Introduction to PostGIS

shortened version of

http://postgis.net/workshops/postgis-intro

(some slides ommitted, no slides changed)

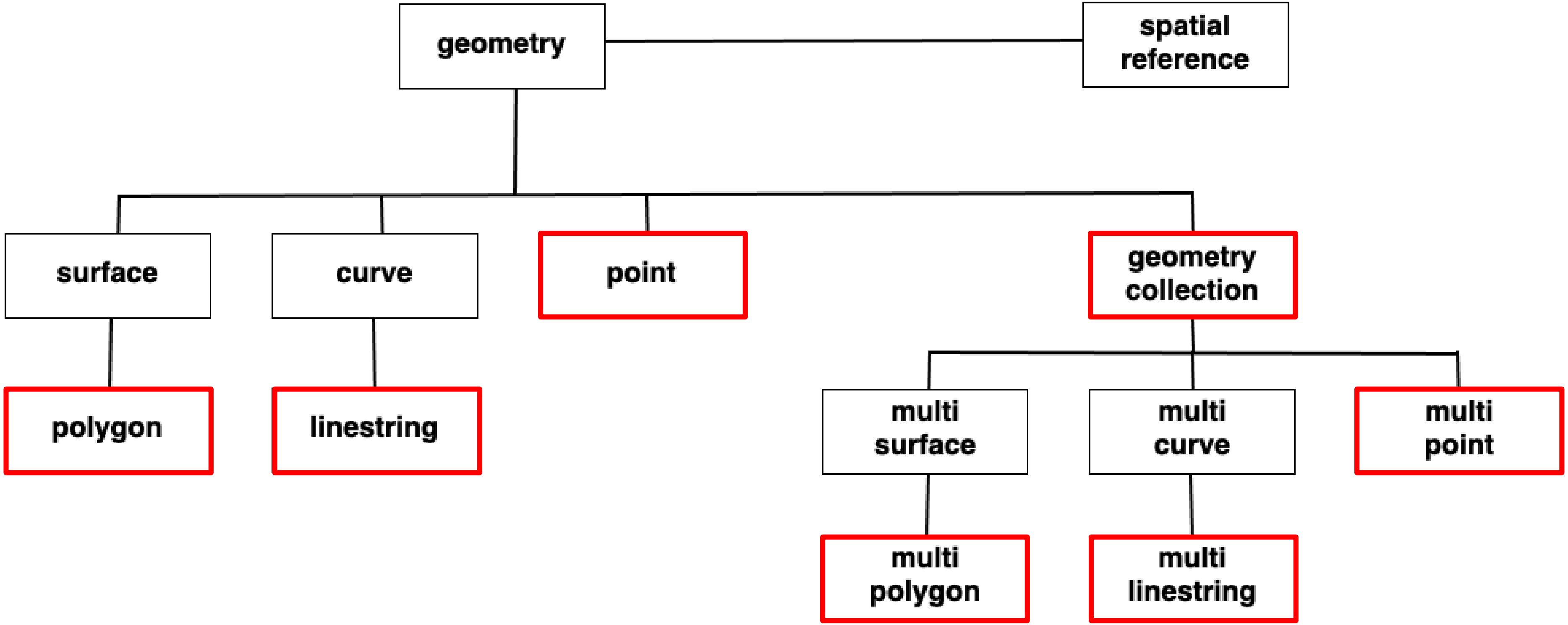
Section 2 - Introduction

What is a spatial database?

System for storage and random access of relationally (tables of rows and columns) structured data, providing the following capabilities for that data.

- **Data Types** including Spatial Types
  - number, date, string, geometry, geography and raster
- **Indexes** including Spatial Indexes
  - b-tree, hash, rtree, quadtree
- **Functions** including Spatial Functions
  - strlen(string), pow(float, float), now(), ST_Area(), ST_Distance()

Spatial Types



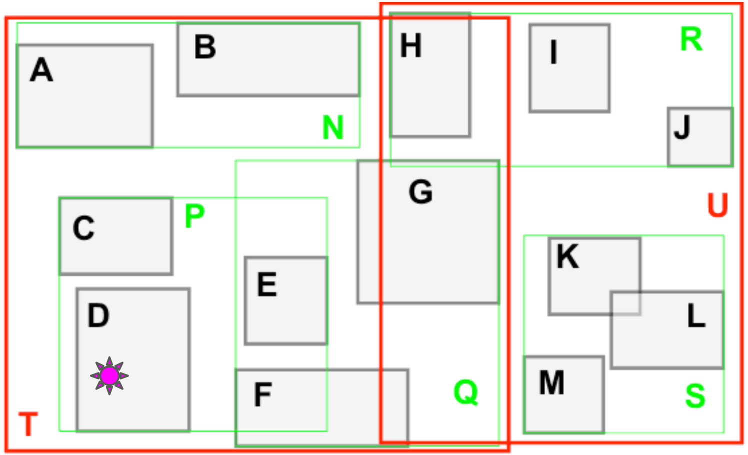
Spatial Indexes

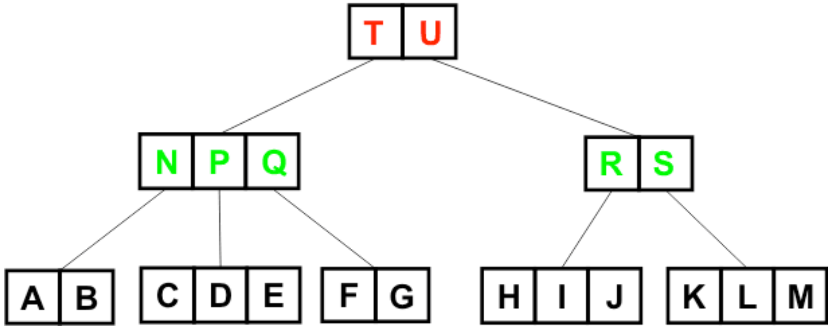
This R-Tree organizes the spatial objects so that a spatial search is a quick walk through the tree.

To find what object contains ?

- The system first checks if it is in **T** or **U** (**T**)
- Then it checks if it is in **N**, **P** or **Q** (**P**)
- Then it checks if it is in **C**, **D** or **E** (**D**)

Only 8 boxes have to be tested. A full table scan would require *all 13 boxes* to be tested. The larger the table, the *more powerful* the index is.





Spatial Functions

For example:

- ST_GeometryType(geometry) → text
- ST_Area(geometry) → float
- ST_Distance(geometry, geometry) → float
- ST_Buffer(geometry, radius) → geometry
- ST_Intersection(geometry, geometry) → geometry
- ST_Union([geometry]) → geometry

What is PostGIS?

PostGIS = + CREATE EXTENSION postgis;

Section 9 - Geometries

Creating a table with geometry

```
CREATE TABLE geometries
(
  name varchar,
  geom geometry
);
```

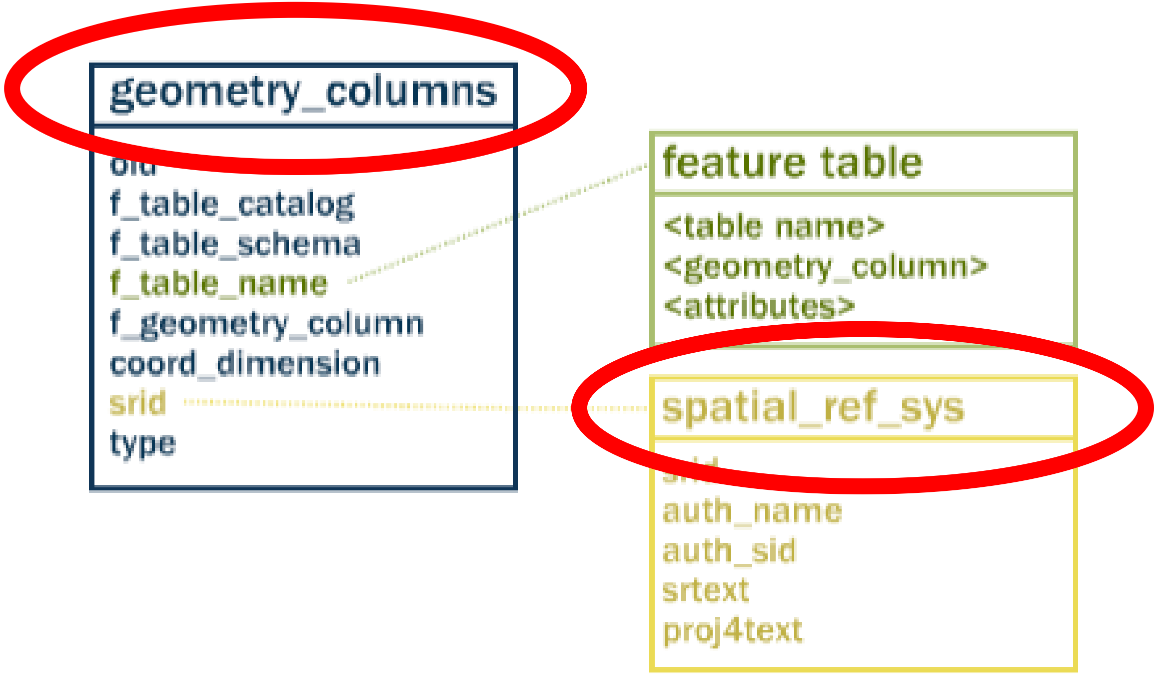
Creating a table with geometry

```
INSERT INTO geometries (name, geom) VALUES
  ('Point', 'POINT(0 0)'),
  ('Linestring', 'LINESTRING(0 0, 1 1, 2 1, 2 2)'),
  ('Polygon', 'POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'),
  ('PolygonWithHole', 'POLYGON((...))'),
  ('Collection', 'GEOMETRYCOLLECTION(...)');
```

Creating a table with geometry

```
SELECT name, ST_AsText(geom)
  FROM geometries;
```

Table Relationships



geometry_columns

```
SELECT *
FROM geometry_columns
```

geometry_columns

nyc/postgres@localhost

Query Editor | Query History | Scratch Pad

```
SELECT * FROM geometry_columns;
```

Data Output | Explain | Messages | Notifications

	f_table_catalog character varying (256)	f_table_schema name	f_table_name name	f_geometry_column name	coord_dimension integer	srid integer	type character varying (30)
1	nyc	public	nyc_census_blocks	geom	2	26918	MULTIPOLYGON
2	nyc	public	nyc_homicides	geom	2	26918	POINT
3	nyc	public	nyc_neighborhoods	geom	2	26918	MULTIPOLYGON
4	nyc	public	nyc_streets	geom	2	26918	MULTILINESTRING
5	nyc	public	nyc_subway_stati...	geom	2	26918	POINT

Metadata functions

```
SELECT
  name,
  ST_GeometryType(geom),
  ST_NDims(geom),
  ST_SRID(geom)
FROM geometries;
```

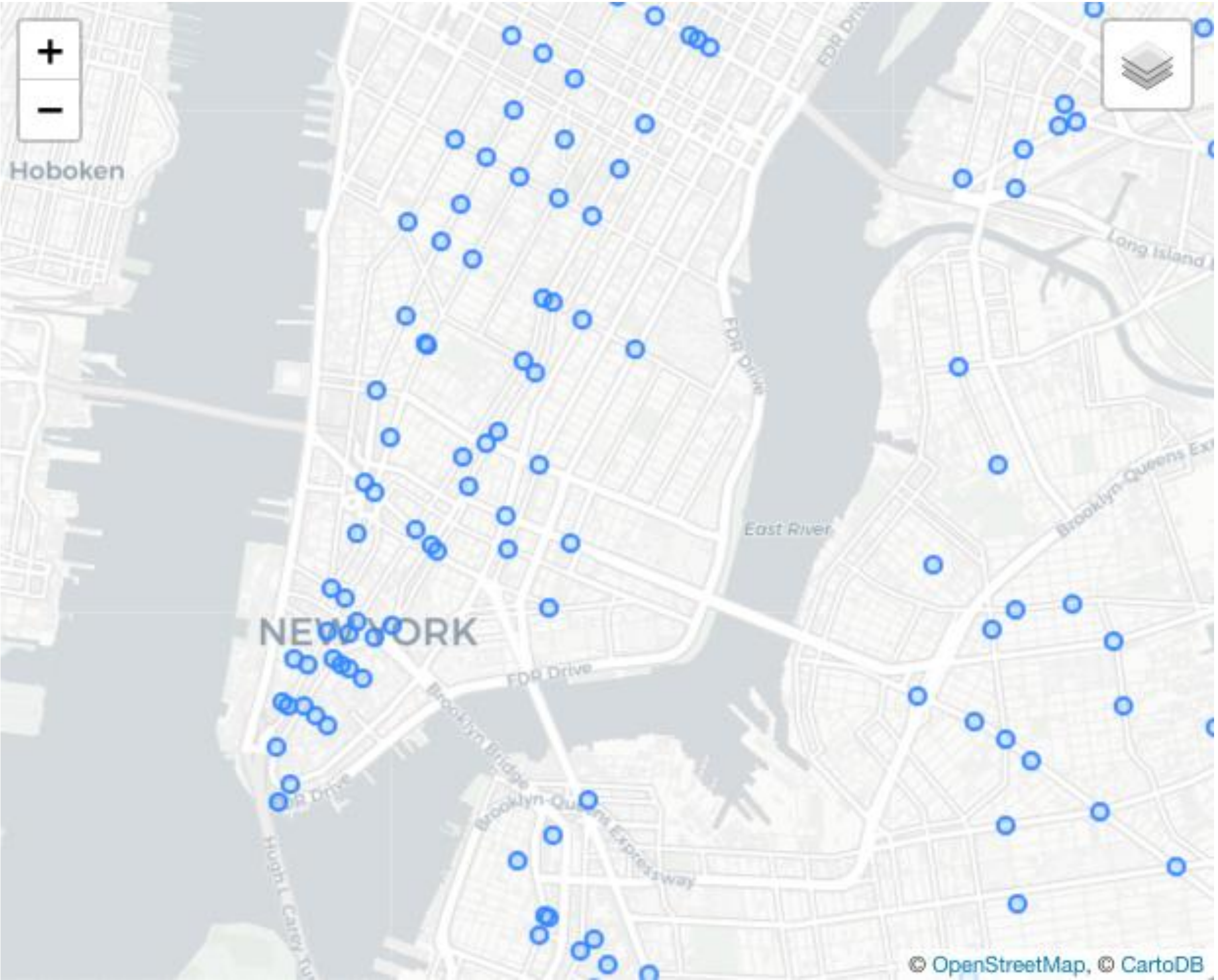
Metadata functions

name	st_geometrytype	st_ndims	st_srid
Point	ST_Point	2	0
Linestring	ST_LineString	2	0
Polygon	ST_Polygon	2	0
PolygonWithHole	ST_Polygon	2	0
Collection	ST_GeometryCollection	2	0

Points

"Point" or "MultiPoint", representing one or more 0-dimensional locations.

New York city subway stations, stop signs, man holes, address points, current locations of vehicles, might all use a "Point" geometry type.



Points

```sql
SELECT ST_AsText(geom)
FROM geometries
WHERE name = 'Point';
```

```
POINT(0 0)
```

Points

```
SELECT
   ST_X(geom),
   ST_Y(geom)
FROM geometries
WHERE name = 'Point'
```

```
0 0
```

Points

```
SELECT
   name,
   ST_AsText(geom)
FROM nyc_subway_stations
LIMIT 1;
```

```
Cortlandt St | POINT(583521 4507077)
```

LineStrings

"LineString" or "MultiLineString", representing one or more 1-dimensional objects.

Streets, streams, bus routes, power lines, driven routes, highways, might all use a "LineString" geometry type.

LineStrings

```
SELECT ST_AsText(geom)
FROM geometries
WHERE name = 'Linestring';
```

```
LINESTRING(0 0,1 1,2 1,2 2)
```

LineStrings

```
SELECT ST_Length(geom)
FROM geometries
WHERE name = 'Linestring';
```

```
3.41421356237309
```

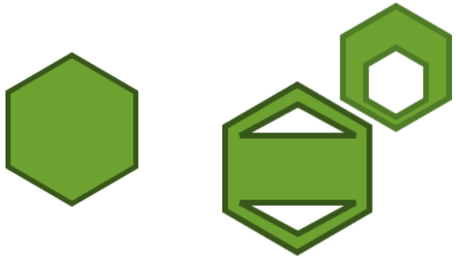
LineStrings

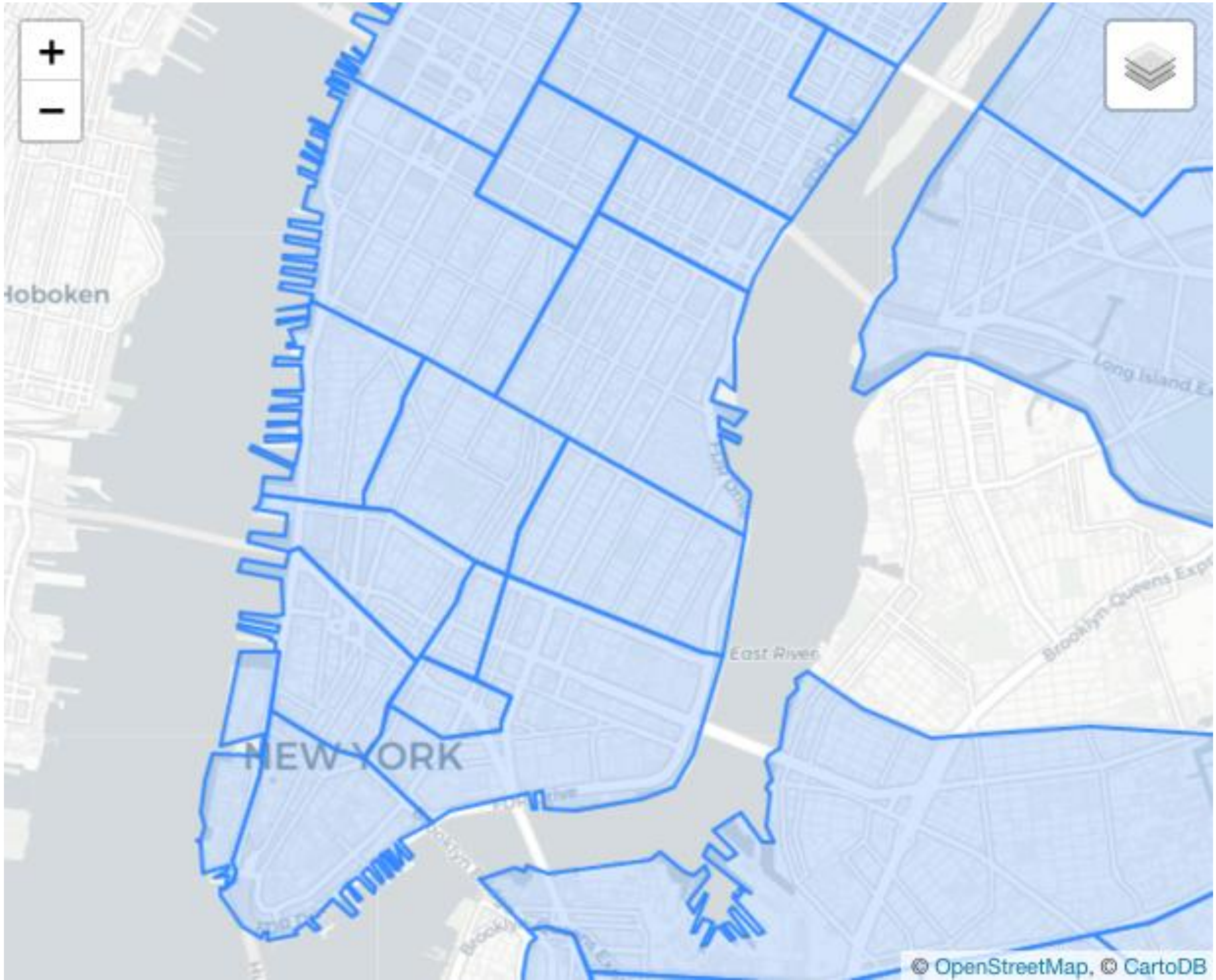
- ST_Length(linestring)
- ST_StartPoint(linestring)
- ST_EndPoint(linestring)
- ST_NumPoints(linestring)

Polygons

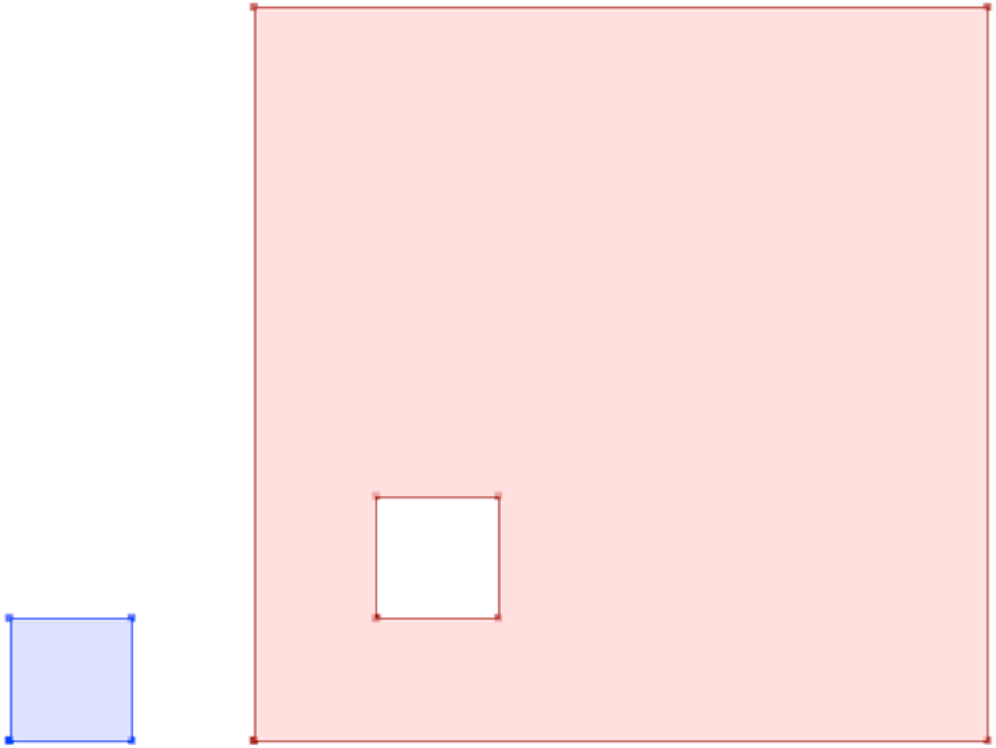
"Polygon" or "MultiPolygon", representing one or more 2-dimensional objects.

Census areas, parcels, counties, countries, neighborhoods, zoning areas, watersheds, and more.





Polygons



Polygons

```
SELECT ST_AsText(geom)
  FROM geometries
  WHERE name LIKE 'Polygon%';
```

```
POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))
POLYGON((0 0, 10 0, 10 10, 0 10, 0 0),
        (1 1, 1 2, 2 2, 2 1, 1 1))
```

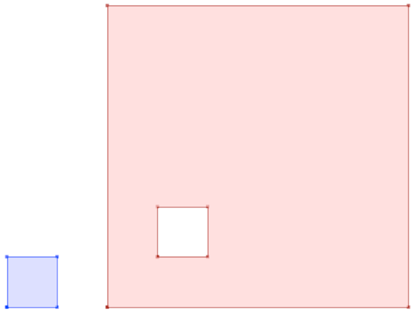
Polygons

- ST_Area(polygon)
- ST_NumInteriorRings(polygon)
- ST_ExteriorRing(polygon)
- ST_InteriorRing(polygon,n)
- ST_Perimeter(polygon)

Polygons

```
SELECT name, ST_Area(geom)
FROM geometries
WHERE name LIKE 'Polygon%';
```

```
Polygon         |  1
PolygonWithHole | 99
```



Geometry Formats

ST_As...

Text, EWKT, GML, KML, SVG, GeoJSON, Binary, EWKB

ST_GeomFrom...

Text, EWKT, GML, KML, GeoJSON, Binary, EWKB

Geometry Formats

```
SELECT ST_AsText(
  ST_GeometryFromText(
    'LINESTRING(0 0 0,1 0 0,1 1 2)'
  )
);
```

```
LINESTRING Z (0 0 0,1 0 0,1 1 2)
```

Geometry Formats

```
SELECT ST_AsGeoJSON(
  ST_GeomFromGML(
    '<gml:Point>
      <gml:coordinates>
       1,1
      </gml:coordinates>
     </gml:Point>'
  ));
```

```
{"type":"Point","coordinates":[1,1]}
```

All roads lead to Rome ... (Geometry construction)

```
SELECT ST_AsEWKT(
  ST_GeomFromText('POINT(1 1)', 4326)
);
```

```
SRID=4326;POINT(1 1)
```

All roads lead to Rome ... (Geometry construction)

```
SELECT ST_AsEWKT(
  ST_SetSRID(
    ST_GeomFromText('POINT(1 1)'),
    4326
  )
);
```

```
SRID=4326;POINT(1 1)
```

All roads lead to Rome … (Geometry construction)

```
SELECT ST_AsEWKT(
  ST_SetSRID(
    ST_MakePoint(1, 1),
    4326
  )
);
```

```
SRID=4326;POINT(1 1)
```

All roads lead to Rome … (Geometry construction)

```
SELECT ST_AsEWKT(
  ST_SetSRID(
    'POINT(1 1)'::geometry,
    4326
  )
);
```

```
SRID=4326;POINT(1 1)
```

All roads lead to Rome … (Geometry construction)

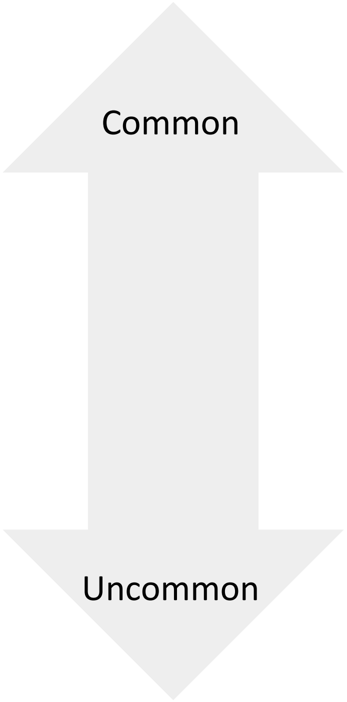
```
SELECT ST_AsEWKT(
   'SRID=4326;POINT(1 1)'::geometry
);
```

```
SRID=4326;POINT(1 1)
```

Section 11 - Spatial Relationships

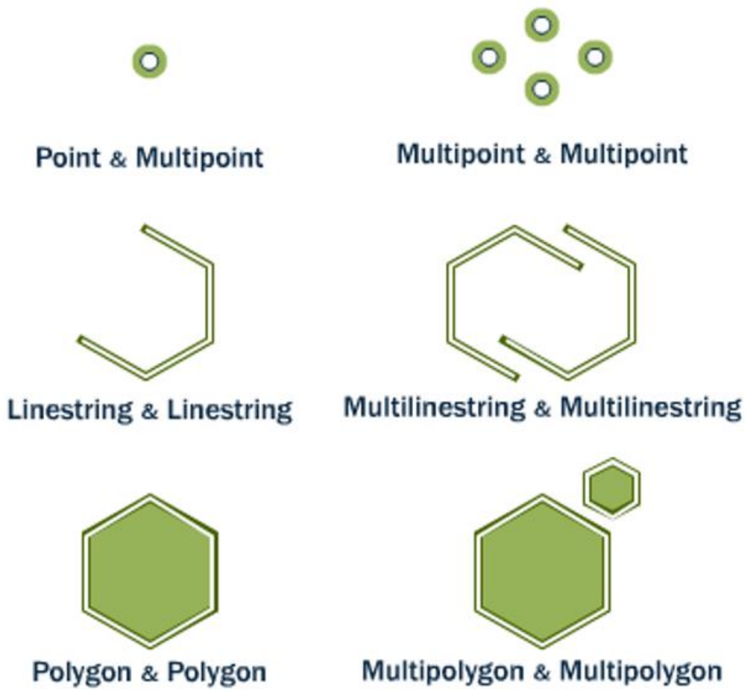
Spatial Relationship Functions

- ST_Intersects(A, B)
- ST_DWithin(A, B, d)
- ST_Distance(A, B)
- ST_Within, ST_Contains(A, B)
- ST_Equals(A, B)
- ST_Touches(A, B)
- ST_Disjoint, ST_Crosses, ST_Overlaps(A, B)



ST_Equals(A, B)
ST_OrderingEquals(A, B)

Equals tests that A and B cover the same space, regardless of representation differences (extra vertices, order of vertices).
OrderingEquals insists on structural identity.



What is geometry of Broad Street subway station?

```
SELECT name, geom
  FROM nyc_subway_stations
  WHERE name = 'Broad St';
```

```
0101000020266900000EEBD4CF27CF2141BC17D69516315141
```

What subway station record matches that geometry?

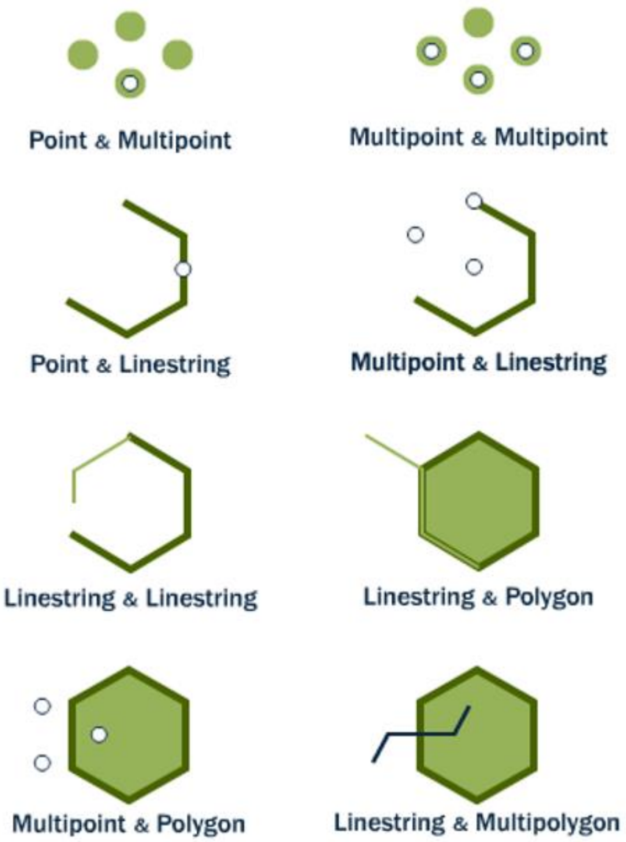
```
SELECT name
FROM nyc_subway_stations
WHERE ST_Equals(
  geom,
  '0101000020266900000EEBD4CF27CF2141BC17D69516315141'
);
```

```
Broad St
```

ST_Intersects(A, B)
ST_Disjoint(A, B)

Intersects and disjoint are opposites. Any kind of interactions between two shapes is an intersection, and implies the pair are not disjoint, and vice versa.

A intersects B ⇒ A not disjoint B
A disjoint B ⇒ A not intersects B



What is the well-known text (WKT) of Broad Street subway station?

```
SELECT name, ST_AsText(geom, 0)
  FROM nyc_subway_stations
  WHERE name = 'Broad St';
```

```
POINT(583571 4506714)
```

What neighborhood intersects that subway station?

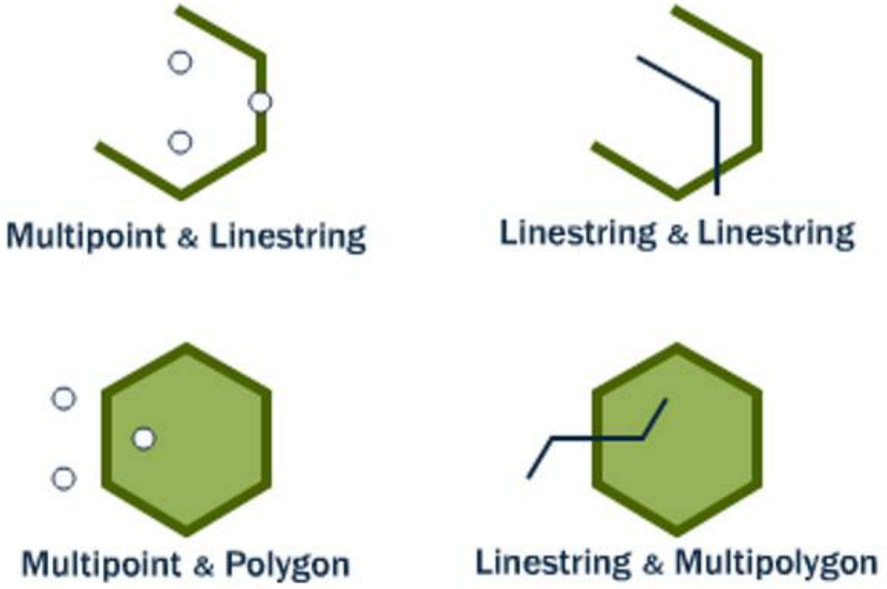
```
SELECT name, boroname
FROM nyc_neighborhoods
WHERE ST_Intersects(
   geom,
   ST_GeomFromText(
     'POINT(583571 4506714)',
     26918));
```

```
Financial District | Manhattan
```

ST_Crosses(A, B)

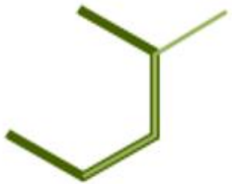
Mostly used to test linestrings, which can be said to cross when their interiors have interactions.

When linestrings cross polygon boundaries, the crosses condition is also true.



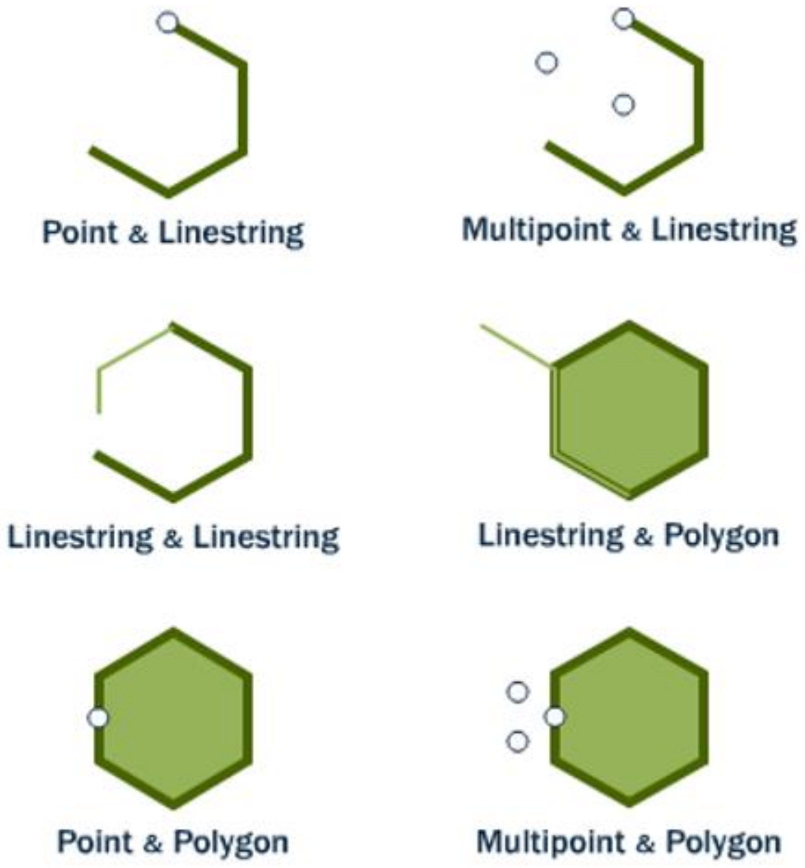
ST_Overlaps(A, B)

Shapes overlap when their interiors interact with each other and also with the exterior of the shape.
So objects that are contained or within do not overlap, overlaps is what normal people might call “partial overlap”.



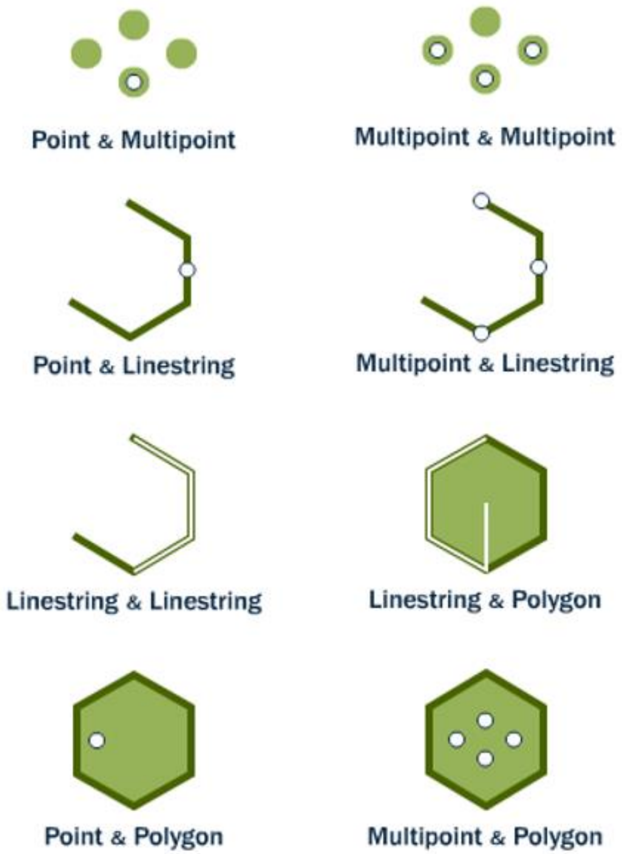
ST_Touches(A, B)

Shapes touch when their boundaries interact but their interiors do not. End points for lines, exterior rings for polygons. Usually used for testing that polygons have ring-touching only.



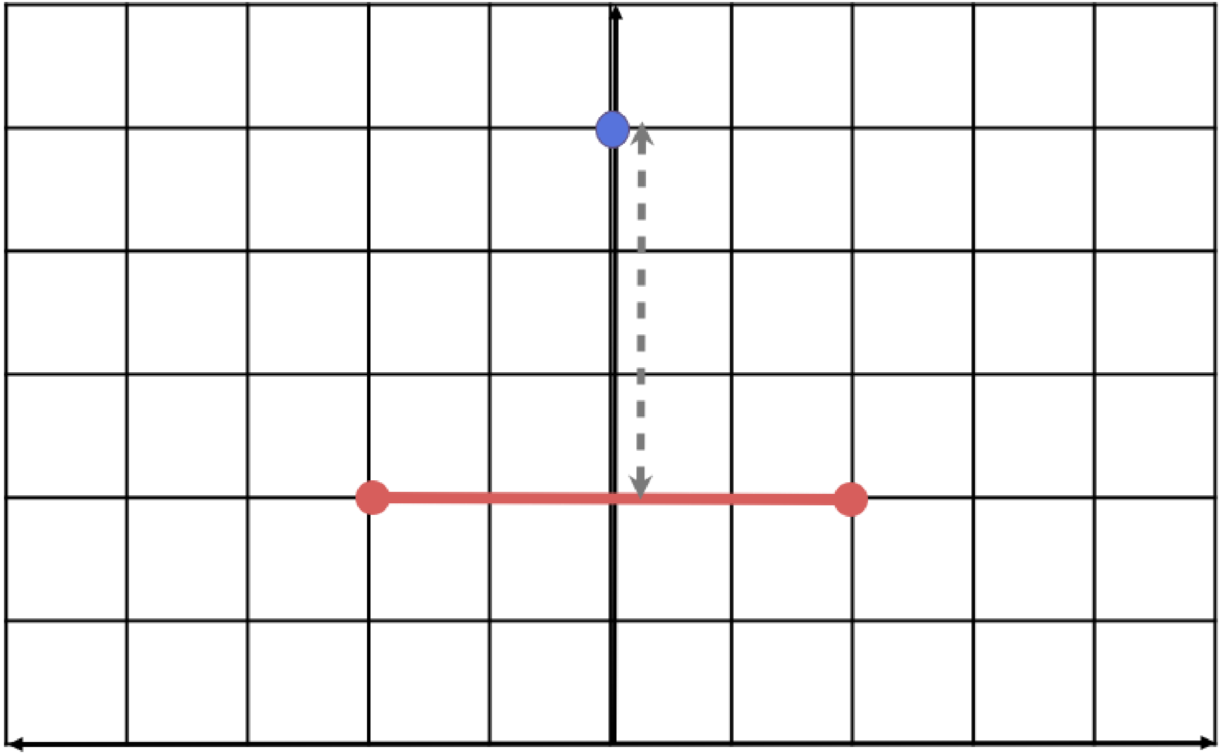
ST_Within(A, B)
ST_Contains(B, A)

Within and contains are about objects being fully inside. One important caveat, for both functions an object on the **boundary** is not considered within. So a point on the outer ring of a polygon is not within the polygon.



ST_Distance(A, B)

Returns the **shortest** distance between the two geometries, in this case the distance from the point to the line mid-point.



ST_Distance(A, B)

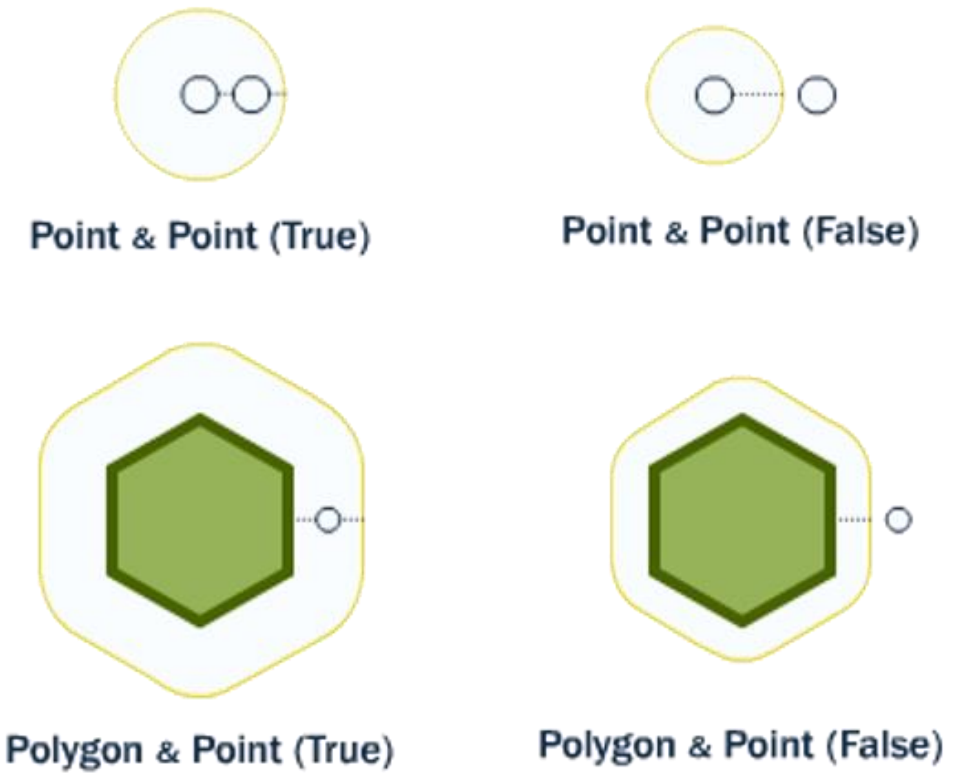
```
SELECT ST_Distance(
   'POINT(0 5)'::geometry,
   'LINESTRING(-2 2, 2 2)'::geometry
   );
```

```
3
```

ST_DWithin(A, B, R)

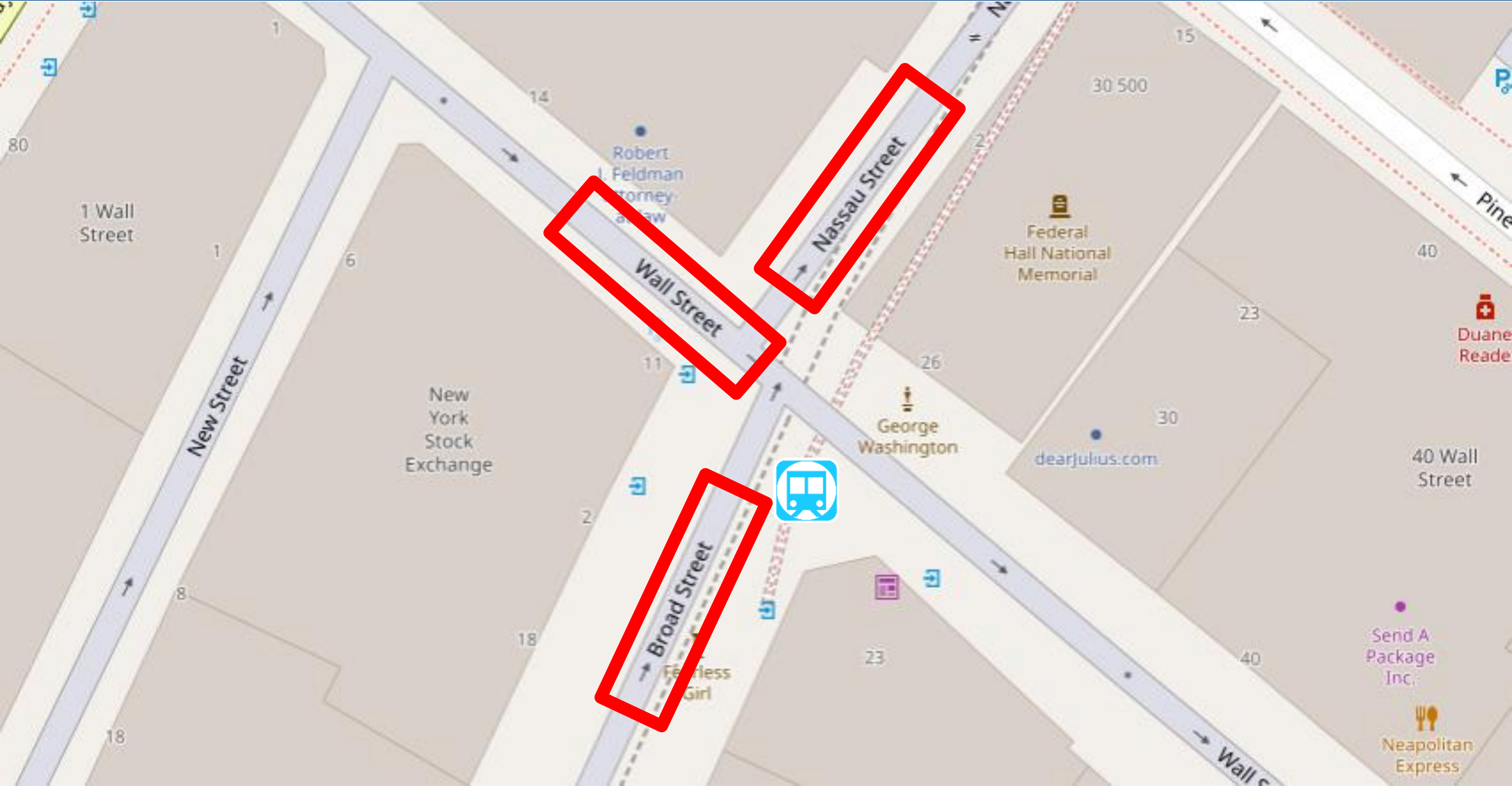
Index-enabled radius search function. True when the distance from geometry A to geometry B is less than radius R. False otherwise.

Use instead of **ST_Distance(A, B) < R**, in order to get benefit of spatial index.



What streets are within 10 meters of Broad Street subway station?

```
SELECT name
FROM nyc_streets
WHERE ST_DWithin(
  geom,
  ST_GeomFromText('POINT(583571 4506714)',26918),
  10
);
```


Wall Street
Nassau Street
Broad Street
New Street
1 Wall Street
New York Stock Exchange
Federal Hall National Memorial
George Washington
dearjulius.com
40 Wall Street
Duane Reade
Send A Package Inc.
Neapolitan Express
Robert J. Feldman
Pine
30 500

Section 13 - Spatial Joins

What neighborhood is the 'Broad St' station in?

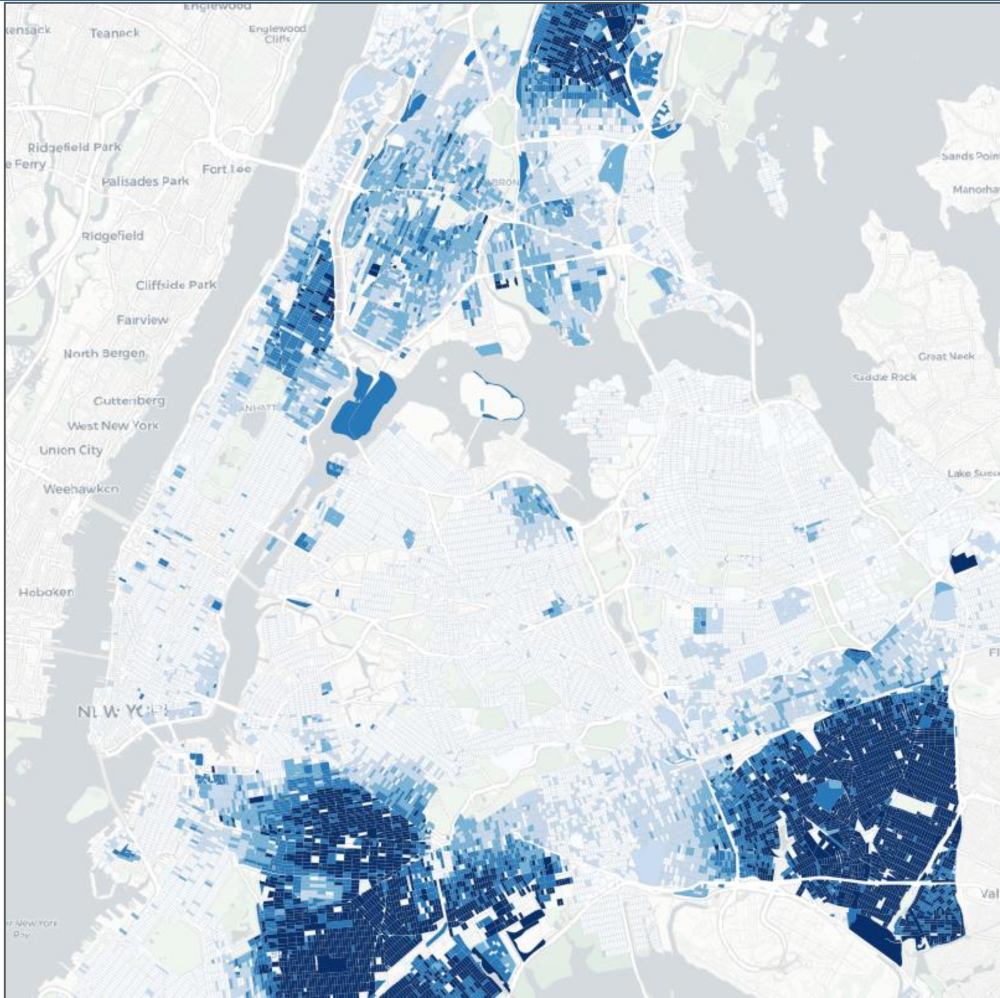
Remember...

```
SELECT name, boroname
FROM nyc_neighborhoods
WHERE
  ST_Intersects(
    geom,
    ST_GeomFromText(
      'POINT(583571 4506714)',
      26918));
```

Do it in one step, with a spatial join!

```sql
SELECT s.name, n.name, n.boroname
FROM nyc_neighborhoods AS n
JOIN nyc_subway_stations AS s
  ON ST_Contains(
    n.geom,
    s.geom
  )
WHERE s.name = 'Broad St';
```

What is the **population** and **racial** make-up of the **neighborhoods** of Manhattan?

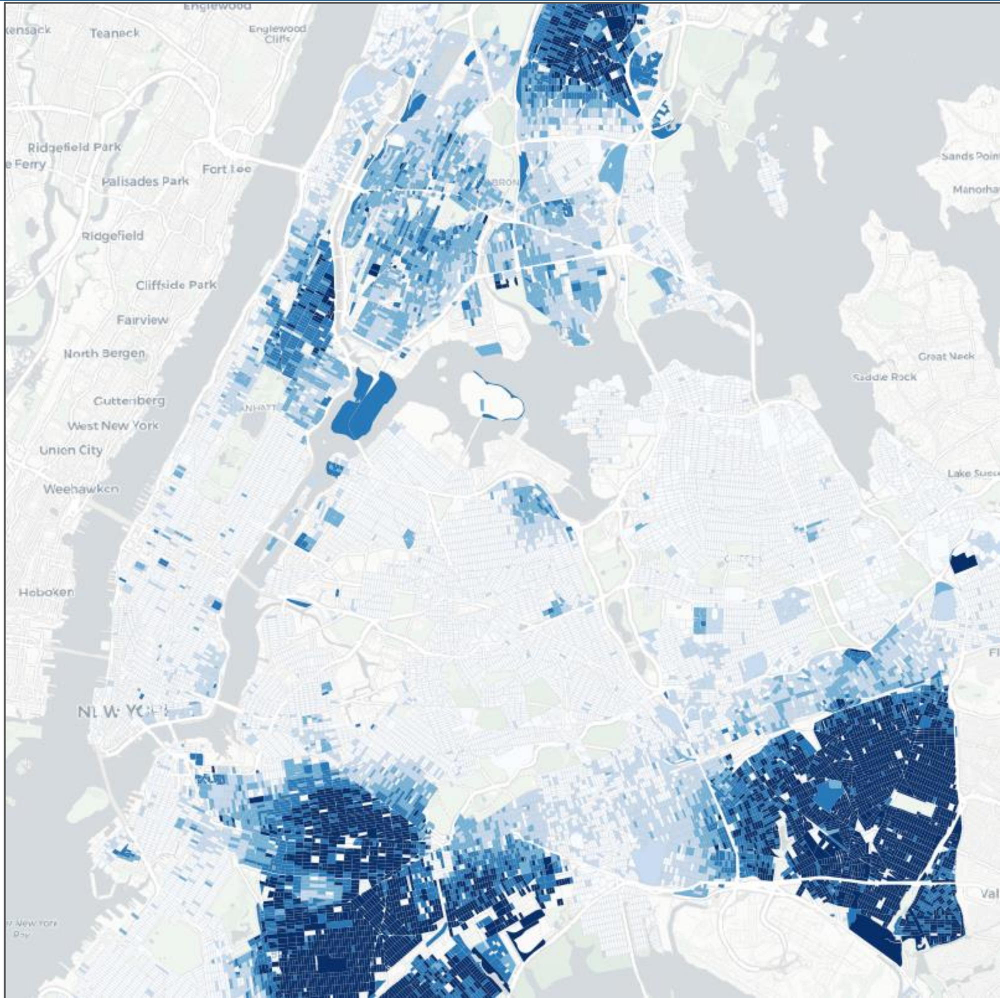


```sql
SELECT
  n.name AS neighborhood_name,
  SUM(c.popn_total) AS population,
  100*SUM(c.popn_white)/SUM(c.popn_total) AS white_pct,
  100*SUM(c.popn_black)/SUM(c.popn_total) AS black_pct
FROM nyc_neighborhoods AS n
JOIN nyc_census_blocks AS c
  ON ST_Intersects(
    n.geom,
    c.geom
  )
WHERE n.boroname = 'Manhattan'
GROUP BY n.name
ORDER BY white_pct DESC;
```

neighborhood_name	popn	white %	black %
Carnegie Hill	18763	90.1	1.4
West Village	26718	87.6	2.2
North Sutton Area	22460	87.6	1.6
Upper East Side	203741	85.0	2.7
Soho	15436	84.6	2.2
Greenwich Village	57224	82.0	2.4
Central Park	46600	79.5	8.0
Tribeca	20908	79.1	3.5
Gramercy	104876	75.5	4.7
Murray Hill	29655	75.0	2.5
Chelsea	61340	74.8	6.4
Upper West Side	214761	74.6	9.2
Midtown	76840	72.6	5.2
Battery Park	17153	71.8	3.4

neighborhood_name	popn	white %	black %
Financial District	34807	69.9	3.8
Clinton	32201	65.3	7.9
East Village	82266	63.3	8.8
Garment District	10539	55.2	7.1
Morningside Heights	42844	52.7	19.4
Little Italy	12568	49.0	1.8
Yorkville	58450	35.6	29.7
Inwood	50047	35.2	16.8
Washington Heights	169013	34.9	16.8
Lower East Side	96156	33.5	9.1
East Harlem	60576	26.4	40.4
Hamilton Heights	67432	23.9	35.8
Chinatown	16209	15.2	3.8
Harlem	134955	15.1	67.1

Let's explore the racial geography of New York City...



Overall Racial Make-up of NYC

```sql
SELECT
  100.0*SUM(popn_white)/SUM(popn_total) AS white_pct,
  100.0*SUM(popn_black)/SUM(popn_total) AS black_pct,
  SUM(popn_total) AS popn_total
FROM nyc_census_blocks;
```

```
     white_pct      |      black_pct      | popn_total
--------------------+---------------------+------------
 44.0039500762811 | 25.5465789002416 |    8175032
```

You, must take the A train. To, go to Sugar Hill way up in Harlem.

What is the racial make-up of the areas served by the A train?

First, we must determine where the A train stops.

Our routes are comma-separated strings!

```
SELECT DISTINCT routes
FROM nyc_subway_stations;
```

```
4,5
N,Q,R,W
J
B,M,Q,R
D,F,N,Q
J,M
E,F
...
```

Postgres string function: strpos()

`strpos(routes,'A')` returns a non-zero number if 'A' is in the `routes` field

Check out https://www.postgresql.org/docs/current/functions-string.html

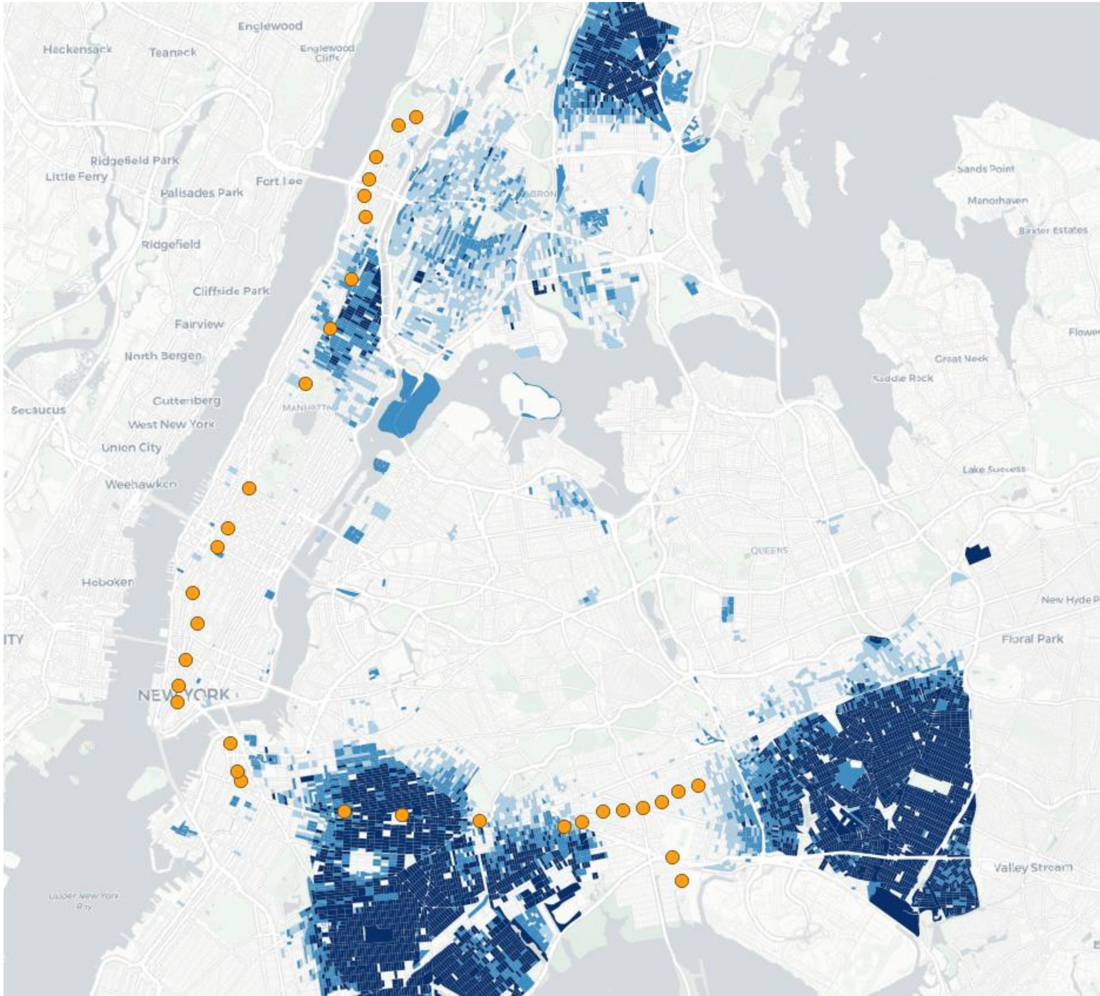
Find all routes with an “A”

```
SELECT DISTINCT routes
FROM nyc_subway_stations AS subways
WHERE strpos(subways.routes,'A') > 0;
```

```
A,C
A,B,C,D
A,C,E,L
A,C,F
A,B,C
A,S
A,C,E
...
```

The route of the A train.

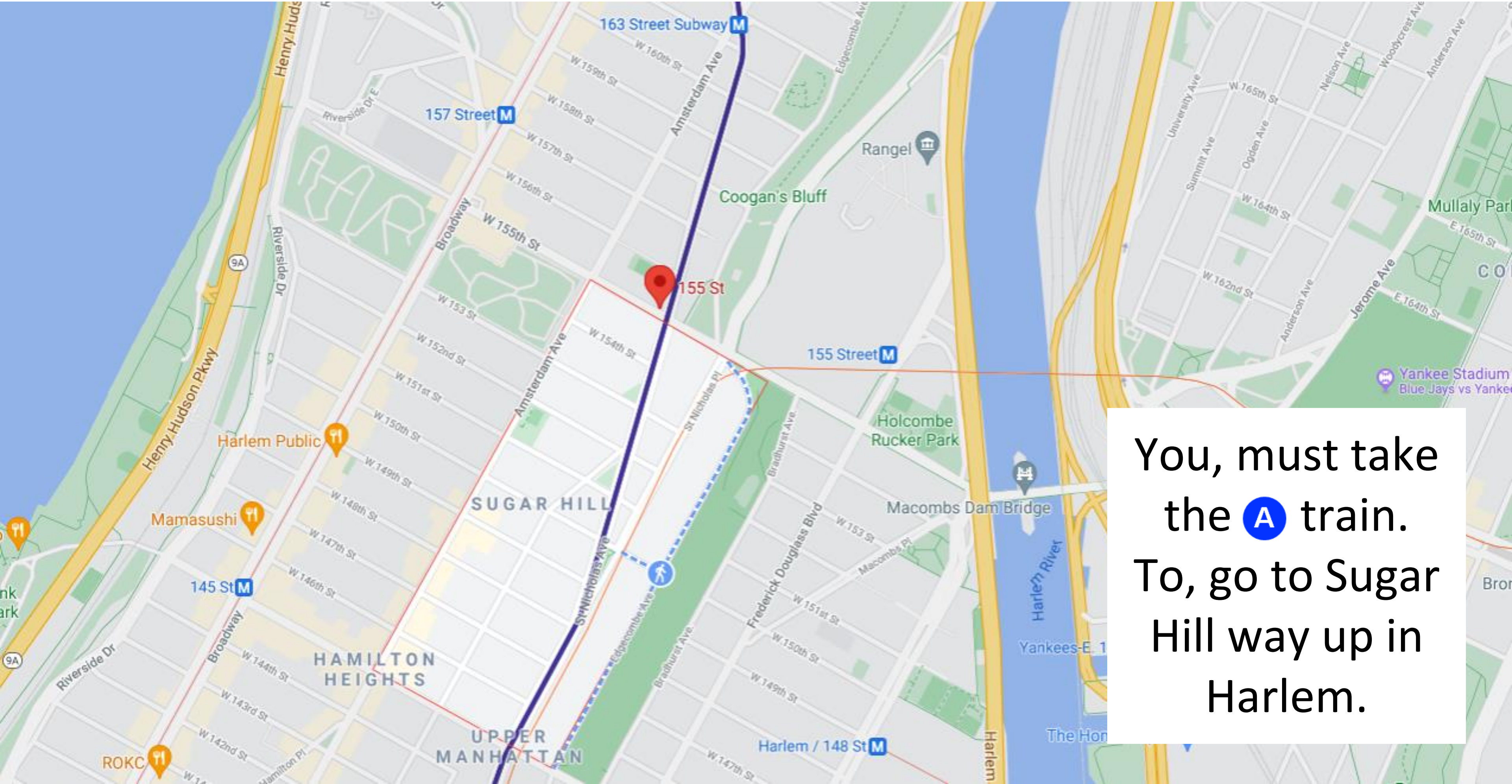
What is the racial makeup within 200 meters of each stop? Who is served by the A train?



Summarize population 200m from A train stops

```sql
SELECT
  100*SUM(c.popn_white)/SUM(c.popn_total) AS white_pct,
  100*SUM(c.popn_black)/SUM(c.popn_total) AS black_pct,
  SUM(popn_total) AS popn_total
FROM nyc_census_blocks AS c
JOIN nyc_subway_stations AS s
  ON ST_DWithin(
    c.geom,
    s.geom,
    200
  )
WHERE strpos(s.routes,'A') > 0;
```

New York	A Train
44.00% white	45.59% white
25.55% black	22.09% black



You, must take the A train. To, go to Sugar Hill way up in Harlem.

Section 15 - Spatial Indexing

A spatial database has...

- Spatial Data Types
  - geometry, geography
- Spatial Indexes
  - r-tree, quad-tree, kd-tree
- Spatial Functions
  - ST_Length(geometry), ST_X(geometry)

A spatial index speeds spatial query ...

- Join two tables of 10,000 records each

Without Index	With Index
10,000 * 10,000 = **100,000,000** comparisons	10,000 + 10,000 = **20,000** comparisons

To prove it... remove the index.

```
DROP INDEX nyc_census_blocks_geom_idx;
```

Run a spatial join.

```
SELECT b.blkid
FROM nyc_census_blocks b
JOIN nyc_subway_stations s
  ON ST_Contains(b.geom, s.geom)
WHERE s.name LIKE 'B%';
```

~300 ms

Create the index again.

```
CREATE INDEX nyc_census_blocks_geom_idx
ON nyc_census_blocks USING GIST (geom);
```

Run the join again.

```
SELECT blocks.blkid
FROM nyc_census_blocks b
JOIN nyc_subway_stations s
  ON ST_Contains(b.geom, s.geom)
WHERE s.name LIKE 'B%';
```

~90 ms

Spatial Index Cliff Notes

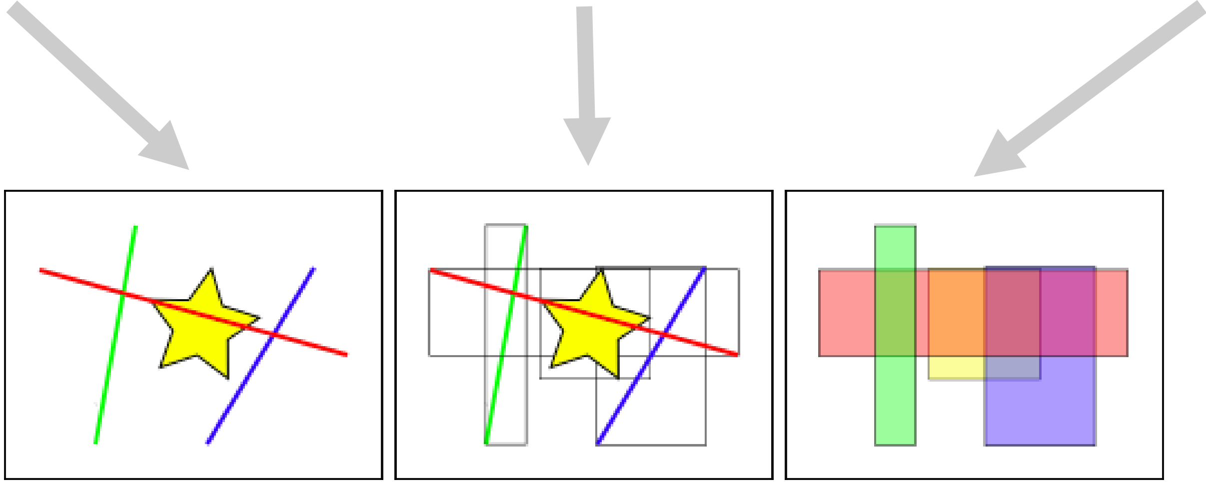
- **CREATE INDEX** index_name **ON** table_name **USING GIST** (geom)
- Use a “spatially indexed function” in **JOIN** or **WHERE** clause
  - ST_Intersects(A, B), ST_Contains(A, B), ST_Within(A, B)
  - ST_DWithin(A, B, R)

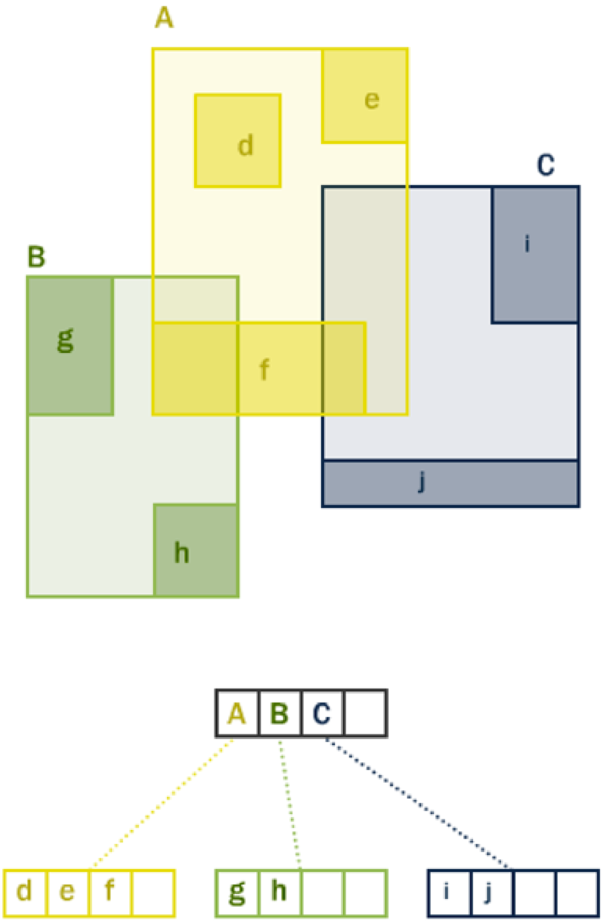
Spatial Index Internals

Some spatial objects (like the star) are quite large and complex. Comparing complex objects is expensive!

Instead of indexing objects directly, spatial indexes work on the bounding boxes of the objects.

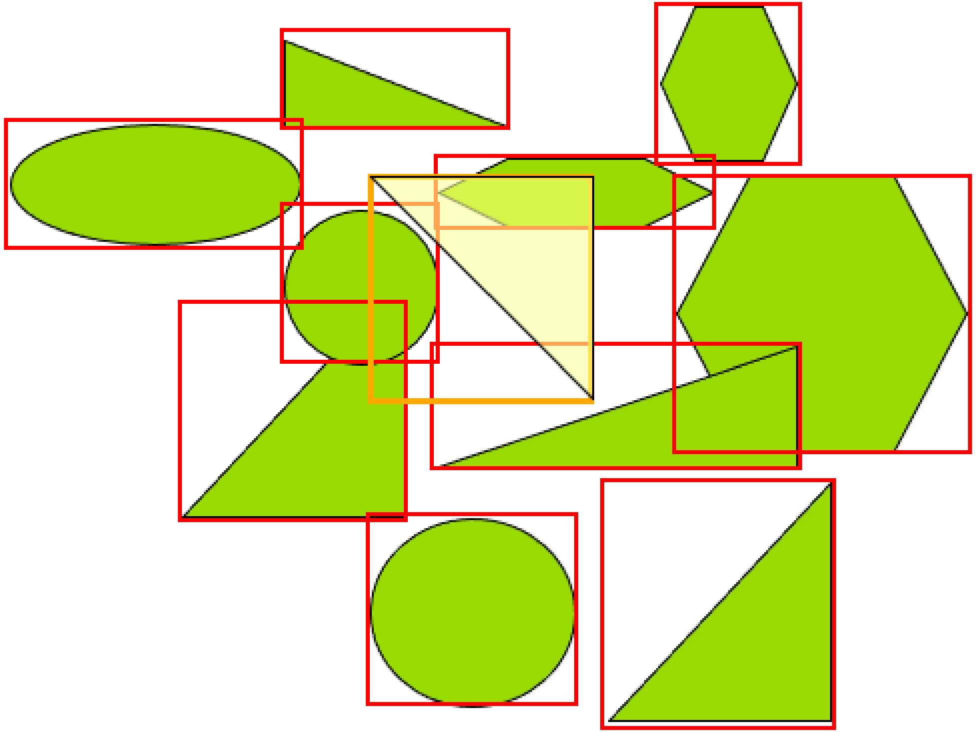
The boxes are of uniform size, and can be compared to determine spatial relationships very quickly.



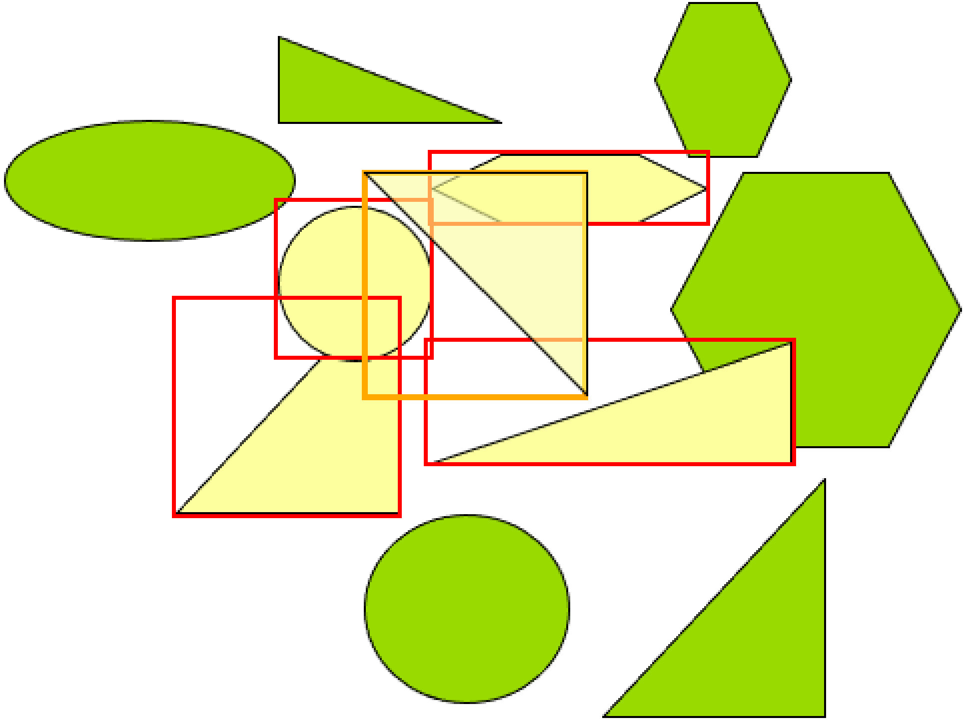


The boxes can be arranged in a hierarchy, so that a query can quickly discard portions of the search space that will not interact with a query box. Depending on the algorithm, different hierarchies can be build. PostGIS uses an “R*tree” algorithm.

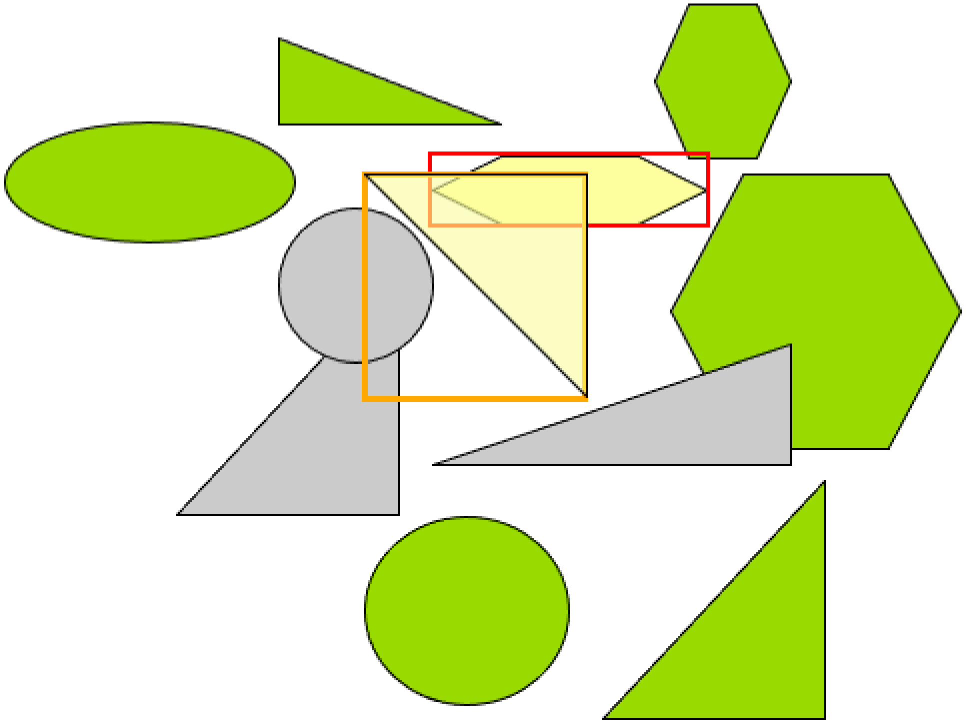
What green objects intersect the yellow query shape?



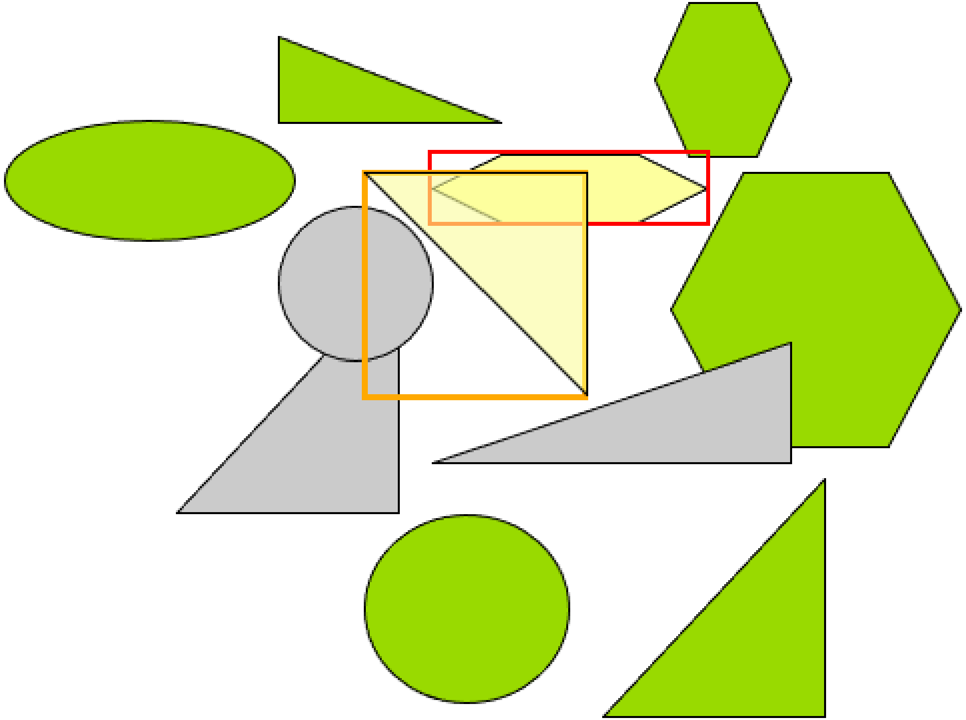
Use **index** to quickly finds the objects with **bounding box intersection**.



Exactly compute relationships in index result to find true intersection.



Index-only queries



Index-enabled Spatial Functions

- **ST_Intersects()**
- **ST_Contains()**
- **ST_Within()**
- **ST_DWithin()**
- ST_ContainsProperly()
- ST_CoveredBy()
- ST_Covers()
- ST_Overlaps()
- ST_Crosses()
- ST_DFullyWithin()
- ST_3DIntersects()
- ST_3DDWithin()
- ST_3DDFullyWithin()
- ST_LineCrossingDirection()
- ST_OrderingEquals()
- ST_Equals()

Index-only queries

geom_a && geom_b

The “&&” operator is the “bounding boxes overlap” operator.

It returns “true” when the bounds of the left and right arguments overlap.

Operators like “=” or “>” are symbols that express relationships between the left- and right-hand side arguments. “&&” is just another operator like any other.

What is the population of the West Village?

```
SELECT Sum(blk.opn_total)
FROM nyc_neighborhoods nh
  JOIN nyc_census_blocks blk
  ON nh.geom && blk.geom
WHERE nh.name = 'West Village';
```

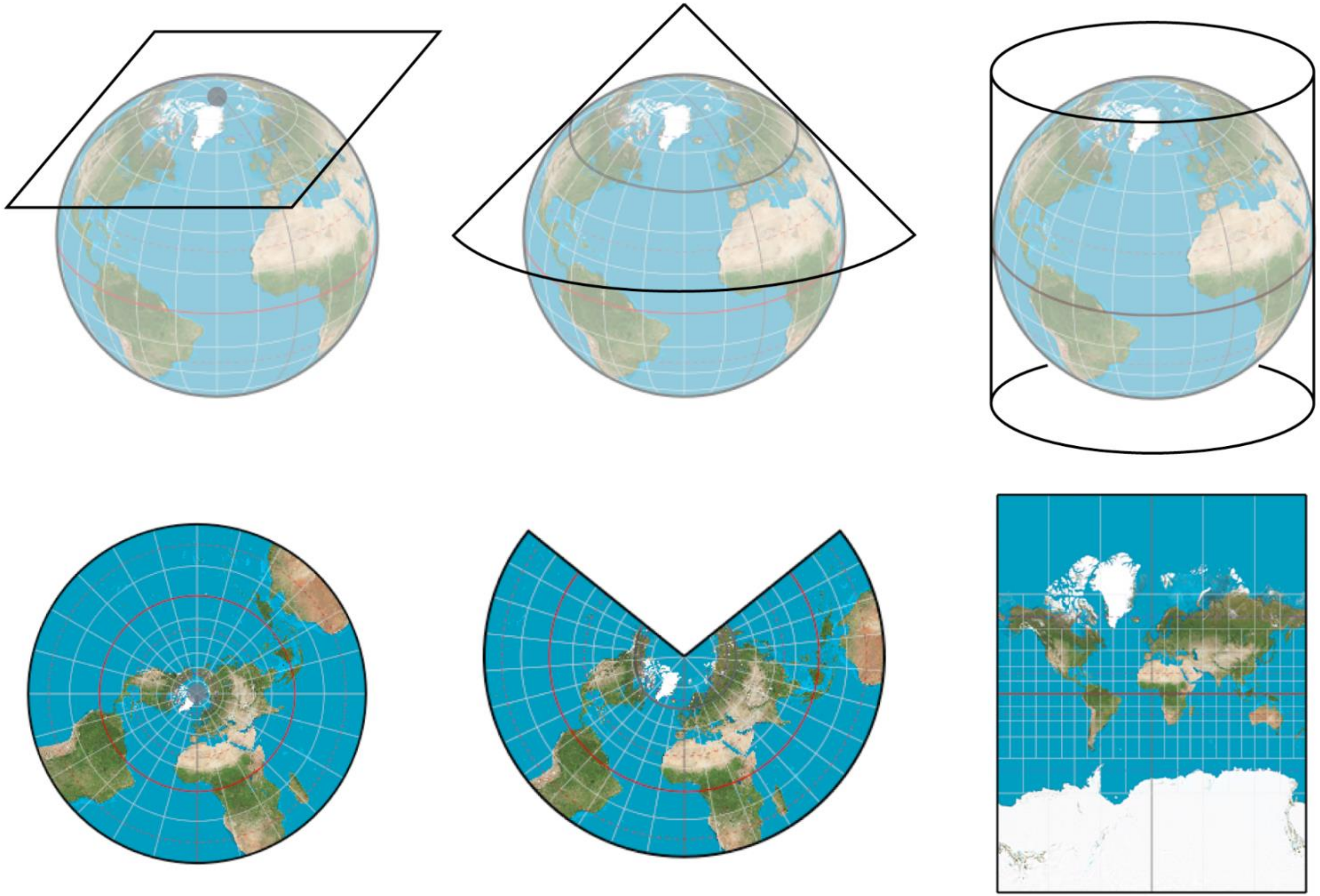
49821

What is the population of the West Village?

```
SELECT Sum(blk.opn_total)
FROM nyc_neighborhoods nh
  JOIN nyc_census_blocks blk
  ON ST_Intersects(nh.geom, blk.geom)
WHERE nh.name = 'West Village';
```

26718

Section 16 - Projecting Data



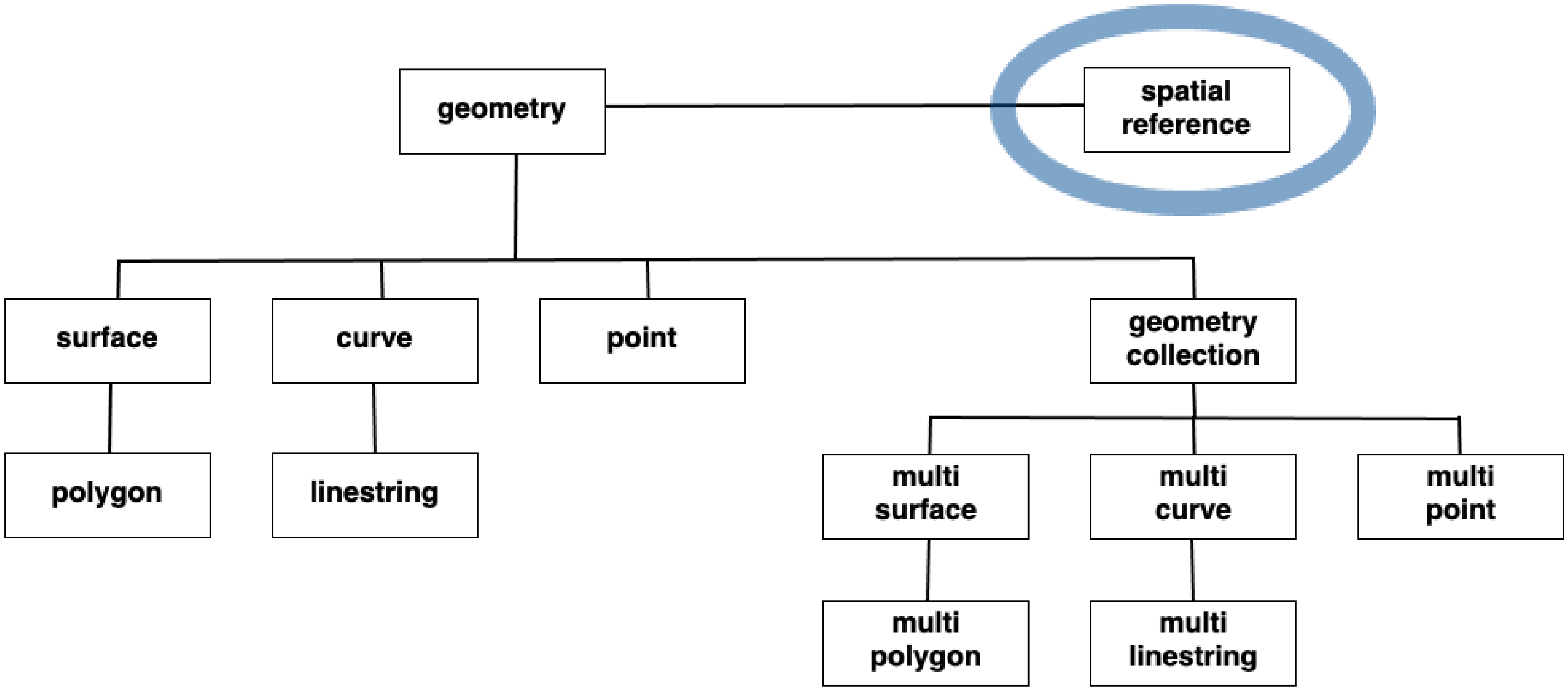
The earth is not flat, and there is no simple way of putting it down on a flat paper map (or computer screen), so people have come up with all sorts of ingenious solutions, each with pros and cons.

$$f(\theta,\Phi) \rightarrow (x,y)$$

Forward projection converts spherical coordinates (longitude, latitude) to cartesian coordinates (x and y)

$$f^{-1}(x,y) \rightarrow (\theta,\Phi)$$

Inverse projection converts cartesian coordinates (x, y) to spherical coordinates (longitude, latitude)

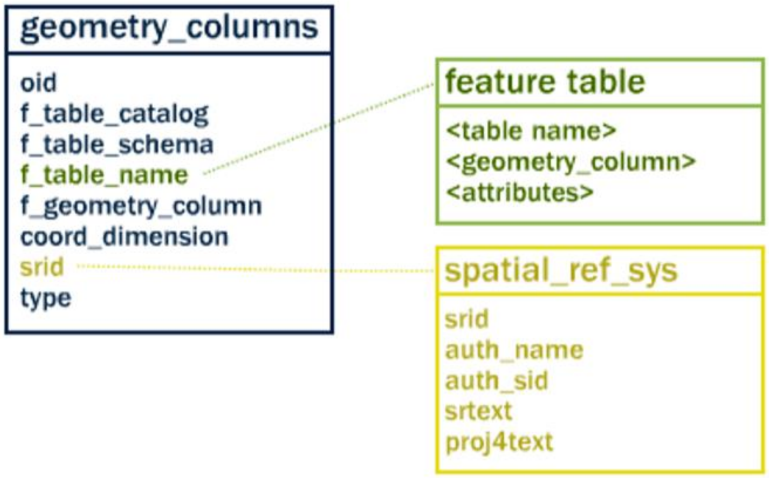

geometry
spatial reference
surface
curve
point
geometry collection
polygon
linestring
multi surface
multi curve
multi point
multi polygon
multi linestring

What is the SRID of our subways?

```
SELECT ST_SRID(geom)
FROM nyc_subway_stations
LIMIT 1;
```

```
26918
```

What does SRID 26918 mean though?



SRID is a foreign key relating to spatial_ref_sys

What does SRID 26918 mean though?

```sql
SELECT srtext
FROM spatial_ref_sys
WHERE srid = 26918;
```

Also, see: https://epsg.io/26918

What does SRID 26918 mean though?

```
PROJCS["NAD83 / UTM zone 18N",
  GEOGCS["NAD83",
    DATUM["North_American_Datum_1983",
      SPHEROID["GRS 1980",6378137,298.257222101],
      TOWGS84[0,0,0,0,0,0,0],
      AUTHORITY["EPSG","6269"]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433],
    AUTHORITY["EPSG","4269"]],
  PROJECTION["Transverse_Mercator"],
  PARAMETER["latitude_of_origin",0],
  PARAMETER["central_meridian",-75],
  PARAMETER["scale_factor",0.9996],
  PARAMETER["false_easting",500000],
  PARAMETER["false_northing",0],
  UNIT["metre",1],
  AXIS["Easting",EAST],
  AXIS["Northing",NORTH]]
```

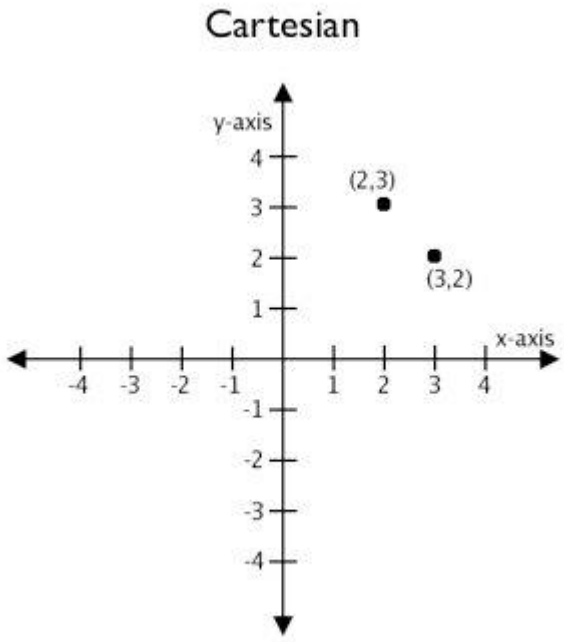
What are coordinates of the "Broad St" subway station in geographic?

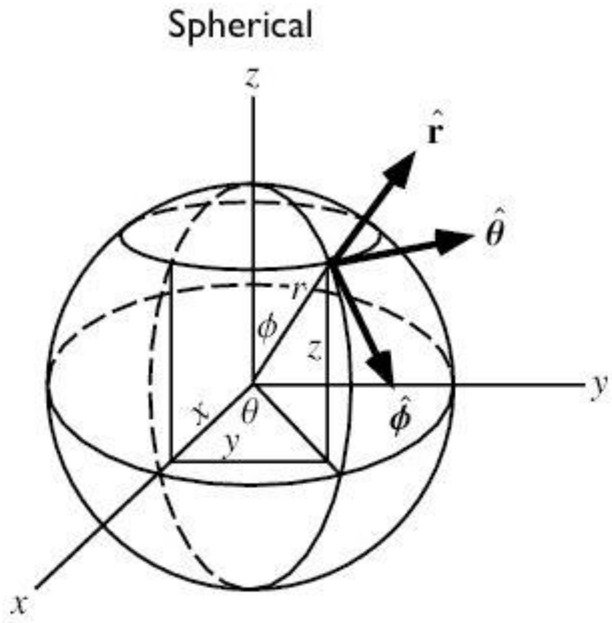
```
SELECT
  ST_AsText(ST_Transform(geom,4326))
FROM nyc_subway_stations
WHERE name = 'Broad St';
```

```
POINT(-74.0106714 40.7071048)
```

Section 18 - Geography

Geographic Coordinate Systems





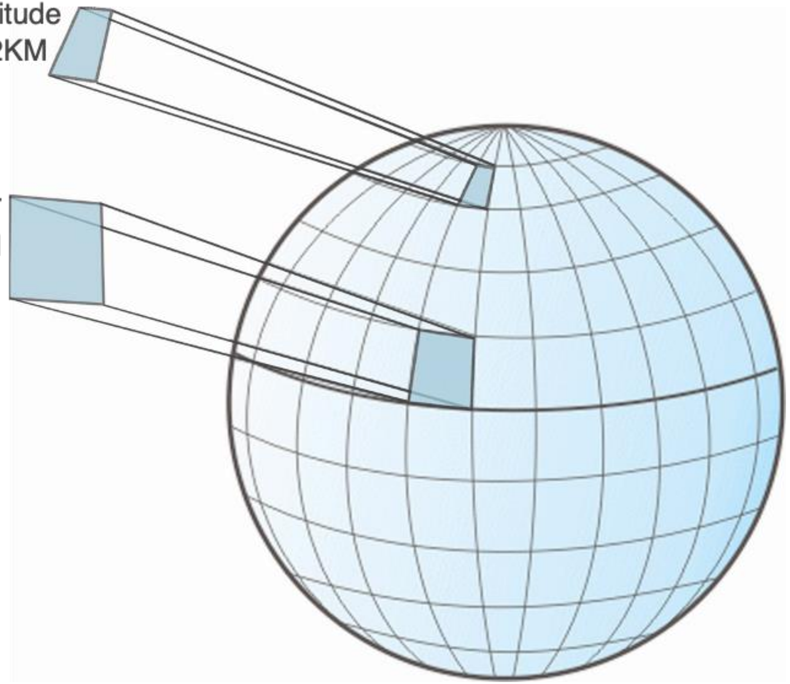
What is the distance between Los Angeles and Paris using ST_Distance(geometry, geometry)?

```
SELECT ST_Distance(
  -- Los Angeles (LAX)
  'SRID=4326;POINT(-118.4079 33.9434)'::geometry,
  -- Paris (CDG)
  'SRID=4326;POINT(2.5559 49.0083)'::geometry
  );
```

```
121.898285970107
```


Arctic Circle (66° 33
121°
PARIS
LOS ANGELES
Tropic of Cancer (23° 27
Equator
Tropic of Capricorn (23° 27

Degrees are not units of distance
Degrees are not units of area



What is the distance between Los Angeles and Paris using ST_Distance(geography, geography)?

```
SELECT ST_Distance(
  -- Los Angeles (LAX)
  'SRID=4326;POINT(-118.4079 33.9434)'::geography,
  -- Paris (CDG)
  'SRID=4326;POINT(2.5559 49.0083)'::geography
  );
```

```
9124665.27317673
```


Arctic Cirlce (66° 33
9124km
PARIS
LOS ANGELES
Tropic of Cancer (23° 27
Equator
Tropic of Capricorn (23° 27

How close will a flight from Los Angeles to Paris come to Iceland?

```
SELECT ST_Distance(
  -- LAX-CDG
  'SRID=4326;LINESTRING(
    -118.4079 33.9434,
       2.5559 49.0083)'::geography,
  -- Iceland
  'SRID=4326;POINT(-21.8628 64.1286)'::geography
);
```

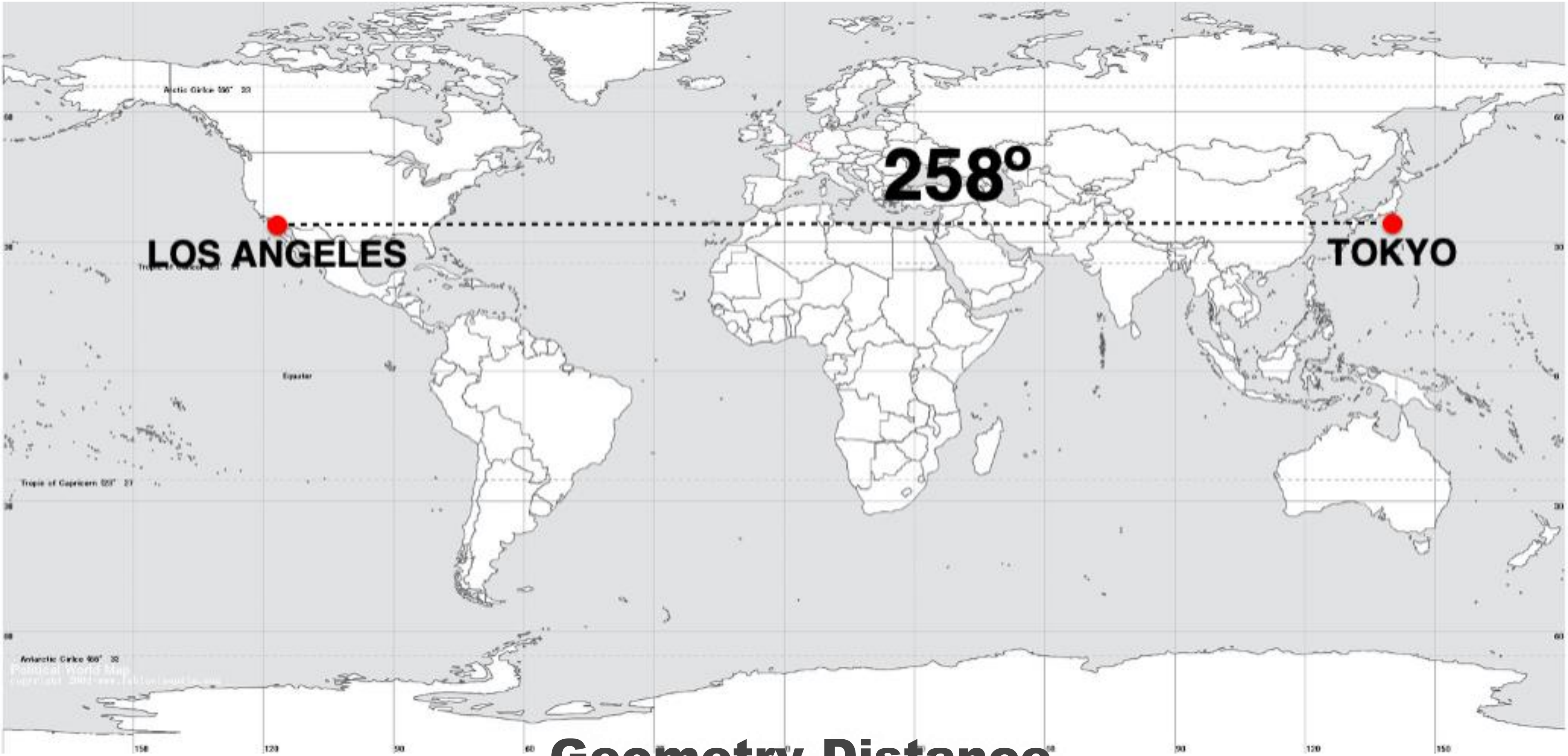
```
531773.75711106
```

ICELAND
PARIS
LOS ANGELES
Arctic Cirlce (66° 33
Tropic of Cancer (23° 27
Equator
Tropic of Capricorn (23° 27

What is the shortest great-circle route from Los Angeles to Tokyo?

```
SELECT ST_Distance(
 'SRID=4326;POINT(-118.408 33.943)'::geometry, -- LAX
 'SRID=4326;POINT( 139.733 35.567)'::geometry) -- NRT
    AS geometry_distance,
ST_Distance(
 'POINT(-118.408 33.943)'::geography,  -- LAX
 'POINT( 139.733 35.567)'::geography)  -- NRT
    AS geography_distance;
```

```
 geometry_distance:      258.14610835
geography_distance: 8833973.30246194
```



Geometry Distance


Arctic Cirlce (66° 33
60
8834km
TOKYO
LOS ANGELES
30
Tropic of Cancer (23° 27
0
Equator
Tropic of Capricorn (23° 27
30
Geographic Distance
60

Using Geography - Casting

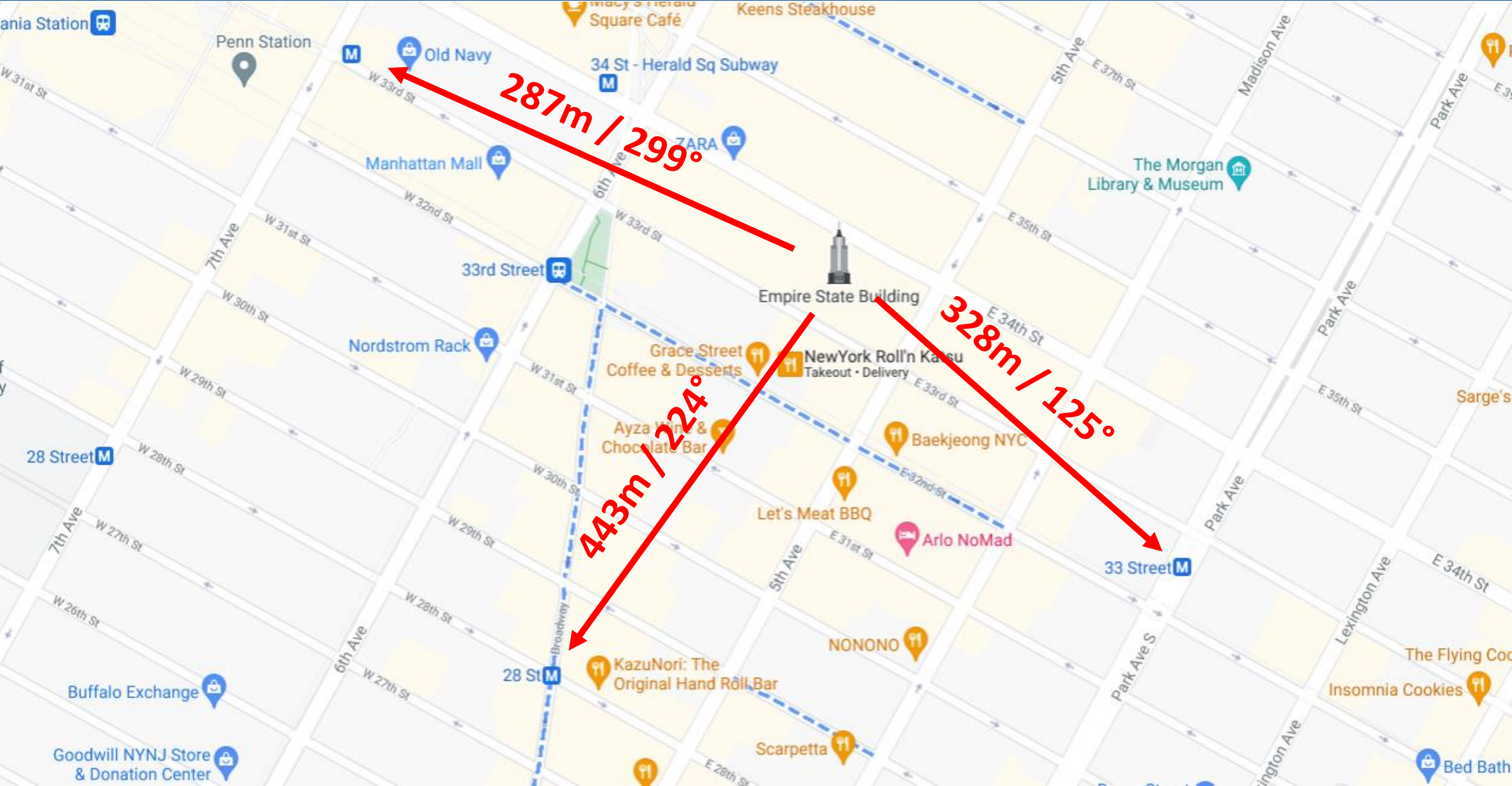
```
CREATE TABLE nyc_subway_stations_geog AS
SELECT
   ST_Transform(geom, 4326)::geography AS geog,
   name,
   routes
FROM nyc_subway_stations;
```

Using Geography - Indexing

```
CREATE INDEX nyc_subway_stations_geog_gix
  ON nyc_subway_stations_geog
  USING GIST (geog);
```

Using Geography - Querying

```
WITH empire_state_building AS (
  SELECT 'POINT(-73.98501 40.74812)'::geography AS geog
)
SELECT name,
  ST_Distance(esb.geog, ss.geog) AS distance,
  degrees(ST_Azimuth(esb.geog, ss.geog)) AS direction
FROM nyc_subway_stations_geog ss,
     empire_state_building esb
WHERE ST_DWithin(ss.geog, esb.geog, 500);
```


287m / 299°
328m / 125°
443m / 224°
Empire State Building
Penn Station
Old Navy
34 St - Herald Sq Subway
Keens Steakhouse
Manhattan Mall
ZARA
33rd Street
Nordstrom Rack
Grace Street Coffee & Desserts
NewYork Roll'n Katsu
Takeout · Delivery
Baekjeong NYC
Let's Meat BBQ
Arlo NoMad
33 Street
The Morgan Library & Museum
28 Street
28 St
KazuNori: The Original Hand Roll Bar
NONONO
Scarpetta
Buffalo Exchange
Goodwill NYNJ Store & Donation Center
Insomnia Cookies
5th Ave
6th Ave
7th Ave
Madison Ave
Park Ave
Park Ave S
Lexington Ave
Broadway
W 33rd St
W 32nd St
W 31st St
W 30th St
W 29th St
W 28th St
W 27th St
W 26th St
E 37th St
E 35th St
E 34th St
E 33rd St
E 32nd St
E 31st St
E 28th St

Using Geography - From Scratch

```
CREATE TABLE airports (
    code VARCHAR(3),
    geog GEOGRAPHY(Point)
  );

INSERT INTO airports
  VALUES ('LAX', 'POINT(-118.4079 33.9434)');
INSERT INTO airports
  VALUES ('CDG', 'POINT(2.5559 49.0083)');
INSERT INTO airports
  VALUES ('KEF', 'POINT(-22.6056 63.9850)');
```

Using Geography - From Scratch

```
SELECT * FROM geography_columns;
```

f_table_name	f_geography_column	srid	type
nyc_subway_stations_geog	geog	0	Geometry
airports	geog	4326	Point

Casting to Geometry

```
SELECT
   code,
   ST_X(geog::geometry) AS longitude
FROM airports;
```

The "::" syntax tells PostgreSQL to attempt to coerce the data into the new data type, if there is an available path.

Geography Native Functions

- ST_Distance(G1, G2)
- ST_DWithin(G1, G2, R)
- ST_Area(geog)
- ST_Length(geography)
- ST_Covers(G1, G2)
- ST_CoveredBy(G1, G2)
- ST_Intersects(G1, G2)
- ST_AsText(G1)
- ST_AsBinary(G1)
- ST_AsSVG(G1)
- ST_AsGML(G1)
- ST_AsKML(G1)
- ST_AsGeoJson(G1)
- ST_Buffer(G1, R)
- ST_Intersection(G1, G2)

Geography is the Magic Solution?

The complexity of dealing with planar projections (choosing one, getting used to it) drives some users to fixate on the **geography** type as a simple cure-all.

However:

- Not all functions in geography have native on-the-sphere implementations yet.
- The computational cost of geography compared to geometry is quite high.

geography distance

```
double R = 6371000; /* meters */
double d_lat = lat2-lat1; /* radians */
double d_lon = lon2-lon1; /* radians */
double sin_lat = sin(d_lat/2);
double sin_lon = sin(d_lon/2);
double a = sin_lat * sin_lat +
           cos(lat1) * cos(lat2) *
           sin_lon * sin_lon;
double c = 2 * atan2(sqrt(a),
                     sqrt(1-a));
double d = R * c;
```

geometry distance

```
double dx = x2 - x1;
double dy = y2 - y1;
double d2 = dx * dx +
            dy * dy;
double d = sqrt(d2);
```

Section 20 - Geometry Constructing Functions

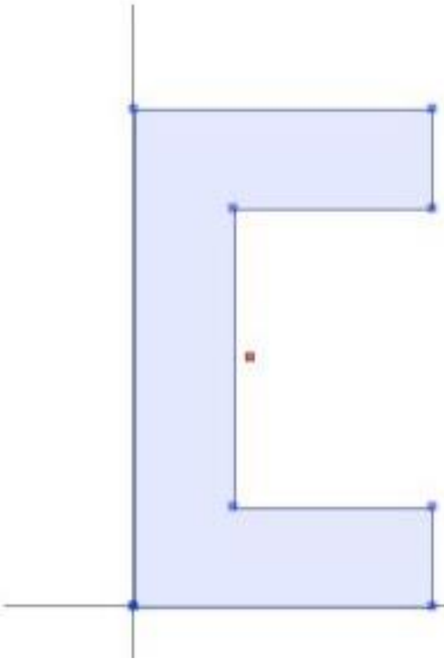
Functions so far...

- Analysis
  - ST_Length(geometry) → float
  - ST_Area(geometry) → float
- Conversion
  - ST_AsText(geometry) → text
  - ST_AsGML(geometry) → text
- Retrieval
  - ST_RingN(geometry,n) → geometry
- Comparison
  - ST_Contains(geometry,geometry) → boolean

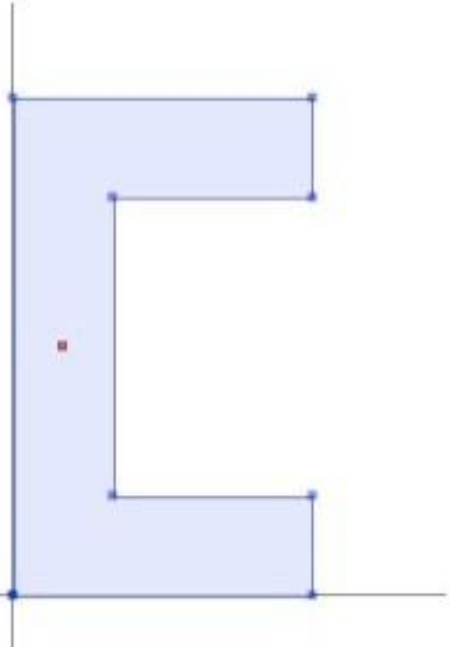
Geometry constructing functions!

- **ST_Buffer**(geometry) → geometry
- **ST_Centroid**(geometry) → geometry
- **ST_Intersection**(geometry, geometry) → geometry
- **ST_Union**(geometry[]) → geometry
- **ST_Collect**(geometry[]) → geometry

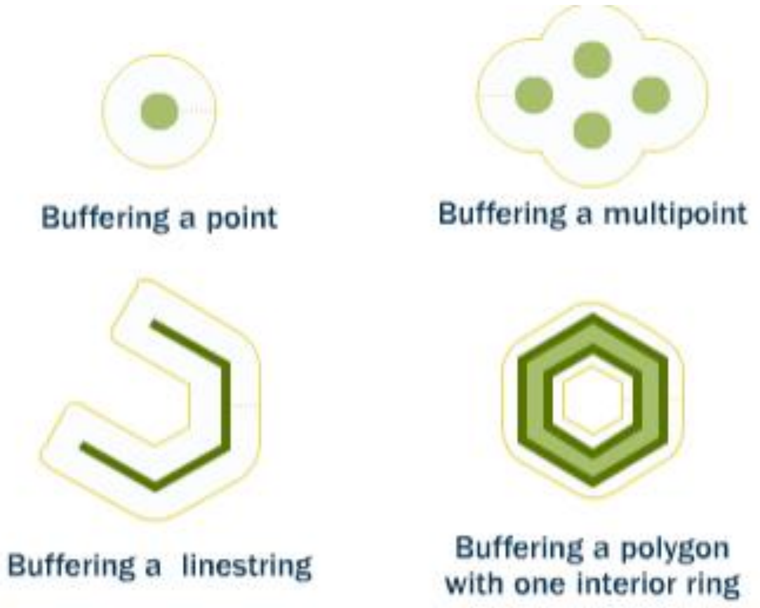
ST_Centroid



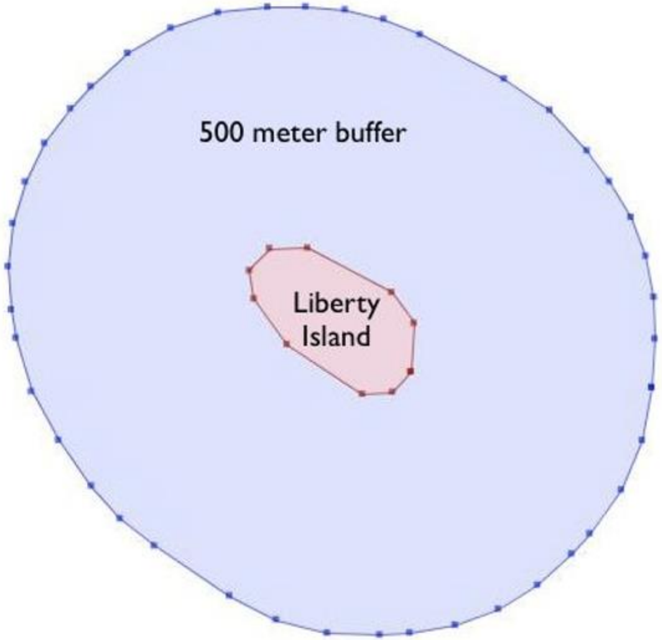
ST_PointOnSurface



ST_Buffer


Buffering a point
Buffering a multipoint
Buffering a linestring
Buffering a polygon with one interior ring

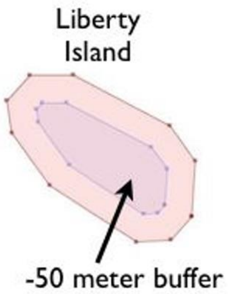
“What would a 500 meter marine traffic zone around Liberty Island look like?”



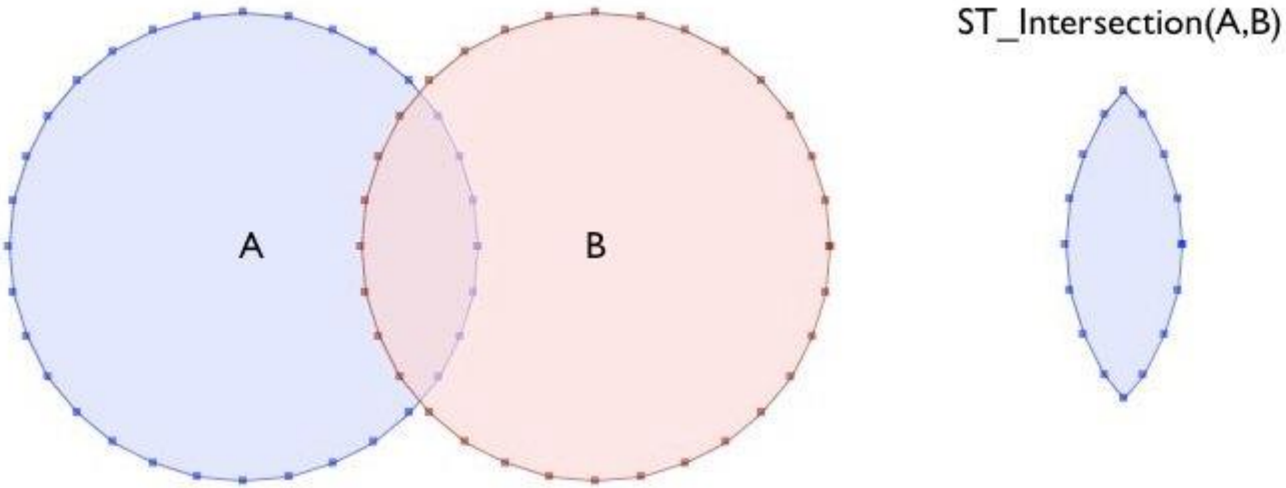
“What would a 500 meter marine traffic zone around Liberty Island look like?”

```
-- New table with a Liberty Island
-- 500m buffer zone
CREATE TABLE liberty_island_zone AS
SELECT
  ST_Buffer(geom, 500)::Geometry(Polygon,26918)
AS geom
FROM nyc_census_blocks
WHERE blkid = '360610001001001';
```

"What would a negative 50 meter marine traffic zone around Liberty Island look like?"

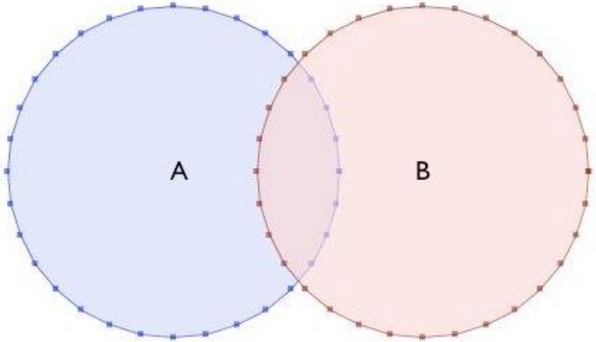


ST_Intersection(A, B)



“What is the area these two circles have in common?”

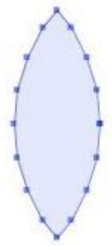
```
SELECT ST_AsText(ST_Intersection(
  ST_Buffer('POINT(0 0)', 2),
  ST_Buffer('POINT(3 0)', 2)
));
```



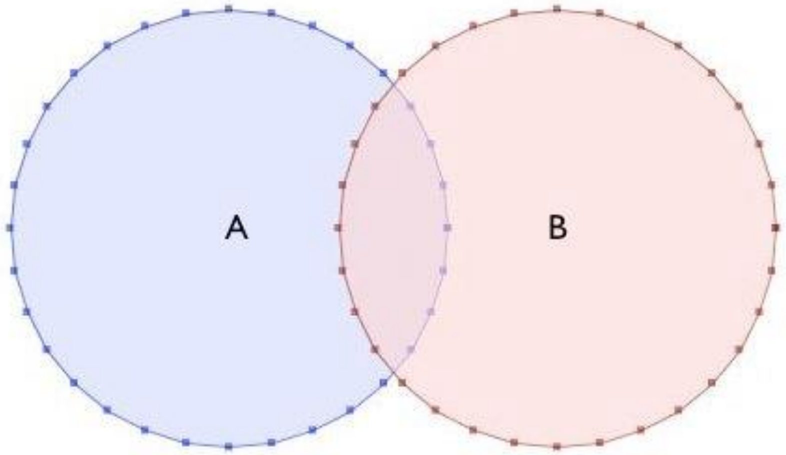
“What is the area these two circles have in common?”

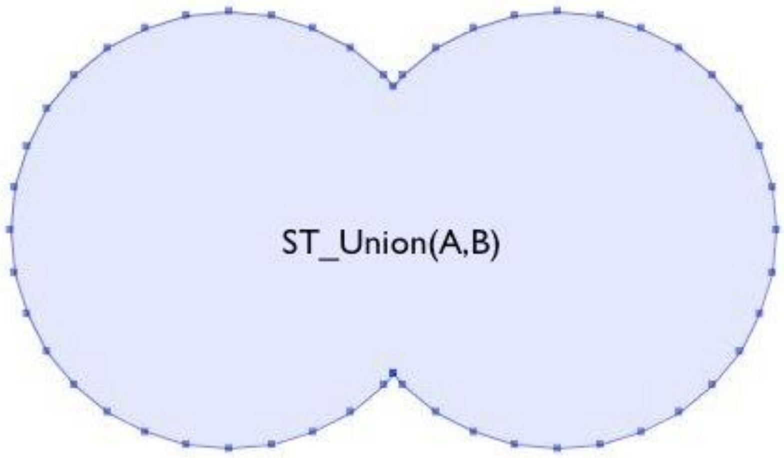
```
POLYGON((
 2 0,
 1.96157056080646 -0.390180644032256,
 1.84775906502257 -0.765366864730179,
 1.66293922460509 -1.1111404660392,
 1.5 -1.30968248567708,
 1.33706077539491 -1.11114046603921,
 1.15224093497743 -0.765366864730185,
 1.03842943919354 -0.390180644032262,
 1 -6.46217829773035e-15,
 1.03842943919354 0.39018064403225,
 1.15224093497742 0.765366864730173,
 1.33706077539491 1.1111404660392,
 1.5 1.30968248567708,
 1.66293922460509 1.11114046603921,
 1.84775906502257 0.765366864730184,
 1.96157056080646 0.390180644032261,
 2 0))
```

ST_Intersection(A,B)



ST_Union(A, B)





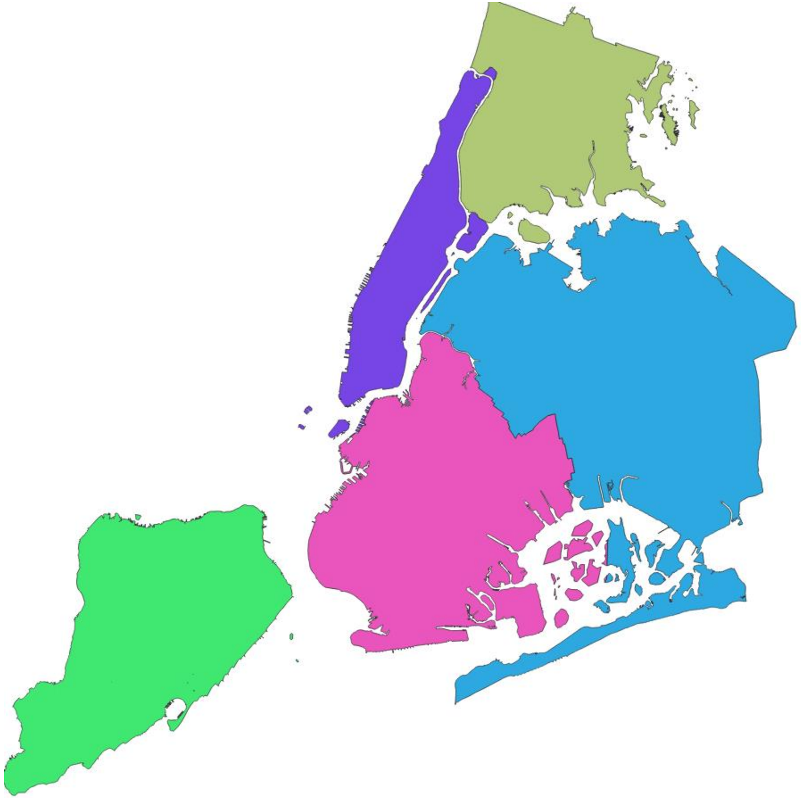
Terminology

- Esri "dissolve" == PostGIS "union"
  - Melt together small things into larger things.
- Esri "union" == PostGIS "overlay"
  - Cookie cut larger things into smaller things.

Forms

- ST_Union(geom1, geom2)
  - Melt together two geometries.
- ST_Union(geometry[])
  - Melt together a set of geometries. "Aggregate" function like Sum() or Average(). Use with GROUP BY.

"How would you make a county map from census blocks?"



Census Block ID

US Census Block IDs encode the ***geographic hierarchy*** used by the census.

```
360610001001001 = 36 061 000100 1 001

36     = State of New York
061    = New York County (Manhattan)
000100 = Census Tract
1      = Census Block Group
001    = Census Block
```

"How would you make a county map from census blocks?"

```
-- An nyc_census_counties table
-- by merging census blocks
CREATE TABLE nyc_census_counties AS
SELECT
  ST_Union(geom) AS geom,
  SubStr(blkid,1,5) AS countyid
FROM nyc_census_blocks
GROUP BY countyid;
```

Section 22 - More Spatial Joins!

Load the nyc_census_sociodata.sql table

1. Open the **Query Tool** in pgAdmin
2. Select **Open File**
3. Browse to the nyc_census_sociodata.sql file
4. Run query

```
-- 2167
SELECT
  Count(*)
FROM nyc_census_sociodata;
```

How would you make a census tract map from census blocks?

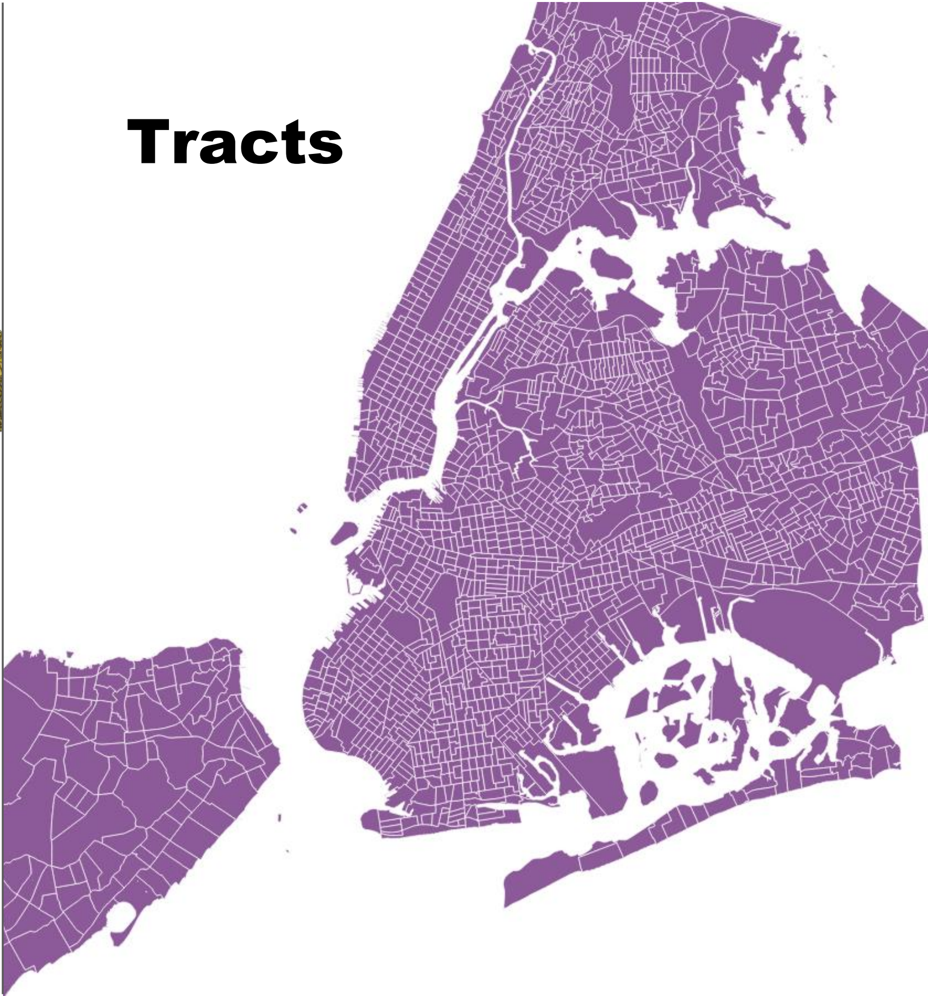
Recall...

Liberty Island blkid

```
360610001001001 = 36 061 000100 1 001

36     = State of New York
061    = New York County (Manhattan)
000100 = Census Tract
1      = Census Block Group
001    = Census Block
```


Blocks


Tracts

ST_Union() Blocks into Tracts

```
-- Make the tracts table
CREATE TABLE nyc_census_tract_geoms AS
SELECT
  ST_Union(geom) AS geom,
  substr(blkid,1,11) AS tractid
FROM nyc_census_blocks
GROUP BY tractid;

-- Index the tractid
CREATE INDEX nyc_census_tract_geoms_tractid_idx
ON nyc_census_tract_geoms (tractid);
```

How can you associate census data with your census tract map?

ST_Union() Blocks into Tracts

```
-- Make the tracts table
CREATE TABLE nyc_census_tracts AS
SELECT g.geom, a.*
FROM nyc_census_tract_geoms g
JOIN nyc_census_sociodata a
ON g.tractid = a.tractid;

-- Index the geometries
CREATE INDEX nyc_census_tract_gidx
ON nyc_census_tracts USING GIST (geom);
```

“List top 10 New York neighborhoods ordered by the proportion of people who have graduate degrees?”

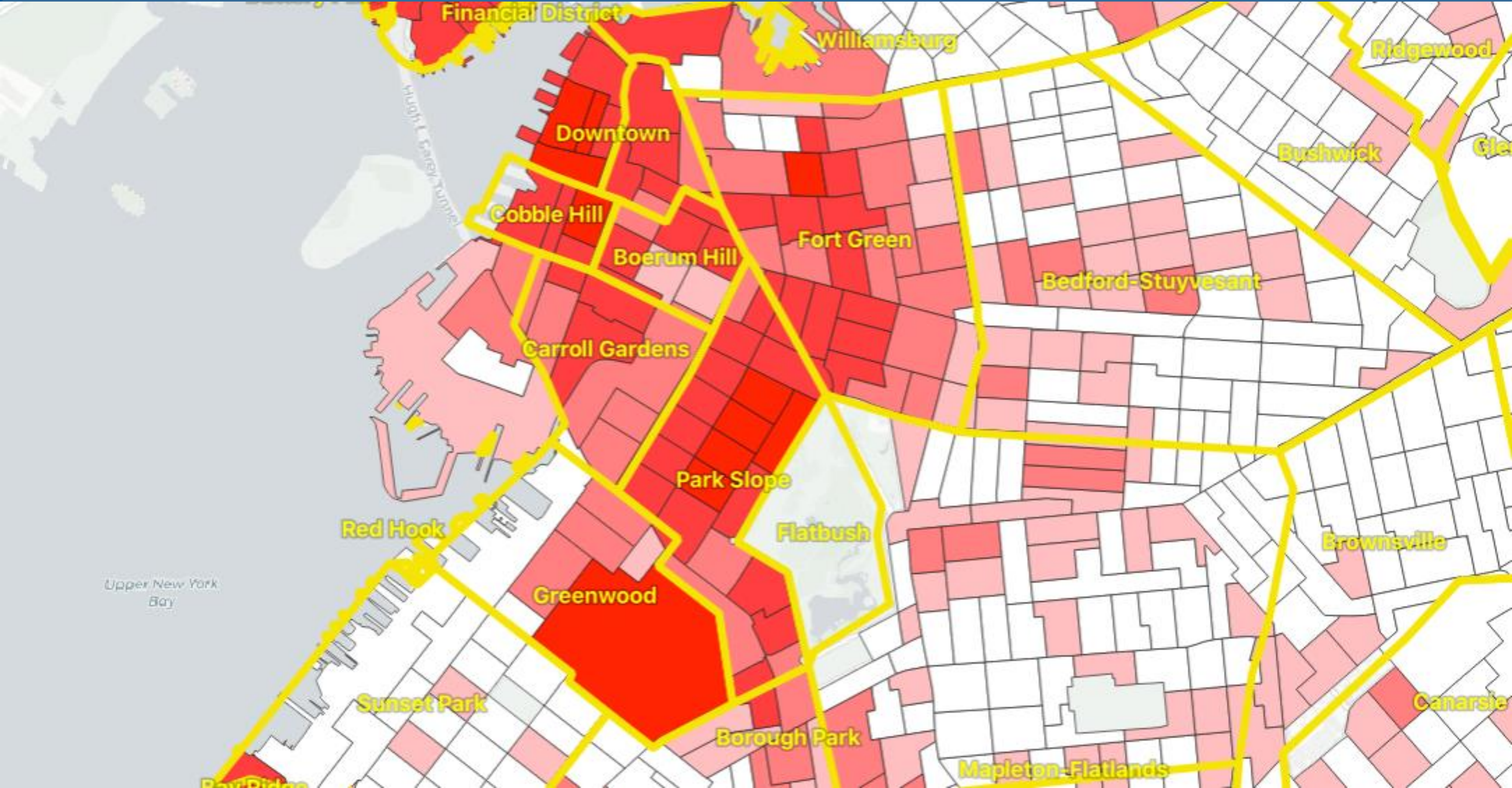
Graduate Degree Population %

```
SELECT
  100.0 * Sum(t.edu_graduate_dipl) /
          Sum(t.edu_total) AS graduate_pct,
  n.name, n.boroname
FROM nyc_neighborhoods n
JOIN nyc_census_tracts t
ON ST_Intersects(n.geom, t.geom)
WHERE t.edu_total > 0
GROUP BY n.name, n.boroname
ORDER BY graduate_pct DESC
LIMIT 10;
```

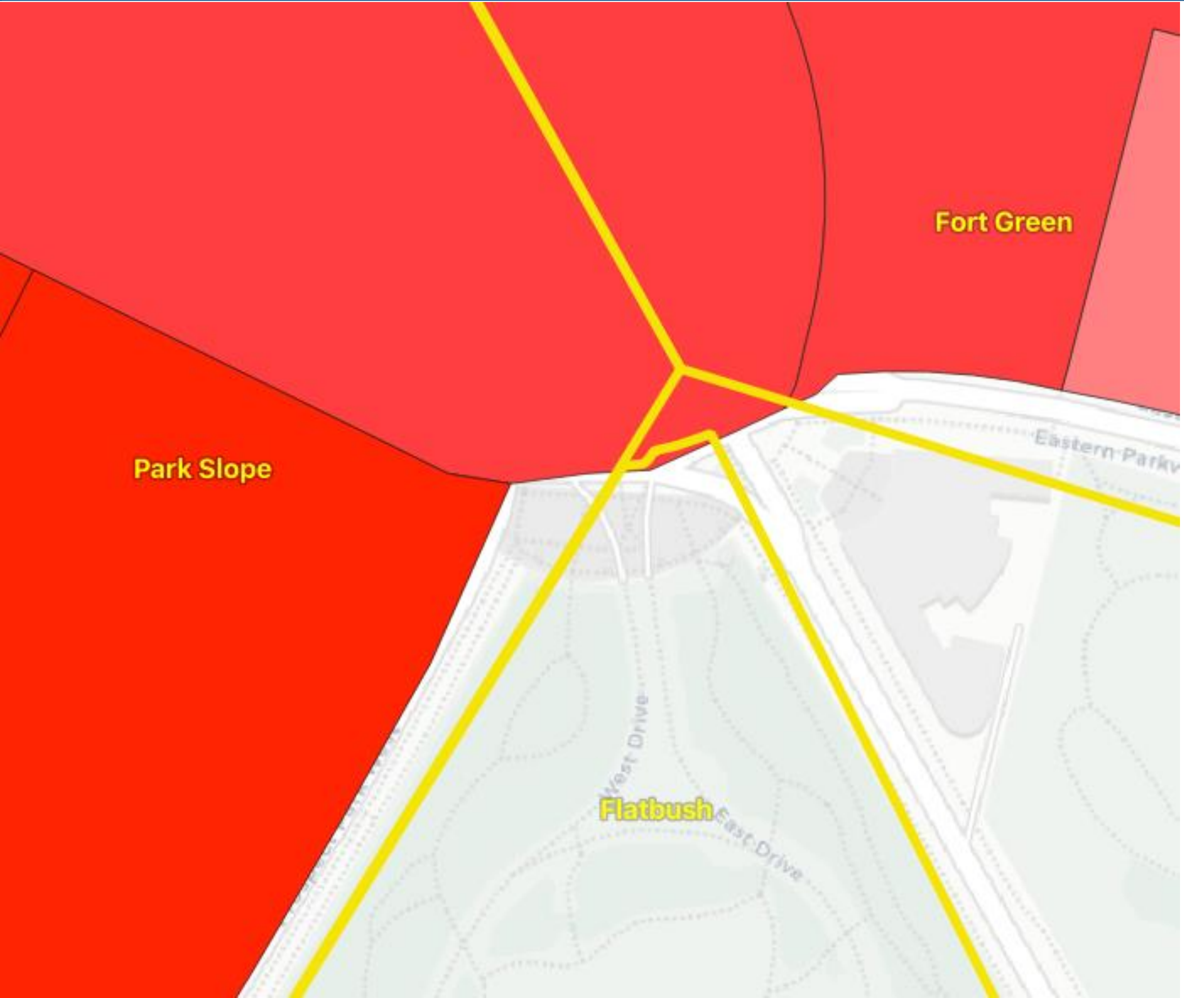
Graduate Degree Population %

```
   graduate_pct       |        name         | boroname
----------------------+---------------------+-----------
 47.6469321851453175 | Carnegie Hill       | Manhattan
 42.1632365492235696 | Upper West Side     | Manhattan
 41.0656645950763598 | Battery Park        | Manhattan
 39.5611557679774060 | Flatbush            | Brooklyn
 39.3409549428379287 | Tribeca             | Manhattan
 39.2188240872451399 | North Sutton Area   | Manhattan
 38.6922550118291620 | Greenwich Village   | Manhattan
 38.6054942073575506 | Upper East Side     | Manhattan
 37.8834795573140662 | Murray Hill         | Manhattan
 37.3714416181491744 | Central Park        | Manhattan
```

PostGIS

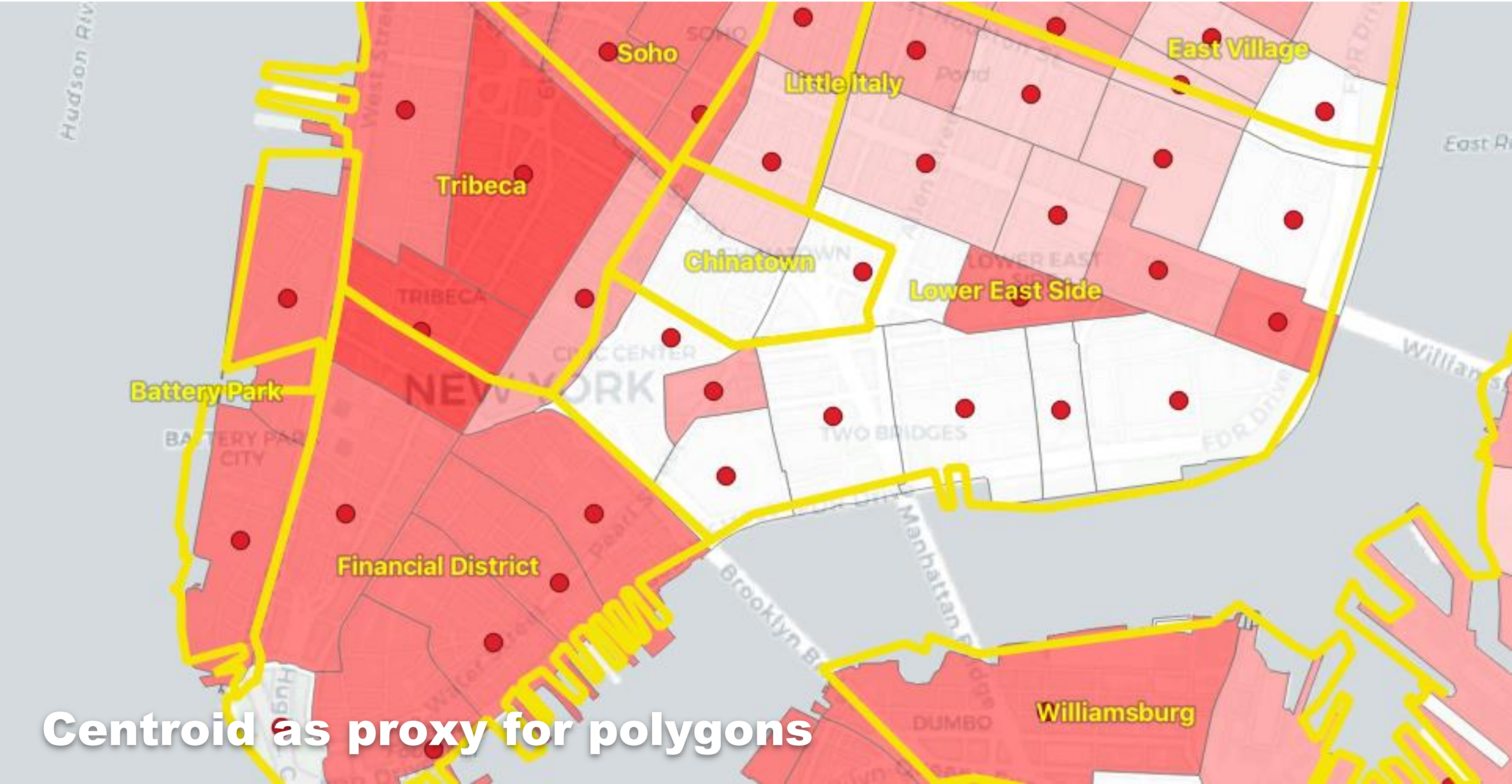


Financial District
Williamsburg
Ridgewood
Downtown
Bushwick
Cobble Hill
Fort Green
Boerum Hill
Bedford-Stuyvesant
Carroll Gardens
Park Slope
Red Hook
Flatbush
Brownsville
Upper New York Bay
Greenwood
Sunset Park
Borough Park
Canarsie
Mapleton-Flatlands
Bay Ridge



The otherwise-empty "Flatbush" neighborhood polygon (which mostly covers Prospect Park) just grazes one high-education tract polygon, resulting in a spurious high measurement for the neighborhood.

What if a tract falls on the border between two neighborhoods?



Centroid as proxy for polygons

Join on ST_Centroid

```
SELECT
    100.0 * Sum(t.edu_graduate_dipl) /
            Sum(t.edu_total) AS graduate_pct,
    n.name,
    n.boroname
FROM nyc_neighborhoods n
JOIN nyc_census_tracts t
ON ST_Contains(n.geom, ST_Centroid(t.geom))
WHERE t.edu_total > 0
GROUP BY n.name, n.boroname
ORDER BY graduate_pct DESC
LIMIT 10;
```

Join on intersects vs centroid

ST_Intersects()

graduate_pct	name	boroname
47.6	Carnegie Hill	Manhattan
42.2	Upper West Side	Manhattan
41.1	Battery Park	Manhattan
39.6	Flatbush	Brooklyn
39.3	Tribeca	Manhattan
39.2	North Sutton Area	Manhattan
38.7	Greenwich Village	Manhattan
38.6	Upper East Side	Manhattan
37.9	Murray Hill	Manhattan
37.4	Central Park	Manhattan

ST_Centroid()

graduate_pct	name	boroname
48.0	Carnegie Hill	Manhattan
44.2	Morningside Heights	Manhattan
42.1	Greenwich Village	Manhattan
42.0	Upper West Side	Manhattan
41.4	Tribeca	Manhattan
40.7	Battery Park	Manhattan
39.5	Upper East Side	Manhattan
39.3	North Sutton Area	Manhattan
37.4	Cobble Hill	Brooklyn
37.4	Murray Hill	Manhattan

How many people live within 500m of a subway station?

ST_Centroid

How many people in New York?

```
SELECT
  Sum(popn_total)
FROM nyc_census_blocks;
```

ST_Centroid

How many people 500m from a subway station?

```
SELECT
  Sum(popn_total)
FROM nyc_census_blocks census
JOIN nyc_subway_stations subway
  ON ST_DWithin(
       census.geom,
       subway.geom,
       500
     );
```

ST_Centroid

How many people in New York?

8,175,032

How many people 500m from a subway station?

10,855,873 ?!?!!?

Overlapping query areas

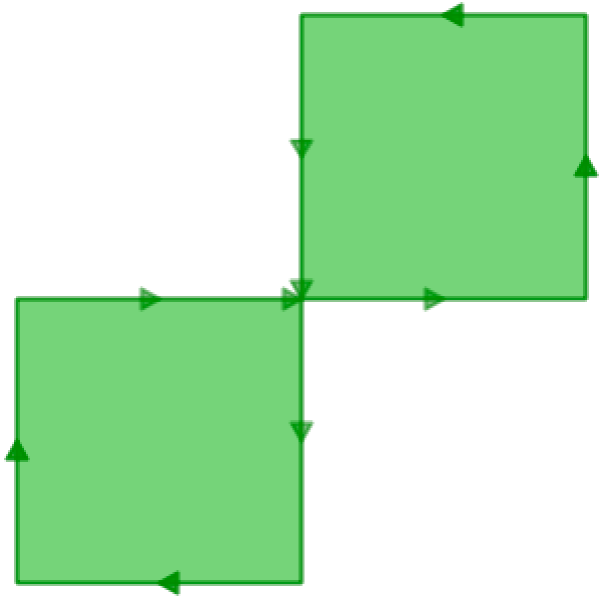
Overlapping query areas

How many people 500m from a subway station?

```
WITH distinct_blocks AS (
  SELECT DISTINCT ON (blkid) popn_total
  FROM nyc_census_blocks census
  JOIN nyc_subway_stations subway
  ON ST_DWithin(
        census.geom,
        subway.geom,
        500)
)
SELECT Sum(popn_total)
FROM distinct_blocks;
```

Section 23 - Validity

POLYGON((0 0, 0 1, 2 1, 2 2, 1 2, 1 0, 0 0))

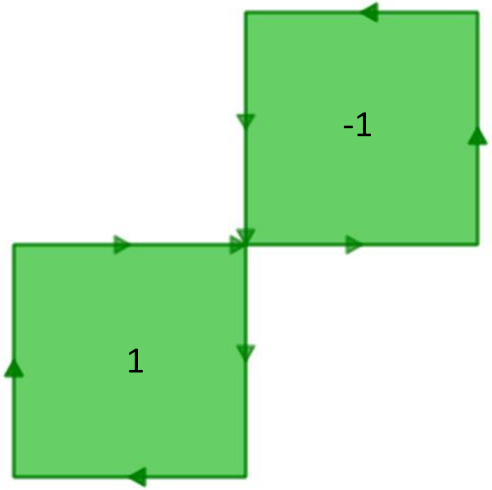


Why does validity matter?

Geometry algorithms rely on properties enforced by validity: ring orientation, self-crossing, self-touching. All can confuse different algorithms.

```
SELECT ST_Area(ST_GeomFromText(
  'POLYGON((0 0, 0 1, 1 1,
            2 1, 2 2, 1 2,
            1 1, 1 0, 0 0))'
));
```

```
0
```

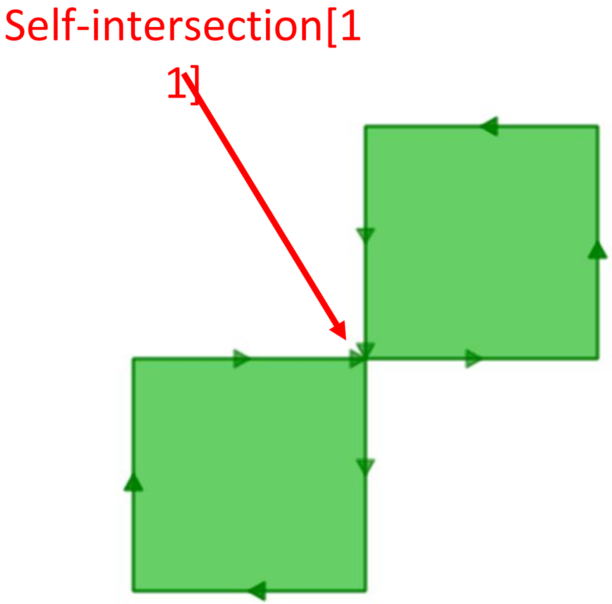


Can we test validity?

ST_IsValid() returns a boolean for validity, and ST_IsValidReason() returns a coordinate of where the error is, and a textual reason.

```
SELECT ST_IsValid(ST_GeomFromText(
  'POLYGON((0 0, 0 1, 1 1,
            2 1, 2 2, 1 2,
            1 1, 1 0, 0 0))'
));
```

```
false
```

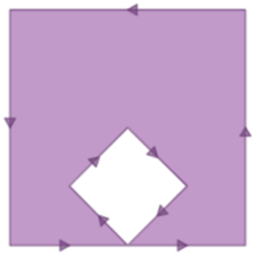


Can we test validity in bulk?

```
SELECT name, boroname,
  ST_IsValidReason(geom)
FROM nyc_neighborhoods
WHERE NOT ST_IsValid(geom);
```

```
 Howard Beach | Queens    | Self-intersection[596394 4500899]
 Corona       | Queens    | Self-intersection[595483 4513817]
 Red Hook     | Brooklyn  | Self-intersection[582655 4500908]
 Steinway     | Queens    | Self-intersection[593198 4515125]
```

Can we fix validity in bulk?



The “banana polygon” is a polygon with a hole, formed by a ring that touches itself at a single point.

```
SELECT ST_AsText(ST_MakeValid(
  'POLYGON((0 0, 2 0, 1 1, 2 2, 3 1, 2 0, 4 0, 4 4, 0 4, 0 0))'))
```

ST_MakeValid() juggles the components of an invalid polygon to form a “best guess” valid interpretation of the rings. The “banana polygon” gets turns into a traditional exterior/interior ring polygon.

```
POLYGON((0 0,0 4,4 4,4 0,2 0,0 0),(3 1,2 2,1 1,2 0,3 1))
```

Can we fix validity in bulk?

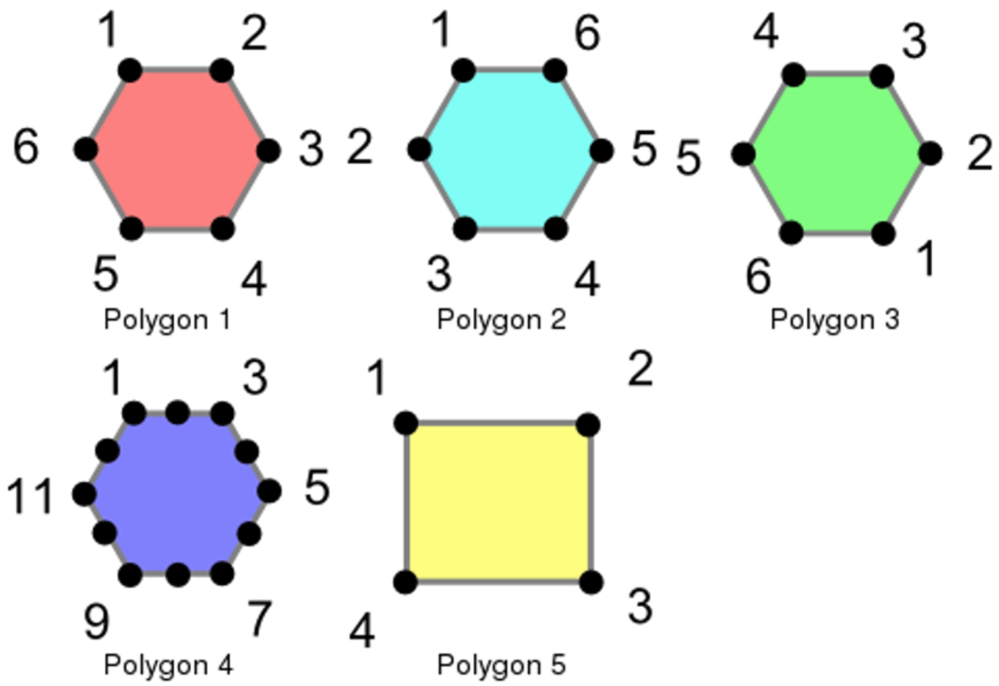
```
UPDATE nyc_neighborhoods
SET geom = ST_MakeValid(geom)
WHERE NOT ST_IsValid(geom);
```

ST_MakeValid(geom, options)

PostGIS 3.2+ includes text options to change the repair algorithm.

```
'method=linework'
'method=structure keepcollapsed=false'
```

Section 24 - Equality


1
2
6
3
5
4
Polygon 1
1
6
2
5
3
4
Polygon 2
4
3
5
2
6
1
Polygon 3
1
3
11
5
9
7
Polygon 4
1
2
4
3
Polygon 5

Create the test polygons

```
CREATE TABLE polygons (id integer, name varchar, poly geometry);

INSERT INTO polygons VALUES
    (1, 'Polygon 1', 'POLYGON((-1 1.732,1 1.732,2 0,1 -1.732,
        -1 -1.732,-2 0,-1 1.732))'),
    (2, 'Polygon 2', 'POLYGON((-1 1.732,-2 0,-1 -1.732,1 -1.732,
        2 0,1 1.732,-1 1.732))'),
    (3, 'Polygon 3', 'POLYGON((1 -1.732,2 0,1 1.732,-1 1.732,
        -2 0,-1 -1.732,1 -1.732))'),
    (4, 'Polygon 4', 'POLYGON((-1 1.732,0 1.732, 1 1.732,1.5 0.866,
        2 0,1.5 -0.866,1 -1.732,0 -1.732,-1 -1.732,-1.5 -0.866,
        -2 0,-1.5 0.866,-1 1.732))'),
    (5, 'Polygon 5', 'POLYGON((-2 -1.732,2 -1.732,2 1.732,
        -2 1.732,-2 -1.732))');
```

Ways of testing equality!

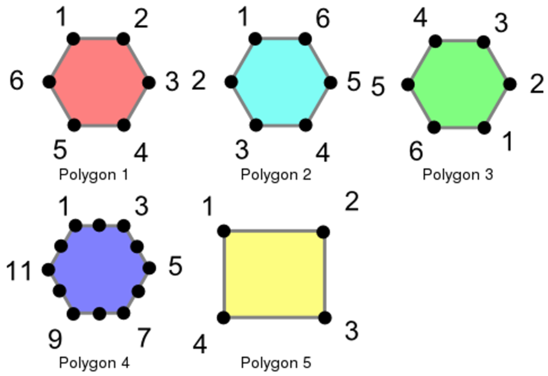
ST_OrderingEquals(A, B)

ST_Equals(A, B)

A = B

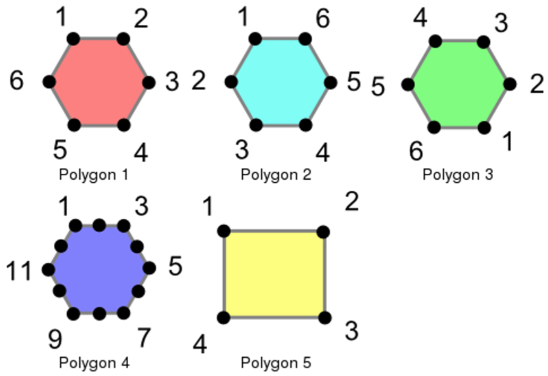
A ~= B

```sql
SELECT a.name, b.name,
  CASE WHEN
    ST_OrderingEquals(a.poly, b.poly)
      THEN 'Exactly Equal'
      ELSE 'Not Exactly Equal' END
FROM polygons AS a,
     polygons AS b;
```



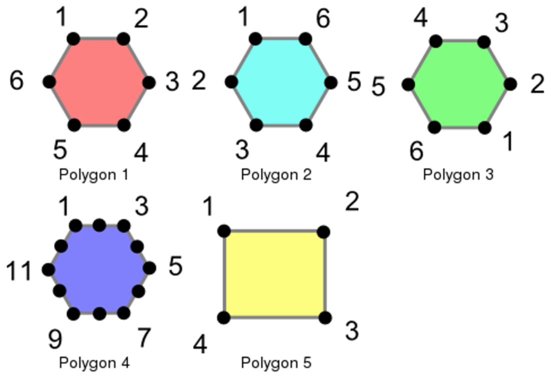
Polygon 1	Polygon 1	Exactly Equal
Polygon 1	Polygon 2	Not Exactly Equal
Polygon 1	Polygon 3	Not Exactly Equal
Polygon 1	Polygon 4	Not Exactly Equal
Polygon 1	Polygon 5	Not Exactly Equal
Polygon 2	Polygon 1	Not Exactly Equal
Polygon 2	Polygon 2	Exactly Equal
Polygon 2	Polygon 3	Not Exactly Equal
Polygon 2	Polygon 4	Not Exactly Equal
Polygon 2	Polygon 5	Not Exactly Equal
Polygon 3	Polygon 1	Not Exactly Equal
Polygon 3	Polygon 2	Not Exactly Equal
Polygon 3	Polygon 3	Exactly Equal
Polygon 3	Polygon 4	Not Exactly Equal
Polygon 3	Polygon 5	Not Exactly Equal
Polygon 4	Polygon 1	Not Exactly Equal
Polygon 4	Polygon 2	Not Exactly Equal
Polygon 4	Polygon 3	Not Exactly Equal
Polygon 4	Polygon 4	Exactly Equal
Polygon 4	Polygon 5	Not Exactly Equal
Polygon 5	Polygon 1	Not Exactly Equal
Polygon 5	Polygon 2	Not Exactly Equal
Polygon 5	Polygon 3	Not Exactly Equal
Polygon 5	Polygon 4	Not Exactly Equal
Polygon 5	Polygon 5	Exactly Equal

```
SELECT a.name, b.name,
  CASE WHEN ST_Equals(a.poly, b.poly)
       THEN 'Spatially Equal'
       ELSE 'Not Equal' END
FROM polygons AS a,
     polygons AS b;
```



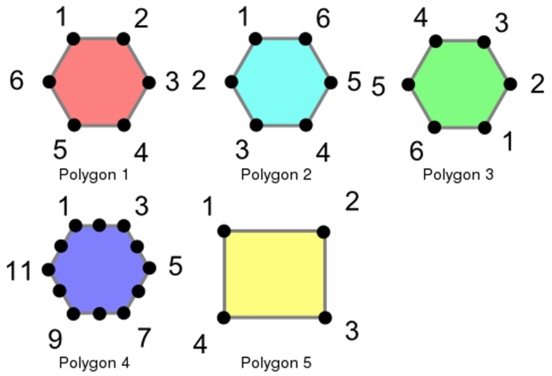

```
Polygon 1 | Polygon 1 | Spatially Equal
Polygon 1 | Polygon 2 | Spatially Equal
Polygon 1 | Polygon 3 | Spatially Equal
Polygon 1 | Polygon 4 | Spatially Equal
Polygon 1 | Polygon 5 | Not Equal
Polygon 2 | Polygon 1 | Spatially Equal
Polygon 2 | Polygon 2 | Spatially Equal
Polygon 2 | Polygon 3 | Spatially Equal
Polygon 2 | Polygon 4 | Spatially Equal
Polygon 2 | Polygon 5 | Not Equal
Polygon 3 | Polygon 1 | Spatially Equal
Polygon 3 | Polygon 2 | Spatially Equal
Polygon 3 | Polygon 3 | Spatially Equal
Polygon 3 | Polygon 4 | Spatially Equal
Polygon 3 | Polygon 5 | Not Equal
Polygon 4 | Polygon 1 | Spatially Equal
Polygon 4 | Polygon 2 | Spatially Equal
Polygon 4 | Polygon 3 | Spatially Equal
Polygon 4 | Polygon 4 | Spatially Equal
Polygon 4 | Polygon 5 | Not Equal
Polygon 5 | Polygon 1 | Not Equal
Polygon 5 | Polygon 2 | Not Equal
Polygon 5 | Polygon 3 | Not Equal
Polygon 5 | Polygon 4 | Not Equal
Polygon 5 | Polygon 5 | Spatially Equal
```

```
SELECT a.name, b.name,
  CASE WHEN a.poly = b.poly
       THEN 'Spatially ='
       ELSE 'Not =' END
FROM polygons AS a,
     polygons AS b;
```



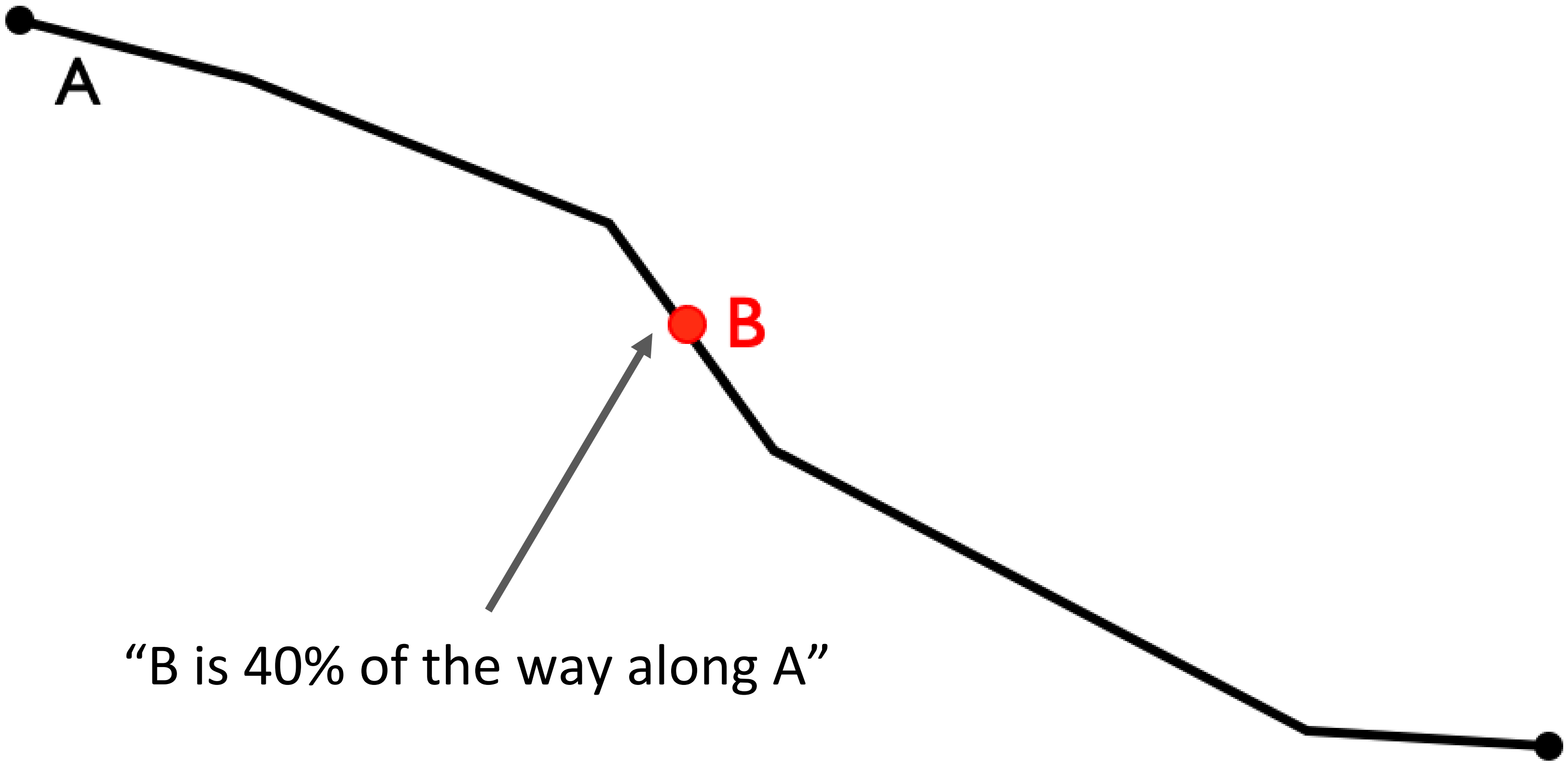

```
Polygon 1 | Polygon 1 | Spatially =
Polygon 1 | Polygon 2 | Not =
Polygon 1 | Polygon 3 | Not =
Polygon 1 | Polygon 4 | Not =
Polygon 1 | Polygon 5 | Not =
Polygon 2 | Polygon 1 | Not =
Polygon 2 | Polygon 2 | Spatially =
Polygon 2 | Polygon 3 | Not =
Polygon 2 | Polygon 4 | Not =
Polygon 2 | Polygon 5 | Not =
Polygon 3 | Polygon 1 | Not =
Polygon 3 | Polygon 2 | Not =
Polygon 3 | Polygon 3 | Spatially =
Polygon 3 | Polygon 4 | Not =
Polygon 3 | Polygon 5 | Not =
Polygon 4 | Polygon 1 | Not =
Polygon 4 | Polygon 2 | Not =
Polygon 4 | Polygon 3 | Not =
Polygon 4 | Polygon 4 | Spatially =
Polygon 4 | Polygon 5 | Not =
Polygon 5 | Polygon 1 | Not =
Polygon 5 | Polygon 2 | Not =
Polygon 5 | Polygon 3 | Not =
Polygon 5 | Polygon 4 | Not =
Polygon 5 | Polygon 5 | Spatially =
```

```
SELECT a.name, b.name,
  CASE WHEN a.poly ~= b.poly
       THEN 'Bounds Equal'
       ELSE 'Bounds Not Equal' END
FROM polygons AS a,
     polygons AS b;
```

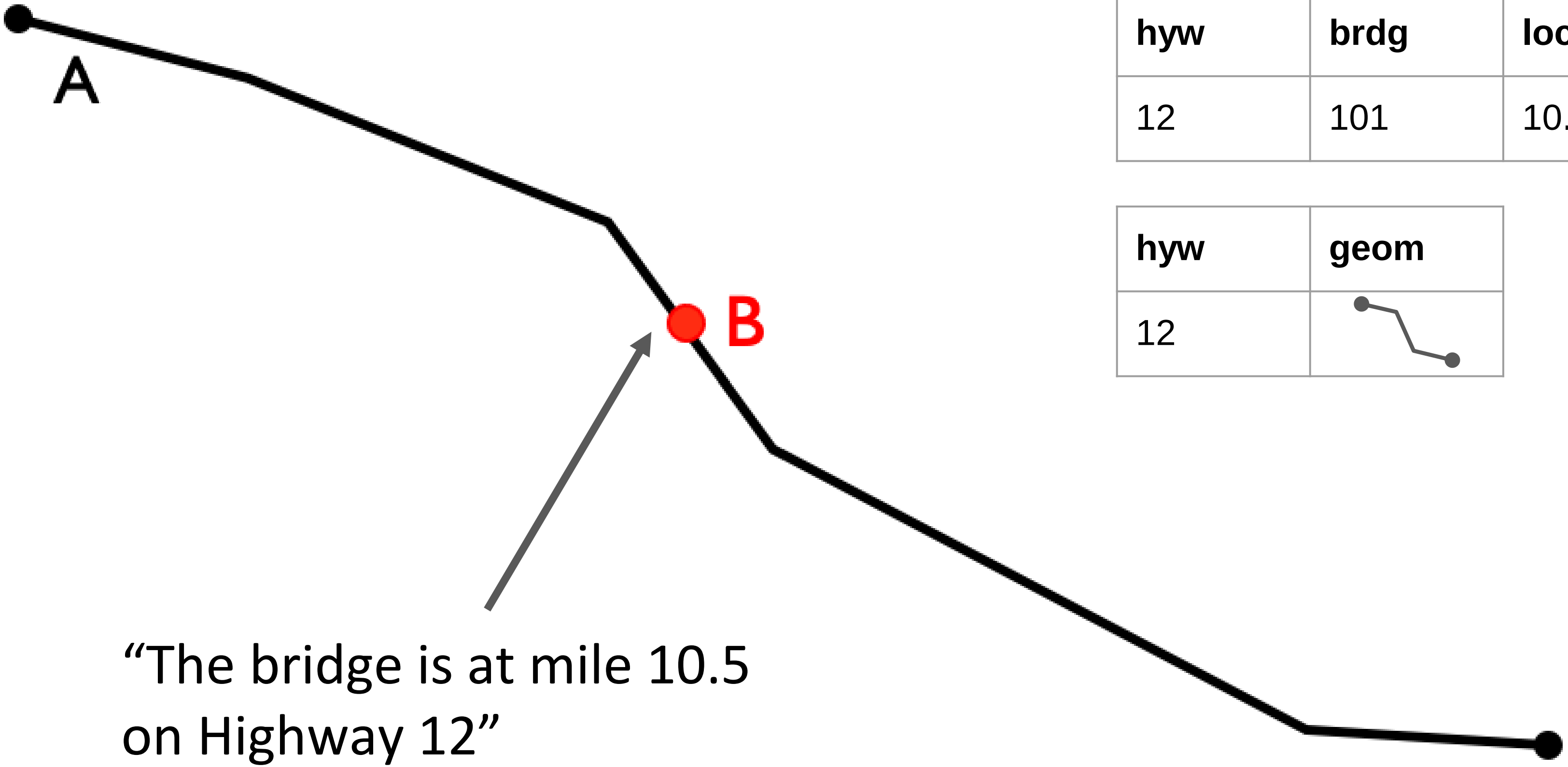


Polygon 1	Polygon 1	Bounds Equal
Polygon 1	Polygon 2	Bounds Equal
Polygon 1	Polygon 3	Bounds Equal
Polygon 1	Polygon 4	Bounds Equal
Polygon 1	Polygon 5	Bounds Equal
Polygon 2	Polygon 1	Bounds Equal
Polygon 2	Polygon 2	Bounds Equal
Polygon 2	Polygon 3	Bounds Equal
Polygon 2	Polygon 4	Bounds Equal
Polygon 2	Polygon 5	Bounds Equal
Polygon 3	Polygon 1	Bounds Equal
Polygon 3	Polygon 2	Bounds Equal
Polygon 3	Polygon 3	Bounds Equal
Polygon 3	Polygon 4	Bounds Equal
Polygon 3	Polygon 5	Bounds Equal
Polygon 4	Polygon 1	Bounds Equal
Polygon 4	Polygon 2	Bounds Equal
Polygon 4	Polygon 3	Bounds Equal
Polygon 4	Polygon 4	Bounds Equal
Polygon 4	Polygon 5	Bounds Equal
Polygon 5	Polygon 1	Bounds Equal
Polygon 5	Polygon 2	Bounds Equal
Polygon 5	Polygon 3	Bounds Equal
Polygon 5	Polygon 4	Bounds Equal
Polygon 5	Polygon 5	Bounds Equal

Section 25 - Linear Referencing



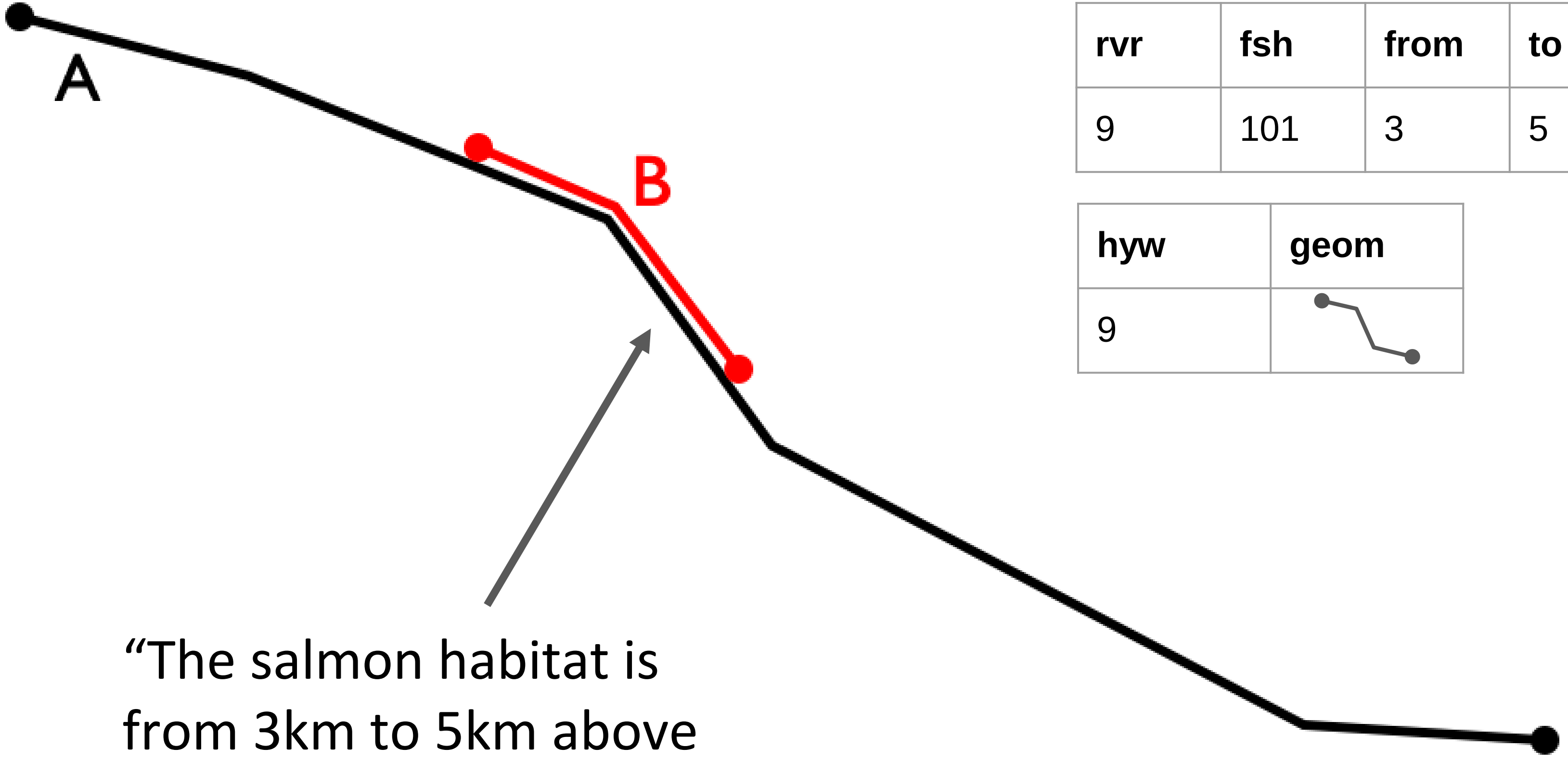
"B is 40% of the way along A"



hyw	brdg	loc
12	101	10.5

hyw	geom
12	

“The bridge is at mile 10.5 on Highway 12”



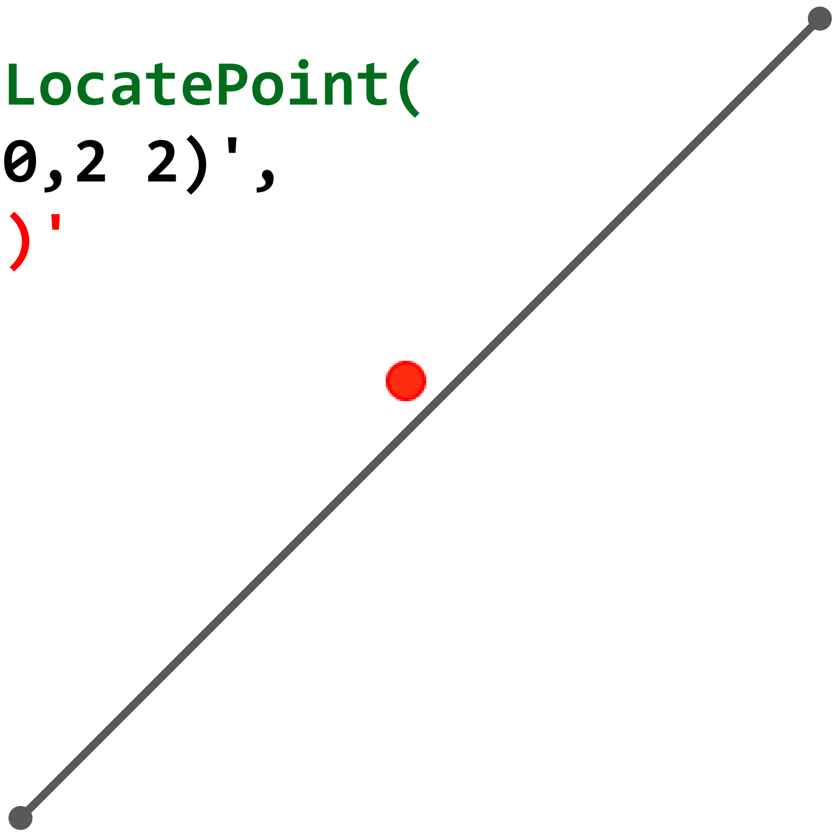
rvr	fsh	from	to
9	101	3	5

hyw	geom
9	

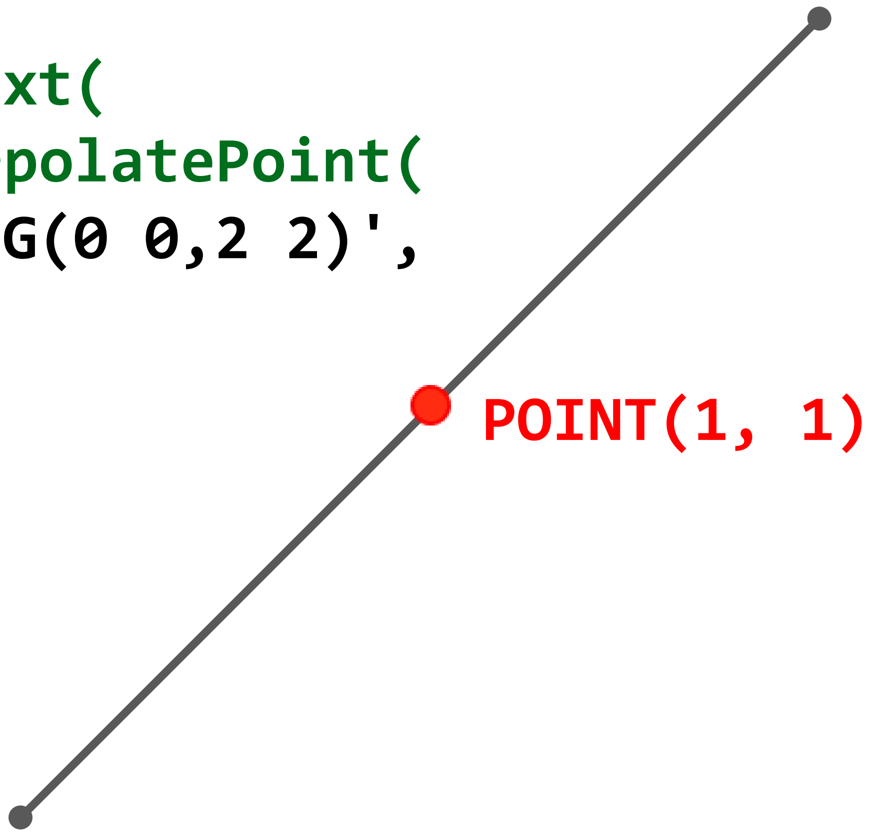
“The salmon habitat is from 3km to 5km above the confluence”

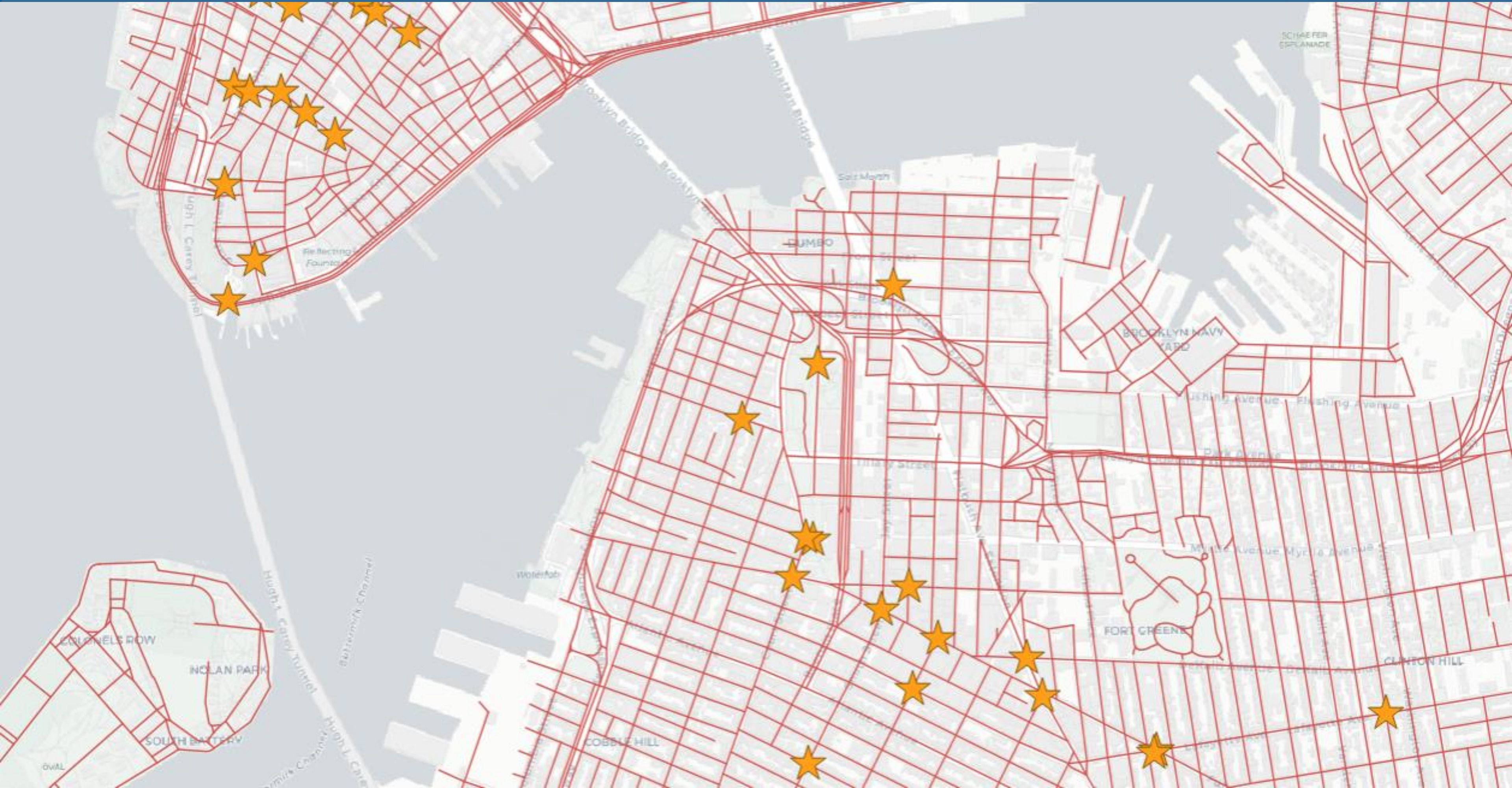
```
SELECT ST_LineLocatePoint(
'LINESTRING(0 0,2 2)',
'POINT(0.9 1.1)'
);


0.5
```



```
SELECT ST_AsText(
  ST_LineInterpolatePoint(
    'LINESTRING(0 0,2 2)',
    0.5
  ));
```




DUMBO
BROOKLYN NAVY YARD
Flushing Avenue
Park Avenue
Myrtle Avenue
FORT GREENE
CLINTON HILL
COBBLE HILL
NOLAN PARK
COLONELS ROW
SOUTH BATTERY
Brooklyn Bridge
Manhattan Bridge
Hugh L. Carey Tunnel
Buttermilk Channel

Find nearest street to each subway station

```
WITH ordered_nearest AS (
  SELECT
    ST_GeometryN(str.geom,1) AS streets_geom,
    str.gid AS str_gid,
    sub.geom AS subways_geom,
    sub.gid AS subways_gid,
    ST_Distance(str.geom, sub.geom) AS distance
  FROM nyc_streets str
  JOIN nyc_subway_stations sub
    ON ST_DWithin(str.geom, sub.geom, 200)
  ORDER BY subways_gid, distance ASC
)
```

Find measure of station on nearest street

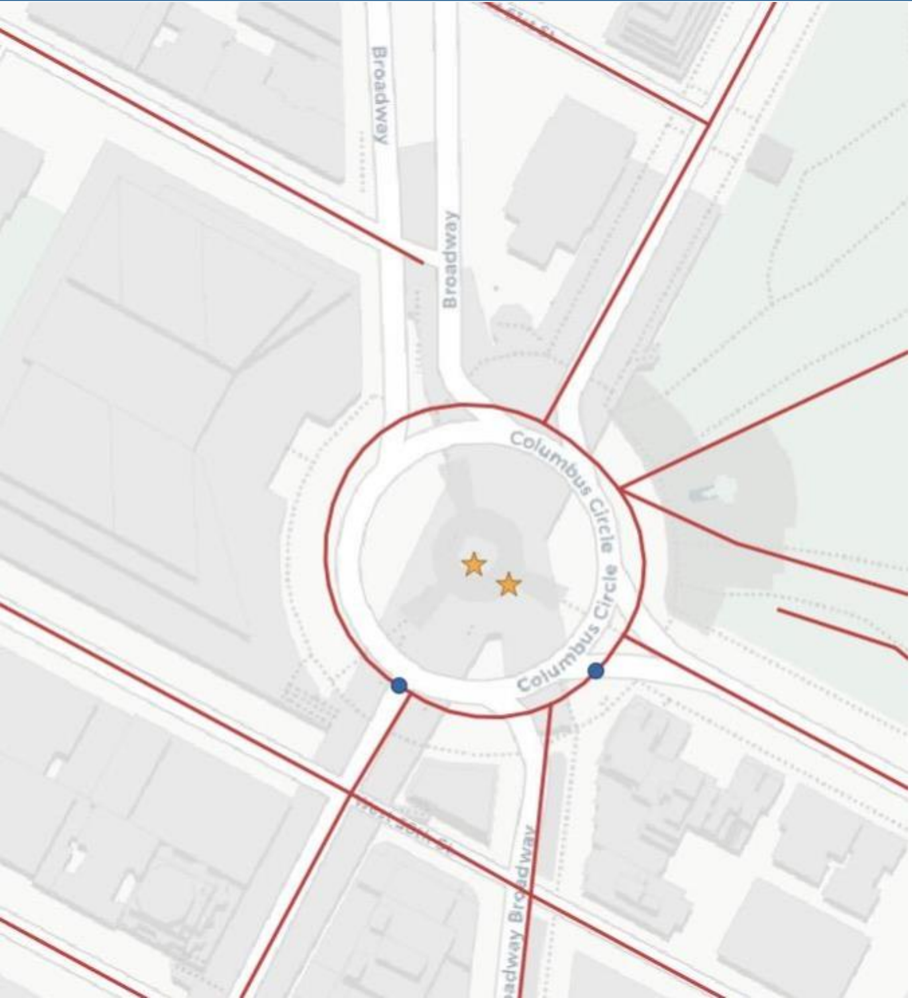
```
SELECT
DISTINCT ON (subways_gid)
  subways_gid,
  streets_gid,
  distance,
  ST_LineLocatePoint(
    streets_geom,
    subways_geom) AS measure
FROM ordered_nearest;
```

Find measure of station on nearest street

```
 subways_gid | streets_gid |        measure
-------------+-------------+----------------------
           1 |       17404 | 0.0023154983819572554
           2 |       17318 |    0.6354078182846773
           3 |       19086 |  0.24946227178552738
           4 |        1924 |  0.11187222763997673
           5 |        2067 |    0.9261874246426975
           6 |        1934 |  0.33457647816803476
           7 |        2024 |    0.5549461001845787
           8 |        2469 |    0.2296616075093935
           9 |        2024 |    0.9069811058590412
          10 |        2067 |    0.6202998183141508
```

How to visualize events? Turn them back into points.

```
-- New view that turns events back
-- into spatial objects
CREATE OR REPLACE
VIEW nyc_subway_stations_lrs AS
SELECT
  events.subways_gid,
  ST_LineInterpolatePoint(
    ST_GeometryN(streets.geom, 1),
    events.measure) AS geom,
  events.streets_gid
FROM nyc_subway_station_events events
JOIN nyc_streets streets
ON (streets.gid = events.streets_gid);
```



Original subway stations (orange stars) on Columbus Circle have been snapped over to the nearby roadways in the LRS view (blue circles)

Shows how LRS functions can be used to snap points to a network, as well as to manage actual LRS data.

Section 29 - Nearest Neighbor Searching

Nearest Neighbor Search

“What is the nearest fire station to this address?”

“What are the 10 nearest gas stations to the current locations?”

Nearest Neighbor Join

“Add the nearest fire station to every parcel in the table.”

Nearest Neighbor Search

```sql
-- The location of Broad St station
-- SRID=26918;POINT(583571.9 4506714.3)
SELECT streets.gid, streets.name,
  ST_Distance(streets.geom,
    'SRID=26918;POINT(583571.9 4506714.3)') AS dist
FROM nyc_streets streets
ORDER BY
  streets.geom <->
  'SRID=26918;POINT(583571.9 4506714.3)'::geometry
LIMIT 3;
```

no WHERE clause

ORDER BY distance

LIMIT clause

gid	name	dist
17385	Wall St	0.749987508809928
17390	Broad St	0.883630623519105 9
17436	Nassau St	1.33682802410704 1

Nearest Neighbor Join

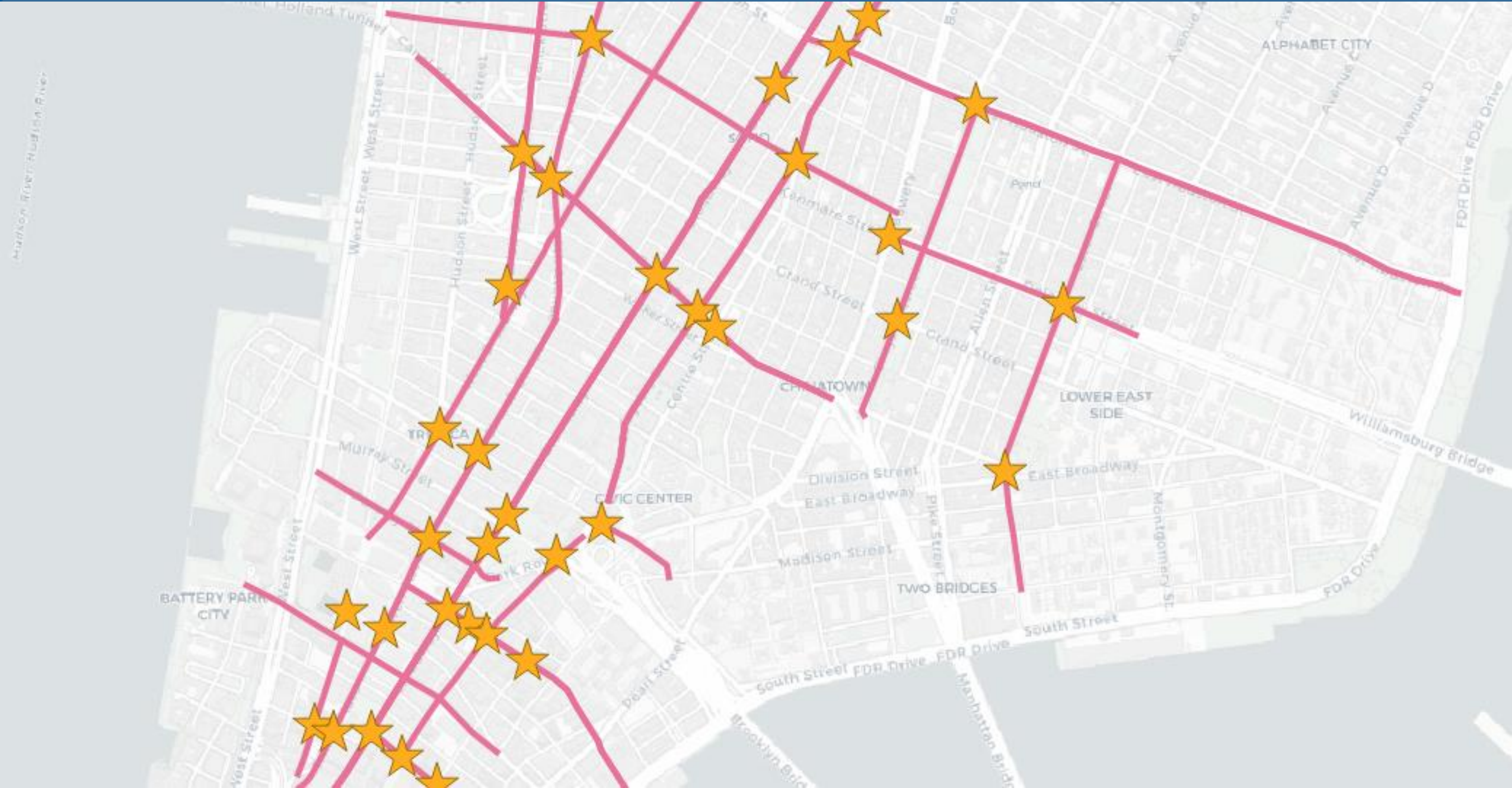
```
SELECT subways.gid AS subway_gid,
       subways.name AS subway,
       streets.name AS street,
       streets.gid AS street_gid
FROM nyc_subway_stations subways
CROSS JOIN LATERAL (
    SELECT streets.name, streets.geom, streets.gid
    FROM nyc_streets streets
    ORDER BY streets.geom <-> subways.geom
    LIMIT 1
) streets;
```

LATERAL join

outer parameter

LIMIT clause

ORDER BY distance


Holland Tunnel
Hudson River
West Street
Hudson Street
SoHo
Kenmare Street
Grand Street
ALPHABET CITY
Avenue D
FDR Drive
Pond
LOWER EAST SIDE
Williamsburg Bridge
East Broadway
CHINATOWN
Centre St
Murray Street
CIVIC CENTER
Division Street
East Broadway
Madison Street
Pike Street
Montgomery St
TWO BRIDGES
BATTERY PARK CITY
South Street
FDR Drive
Pearl Street
Brooklyn Bridge
Manhattan Bridge