What is coming in PostGIS 3.1 and new in PostGIS 3.0

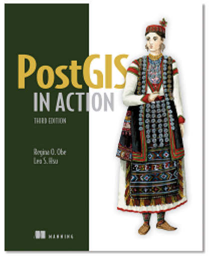
[(https://www.paragoncorporation.com)](https://www.paragoncorporation.com)

Regina Obe

[(https://postgis.net)](https://postgis.net)

PostGIS In Action 3rd Edition available for Purchase

- Get a copy of 2nd edition with purchase of 3rd edition
- Covers PostGIS 3 and 3.1
- Special promo 40% off (Jun 22 - Jun 30 2020) on all Manning E-Books and hard-copy books including PostGIS In Action 3d https://www.manning.com/books/postgis-in-action-third-edition (https://www.manning.com/books/postgis-in-action-third-edition)
- 12 of 17 chapters completed.
- https://www.postgis.us (https://www.postgis.us) find code and data here
- Live Book https://livebook.manning.com/book/postgis-in-action-third-edition (https://livebook.manning.com/book/postgis-in-action-third-edition)



Safe Harbor Statement

Some of the following mentions are forward looking statements and intended to outline the direction of PostGIS development.
They are not a commitment to deliver any code or functionality and should not be relied upon in making life altering decisions. The development, release, and timing of any features or functionality described is subject to change without warning.

What is coming in PostGIS 3.1?

Manual: https://postgis.net/docs/manual-dev/ (https://postgis.net/docs/manual-dev/)

Windows: https://postgis.net/windows_downloads/ (https://postgis.net/windows_downloads/) (has fresh updates for PostgreSQL 11,12,13)

Debian and Ubuntu: https://apt.postgresql.org (https://apt.postgresql.org) (has 3.1.0alpha1)

New Functions in PostGIS 3.1 so far

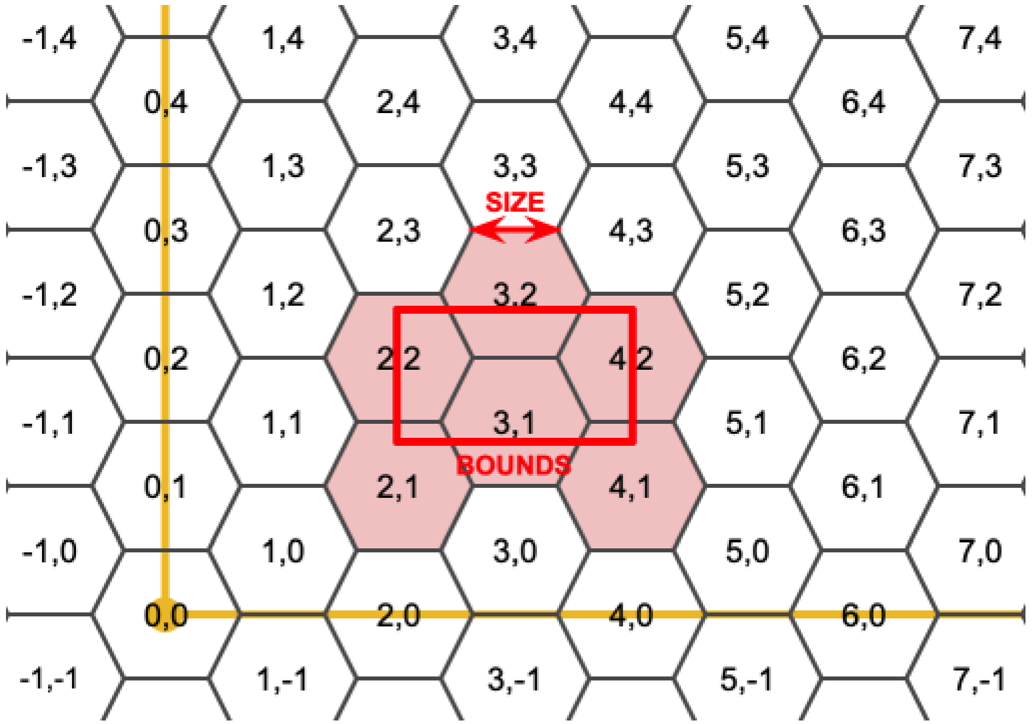
- Gridding functions - ST_HexagonGrid , ST_SquareGrid,(in 3.1.0alpha1)
- ST_MaximumInscribedCircle (in master)

PostGIS 3.1 functions ST_HexagonGrid

[https://postgis.net/docs/manual-dev/ST_HexagonGrid.html (https://postgis.net/docs/manual-dev/ST_HexagonGrid.html)](https://postgis.net/docs/manual-dev/ST_HexagonGrid.html)

1. ST_HexagonGrid creates a grid of the bounding box of geometry passed to it
2. Need ST_Intersects to filter out the hexagons that don't intersect the geometry

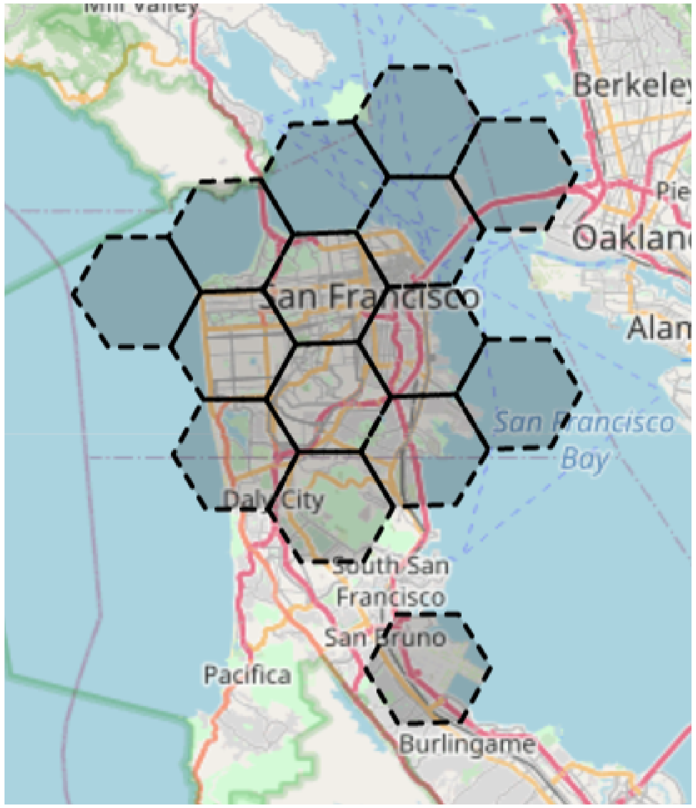
3. The size is length of an edge on the hexagon



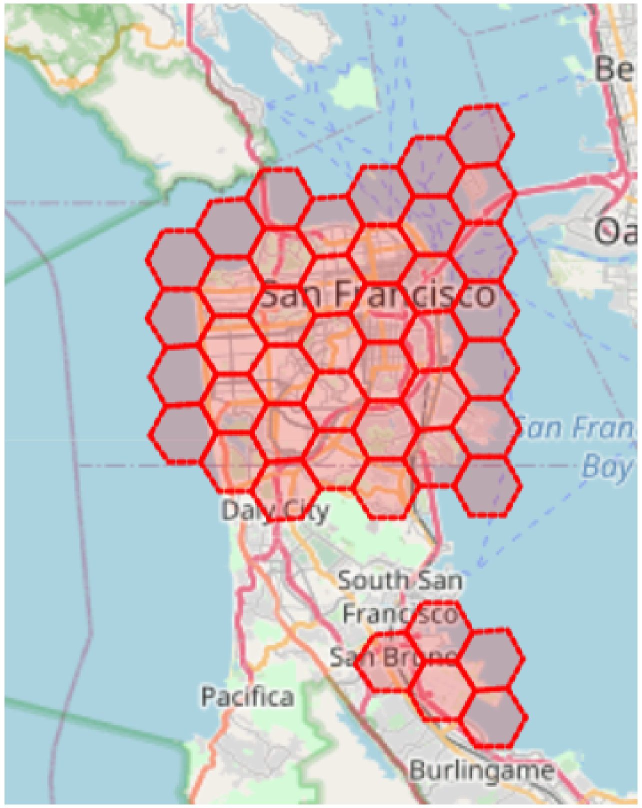
In this case we are using Northern CA State Plane feet.

```
SELECT grid.i, grid.j, ST_Union(grid.geom) AS geom
FROM ch11.cities AS c
    INNER JOIN ST_HexagonGrid(10000, c.geom) AS grid ON ST_Intersects(c.geom, grid.geom)
    WHERE c.city = 'SAN FRANCISCO'
GROUP BY grid.i, grid.j, grid.geom;
```

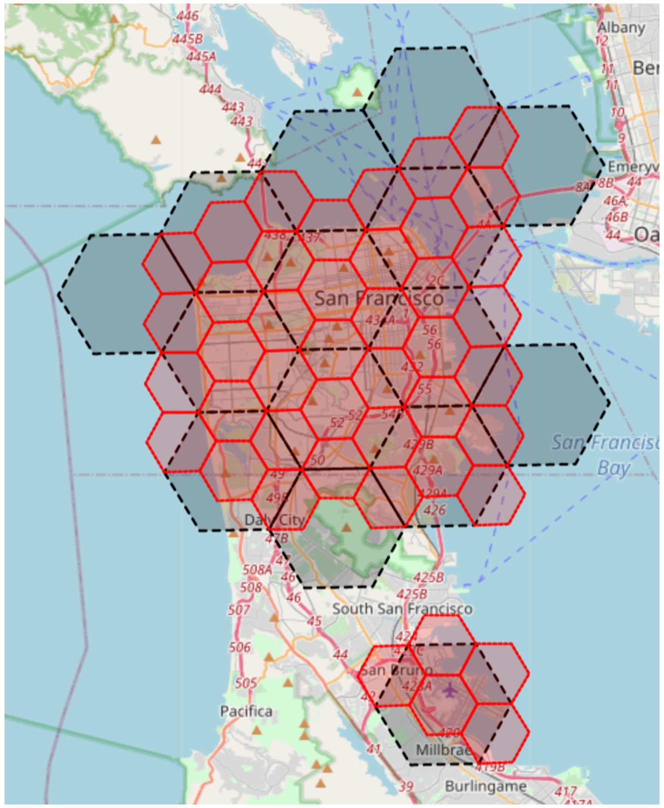
10,000 feet edge size



5,000 feet edge size



Hexagons cut at the middle, not able to nest a hexagon completely in another



PostGIS 3.1 functions ST_SquareGrid

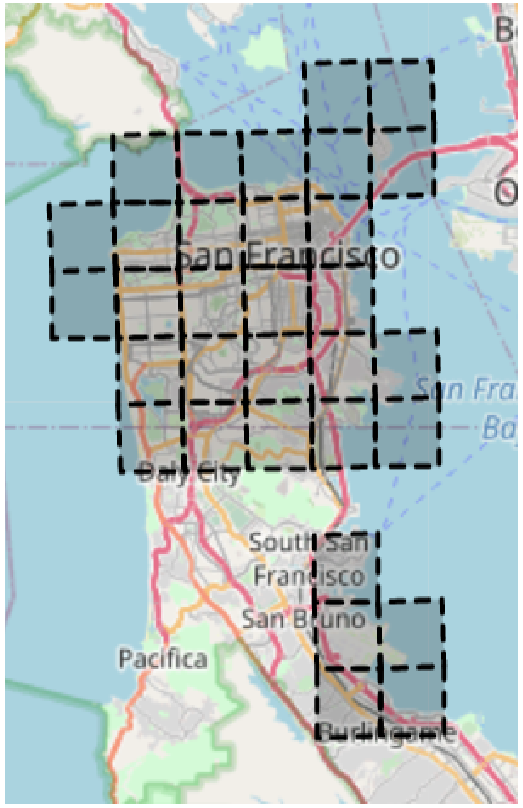
https://postgis.net/docs/manual-dev/ST_SquareGrid.html (https://postgis.net/docs/manual-dev/ST_SquareGrid.html)

1. ST_SquareGrid creates a grid of the bounding box of geometry passed to it
2. Need ST_Intersects to filter out the squares that don't intersect the geometry
3. The size: 10,000 is length of an edge in measure of the units of the spatial reference system, in our case it would be California State Plane feet

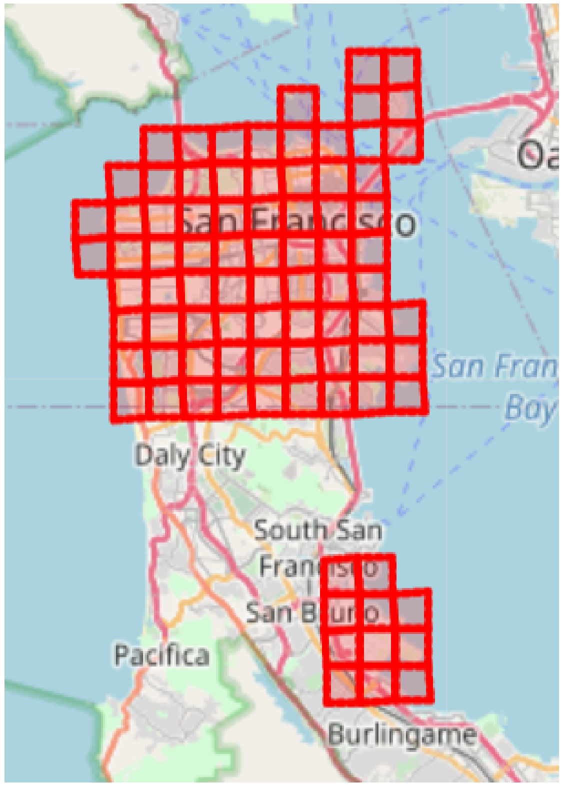
PostGIS 3.1 ST_SquareGrid San Francisco

```
SELECT grid.i, grid.j, ST_Union(grid.geom) AS geom
FROM ch11.cities AS c
    INNER JOIN ST_SquareGrid(10000, c.geom) AS grid ON ST_Intersects(c.geom, grid.geom)
    WHERE c.city = 'SAN FRANCISCO'
GROUP BY grid.i, grid.j, grid.geom;
```

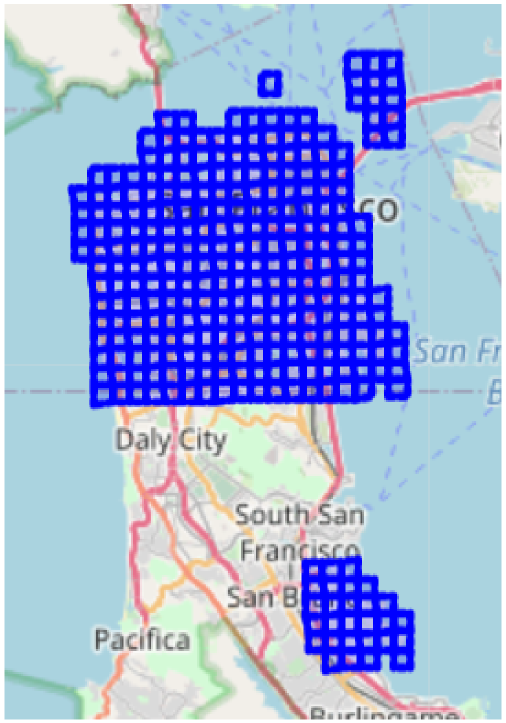
10,000 feet edge size



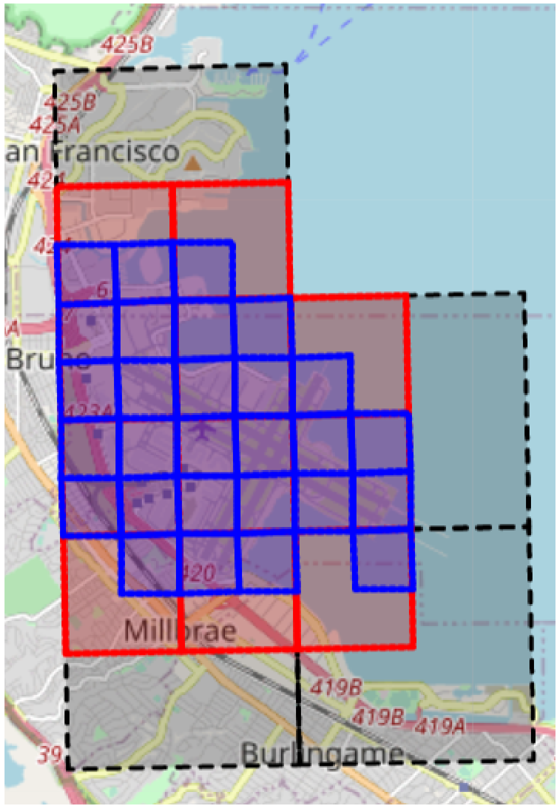
5,000 feet edge size



1,000 feet edge size



Square Grids are neatly divisible fit into smaller grids

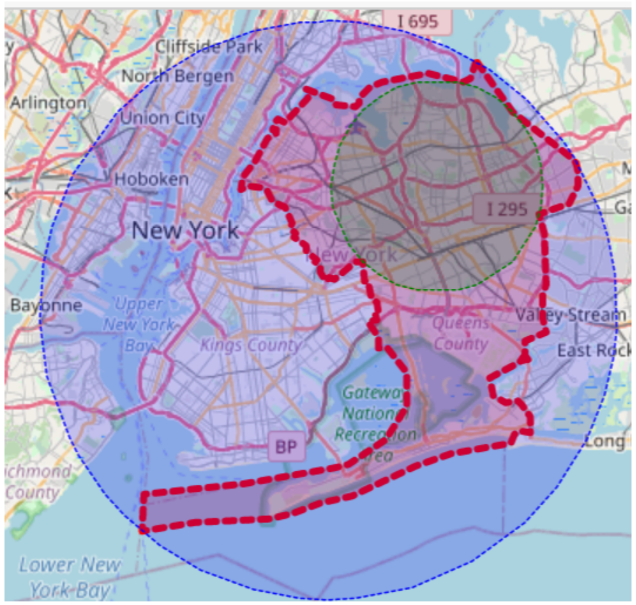


PostGIS 3.1: MaximumInscribedCircle - new vs. MinimumBoundingCircle

http://postgis.net/docs/manual-dev/ST_MaximumInscribedCircle.html
(http://postgis.net/docs/manual-dev/ST_MaximumInscribedCircle.html)
http://postgis.net/docs/ST_MinimumBoundingCircle.html
(http://postgis.net/docs/ST_MinimumBoundingCircle.html)

```
SELECT  ic.radius, ic.center, ic.nearest, ST_Buffer(ic.center, ic.radius) As geom
 FROM ch11.boroughs AS c, ST_MaximumInscribedCircle(c.geom) AS ic
 WHERE boroname = 'Brooklyn';

SELECT  ST_MinimumBoundingCircle(c.geom) AS geom
 FROM ch11.boroughs AS c
 WHERE boroname = 'Brooklyn';
```



Improvements in PostGIS 3.1 so far

- Ability to cast geometry to geojson directly
- Detoasting of geometries avoided with && -- means much faster relation operations
- Textual output functions like ST_ASText, ST_AsGeoJSON etc, 5-100x faster
- topology.GetRingEdges now implemented in C (postgis_topology extension)
- ST_Force3* functions can now take a measure.

Not Committed yet to PostGIS 3.1, but there is hope

- Improved robustness of ST_Intersection and other functions if running GEOS 3.9.0. Should result in fewer `Topology Exception` errors when doing ST_Intersection and ST_Union
- Drop sfcgal (cgal binding) from postgis-3.so and spin-off as postgis_sfcgal

Key changes in PostGIS 3.0

- `postgis_raster` now separate extension from postgis, for easier management
- minor by default dropped, so lib will be postgis-3, postgis_raster-3, postgis_topology-3 (for 3.0 and 3.1)
- Uses supportfn plumbing introduced in PostgreSQL 12 -- means better parallelism support, more cases where parallel will kick in without any changes.
- dropped functions ST_Accum (use array_agg)
- SFCGAL 2-d overloads ST_Intersection, ST_Intersects these functions now natively support 2D TINS and Polyhedral surfaces
- SFCGAL ST_3DIntersects overload dropped - ST_3DIntersects native now supports TINS and solid Polyhedral Surfaces
- ST_AsMVT functions now faster and support feature id
- Enhanced ST_AsGeoJSON now can return a whole feature given a row
- On disk format changed - now has 8 bytes for header (not noticable), your data gets converted to new format when you update data, but not doing a pg_upgrade.

How you upgrade from PostGIS 2.* to PostGIS 3

```
ALTER EXTENSION postgis UPDATE; -- if running pre PostGIS 2.5

SELECT postgis_extensions_upgrade(); -- this unpackages raster

SELECT postgis_extensions_upgrade(); -- do again to repackage raster
```

How you install extensions in PostGIS 3+

```
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_raster; -- used to be part of postgis extension
CREATE EXTENSION postgis_sfcgal; -- much of the functionality like dealing with 2d TINS a
nd Polyhedralsurfaces now supported in PostGIS proper
CREATE EXTENSION postgis_topology;
CREATE EXTENSION postgis_tiger_geocoder CASCADE; -- needs postgis and fuzzystrmatch
```

Upgrading from PostGIS 3.0 to PostGIS 3.1

```sql
SELECT postgis_extensions_upgrade();
```

Minor version dropped from Lib

```
Before:
    postgis-2.5.so
    rtpostgis-2.5.so
    postgis-topology-2.5.so

In 3.0:
    postgis-3.so
    postgis-raster-3.so
    postgis-topology-3.so

In 3.1 for 3.forevermore:
    postgis-3.so
    postgis-raster-3.so
    postgis-topology-3.so
```

For developers who need to test both versions and cook their own PostGIS meal:

```
./configure --with-library-minor-version
```

PostGIS 3.0 uses index supportfn instead of In-lining

PostGIS 3.0 supportfn

In past below did not parallelize well without turning a bunch of knobs, with PostGIS 3+12 it does with very little change in settings aside from number of processors and gathers.

```
SELECT p.pid, p.geom, s.name As street_name
FROM parcels AS p
    INNER JOIN streets AS s ON ST_Intersects(p.geom, s.geom);
```

PostGIS Pre-3.0/PostgreSQL < 12 Relied on SQL Inlining for index magic

In 2.5 or PostgreSQL < 12, geometry ST_Intersects looked like:

```
CREATE OR REPLACE FUNCTION st_intersects(
    geom1 geometry,
    geom2 geometry)
    RETURNS boolean
    LANGUAGE 'sql'
    COST 100
    IMMUTABLE PARALLEL SAFE
AS $$SELECT $1 && $2 AND _ST_Intersects($1,$2)$$;
```

PostGIS 3+/PostgreSQL 12+ Uses magic support fn for index magic

geometry/geography ST_Intersects now looks like:

```
CREATE OR REPLACE FUNCTION st_intersects(
    geom1 geometry,
    geom2 geometry)
    RETURNS boolean
    LANGUAGE 'c'
    COST 10000
    IMMUTABLE STRICT PARALLEL SAFE
    SUPPORT postgis.postgis_index_supportfn
AS '$libdir/postgis-3', 'ST_Intersects';
```

PostGIS 3.0 enhanced GeoJSON support to now accept full features

very very old way of creating a feature collection - https://www.postgresonline.com/journal/archives/267-Creating-GeoJSON-Feature-Collections-with-JSON-and-PostGIS-functions.html (https://www.postgresonline.com/journal/archives/267-Creating-GeoJSON-Feature-Collections-with-JSON-and-PostGIS-functions.html) (painful)

New way

```
SELECT json_build_object('type', 'FeatureCollection', 'features',
    json_agg(ST_AsGeoJSON(c.*)::json) )
 FROM ch11.cities AS c
 -- (transform is just to convert to 2227 North CA stateplane feet)
 WHERE c.geom && ST_Transform(ST_MakeEnvelope(-122, 37.74, -121.5, 38,4326 ), 2227);
```

```
In [ ]: %%sql
        SELECT ST_AsGeoJSON(a.*)
         FROM ch09.airports AS a
            WHERE municipality ILIKE 'BOSTON%' LIMIT 3;
```

PostGIS 3.0 ST_ASMVT faster and more parallelizable

If you are new to Mapbox Vector Tiles, try using https://github.com/CrunchyData/pg_tileserv (https://github.com/CrunchyData/pg_tileserv) (Crunchy Data pg_tileserv). pg_tileserv is a minimalist tile server written in Go that leverages PostGIS MVT functions. General concept behind it detailed - https://info.crunchydata.com/blog/dynamic-vector-tiles-from-postgis (https://info.crunchydata.com/blog/dynamic-vector-tiles-from-postgis)

1. Download the binary for your OS
2. Create a shell script # Nix

   ```
   export DATABASE_URL=postgresql://postgres:password@localhost/postgis_in_action
   pg_tileserv
   ```

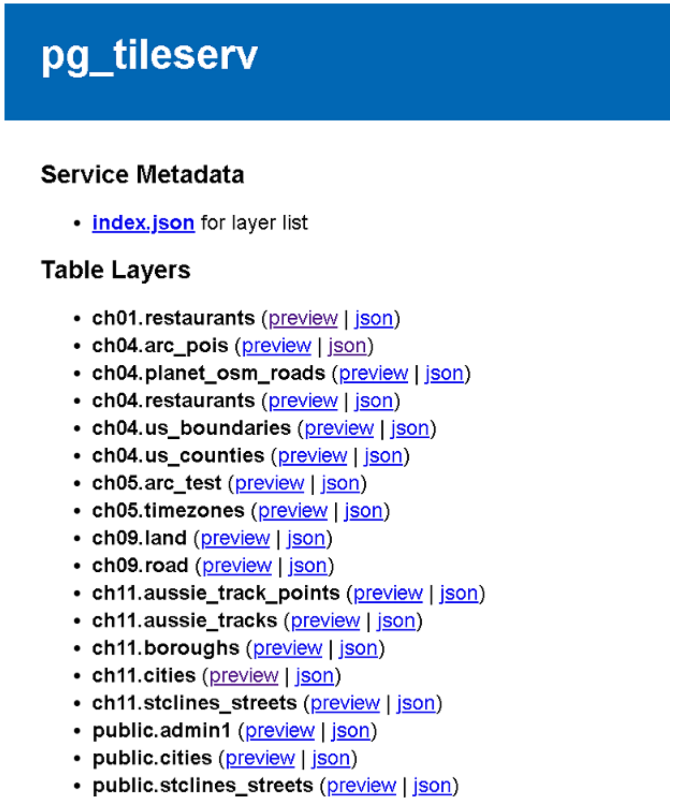
Windows

```
set DATABASE_URL=postgresql://postgres:password@localhost/postgis_in_action
pg_tileserv
```

1. Edit the packaged pg_tileserv.toml file if you want to change the port etc.

2. Browse to http://localhost:7800 (http://localhost:7800)

pg_tileserv catalog



pg_tileserv

Service Metadata

- **index.json** for layer list

Table Layers

- **ch01.restaurants** (preview | json)
- **ch04.arc_pois** (preview | json)
- **ch04.planet_osm_roads** (preview | json)
- **ch04.restaurants** (preview | json)
- **ch04.us_boundaries** (preview | json)
- **ch04.us_counties** (preview | json)
- **ch05.arc_test** (preview | json)
- **ch05.timezones** (preview | json)
- **ch09.land** (preview | json)
- **ch09.road** (preview | json)
- **ch11.aussie_track_points** (preview | json)
- **ch11.aussie_tracks** (preview | json)
- **ch11.boroughs** (preview | json)
- **ch11.cities** (preview | json)
- **ch11.stclines_streets** (preview | json)
- **public.admin1** (preview | json)
- **public.cities** (preview | json)
- **public.stclines_streets** (preview | json)

How do you use mapbox vector tiles for your maps?

In [ ]:

```python
from ipyleaflet import Map, VectorTileLayer, basemaps, LayersControl


# http://localhost:7800/ch04.us_counties/{z}/{x}/{y}.pbf #tile url for counties can use in leaflet
vlparcels = VectorTileLayer(name='Parcels', url='http://localhost:7800/staging.parcels/{z}/{x}/{y}.pbf')
vlff = VectorTileLayer(name='Fast Food', url='http://localhost:7800/ch01.restaurants/{z}/{x}/{y}.pbf')

m = Map(center=(42.38,-71.12), zoom = 15, basemap=basemaps.OpenStreetMap.BlackAndWhite)

m.add_layer(vlparcels)
m.add_layer(vlff)
m.add_control(LayersControl())
m
```

+
−
● OpenStreetMap.BlackAndWhite
✓ Parcels
✓ Fast Food
North Cambridge Carhouse
Davis
West Somerville
Cameron Avenue
Meacham Road
Dover Street
Day Street
Chester Street
College Avenue
Rindge Avenue