Dynamic GIS Layers with QGIS and Spatial Databases

TERRA GIS

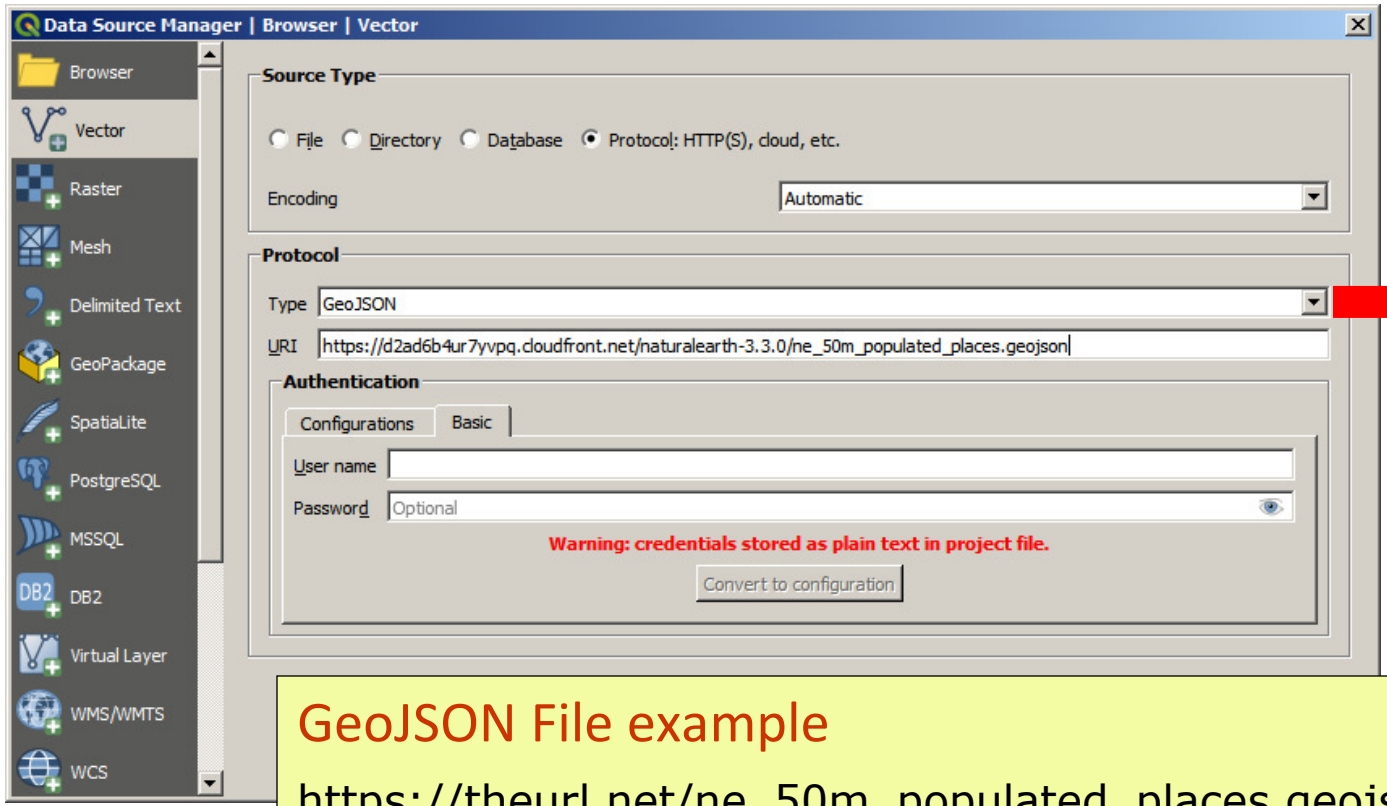
TERRESTRIAL ENVIRONMENT REGIONAL ANALYSIS

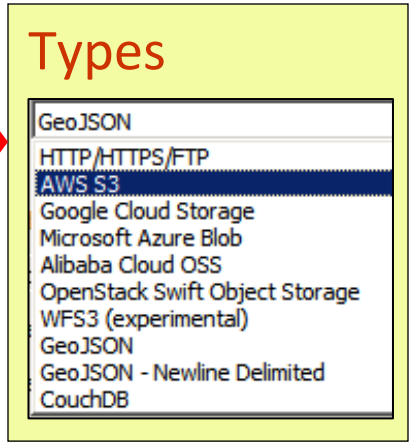
by Karsten Vennemann

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

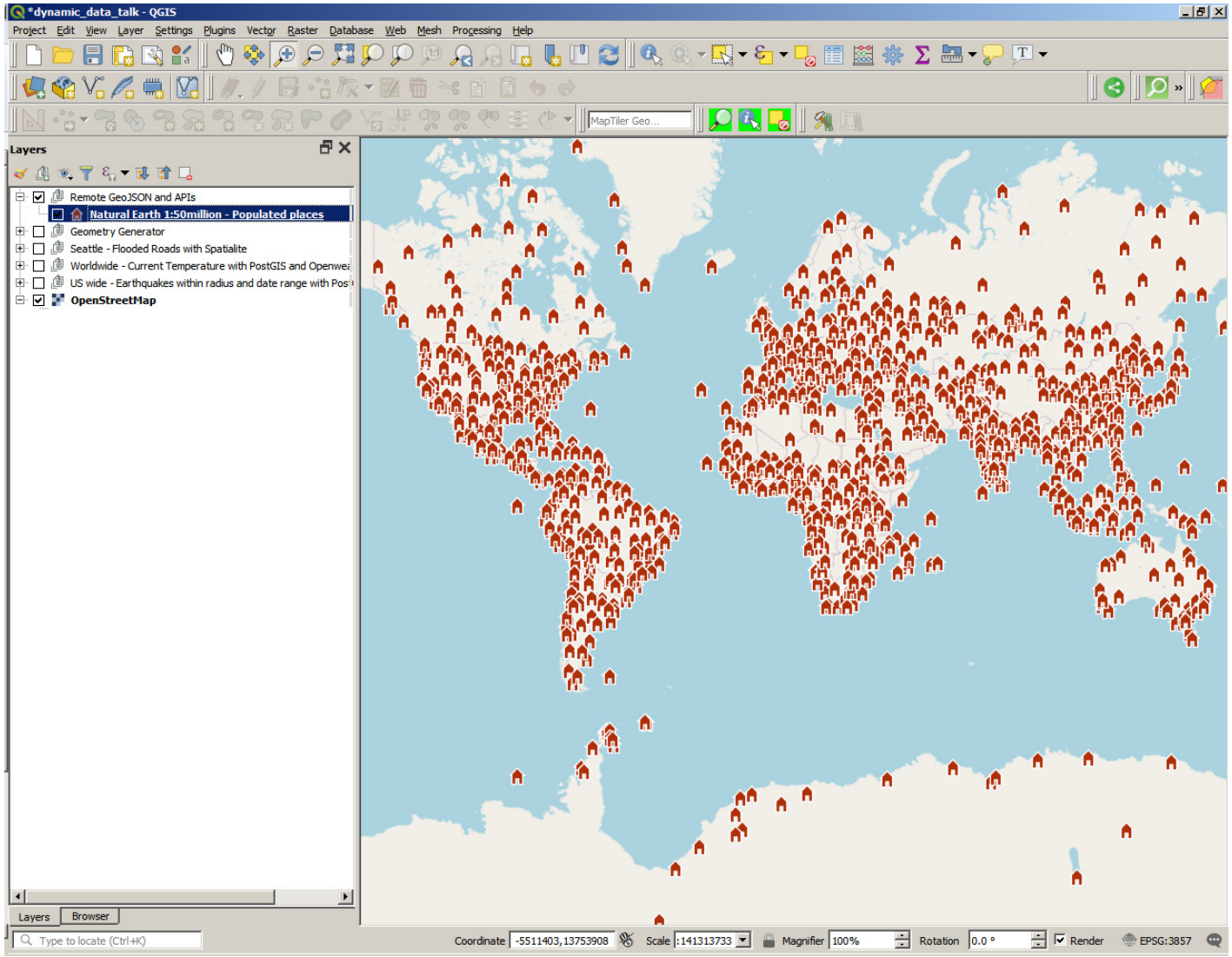




GeoJSON File example

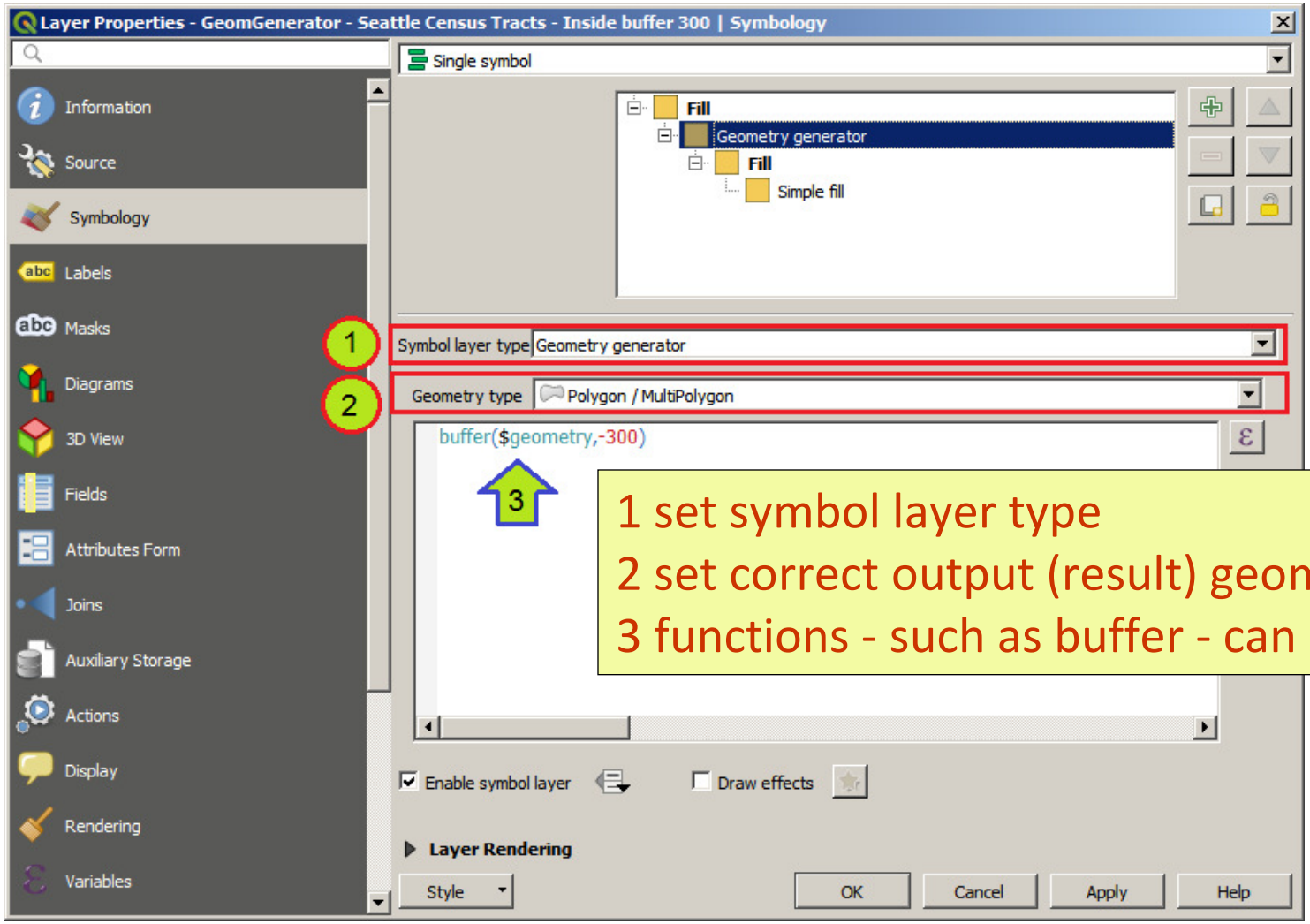
https://theurl.net/ne_50m_populated_places.geojson

Adding Remote Data- GeoJSON Example



Buffer Example – Geometry Generator

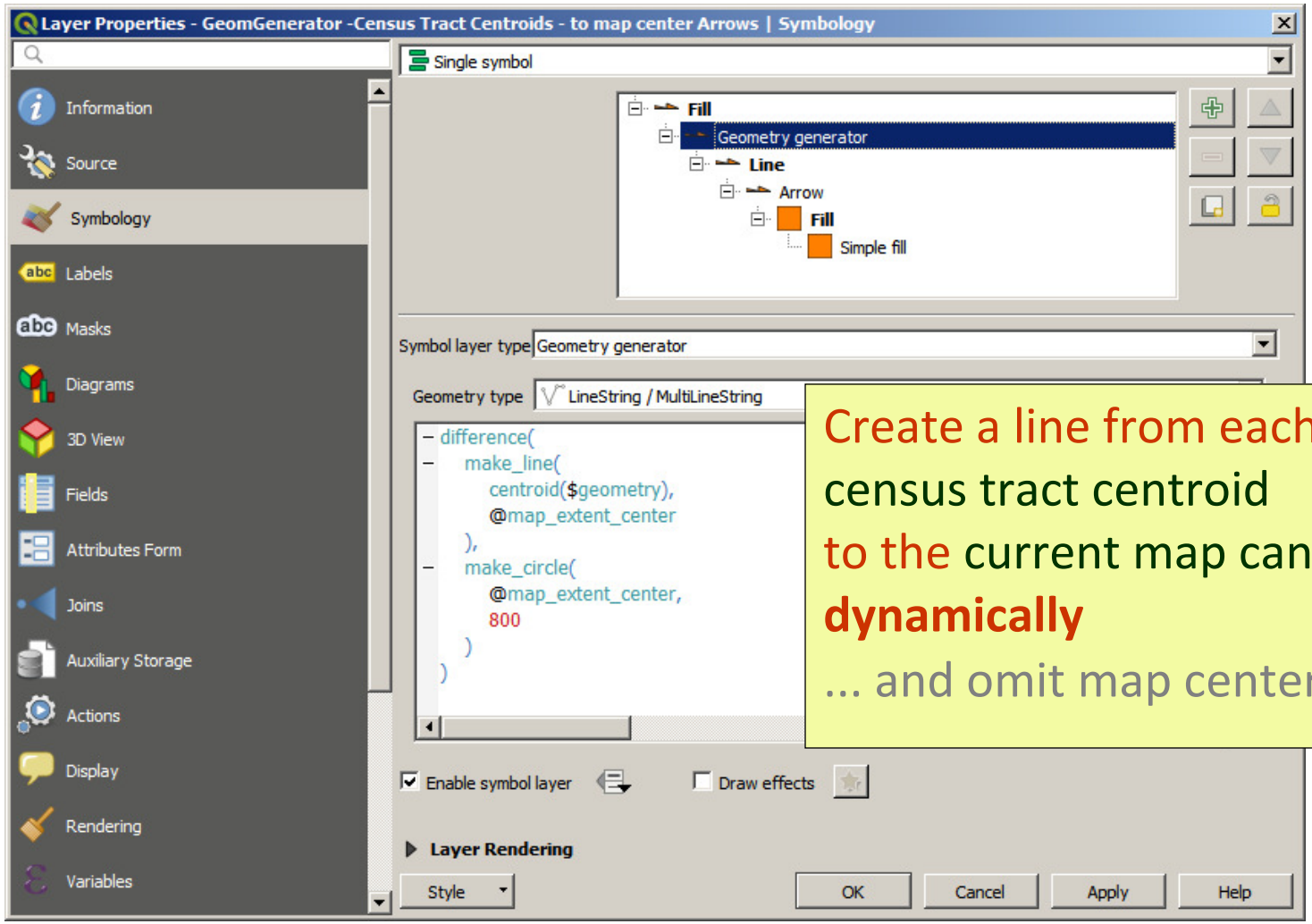
A QGIS symbol style that can change display geometries



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

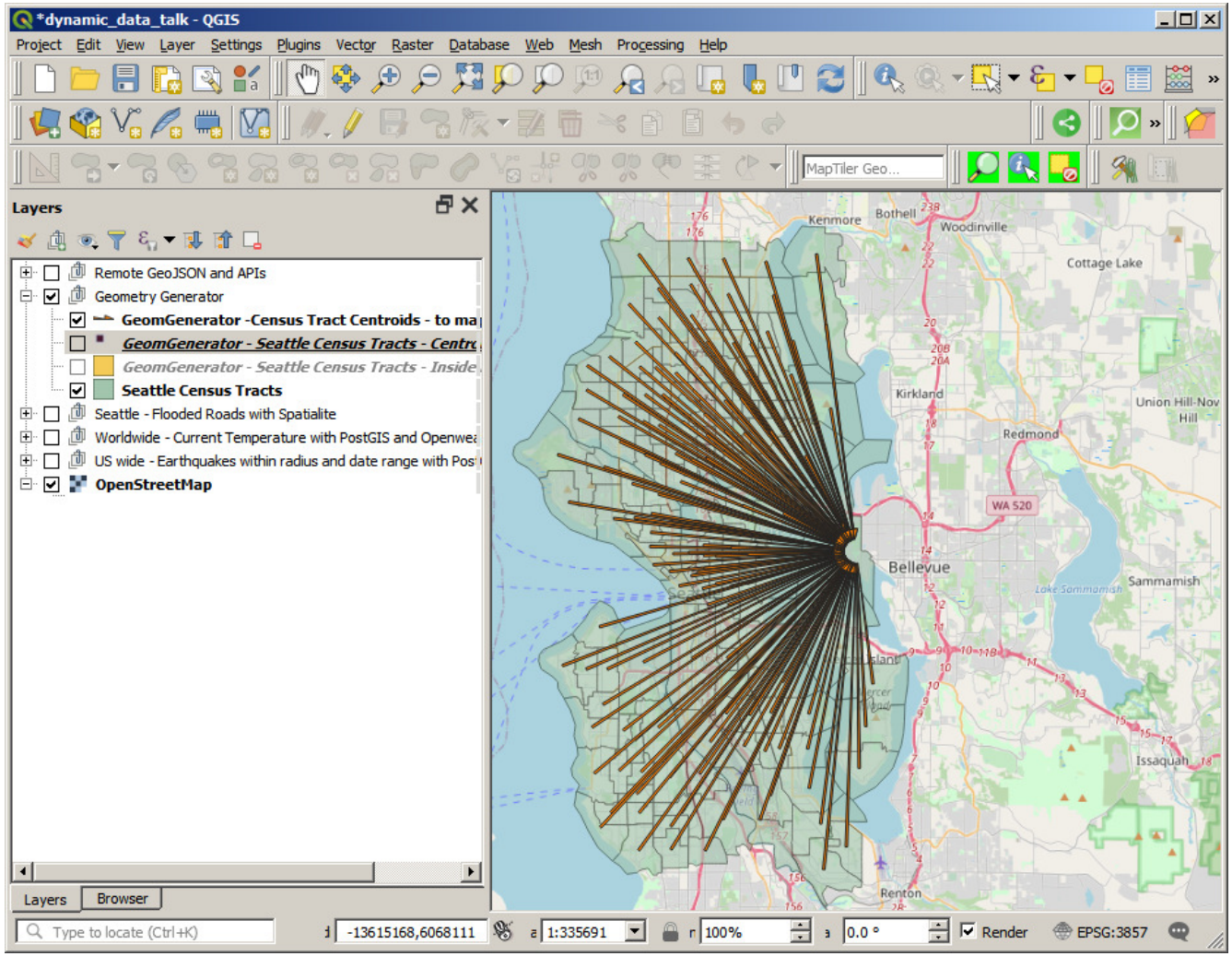
A QGIS symbol style that can change display geometries



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

Dynamic Arrows display

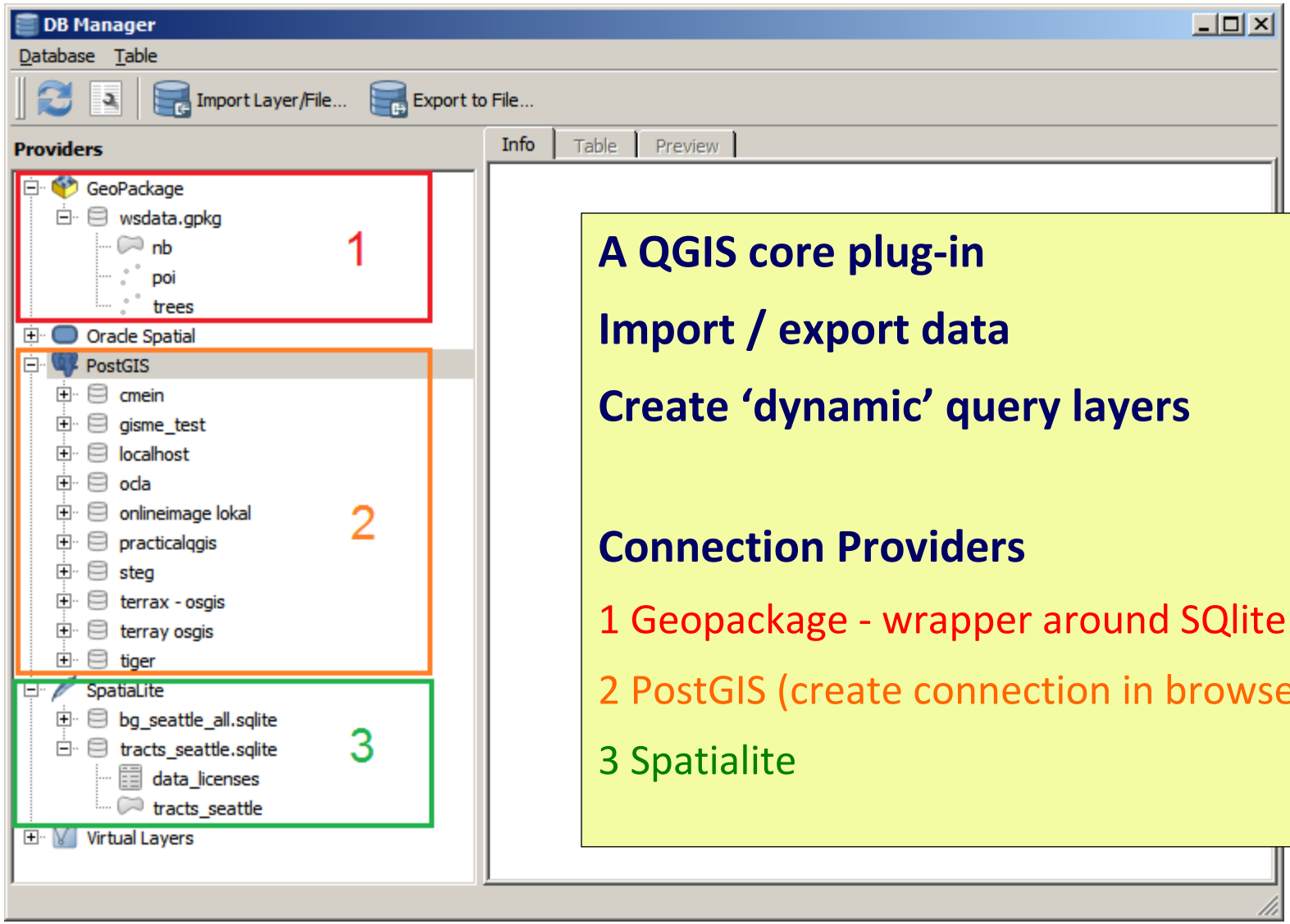
census tract centroid to the current map canvas center **dynamically**



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



A QGIS core plug-in

Import / export data

Create 'dynamic' query layers

Connection Providers

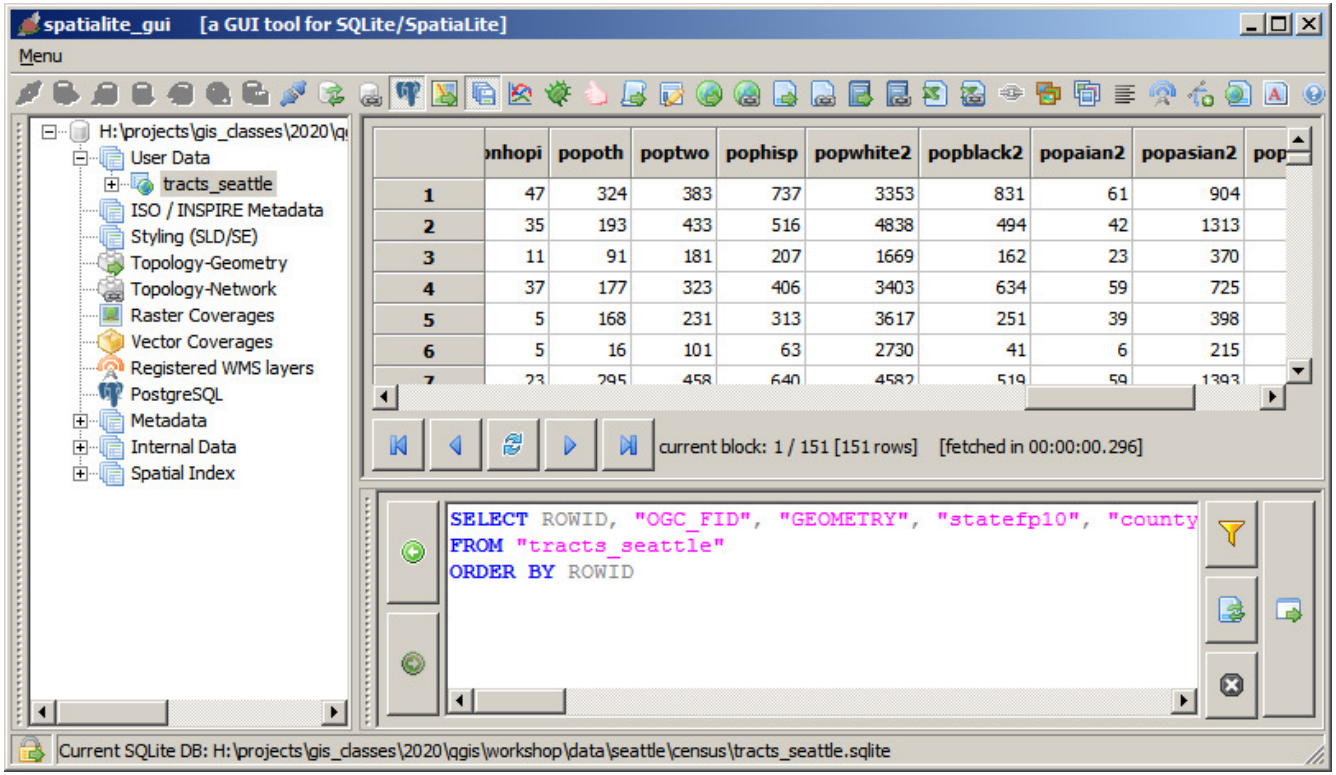
1 Geopackage - wrapper around SQlite

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.

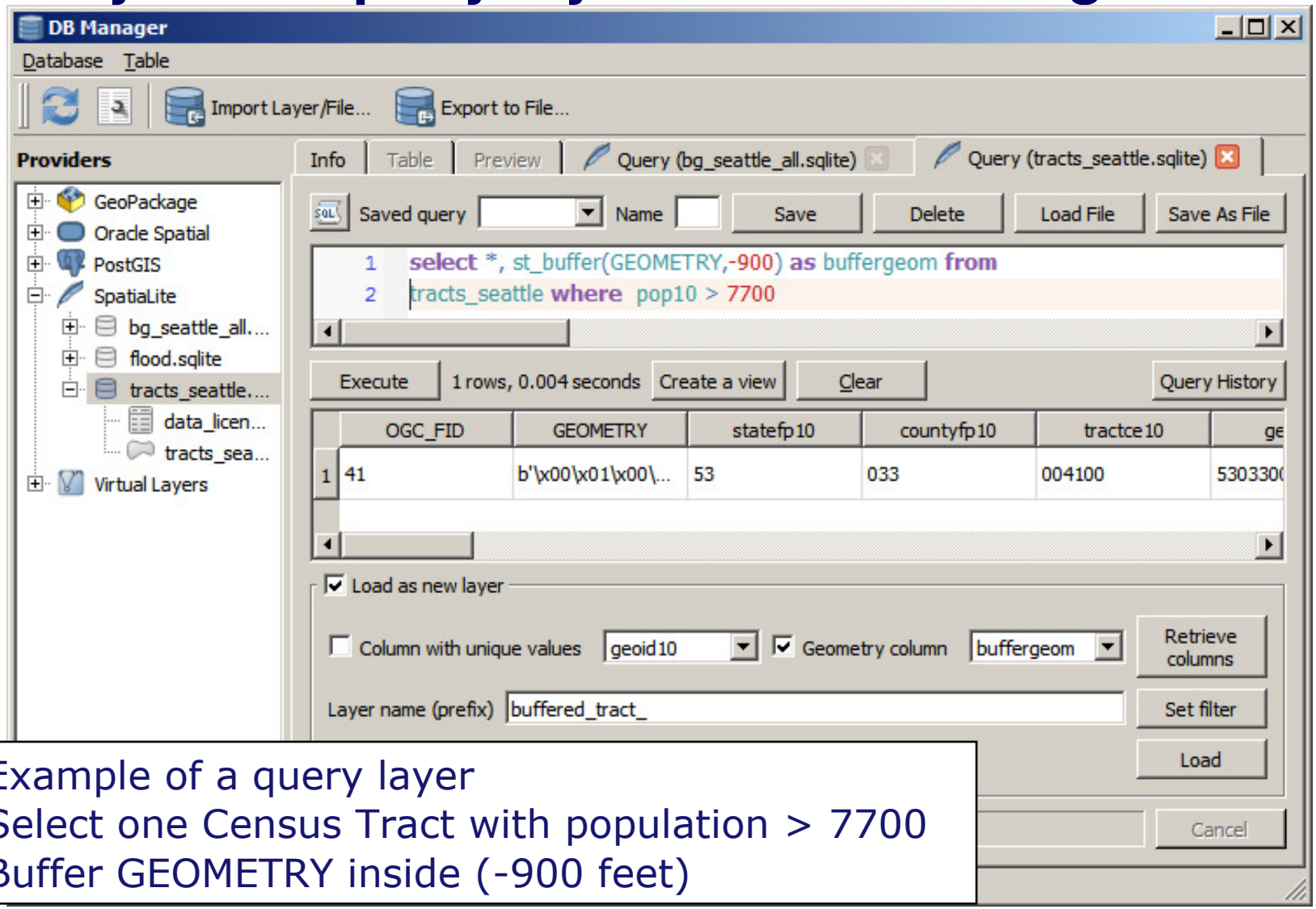


Spatialite GUI

Spatial Database

- 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



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

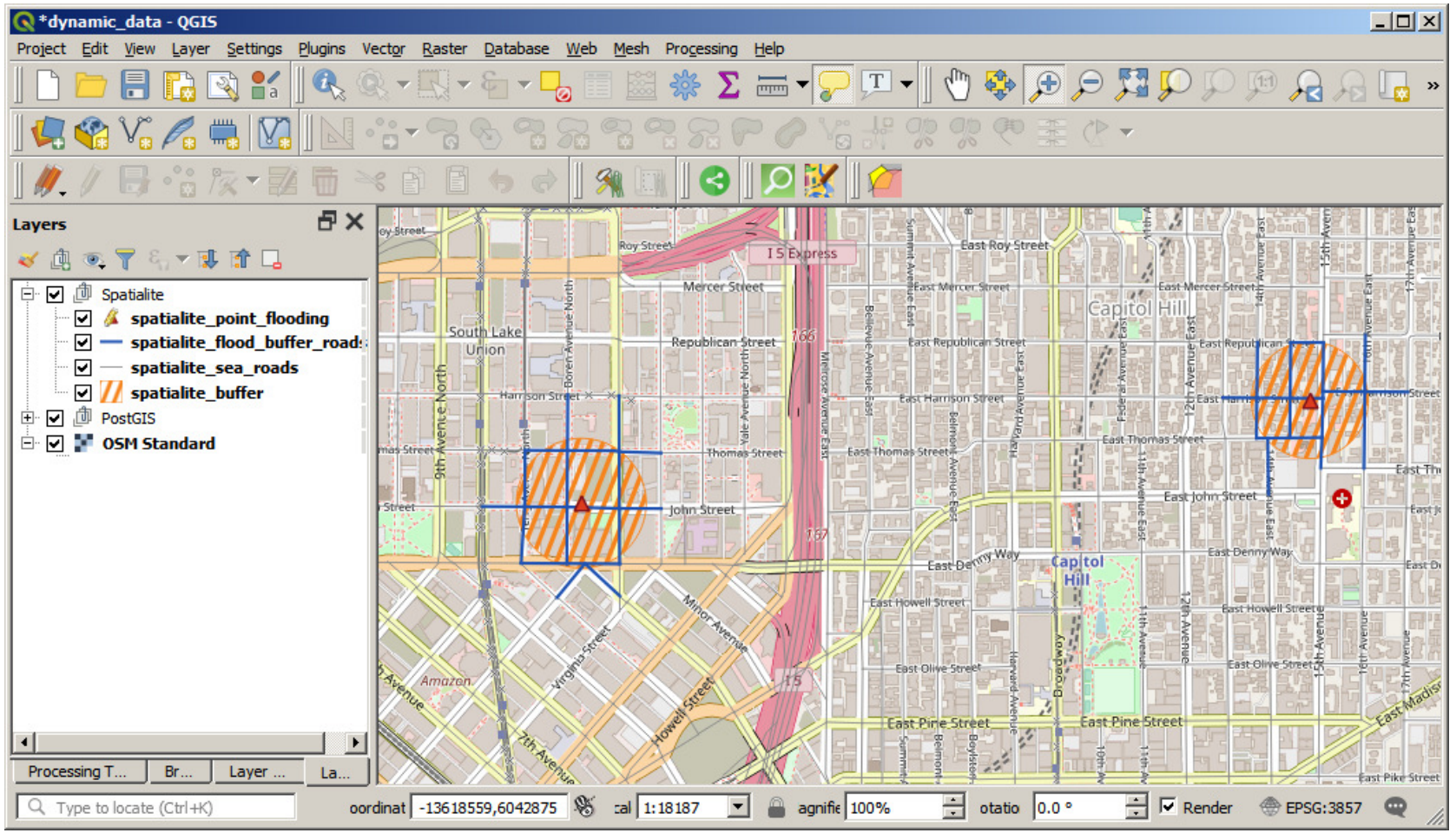
```
SELECT row_number() OVER () AS gid, …
```

  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

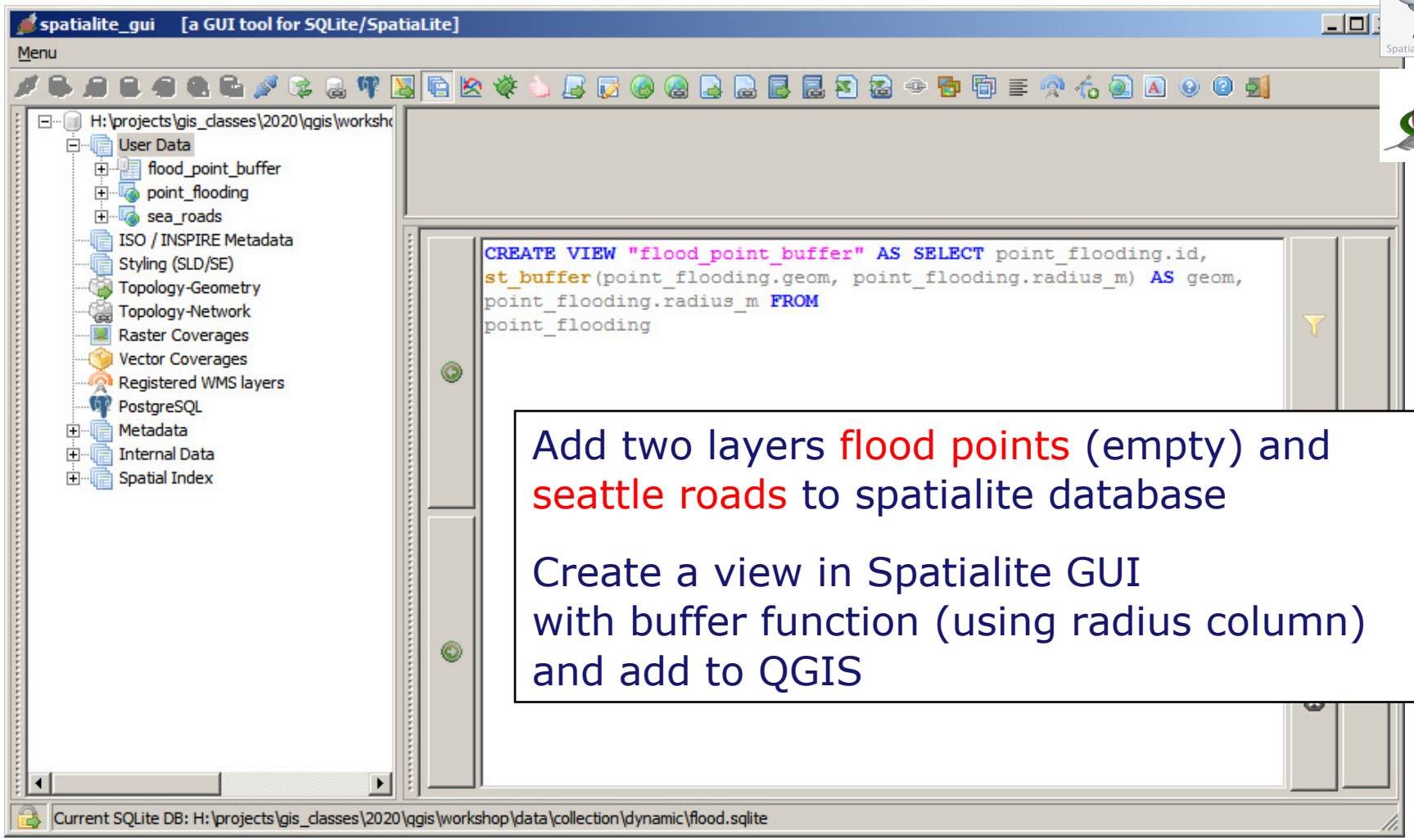
What will happen: buffer point to create flooded area using attribute columns, and display only affected (intersecting) roads



TERRA GIS
TERRESTRIAL ENVIRONMENT REGIONAL ANALYSIS

Example - Dynamic Layers with Spatialite

How was this done ?



Add two layers flood points (empty) and seattle roads to spatialite database

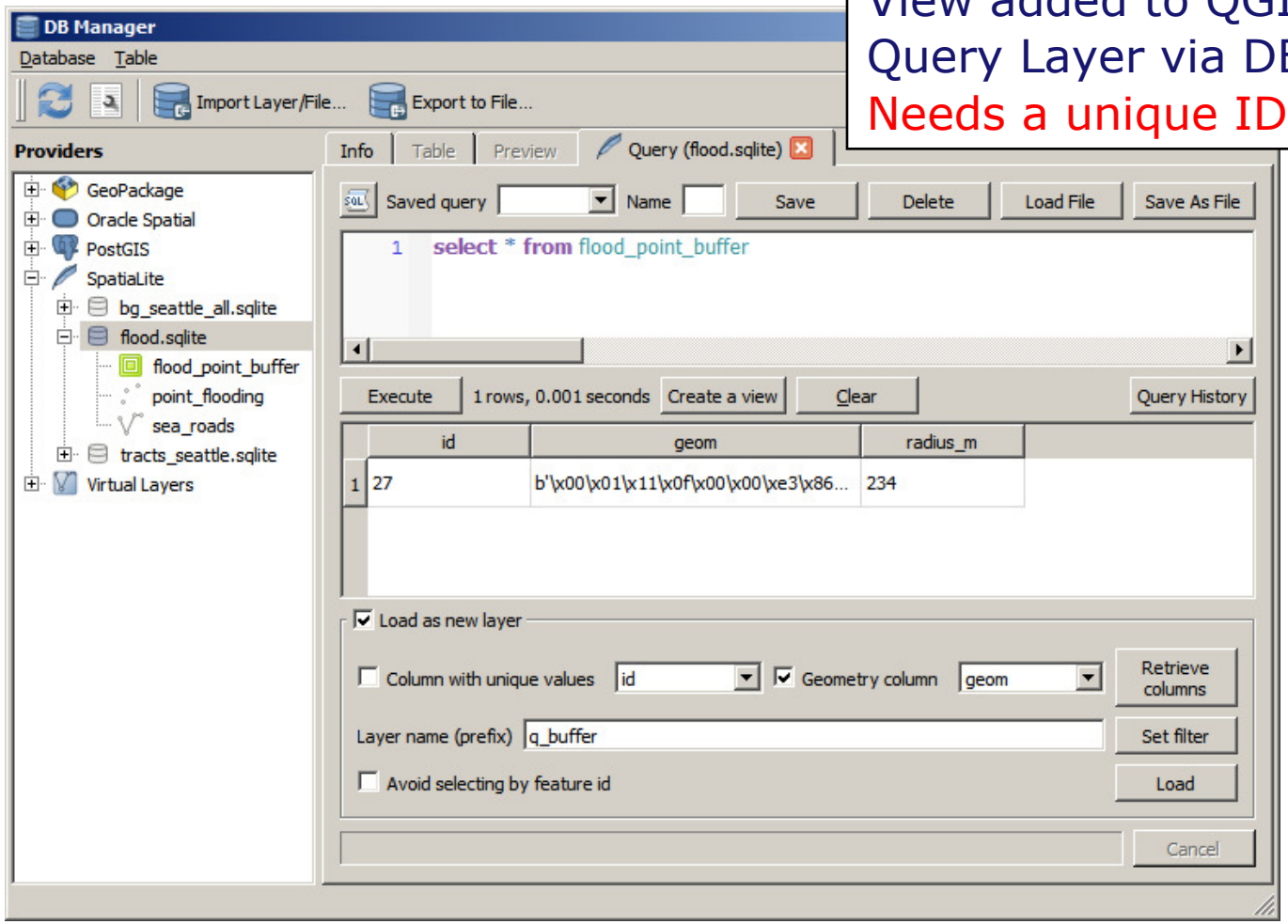
Create a view in Spatialite GUI
with buffer function (using radius column)
and add to QGIS

Example - Dynamic Layers with Spatialite

How was this done ?

View added to QGIS as a Query Layer via DB Manager

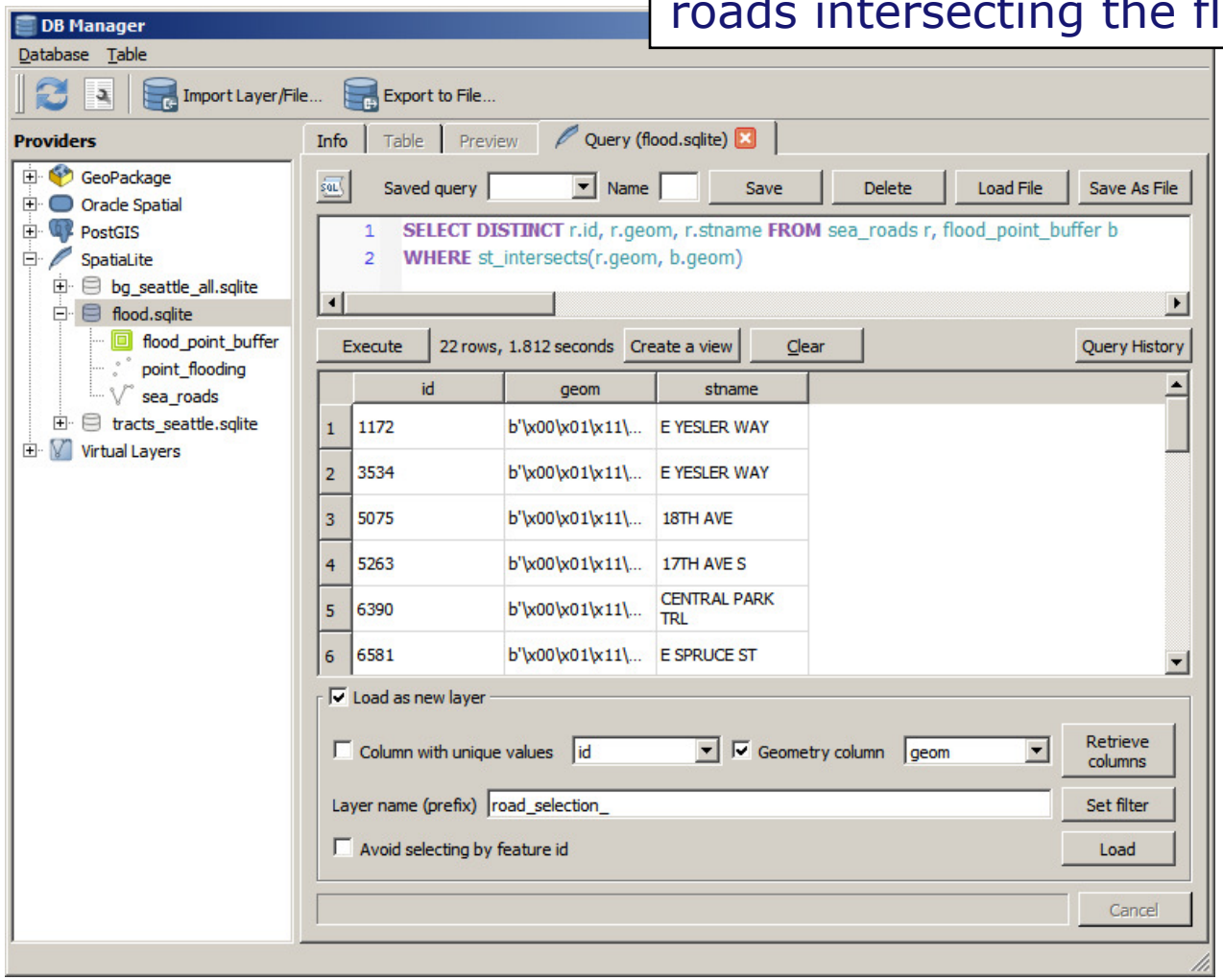
Needs a unique ID column !



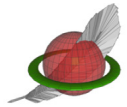
Example - Dynamic Layers with Spatialite

How was this done ?

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



3. Combining PostGIS Database Views and dynamic queries



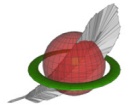
plus we will use

- 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

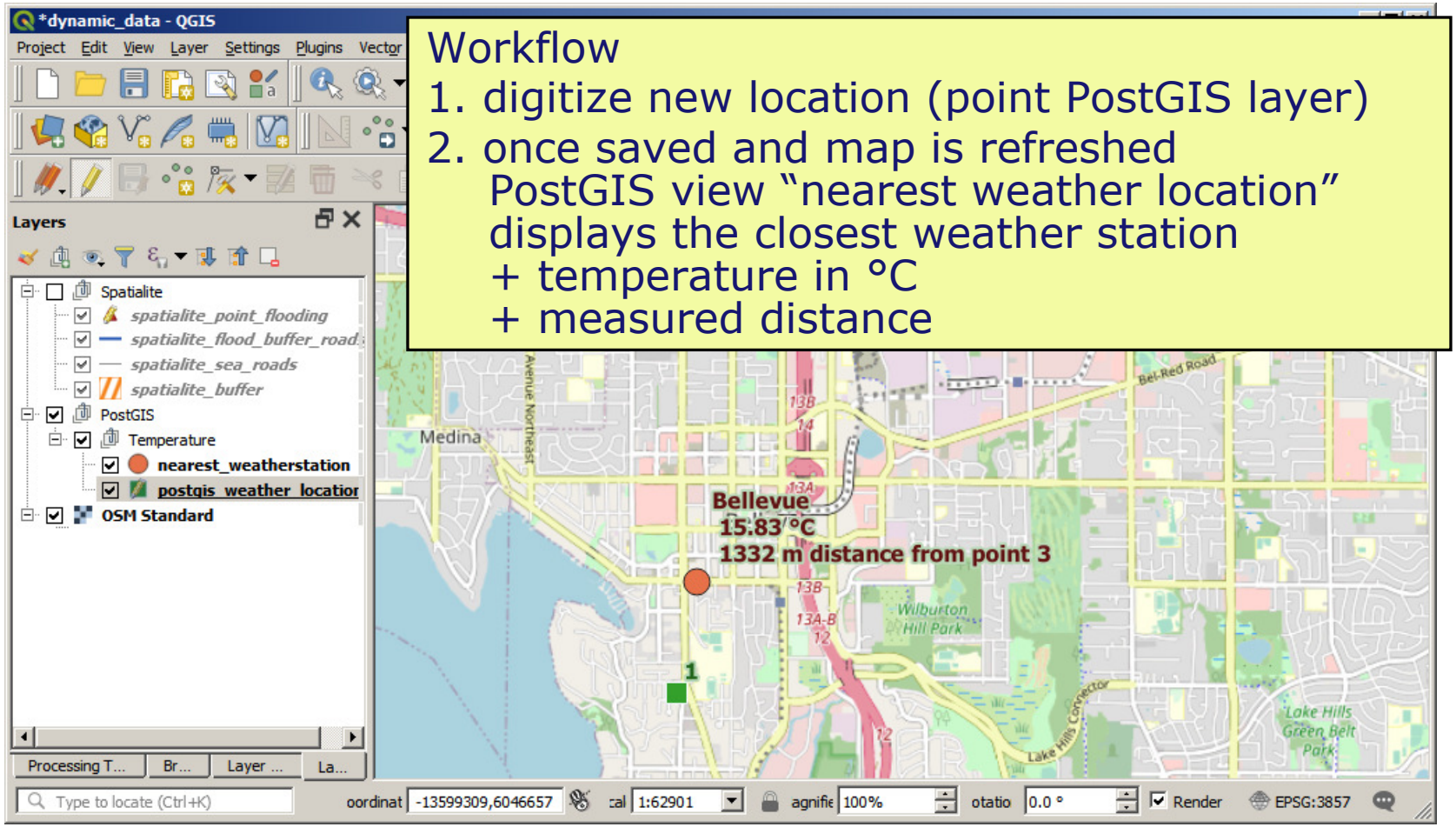
- custom plpython function GetWeatherAll(lon, lat)
  → takes a coordinate pair long/la and retrieves current temperature at closest weather stations plus location coordinates of the closest weather station
- A dynamic display of those data in QGIS is enabled via a PostGIS Database view joining a PostGIS 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***

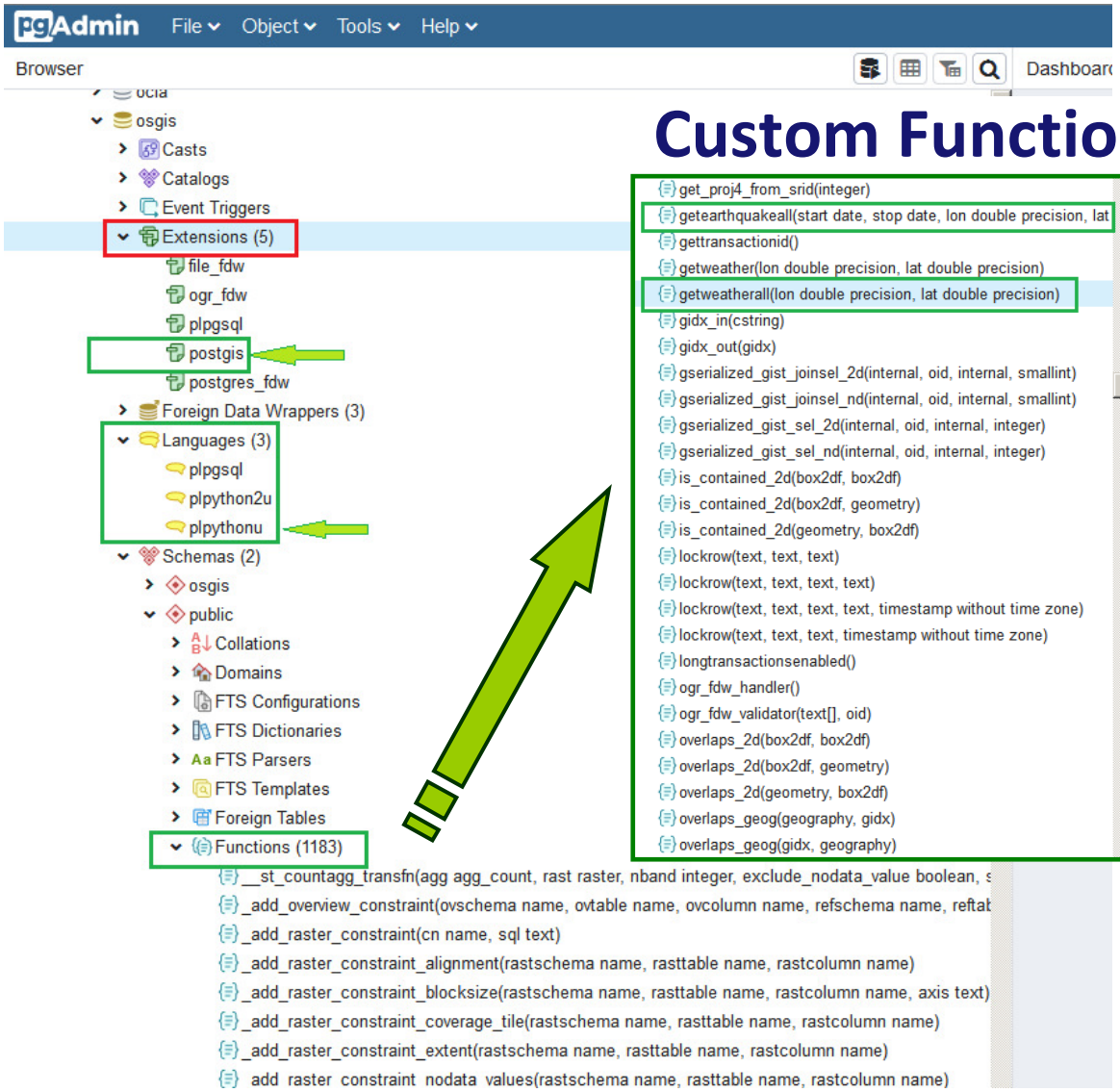
Result

PostGIS Database View with current weather data

via dynamic Weather API query in plpython function



PostGIS Functions - Spatial SQL



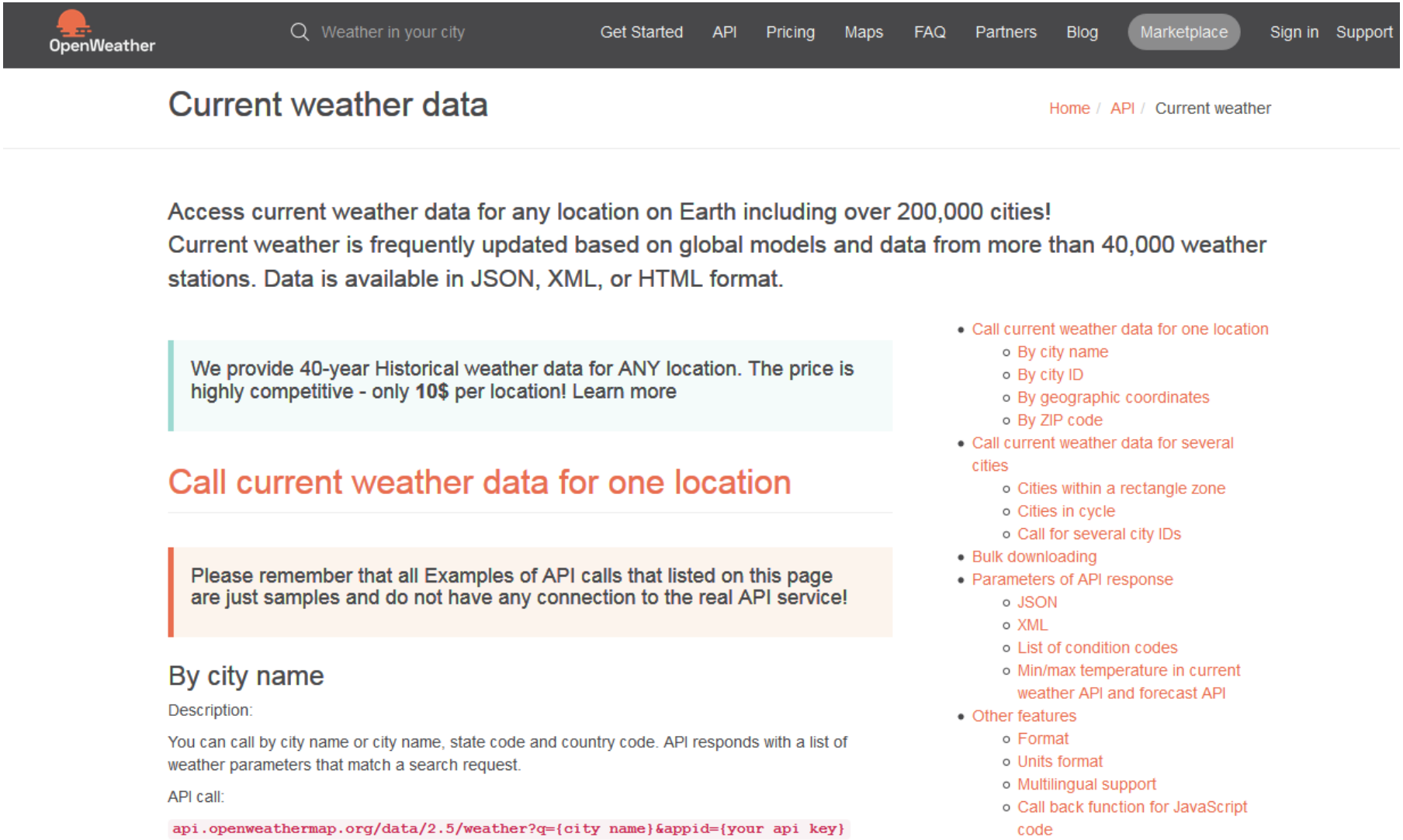
Custom Functions - plpython

get_proj4_from_srid(integer)
getearthquakeall(start date, stop date, lon double precision, lat
gettransactionid()
getweather(lon double precision, lat double precision)
getweatherall(lon double precision, lat double precision)
gidx_in(cstring)
gidx_out(gidx)
gserialized_gist_joinsel_2d(internal, oid, internal, smallint)
gserialized_gist_joinsel_nd(internal, oid, internal, smallint)
gserialized_gist_sel_2d(internal, oid, internal, integer)
gserialized_gist_sel_nd(internal, oid, internal, integer)
is_contained_2d(box2df, box2df)
is_contained_2d(box2df, geometry)
is_contained_2d(geometry, box2df)
lockrow(text, text, text)
lockrow(text, text, text, text)
lockrow(text, text, text, text, timestamp without time zone)
lockrow(text, text, text, timestamp without time zone)
longtransactionsenabled()
ogr_fdw_handler()
ogr_fdw_validator(text[], oid)
overlaps_2d(box2df, box2df)
overlaps_2d(box2df, geometry)
overlaps_2d(geometry, box2df)
overlaps_geog(geography, gidx)
overlaps_geog(gidx, geography)

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

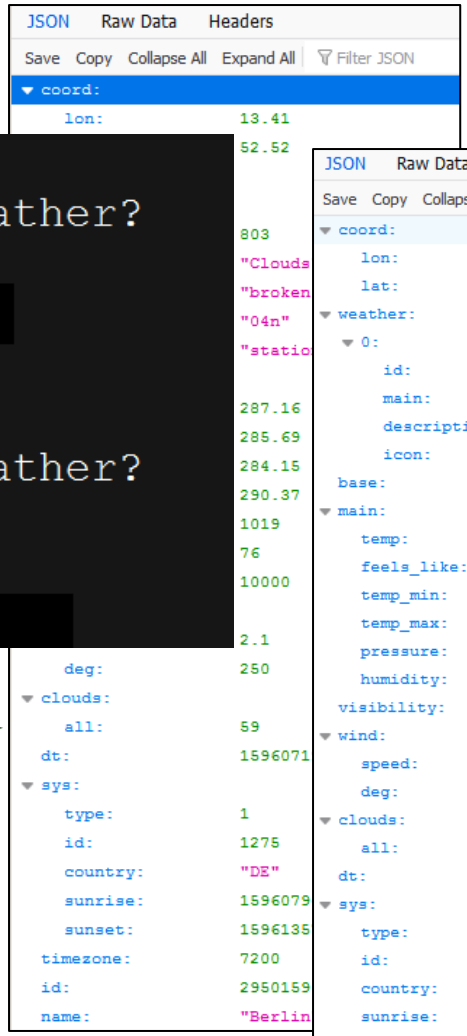


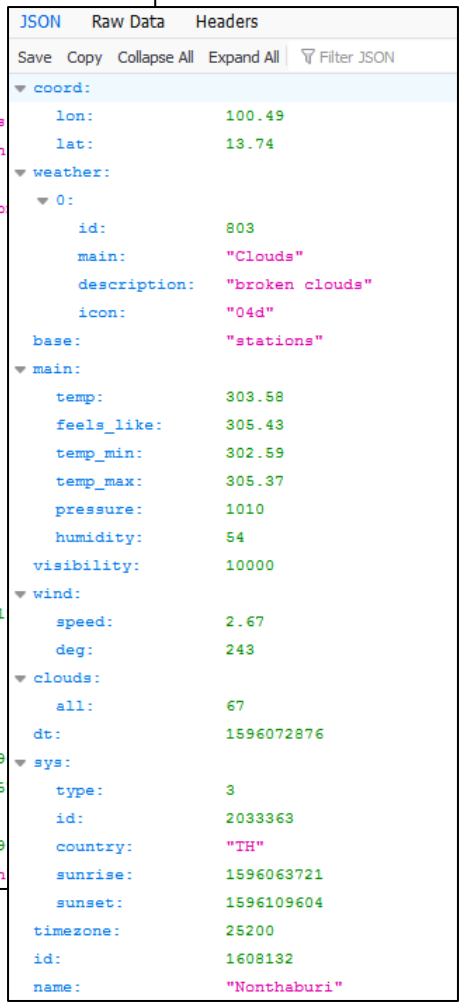
OpenWeathermap

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





Custom plpython (python2) Function in PostgreSQL

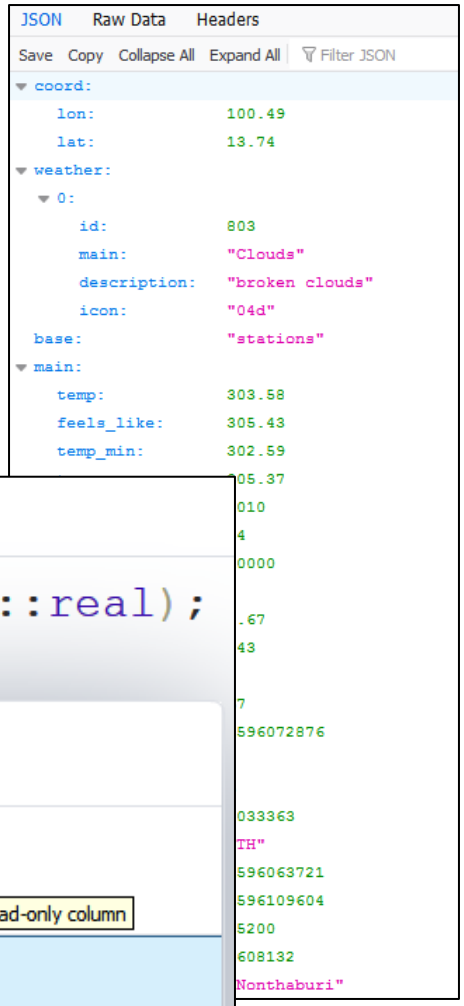
GetWeatherAll(lon float, lat float)

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



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

Read-only column

(31.15,100.5,13.75,Nonthaburi)

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 bigint	loc_x double precision	loc_y double precision	dist_m numeric	temperature double precision	clong double precision	clat double precision	geom geometry	cname text
1	1	4.37059297493893	43.8390708719451	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	43.2528992258199	6471	23.84	5.52	43.28	0101000020E610...	La Penne-sur-Huveaune
4	4	9.03262314990176	48.6905440474365	2266	13.75	9.02	48.68	0101000020E610...	Böblingen
5	5	8.73183280033311	49.4875126335156	3618	16.98	8.75	49.47	0101000020E610...	Wilhelmsfeld
6	6	-122.308166699283	47.6380686773299	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

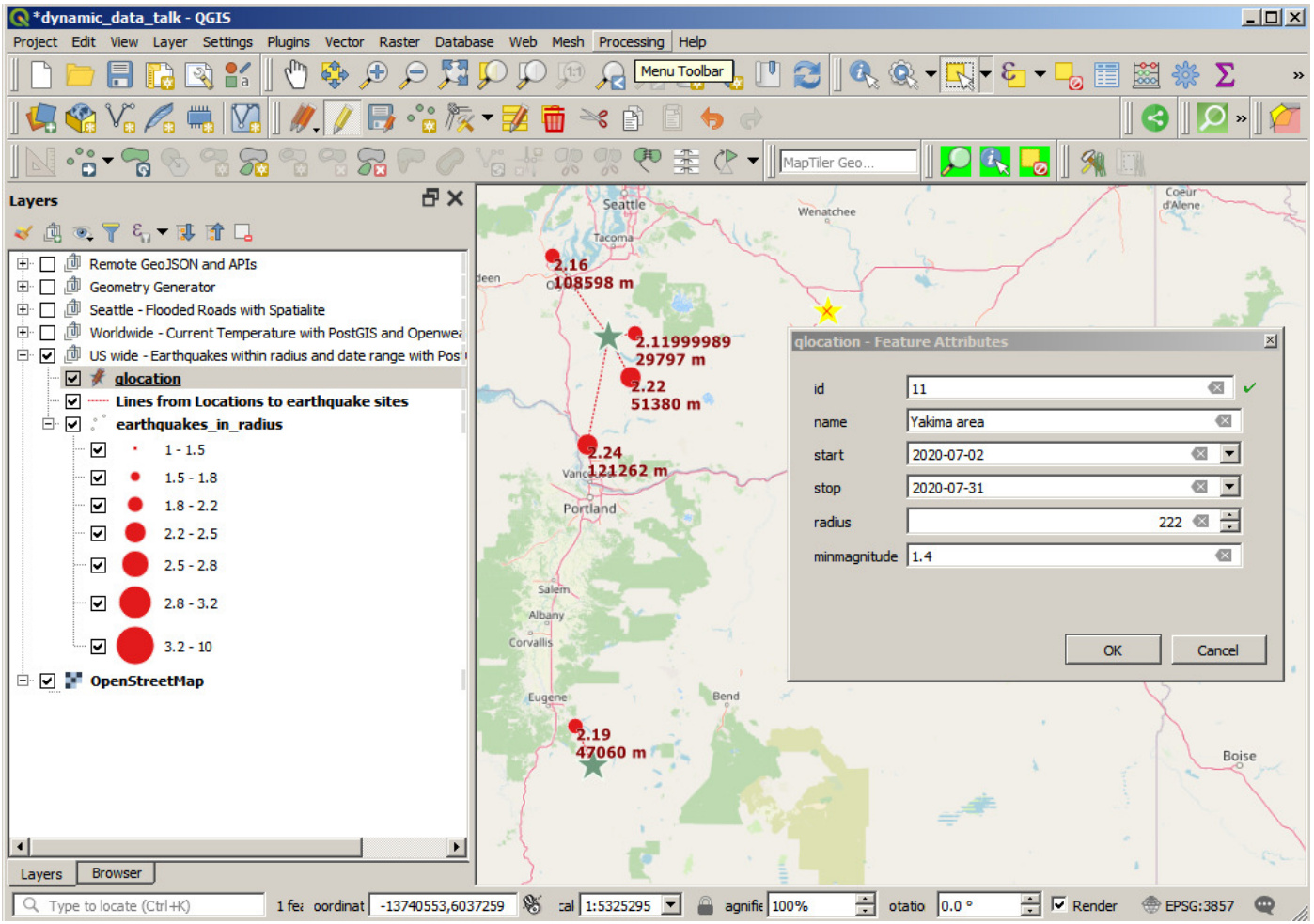
Example 2: USGS Earthquake API

- custom plpython function **getearthquakeall**
  → takes start + stop date, lon,lat, search radius km and min. magnitude and all earthquake events for choosen date range and minmum 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

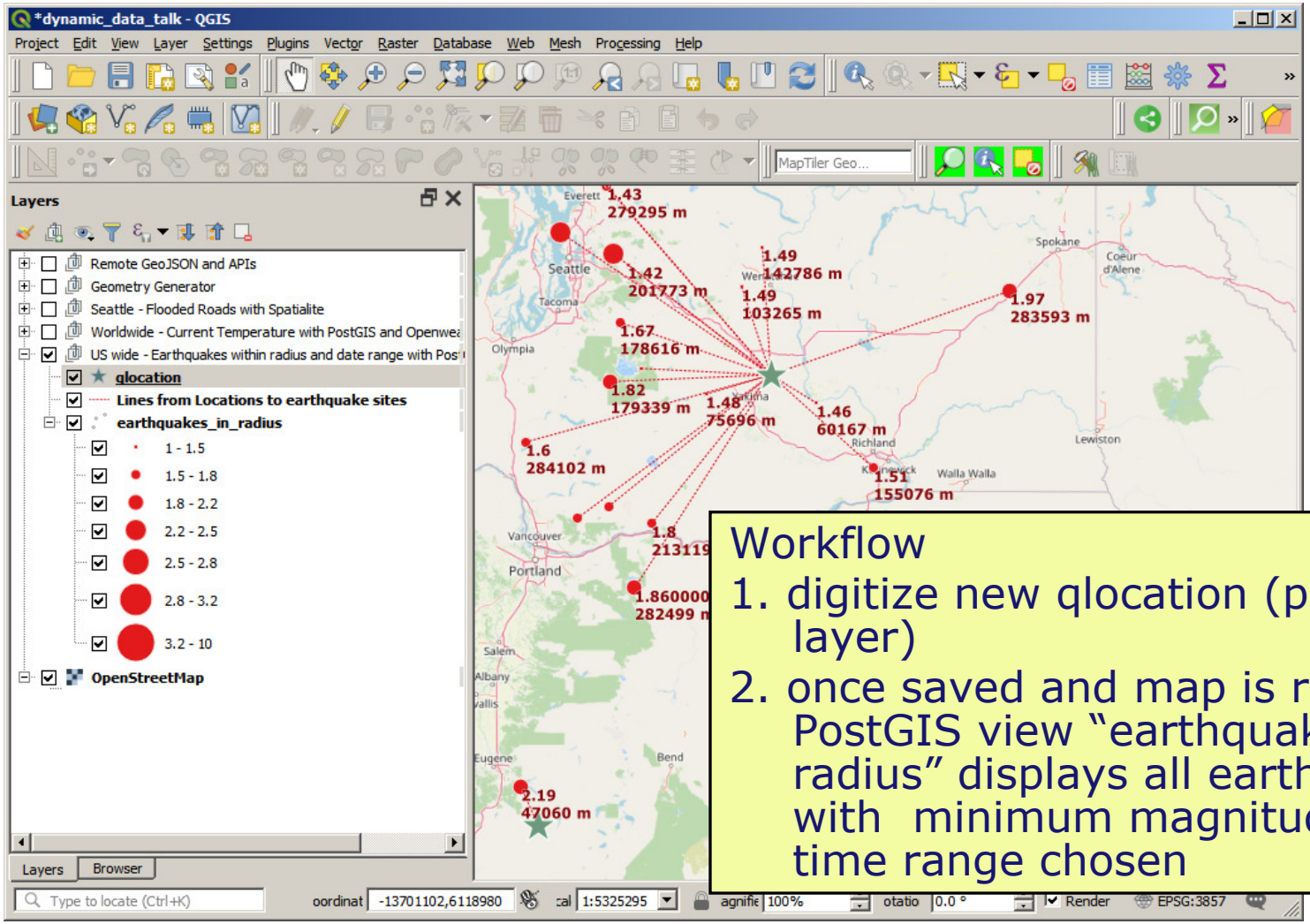
→ *Each time a new location is added to the PostGIS point layer the view will be updated, earthquake events that are corresponding to the minimum magnitude and time range retrieved from the USGS API and displayed in QGIS*

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

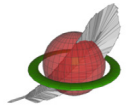


Workflow

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 equake_values AS $$
  import urllib2
  import json as json
  data =
  urllib2.urlopen('https://earthquake.usgs.gov/fdsnws/event/1/query?format=geojson&s
  tarttime=%s&endtime=%s&latitude=%s&longitude=%s&maxradiuskm=%s&minmagnitude=%s&ord
  erby=magnitude' % (start,stop,lat,lon,radius,minmagnitude))
  js_data = json.load(data)
  equake = js_data
  equakearray = []
  a = 0
  for i in equake['features']:
    equakeplace = i['properties']['place'] # tile for earthquake location
    magnitude =   i['properties']['mag']
    qlong =       i['geometry']['coordinates'][0]
    qlat =        i['geometry']['coordinates'][1]
    equakevalue = {"place": equakeplace, "magnitude": magnitude , "qlong": qlong,
    "qlat": qlat}
    equakearray.append(equakevalue)
    a = a+1
  return equakearray
$$ LANGUAGE plpythonu;

# create custom data type that is returned from equake  data API query
CREATE TYPE equake_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;
```

Database view qloc_earthquake_line for display in QGIS

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

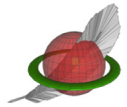
```
CREATE OR REPLACE VIEW osgis.qloc_earthquake_line AS
 SELECT row_number() OVER () AS gid,
    st_makeline(st_setsrid(eq.geom, 4326), st_setsrid(ql.geom, 4326)) AS geom,
    round(st_distance(st_transform(eq.geom, 3857), st_transform(ql.geom,
    3857))::numeric / 1000::numeric, 1) AS dist_km,
    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

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