Using Uber H3 Indexing Library in Postgres for Geospatial Data Analytics

- Locate24

About Me:

- Jashanpreet Singh aka “json singh” - because “JSON” is what I speak
- Freelancer based out of India
- Full Stack GIS Dev - Frontend -> Backend -> Databases -> Cloud
- https://jsonsingh.com
- Also, I work with 2 cats

What's uber h3?

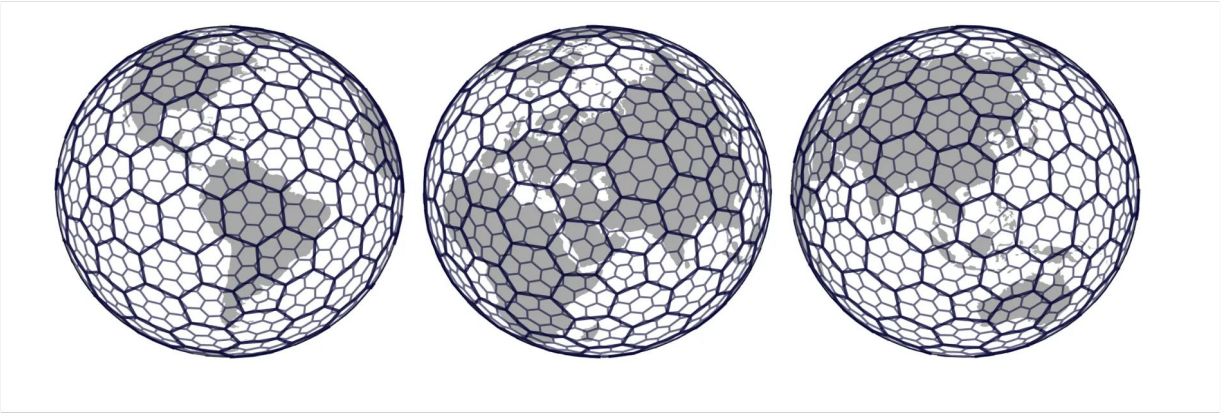
H3 is a hierarchical grid system designed to efficiently index and organize geospatial data. It divides the Earth's surface into hexagonal cells of various sizes, creating a hierarchical structure that allows for spatial indexing and querying.

The H3 grid system offers advantages such as spatial indexing, fast neighbor searches, and efficient spatial aggregation. It can be used in various applications involving geospatial data analysis, routing, visualization, and more. The H3 library provides APIs and tools for working with the grid system in different programming languages, making it accessible for developers.

TLDR;

- Convert spatial problems into relational problems
- Using Discreet GlobalGrid
- Representing your spatial that’s easier for the postgres to work

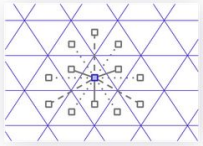
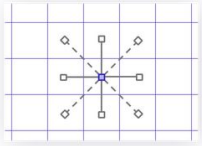
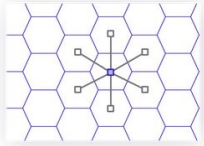
H3 indexing



Aggregation

Analysis of location data, such as locations of cars in a city, can be done by bucketing locations. (Sahr et al., 2003) Using a regular grid provides smooth gradients and the ability to measure differences between cells.

The cell shape of that grid system is an important consideration. For simplicity, it should be a polygon that tiles regularly: the triangle, the square, or the hexagon. Of these, triangles and squares have neighbors with different distances. Triangles have three different distances, and squares have two different distances. For hexagons, all neighbors are equidistant.

Triangle	Square	Hexagon
		
Triangles have 12 neighbors	Squares have 8 neighbors	Hexagons have 6 neighbors

Indexing

Uncompacted (dense)	Compacted (sparse)
	
California represented by 10633 uncompacted cells	California represented by 901 compacted cells

Choosing a resolution

Res	Average Hexagon Area (km^2)	Pentagon Area* (km^2)	Ratio (P/H)
0	4,357,449.416078381	2,562,182.162955496	0.5880
1	609,788.441794133	328,434.586246469	0.5386
2	86,801.780398997	44,930.898497879	0.5176
3	12,393.434655088	6,315.472267516	0.5096
4	1,770.347654491	896.582383141	0.5064
5	252.903858182	127.785583023	0.5053
6	36.129062164	18.238749548	0.5048
7	5.161293360	2.604669397	0.5047
8	0.737327598	0.372048038	0.5046
9	0.105332513	0.053147195	0.5046
10	0.015047502	0.007592318	0.5046
11	0.002149643	0.001084609	0.5046
12	0.000307092	0.000154944	0.5046
13	0.000043870	0.000022135	0.5046
14	0.000006267	0.000003162	0.5046
15	0.000000895	0.000000452	0.5046

Using h3 in postgres

Using h3 in postgres is as simple as installing the extension.

```
CREATE EXTENSION h3;
```

```
CREATE EXTENSION h3_postgis CASCADE;
```

The second line installs the postgis and postgis_raster extension as well which are required to use the h3.

Using h3 in postgres

The most common use case it create a h3index, in case of point it'll be single value where in case of polygon or line it'll be an array of h3index.

```
h3_lat_lng_to_cell(GEOM::POINT, INDEX)
```

And for Polygon:

```
h3_polygon_to_cells(shape, index)
```

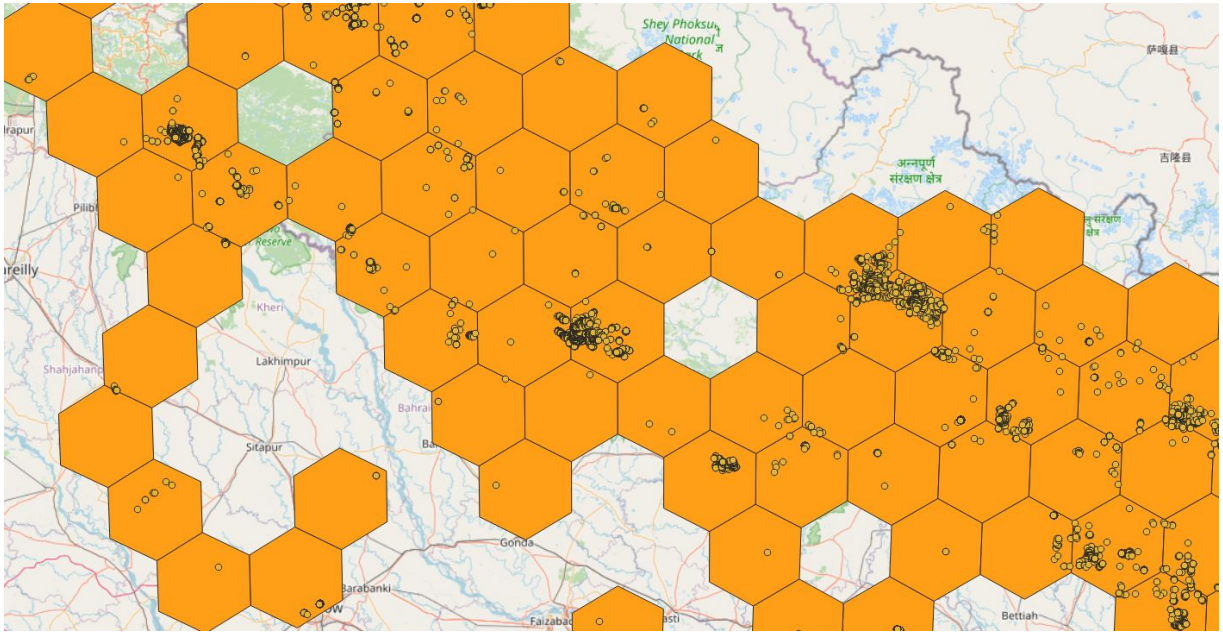
See the index shape

It might be helpful to see the h3index shape with the data it's indexing and for that we have the

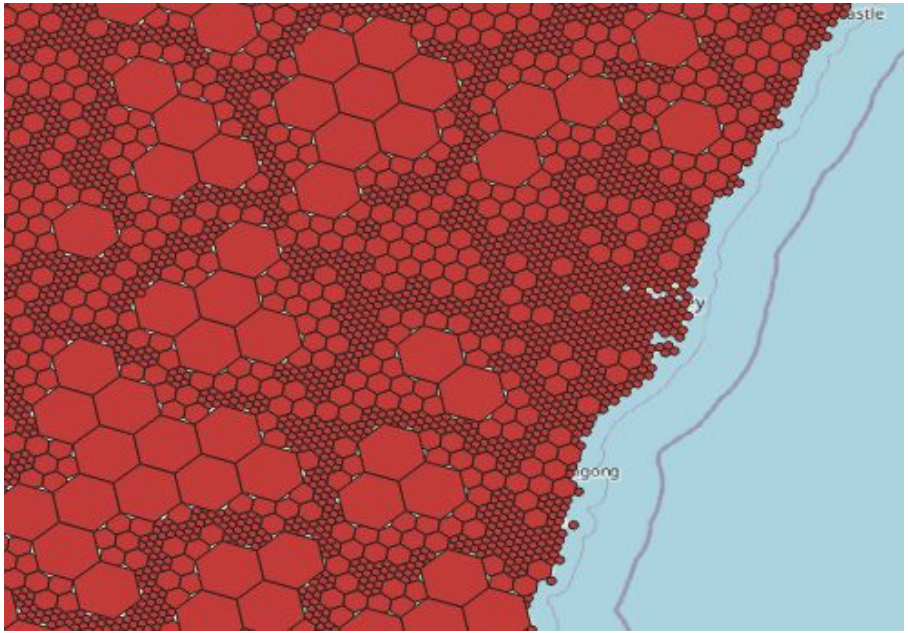
```
h3_cell_to_boundary_geometry(h3index)
```

Which converts the index to a shape which can visualize in QGIS or other places (can also be used for creating visualizations for frontend applications)

H3 index visualized



H3 index visualized



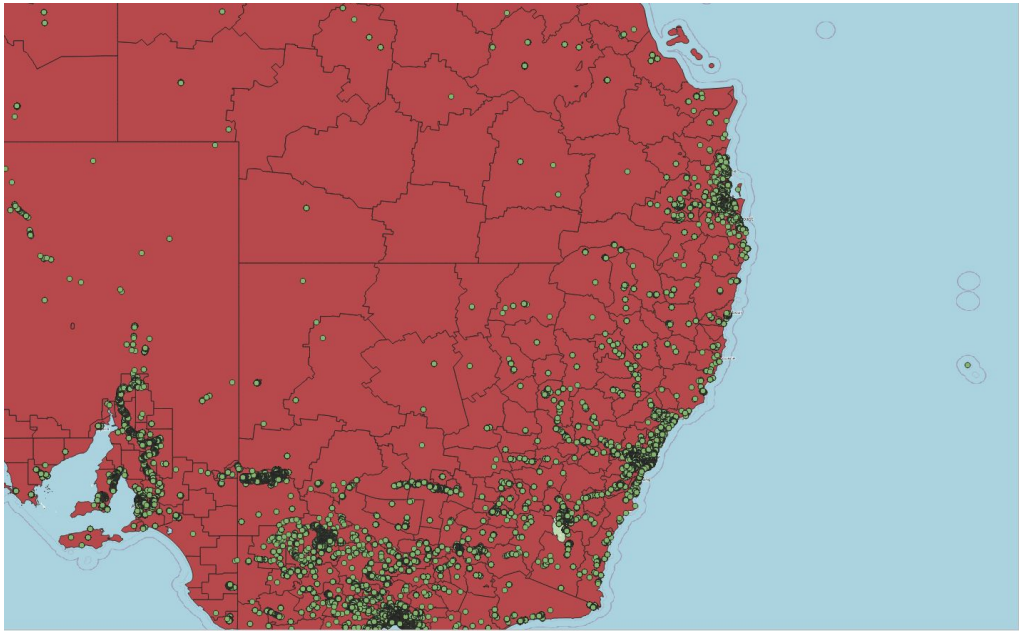
What is `h3index` type?

An unsigned 64-bit integer representing any H3 object (hexagon, pentagon, directed edge ...) represented as a (or 16-character) hexadecimal string, like '8928308280fffff'.

Which means all the optimizations that postgres can provide on integer will be applicable to h3index data type, some of which includes:

- Partitioning
- Faster Indexing

Problem Statement - Point in Polygon (Vector)



SQL Solution

```
select
  b.osm_id,
  count(a.osm_id)
from
  planet_osm_trees a
  right join
planet_osm_polygon_admin_2 b on
ST_Within(a.way, b.way)
group by
  b.osm_id
order by
  count;

– Cancelled after 1000 seconds
```

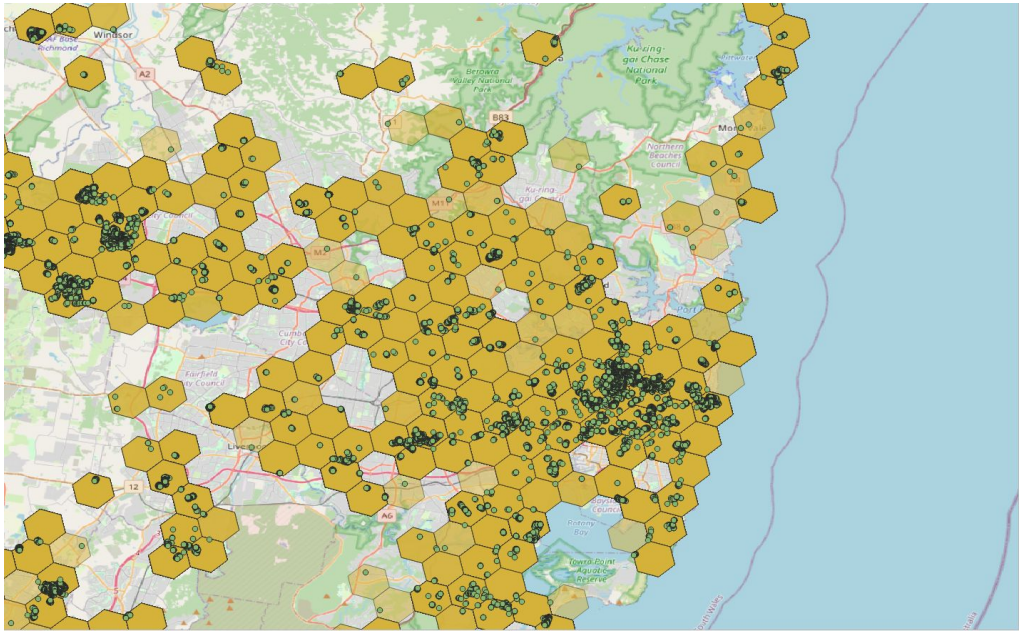
```
select
  b.osm_id,
  count(a.osm_id)
from
  planet_osm_trees a
  join planet_osm_polygon_admin_2_flat
b on a.h3_index = b.h3_index
group by
  b.osm_id
order by
  count;

– takes 4 secs
```

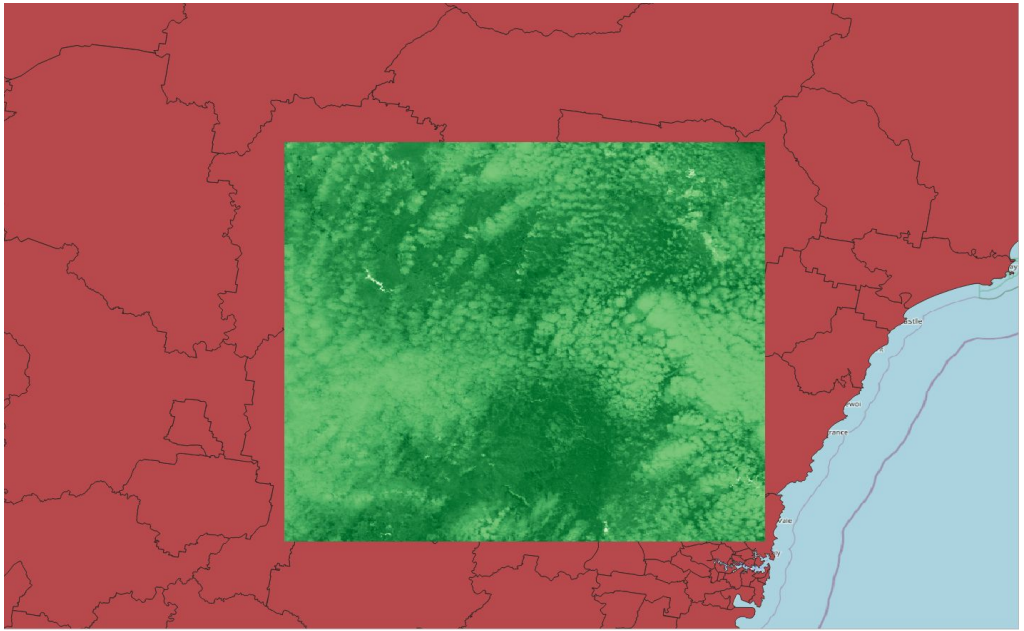
Source: https://github.com/jashanbhullar/foss4g-2023-spatial-analysis-in-postgres-using-uber-h3


This Is Business

Aggregation



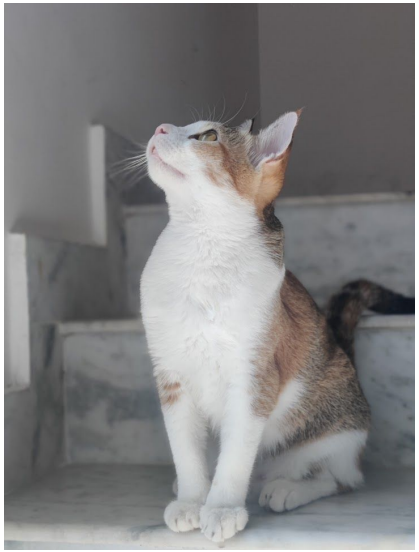
Problem Statement - Raster Analysis (Avg NDVI)



SQL Solution

```
SELECT b.osm_id,
     avg(a.mean)
FROM ndvi_hex a
JOIN planet_osm_polygon_admin_6_flat
b ON a.h3 = b.h3_index
GROUP BY b.osm_id;

– 1.1 seconds
```



Source: https://github.com/jashanbhullar/foss4g-2023-spatial-analysis-in-postgres-using-uber-h3

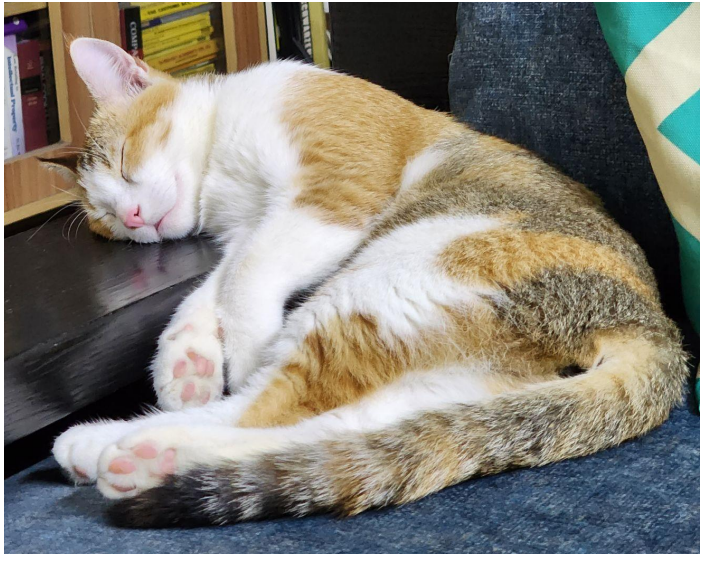
What more can we do?

Beyond representing your spatial data as integer values there are many use cases that can be explored with h3index

- **Raster analysis** - For rasters with pixel values representing continuous data (temperature, humidity, elevation), the data inside H3 cells can be summarized by calculating number of pixels, sum, mean, standard deviation, min and max for each cell inside a raster and grouping these stats across multiple rasters by H3 index
- **Machine Learning** - Since h3 index is built on functions for nearest neighbours, the data generated can be used by ML models for analysis and prediction
- **Visualization** - It might be helpful to visualize a huge amount aggregated in hexagons for the web and mobile applications. Can also be used at multiple resolutions for multiple zoom values.
- **Time Series Data** - You can aggregate and store time series data for a h3 index (since they are static) which can make it really easy to work with extensions like timescale.

Key Take Away

- Represent your spatial data as base64 integer and you'll all the power of postgres behind you
- Trading accuracy for speed
- Hexagons are cool



Sources

- https://www.youtube.com/watch?v=ay2uwtRO3QE&t=1044s&ab_channel=UberEngineering
- https://www.uber.com/en-IN/blog/h3/
- https://h3geo.org/docs/
- https://github.com/zachasme/h3-pg
- https://hub.docker.com/r/postgis/postgis
- https://h3geo.org/docs/community/tutorials
- Code: https://github.com/jashanbhullar/foss4g-2023-spatial-analysis-in-postgres-using-uber-h3
    - Branch - locate24
- Recommended - https://youtu.be/1Ocw_oaw_a8?si=shR_nZeefL3FbdM1

Thank You

You can reach me out on:

https://twitter.com/jsonsingh

https://www.linkedin.com/in/jsonsingh/

Or check more about me at: https://jsonsingh.com