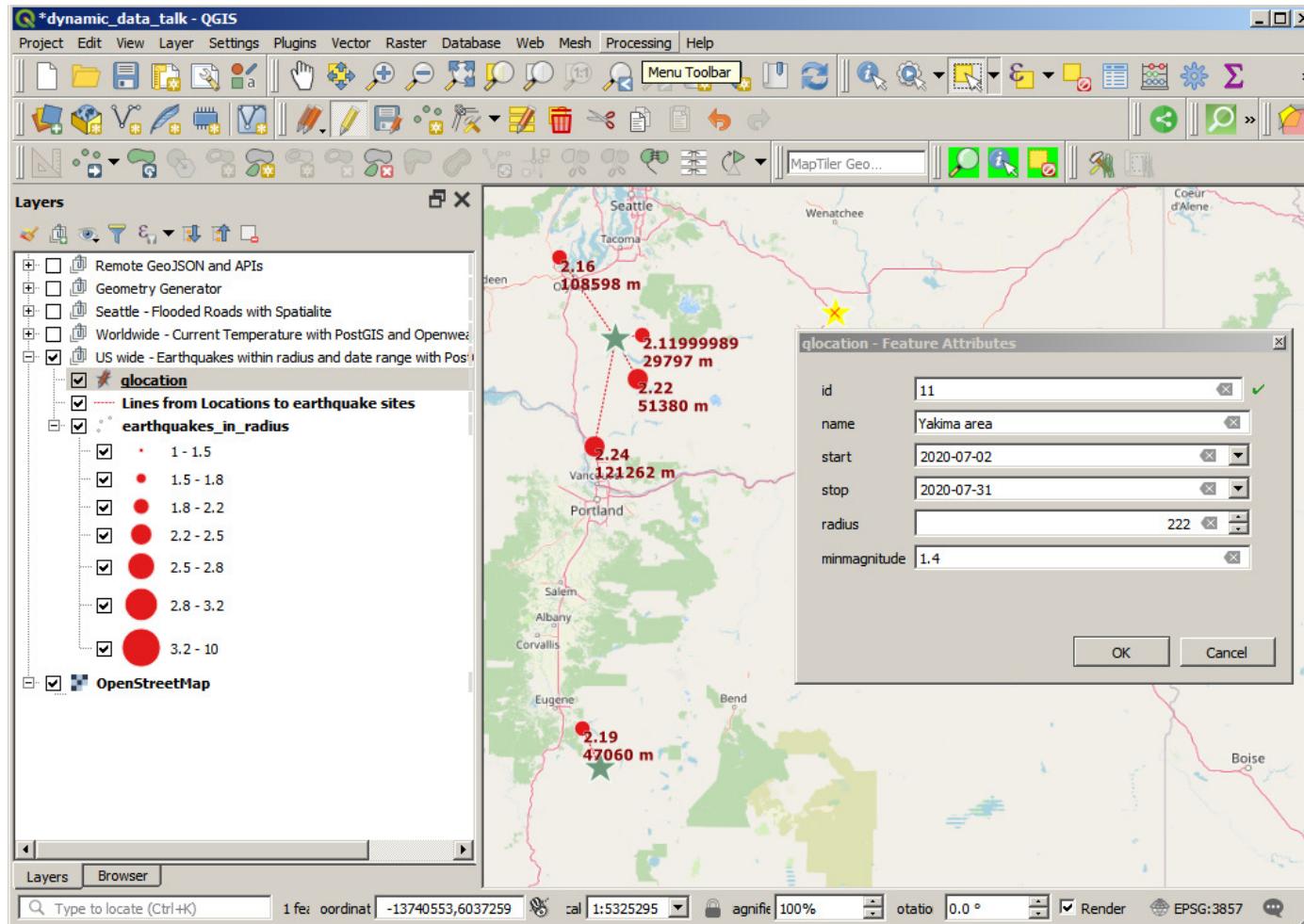
Combining PostGIS Database Views and dynamic queries



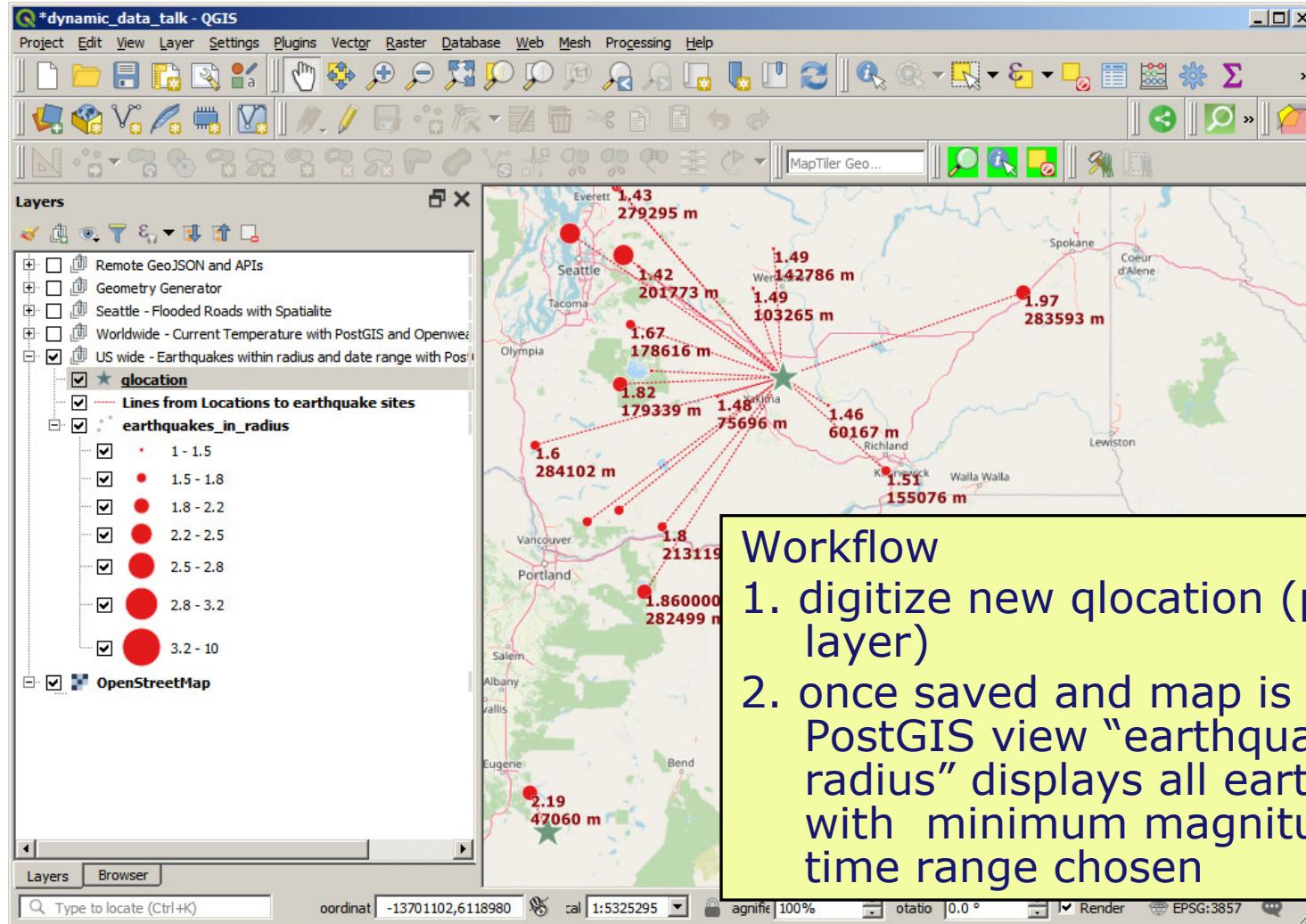
Example 2: USGS Earthquake API

- custom plpython function **getearthquakeall**
 - ➡ takes start + stop date, lon,lat, search radius km and min. magnitude and all earthquake events for chosen date range and minimum magnitude within search radius
 - A dynamic display of those data in QGIS is enabled via a PostGIS Database view joining a PostGIS point layer and the **getearthquakeall** function
- ➡** *Each time a new location is added to the PostGIS point layer the view will be updated, earthquake events that are corresponding to the minimum magnitude and time range retrieved from the USGS API and displayed in QGIS*

Result- PostGIS Database View with earthquake Data in time range and magnitude via dynamic USGS API query in plpython function



Result- PostGIS Database View with earthquake Data in time range and magnitude



Workflow

1. digitize new qlocation (point PostGIS layer)
2. once saved and map is refreshed PostGIS view "earthquakes in radius" displays all earthquakes with minimum magnitude and in time range chosen

plpython function

getearthquakeall(start, stop, lon, lat, radius, minmagnitude)

```
CREATE OR REPLACE FUNCTION GetEarthquakeAll(start_date ,stop date, lon float, lat
float,radius int, minmagnitude float) RETURNS SETOF earthquake_values AS $$  
import urllib2  
import json as json  
data =  
urllib2.urlopen('https://earthquake.usgs.gov/fdsnws/event/1/query?format=geojson&starttime=%s&endtime=%s&latitude=%s&longitude=%s&maxradiuskm=%s&minmagnitude=%s&orderby=magnitude' % (start,stop,lat,lon,radius,minmagnitude))  
js_data = json.load(data)  
equake = js_data  
equakearray = []  
a = 0  
for i in equake['features']:  
    equakeplace = i['properties']['place'] # tile for earthquake location  
    magnitude = i['properties']['mag']  
    qlong = i['geometry']['coordinates'][0]  
    qlat = i['geometry']['coordinates'][1]  
    equakevalue = {"place": equakeplace, "magnitude": magnitude, "qlong": qlong,  
    "qlat": qlat}  
    equakearray.append(equakevalue)  
    a = a+1  
return equakearray  
$$ LANGUAGE plpythonu;  
  
# create custom data type that is returned from equake data API query  
CREATE TYPE earthquake_values AS (  
    place text,  
    magnitude float,  
    qlong float,  
    qlat float  
) ;
```





Database view with **LATERAL** join type

qlocation (points)

LATERAL joined with
getearthquakeall (...) (function results)

```
CREATE OR REPLACE VIEW osgis.earthquakes_in_radius AS
SELECT row_number() OVER () AS gid,
       st_x(st_transform(w.geom, 4326)) AS loc_x,
       st_y(st_transform(w.geom, 4326)) AS loc_y,
       round(st_distance(st_transform(w.geom, 3857),
                         st_transform(st_setsrid(st_makepoint(f.qlong, f.qlat), 4326), 3857)))::numeric, 0)
AS dist_m,
       f.magnitude,
       f.qlong,
       f.qlat,
       st_setsrid(st_makepoint(f.qlong, f.qlat), 4326) AS geom,
       f.place,
       w.id AS qlocation_id
FROM osgis.qlocation w,
     LATERAL getearthquakeall(w.start, w.stop, st_x(st_transform(w.geom, 4326)),
                               st_y(st_transform(w.geom, 4326)), w.radius, w.minmagnitude::double precision)
     f(place, magnitude, qlong, qlat);
```

```
ALTER TABLE osgis.earthquakes_in_radius
OWNER TO postgres;
```



TERRA GIS
TERRESTRIAL ENVIRONMENT REGIONAL ANALYSIS



Database view qloc_earthquake_line for display in QGIS

Creating lines from qlocation points to all earthquake events found for that point

```
CREATE OR REPLACE VIEW osgis.qloc_earthquake_line AS
SELECT row_number() OVER () AS qid,
       st_makeline(st_setsrid(eq.geom, 4326), st_setsrid(ql.geom, 4326)) AS geom,
       round(st_distance(st_transform(eq.geom, 3857), st_transform(ql.geom,
       3857))::numeric / 1000::numeric, 1) AS dist_km,
       eq.place, eq.qlocation_id
  FROM osgis.earthquakes_in_radius eq,
       osgis.qlocation ql WHERE ql.id = eq.qlocation_id;

ALTER TABLE osgis.qloc_earthquake_line
OWNER TO postgres;
```

	gid bigint	geom geometry	dist_km numeric	place text	qlocation_id bigint
7	7	0102000020E61...	6.7	74km ESE of Cascade, Idaho	3
8	8	0102000020E61...	17.2	29 km NW of Stanley, Idaho	3
9	9	0102000020E61...	35.8	18 km SW of Stanley, Idaho	3
10	10	0102000020E61...	47.1	3km SW of Lowell, Oregon	4
11	11	0102000020E61...	81.4	2km NW of The Geysers, CA	5
12	12	0102000020E61...	145.7	17km NE of Upper Lake, CA	5

Thank you and happy

dynamic
mapping !



TERRA GIS
TERRESTRIAL ENVIRONMENT REGIONAL ANALYSIS

Links

■ **Geometry Generator**

[Video - Basic look at the QGIS Geometry Generator style](#)

[Insightful slides / talk](#)

[QGIS Docs](#)



■ **Spatialite**

■ **PostGIS and plpython**

■ **Presentation slides and code snippets**

https://github.com/karstenv/dynamic_layers_qgis

■ **Contact**

Karsten Vennemann

karsten@terragis.net