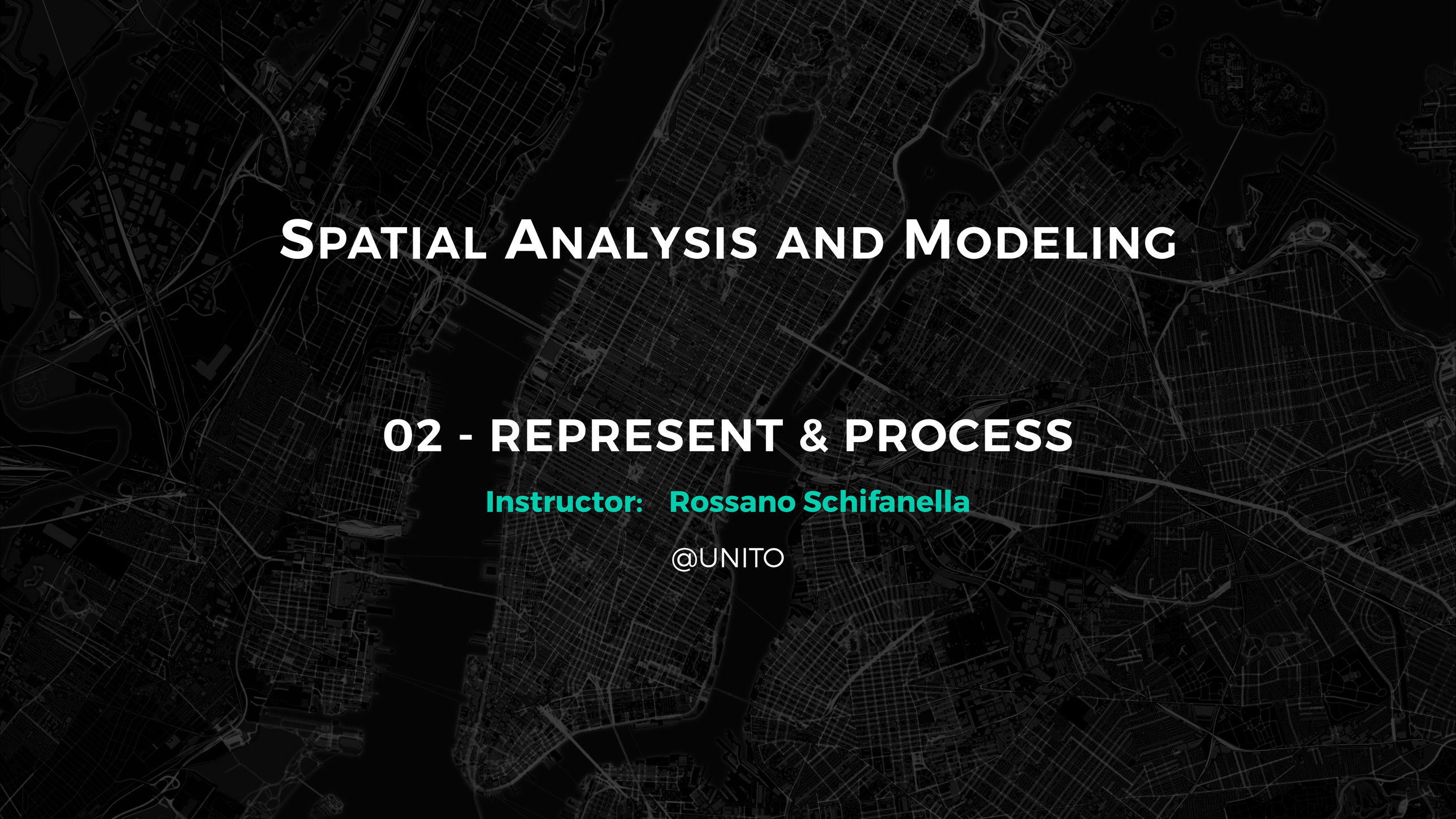


Spatial Analysis and Modeling

02 - REPRESENT & PROCESS

Instructor: Rossano Schifanella

@UNITO

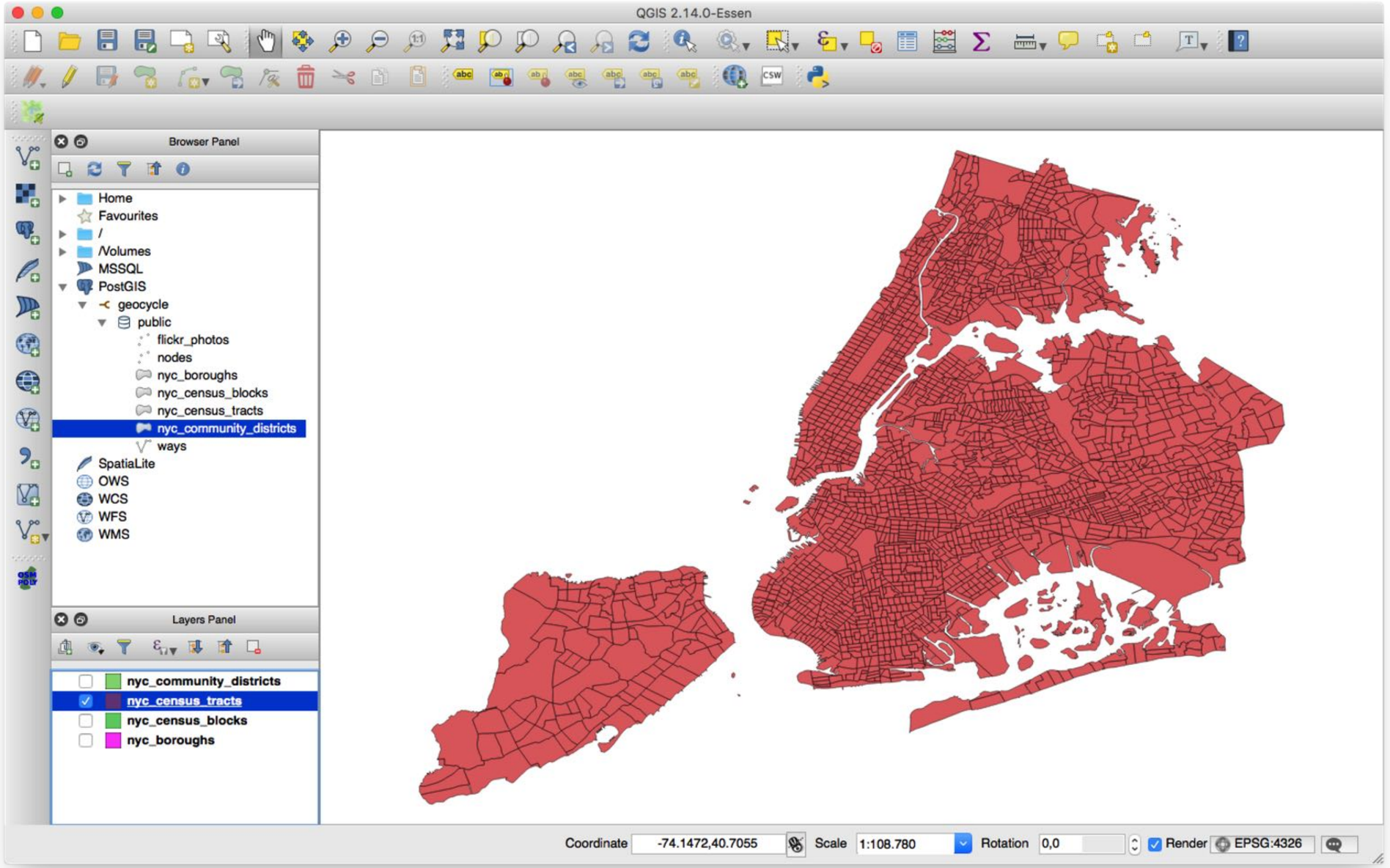
Spatial Data

- **Vectorial**
  - **Points**
    - location as a random event
    - locations of crimes, accidents, grocery stores
  - **Discrete spatial data - lattice data**
    - areal units
    - census tracts, counties, countries
  - **Networks**
    - nodes and links
    - street network, river network, social network
  - **Surfaces**
    - continuous spatial field
    - air quality surface, noise surface, price surface
- **Raster**

points data

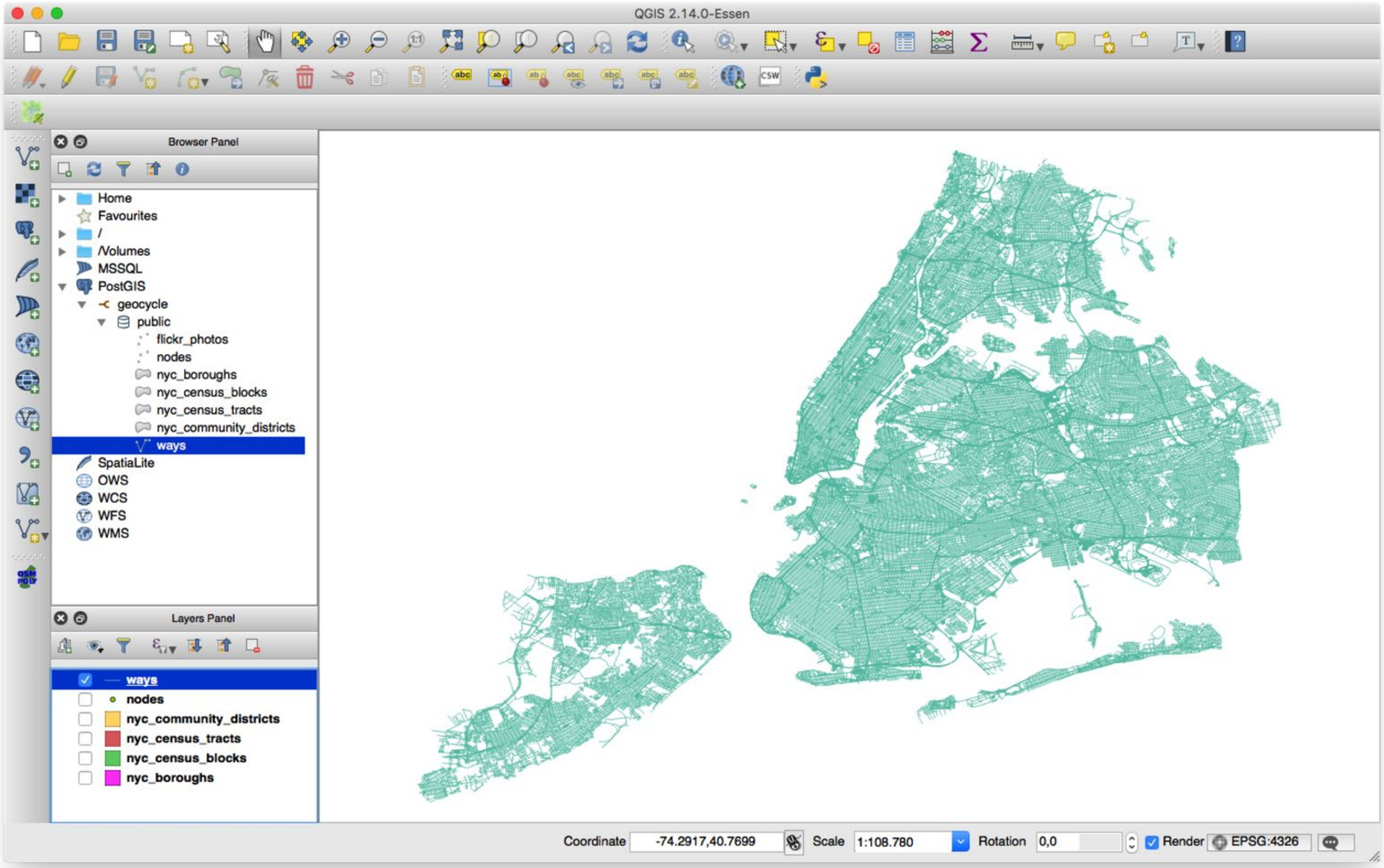
Instagram photos in NYC

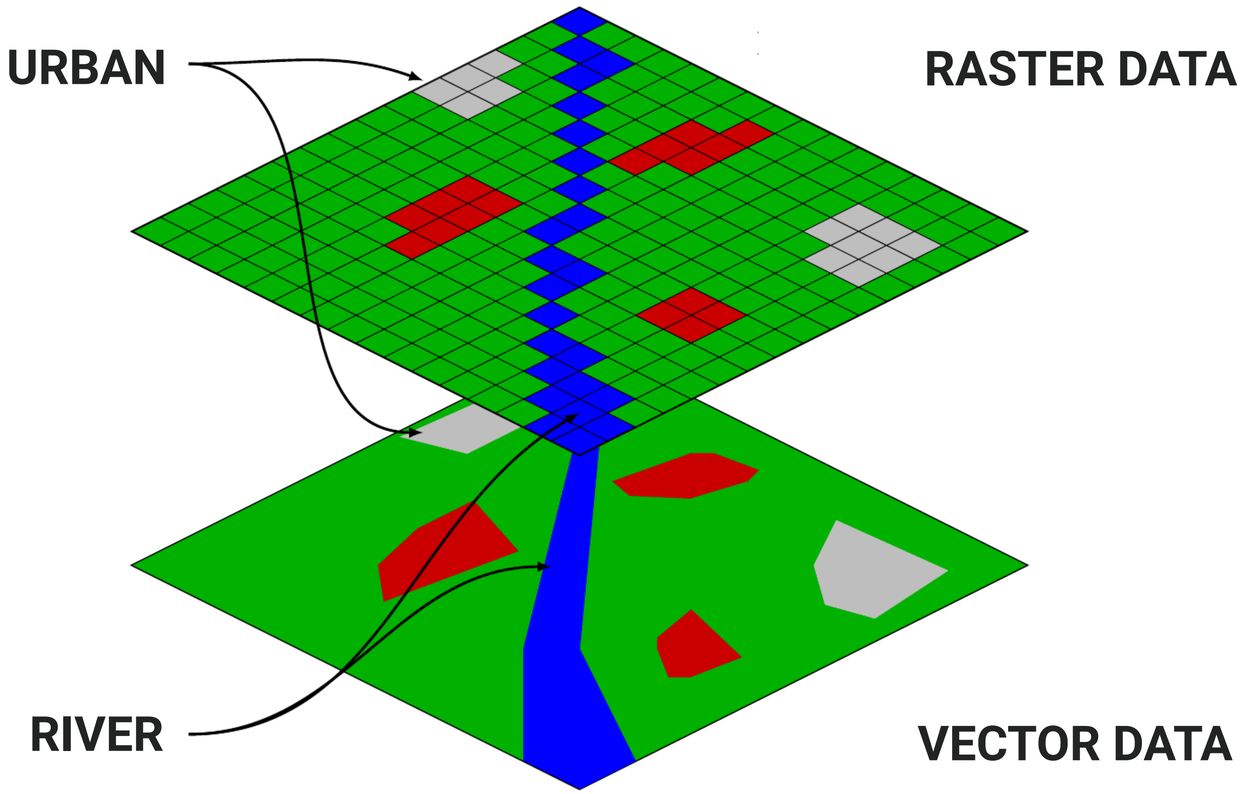
aerial data
NYC census tracts



lines and networks

Open Street Map ways in NYC




URBAN
RASTER DATA
RIVER
VECTOR DATA

georeferenced data:
sources

- **Open Data Portals**
  - NYC open data portal
  - London Open Data Portal
- **Administrative/Boundary Data, Census Data,**
- **Open Street Map (Roads, Parks, Water (lakes and rivers!), Land use)**
- **Social Media – User-generated data**
  - Instagram, Facebook, Flickr, Twitter and so on
- **Mobility traces**
- **CDR**
- **Many many others!**

OpenStreetMap

Tasks

- How to store spatial data?
- How to query spatial data?
- How to compute complex information from spatial data?

SPATIAL DATABASES vs FILE-BASED STORAGE

Example

- Geolocated images from Instagram or Flickr (YFCC100M)
- Neighbourhoods shapefile
- Street segments from Open Street Maps

- How do we compute the photos that belong to a particular neighbourhood or that have been taken within a certain distance from a street segment?

Advantages of a DB over files

- Avoids **redundancy** and **duplication**
- Reduces data **maintenance costs**
- Provide a **unified** means of accessing and analysing data (SQL)
- Applications are **separated** from the data
- Support **multiple concurrent applications** and **multiple users** editing and accessing the same data at the same time.
- Support **large data volumes** better than files (for some kinds of data!)
- Better **data sharing**
- **Security** and standards can be defined and enforced (access control)

Disadvantages of a DB over Files

- **Cost**
- **Complexity**
- **Performance** – especially complex data types
- **Integration** with other systems can be difficult

What is a spatial database?

Extension mechanism

Spatial Data Types

Spatial Join

Spatial Relationships

Spatial Indexing

Database System

Some Spatial DBMS Systems

PostGIS

Microsoft® SQL Server®

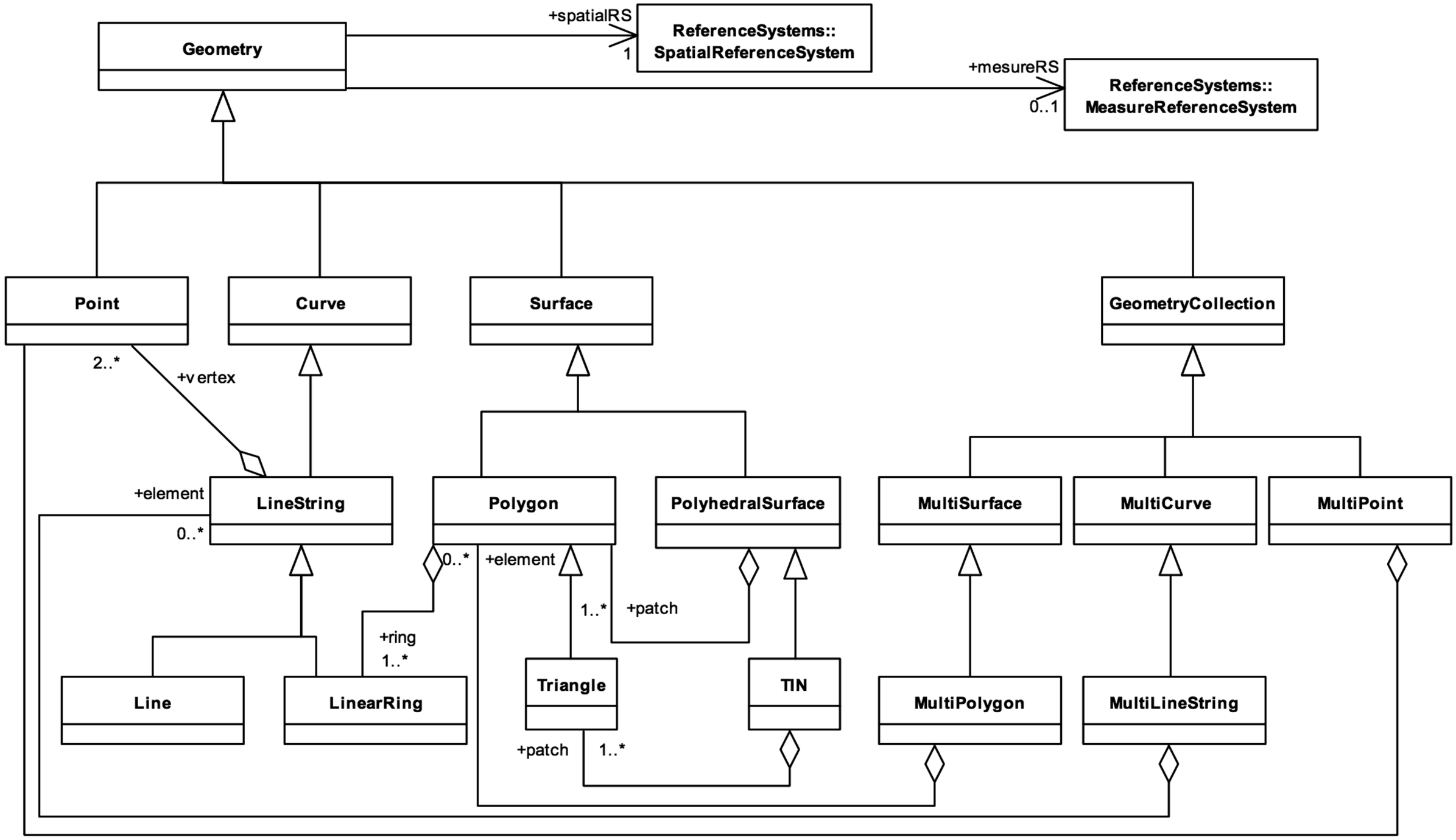
IBM DB2

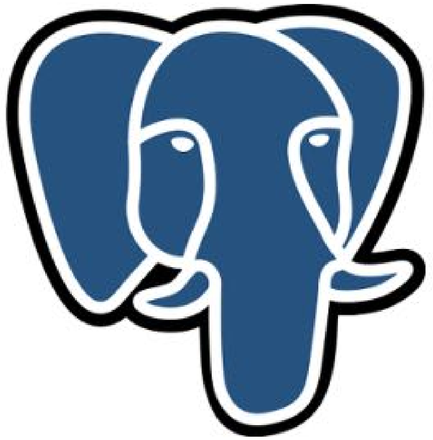
MySQL®

ORACLE® SPATIAL

Standards and Specifications

- **The OGC is an international not for profit organization committed to making quality open standards for the global geospatial community**
  - http://www.opengeospatial.org
- **Defines standard Geographic Information Systems (GIS) object types, functions associated with them, and metadata tables**
  - Simple Feature Access Architecture (common architecture for simple feature geometry)
    - http://www.opengeospatial.org/standards/sfa
  - Simple Features Access SQL (schema that supports storage, retrieval, query and update of feature collections via the SQL)
    - www.opengeospatial.org/standards/sfs


Geometry
+spatialRS
1
ReferenceSystems::
SpatialReferenceSystem
+mesureRS
0..1
ReferenceSystems::
MeasureReferenceSystem
Point
Curve
Surface
GeometryCollection
2..*
+vertex
+element
0..*
LineString
Polygon
PolyhedralSurface
MultiSurface
MultiCurve
MultiPoint
0..*
+element
1..*
+patch
+ring
1..*
Line
LinearRing
Triangle
TIN
MultiPolygon
MultiLineString
+patch
1..*



+

=

PostGIS

Why PostGIS?

- **Open source**
- Follows the **Simple Features** for SQL specification from OCG
- **Widely adopted**
- **Flexible** and **powerful**
  - adds extra types (geometry, geography, raster and others)
  - adds functions, operators, and index enhancements.
- **Extensible**
- **Compatible** with a large ecosystem of GIS open source tools and libraries

PYTHON

CONNECT

LOAD

ESRI SHAPEFILE
OPENSTREETMAP
CSV (WKT)

EXPORT

ESRI SHAPEFILE
CSV (WKT)
GEOJSON

CREATE

SPATIAL TABLE
SPATIAL INDEX

QUERY

SPATIAL RELATIONS
SPATIAL JOIN

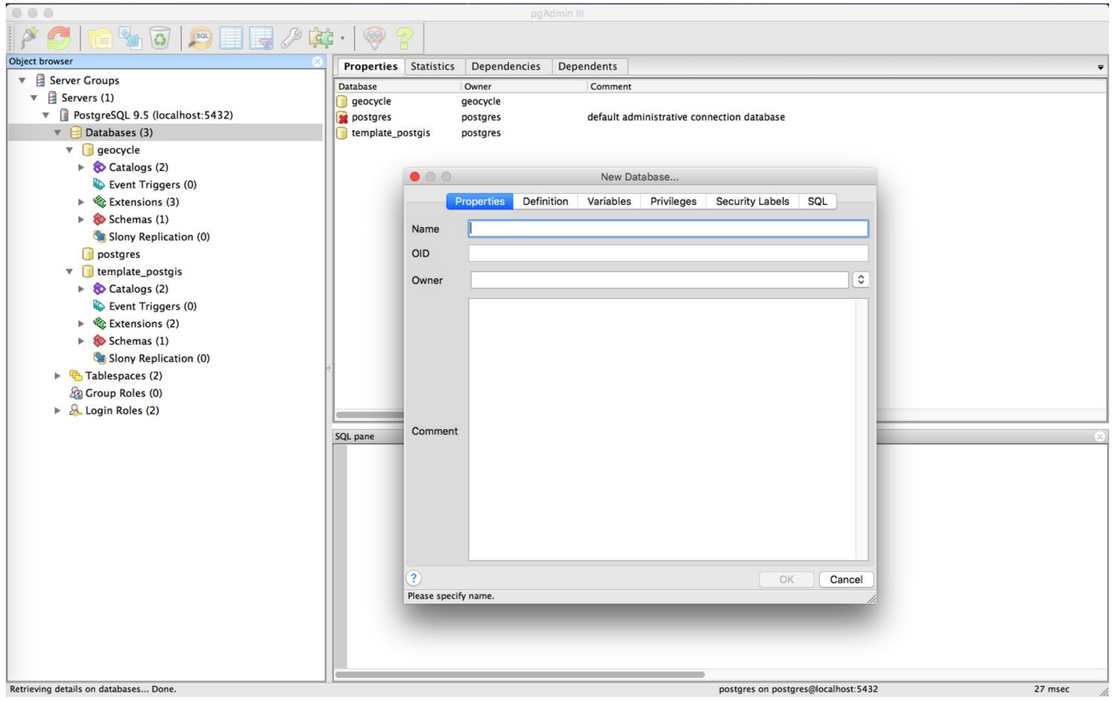
CREATE

SPATIAL TABLE
SPATIAL INDEX

Create a Spatial Database

```
CREATE DATABASE geocycle
TEMPLATE template_postgis;
# Or
CREATE DATABASE geocycle;
\connect geocycle
CREATE EXTENSION postgis;

# Check the installation
SELECT PostGIS_Full_Version();
```

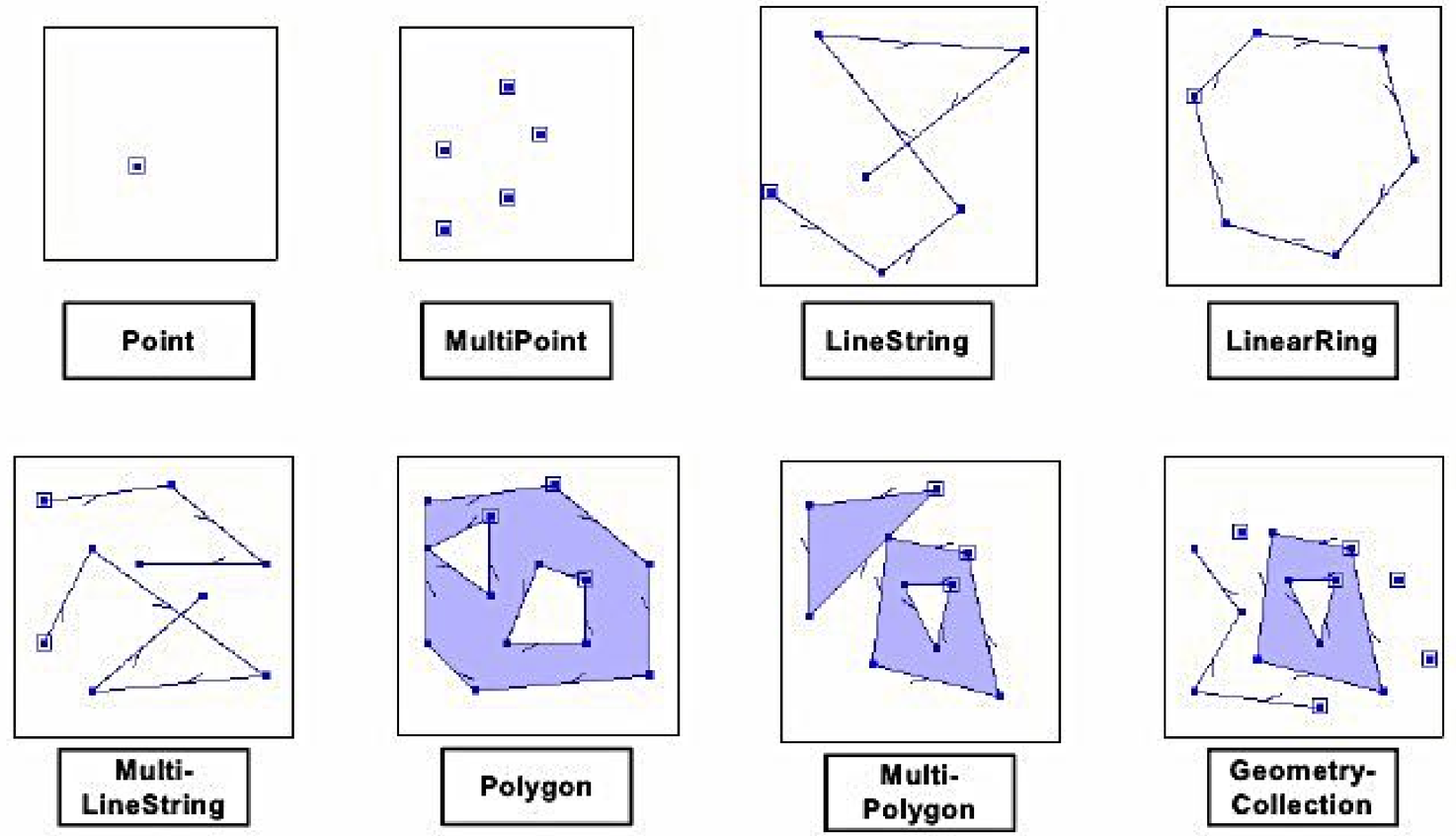


pgAdmin

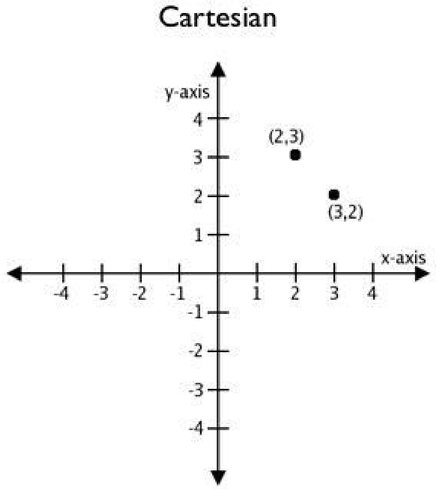
Extensions useful for Spatial Computation (some!)

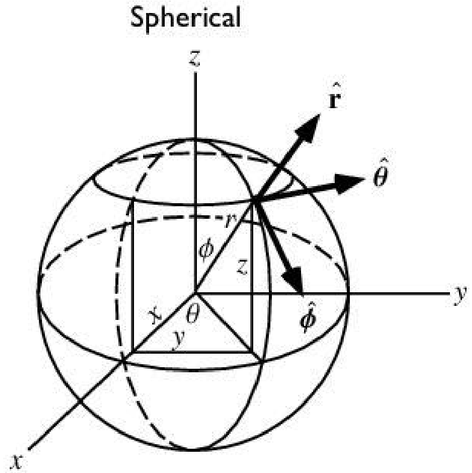
- **HSTORE**
  - storing sets of key/value pairs
- **ADDRESS_STANDARDIZER**
  - single line address parser that takes an input address and normalizes it based on a set of rules stored in a table and helper lex and gaz tables.
- **POSTGIS_TIGER_GEOCODER**
  - utilizes US Census Tiger data for geocoding
- **POSTGIS_TOPOLOGY**
  - Manage topological objects such as faces, edges and nodes
- **FUZZYSTRMATCH**
  - fuzzy string matching functions

Geometry Data Types



Geometry vs Geography





PostGIS supports **geography** type

GEOMETRY

Diverse representation
Diverse unit systems
(Default) planar measurements
Wide set of operations supported
Faster

GEOGRAPHY

WGS 84 lon lat degrees
Unit in meters
Geodetic measurements
Not all the operations supported
Slower

Create a Spatial Table

```
CREATE TABLE photos (
   pid int NOT NULL,
   geom geometry(Point, 4326));
```

Point

- specify the type of geometry (Point, Polygon, LineString, etc.)

4326

- SRID (Spatial Reference Identifier) of the geometry

Add Geometry Column

```
CREATE TABLE photos (
    pid int NOT NULL,
    longitude float8,
    latitude float8);

#Add a geometry column geom as a two dimensional point
SELECT AddGeometryColumn('photos', 'geom', 4326, 'POINT', 2);

#Convert longitude and latitude columns in a Point geometry
UPDATE instagram_photos SET geom = ST_SetSRID(ST_Point(longitude,
latitude), 4326);
```

Transform SRID

```
SELECT UpdateGeometrySRID('photos','geom',3157);

# The prior example is equivalent to this DDL statement

ALTER TABLE photos ALTER COLUMN geom
TYPE geometry(POINT, 3157) USING ST_SetSRID(geom, 3157);
```

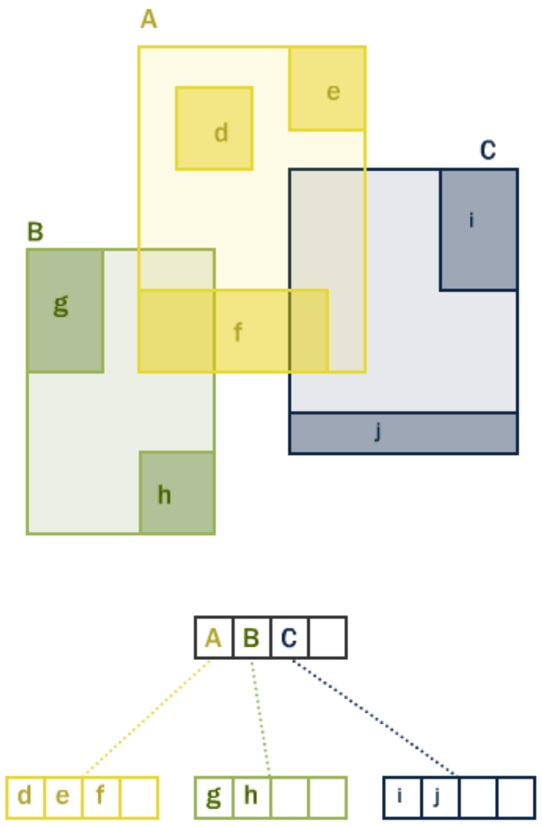
http://postgis.net/docs/UpdateGeometrySRID.html

Add Spatial Index

- **Indexes are what make using a spatial database for large data sets possible.**
- **PostgreSQL supports three kinds of indexes by default:**
  - B-Tree (default):
    - data which can be sorted along one axis
  - R-Tree:
    - break up data into rectangles, and sub-rectangles, etc
  - GiST (Generalized Search Trees):
    - indexes break up data into "things to one side", "things which overlap", "things which are inside" and can be used on a wide range of data-types

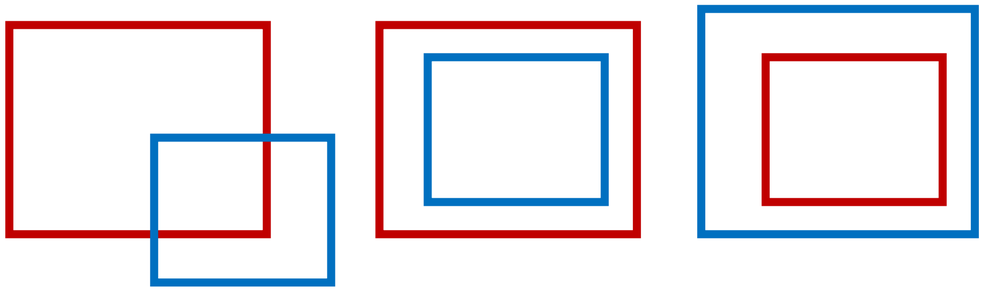
http://workshops.boundlessgeo.com/postgis-intro/indexing.html

R-tree Hierarchy

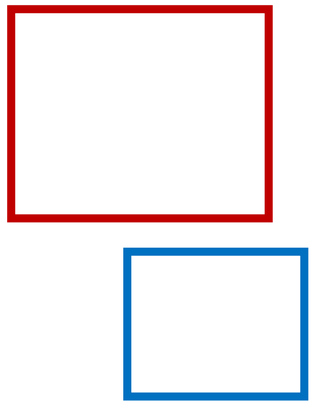


Using Spatial Indexes

Spatial index operator for intersection is “&&”



A && B = TRUE

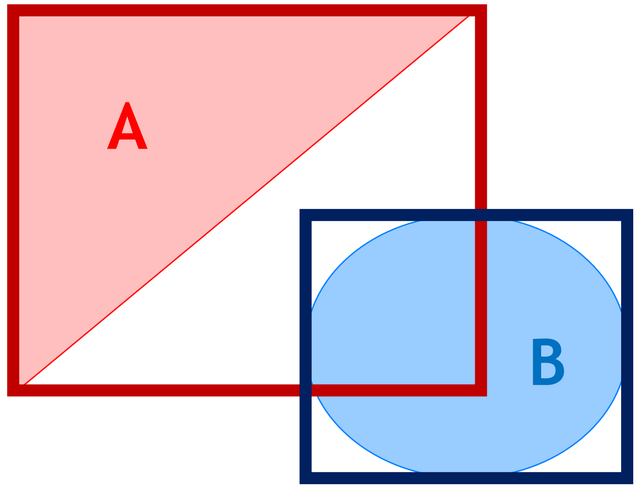


A && B = FALSE

SPATIAL INDEXES CONSIDER ONLY BOUNDING BOXES!

Using Spatial Indexes

- **Bounding Boxes are not enough!**
- **Two pass system required**
  - Use bounding boxes to reduce candidates
  - Use real topological test to get final answer
- Index operations (&&) are built into common functions for automatic use, but you can use them separately if you like.
  - ST_Intersects(G1,G2)
  - G1 && G2 AND _ST_Intersects(G1,G2)

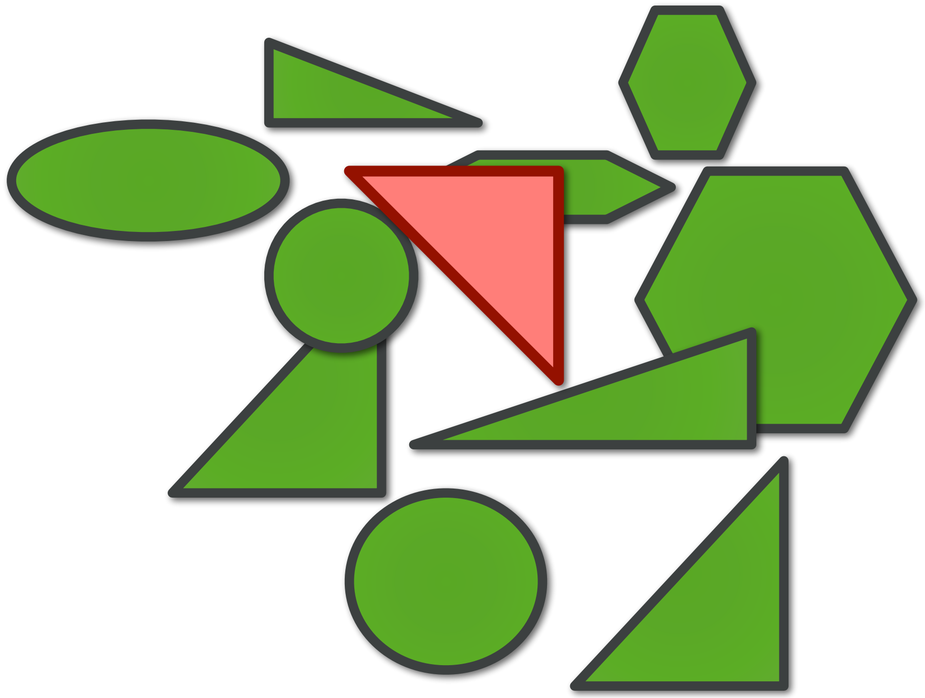


A && B = **TRUE**

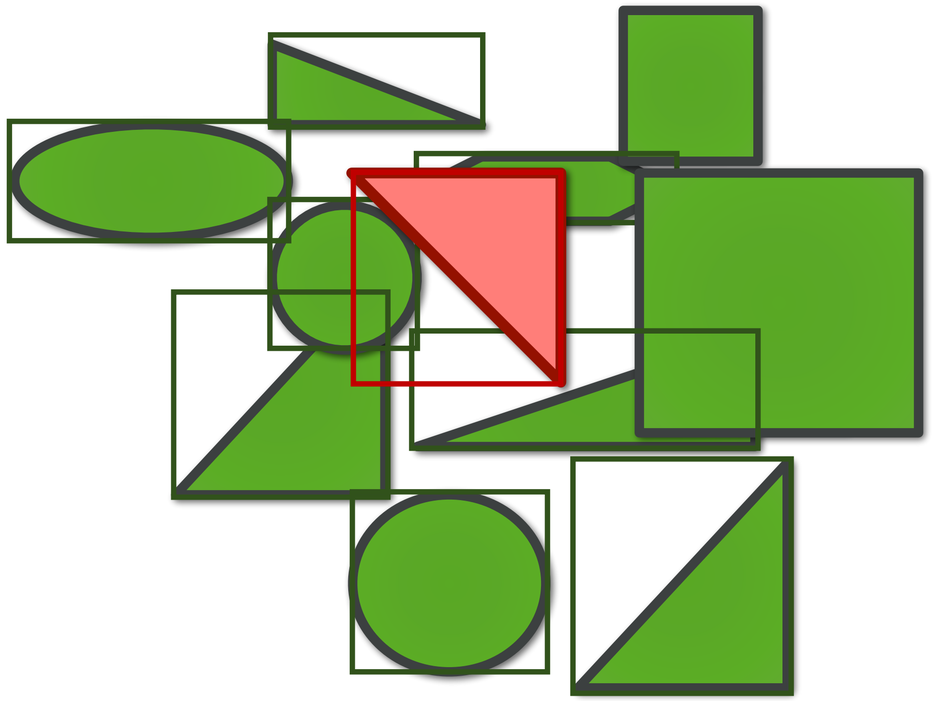
_ST_Intersects(A && B) = **FALSE**

* _ST_Intersects: geometric intersection

Using Spatial Indexes

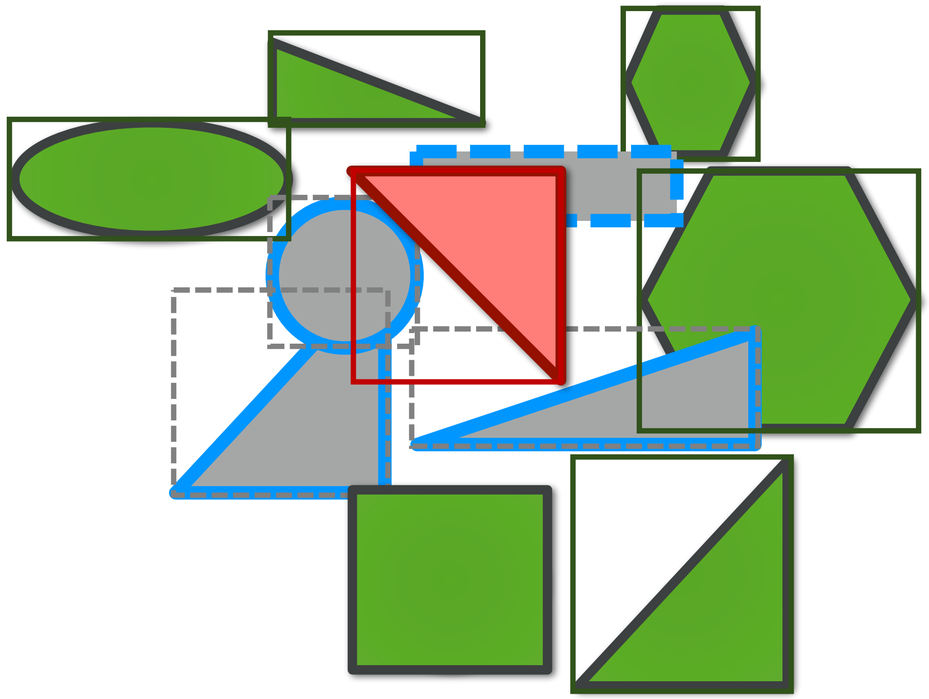


Using Spatial Indexes (first step)



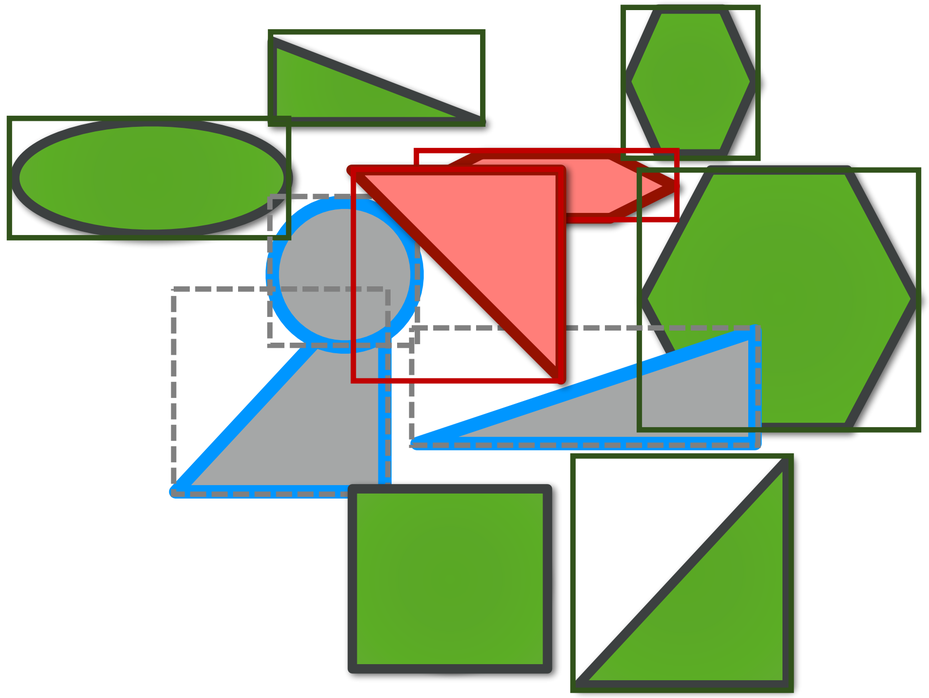
A && B

Using Spatial Indexes (first step)



A && B

Using Spatial Indexes (first step)



_ST_Intersects(A,B)
between candidates

Taking Advantage of Indexes

- **Remember that:**
  - Only the bounding-box-based operators such as && can take advantage of the GiST spatial index.
  - Functions such as ST_Distance() cannot use the index advantages
- **Example: “Find all geometries that are within 100 units from a point”**

```
SELECT geom FROM geom_table
WHERE ST_Distance(the_geom, ST_GeomFromText('POINT(100000 200000)',
312)) < 100
```

VERY INEFFICIENT!

Taking Advantage of Indexes

```
SELECT the_geom
FROM geom_table
WHERE ST_DWithin(the_geom, ST_MakeEnvelope(90900, 190900, 100100,
       200100,312), 100)
```

MUCH BETTER!

ESRI Shapefile
OpenStreetMap
CSV (WKT)

LOAD

CREATE

Spatial Table
Spatial Index

Loading Data Using SQL

Standard SQL INSERT statement

```
INSERT INTO roads (road_id, roads_geom, road_name)
VALUES (1, ST_GeomFromText('LINESTRING(191232 243118,191108 243242)',
 ◦         -1),'Jeff Rd');
```

Batch load

```
COPY roads FROM 'absolute_path_to_the_input_file.csv' CSV HEADER
ENCODING 'utf-8';
```

Load ESRI Shapefile

- **Command line tool shp2pgsql**
  - http://www.bostongis.com/pgsql2shp_shp2pgsql_quickguide.bqg
- **Example:**

```
shp2pgsql -I -d -s 102718:4326 -g geom data/shp/nyc_boroughs.shp
nyc_boroughs | psql -d geocycle -U geocycle
```

- **-d** Drops the table, then recreates it and populates it with current shape file data.
- **-I** Create a GiST index on the geometry column.
- **-s** from_srid:to_srid If -s :to_srid is not specified then from_srid is assumed and no transformation happens.
- -**g** Specify the name of the geometry column to be (S) created (P) exported.

Load GeoJSON to PostGIS

- **ST_GeomFromGeoJSON to bring in just the geometry part of the GeoJSON.**
- **ogr2ogr to import the entire GeoJSON document**
  - http://www.gdal.org/ogr2ogr.html

```
ogr2ogr -f "PostgreSQL" PG:"dbname=my_database user=postgres"
"source_data.json" -nln destination_table -append
```

- **ogr2ogr is a very powerful conversion tool that is part of the GDAL library (a must for many tasks!)**
- **ogrinfo provide information about a datasource (same family of ogr2ogr)**

Load OpenStreetMap data

Many tools available out there:

- **osmosis**
  - http://wiki.openstreetmap.org/wiki/Osmosis
- **osm2pgsql**
  - http://wiki.openstreetmap.org/wiki/Osm2pgsql
- **Imposm**
  - http://imposm.org/

Example:

```
osmosis --read-pbf ../../data/osm/nyc_poly_highways.osm.pbf --log-
progress --write-pgsql database=geocycle user=geocycle
password=geocycle
```

Use QGIS to import data or explore a spatial database

- **QGIS is a valid alternative to some of the importing tasks**
- **Many plugins for different data formats**
- **Ability to connect to postGIS databases**
  - Load layers
  - Manipulate layers
  - Export layers

ESRI Shapefile
OpenStreetMap
CSV (WKT)

LOAD

CREATE

Spatial Table
Spatial Index

QUERY

Spatial Relations
Spatial Join

Spatial Functions

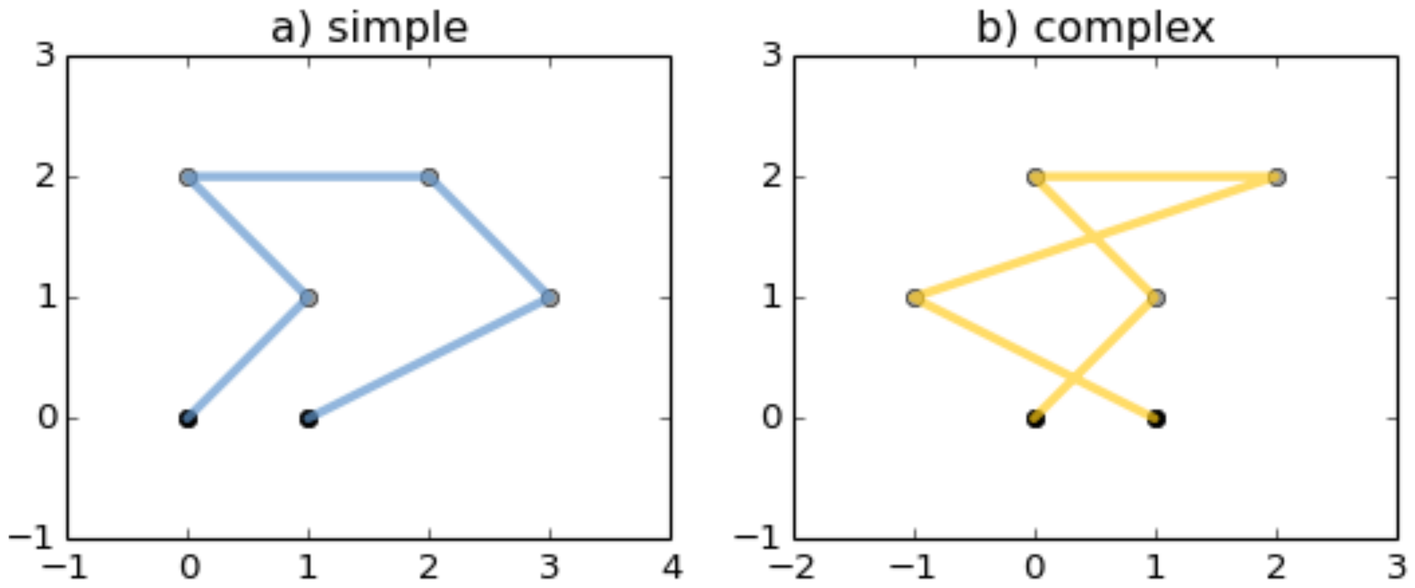
- **Constructors: use these functions to create geometries:**
  - ST_GeomFromText
  - ST_GeomFromWKT
  - ST_GeomFromWKB
- **Outputs: return a geometry representation in another industry-standard format**
  - ST_AsText
  - ST_AsWKT
  - ST_AsGeoJSON

Spatial Functions

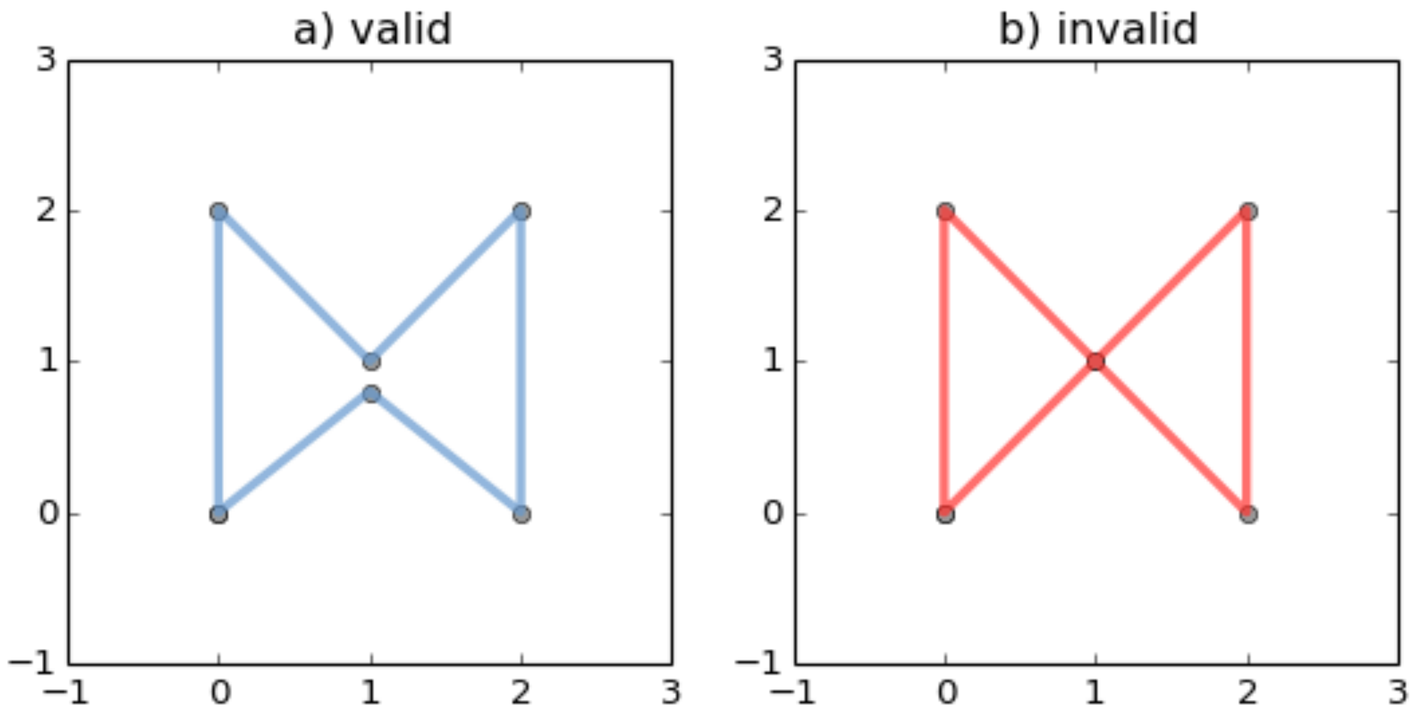
- **Accessor: getters and setters functions to read/modify properties.**
  - ST_SRID
  - ST_Transform
  - ST_GeometryType
  - ST_IsValid
  - ST_IsSimple
- **Measurement:**
  - ST_Length
  - ST_Area
  - ST_Perimeter
  - ST_Length_Spheroid

ST_IsSimple

- **Description of validity and simplicity in geometries**



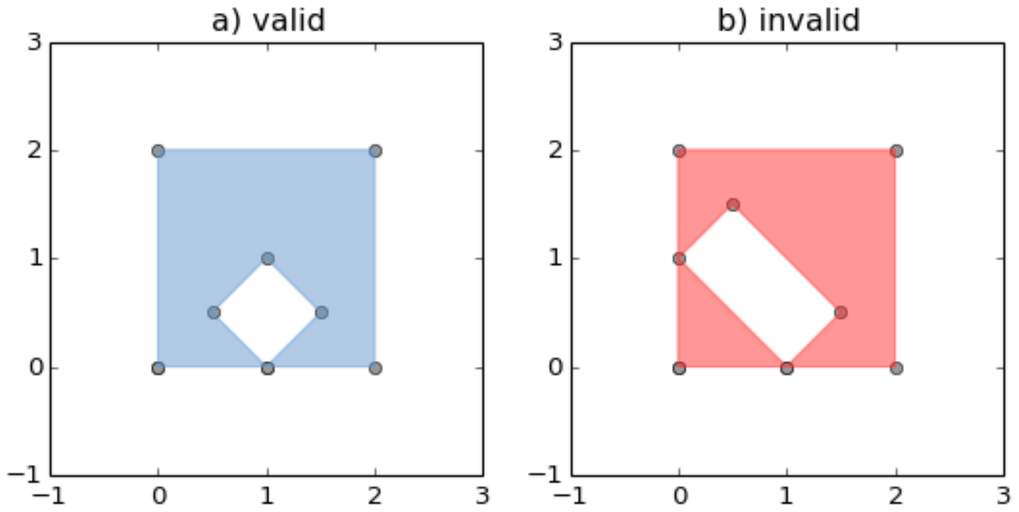
ST_IsValid



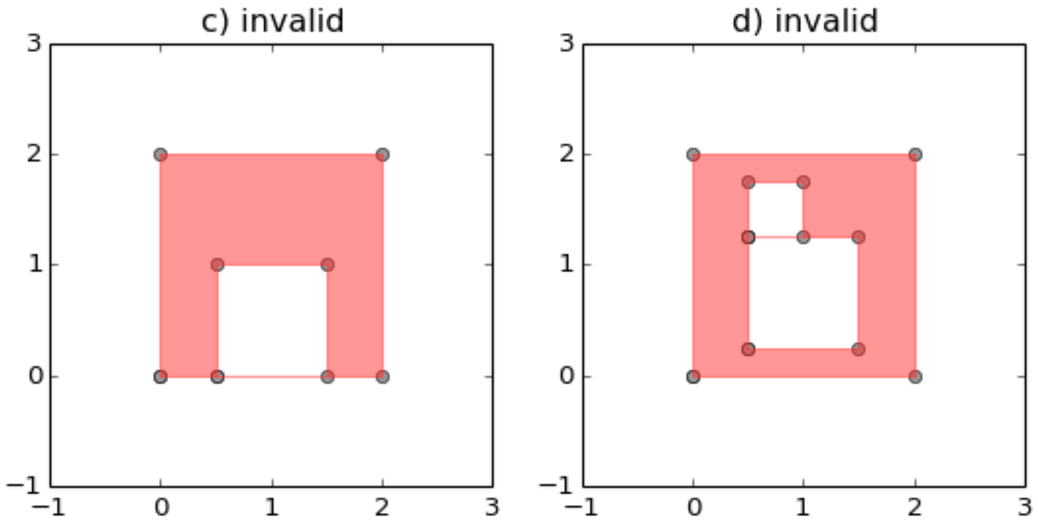
A valid LinearRing on the left, an invalid self-touching LinearRing on the right.

https://postgis.net/docs/using_postgis_dbmanagement.html#OGC_Validity

ST_IsValid



On the left, a valid Polygon with one interior ring that touches the exterior ring at one point, and on the right a Polygon that isinvalid because its interior ring touches the exterior ring at more than one point.



On the left, a Polygon that is invalid because its exterior and interior rings touch along a line, and on the right, a Polygon that is invalid because its interior rings touch along a line.

Spatial Functions

- **Decomposition**
  - Extract parts of an existing geometry. You may need to find the closed linestring that encloses a polygon or the multipoint that constitutes a linestring. We call functions that extract and return one or more geometries.
    - ST_Envelope
    - ST_X
    - ST_Y
    - ST_Boundary
    - ST_Centroid

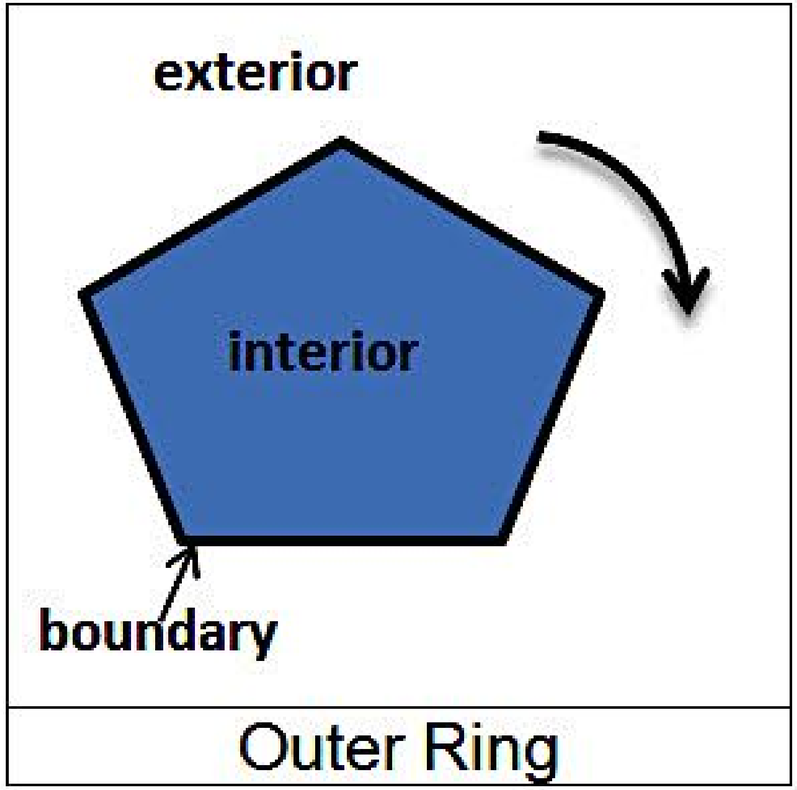
Spatial Functions

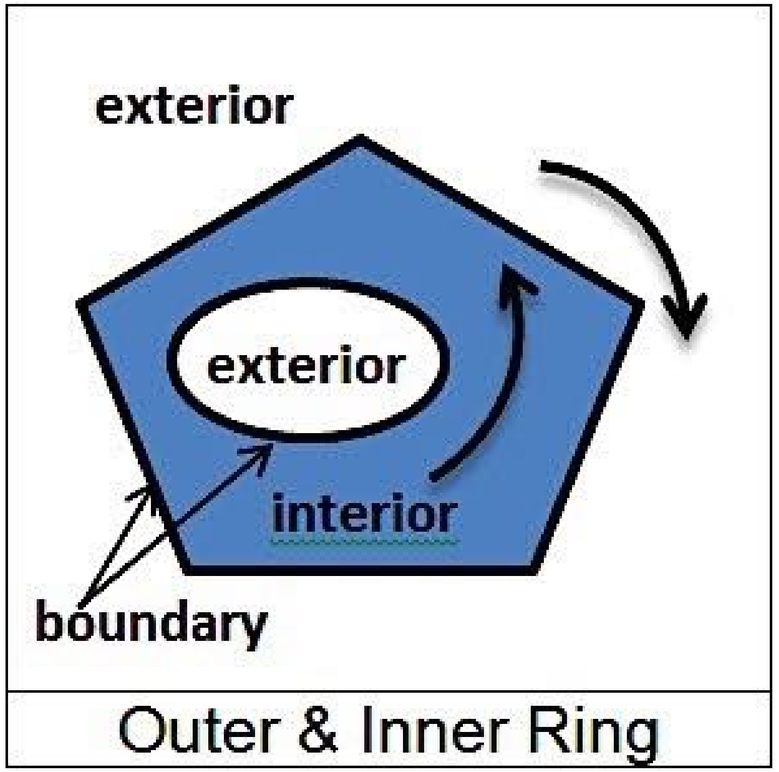
- **Composition:**
  - ST_MakePoint
  - ST_MakePolygon
  - ST_BuildArea
- **Simplification:**
  - ST_Simplify
  - ST_SimplifyPreserveTopology

Spatial Relationships Model

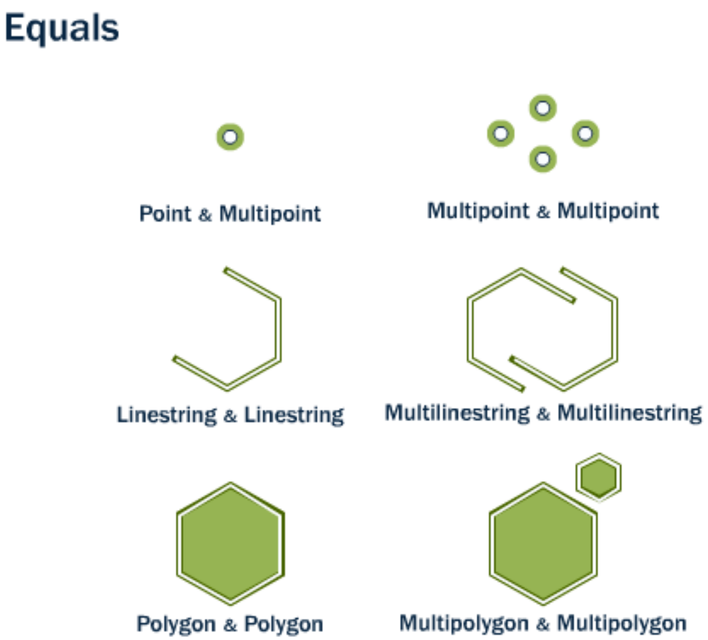
- **A geometry can be partitioned in :**
  - **Interior**: that portion of a geometry that’s inside the geometry and not on the boundary.
  - **Exterior**: the coordinate space outside a geometry but not including the boundary.
  - **Boundary**: the coordinate space neither interior nor exterior to the geometry. It’s the space that separates the interior from the exterior (the rest of the coordinate space).

Spatial Relationships Model

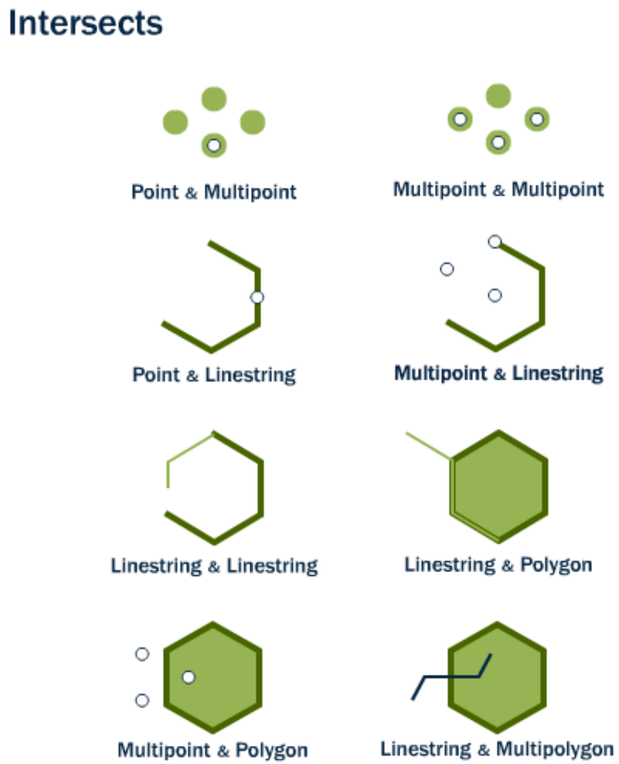




Spatial Relationships

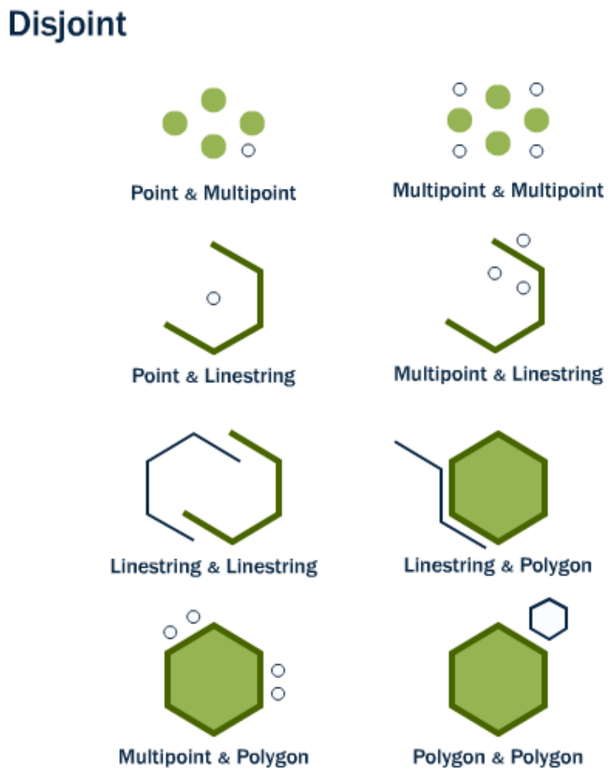


ST_Equals

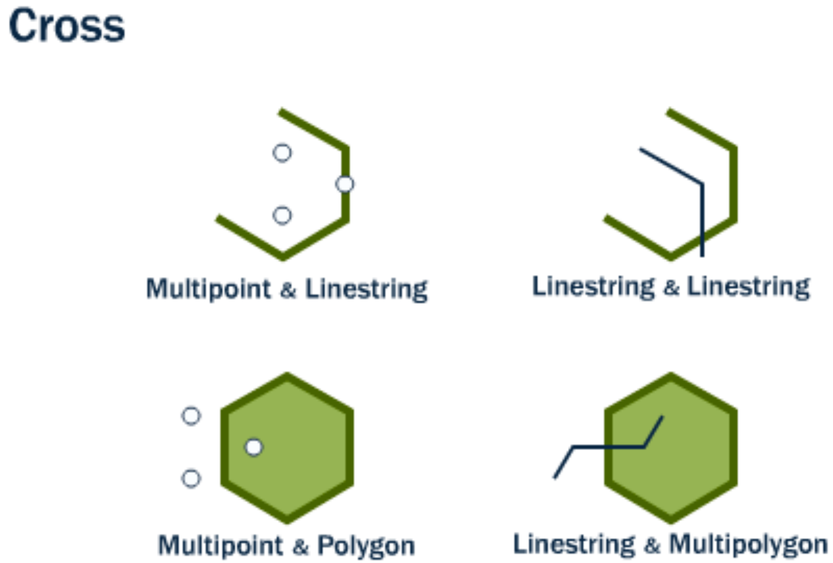


ST_Intersects

Spatial Relationships



ST_Disjoint

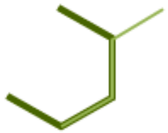


ST_Crosses

Spatial Relationships

Overlap

Multipoint & Multipoint

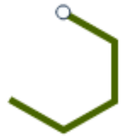


Linestring & Linestring

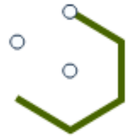
Polygon & Polygon

ST_Overlaps

Touch



Point & Linestring

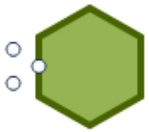


Multipoint & Linestring

Linestring & Linestring

Linestring & Polygon

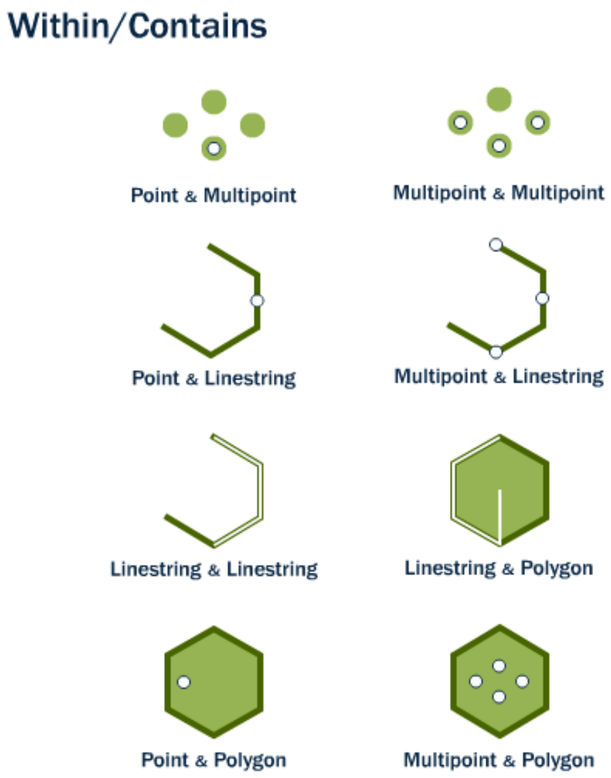
Point & Polygon



Multipoint & Polygon

ST_Touches

Spatial Relationships



ST_Within
ST_Contains

ST_DWithin

Spatial Relationships (Recap)

- **ST_Equals(geometry A, geometry B):** Returns true if the given geometries represent the same geometry. Directionality is ignored.
- **ST_Intersects(geometry A, geometry B):** Returns TRUE if the Geometries/Geography “spatially intersect” - (share any portion of space) and FALSE if they don’t (they are Disjoint).
- **ST_Overlaps(geometry A, geometry B):** Returns TRUE if the Geometries share space, are of the same dimension, but are not completely contained by each other.
- **ST_Touches(geometry A, geometry B):** Returns TRUE if the geometries have at least one point in common, but their interiors do not intersect.
- **ST_Within(geometry A , geometry B):** Returns true if the geometry A is completely inside geometry B

Spatial Relationships (Recap)

- **ST_Contains(geometry A, geometry B):** Returns true if and only if no points of B lie in the exterior of A, and at least one point of the interior of B lies in the interior of A.
- **ST_Crosses(geometry A, geometry B):** Returns TRUE if the supplied geometries have some, but not all, interior points in common.
- **ST_Disjoint(geometry A , geometry B):** Returns TRUE if the Geometries do not “spatially intersect” - if they do not share any space together.
- **ST_Distance(geometry A, geometry B):** Returns the 2-dimensional cartesian minimum distance (based on spatial ref) between two geometries in projected units.
- **ST_DWithin(geometry A, geometry B, radius):** Returns true if the geometries are within the specified distance (radius) of one another.

Operators

Operator	What is checks	Index
&&	Returns true if A's bounding box intersects B's	gist
&<	Returns true if A's bounding box overlaps or is to the left of B's.	gist
&<|	Returns true if A's bounding box overlaps or is below B's.	gist
&>	Returns true if A's bounding box overlaps or is to the right of B's	gist
<<	Returns true if A's bounding box is strictly to the left of B's	gist
<<|	Returns true if A's bounding box is strictly below B's	gist
=	Returns true if A's bounding box is the same as B's	B-tree
>>	Returns true if A's bounding box is strictly to the right of B's	gist
@	Returns true if A's bounding box is contained by B's	gist
|&>	Returns true if A's bounding box overlaps or is above B's	gist
|>>	Returns true if A's bounding box is strictly above B's	gist
~ -	Returns true if A's bounding box contains B's	gist
~=	Obsolete; superseded by ST_OrderingEquals	gist

Equality

- **Different kind of equalities**
- **Three basic kinds of equality are specific to geometries in PostGIS**
  - Spatial equality (occupying the same space)
    - ST_Equals
  - Geometric equality (same space, more or less same points and same point order, although with subtleties)
    - ST_OrderingEquals
  - Bounding box equality (the geometries have the same smallest box that can enclose them)
    - = operator

Querying spatial databases

- **Use SQL like for standard relational databases! ;)**
- **The biggest difference is that you can do spatial joins:**
  - Spatial joins are the bread-and-butter of spatial databases. They allow you to combine information from different tables by using spatial relationships as the join key. Much of what we think of as “standard GIS analysis” can be expressed as spatial joins.
- **We will do some examples in the practical session.**
- **Examples:**
  - How many Instagram pictures has been taken in each NYC borough during March 2015?
  - Which are the 10 most dangerous streets in NYC?

ESRI Shapefile
OpenStreetMap
CSV (WKT)

LOAD

EXPORT

ESRI Shapefile
CSV (WKT)
GeoJSON

CREATE

Spatial Table
Spatial Index

QUERY

Spatial Relations
Spatial Join

Export to ESRI Shapefile

- **Command line pgsql2shp**
  - http://www.bostongis.com/pgsql2shp_shp2pgsql_quickguide.bqg

```
pgsql2shp -f <path to output shapefile> -h <hostname> -u <username> -P
<password> databasename "<query>" –g geom_column
```

- **You can also use the ogr2ogr utility.**
- **NB: you can use the command COPY to export a table to a CSV file**
  - Example:

```
COPY products_273 TO '/tmp/products_199.csv' DELIMITER ',' CSV HEADER;
```

Export data to GeoJSON

- **Again you can use the ogr2ogr tool**
  - http://www.postgresonline.com/journal/archives/267-Creating-GeoJSON-Feature-Collections-with-JSON-and-PostGIS-functions.html
  - https://www.mapbox.com/help/postgis-data-studio-format/
- **Example:**

```
ogr2ogr -f GeoJSON -t_srs EPSG:4326 instagram_photos.geojson
"PG:host=localhost dbname=geocycle user=geocycle" -sql "select * from
instagram_photos";
```

Export to GeoJSON

- **Alternatively, you can export as a shapefile and then use ogr2ogr to convert to GeoJSON and apply some additional filters:**

```
ogr2ogr -f GeoJSON -t_srs EPSG:4326
-lco COORDINATE_PRECISION=5
-simplify tolerance 0.5
nyc_census_tracts_manhattan.geojson nyc_census_tracts.shp
-SQL "SELECT boroname, Shape_Leng as perimeter, Shape_Area as area FROM
nyc_census_tracts WHERE BORONAME='Manhattan'”
```

- **-lco COORDINATE_PRECISION: for space reason, you might want to round coordinates**
- **-simplify tolerance: distance tolerance for simplification preserving topology**

PYTHON

CONNECT

LOAD

ESRI SHAPEFILE
OPENSTREETMAP
CSV (WKT)

EXPORT

ESRI SHAPEFILE
CSV (WKT)
GEOJSON

CREATE

SPATIAL TABLE
SPATIAL INDEX

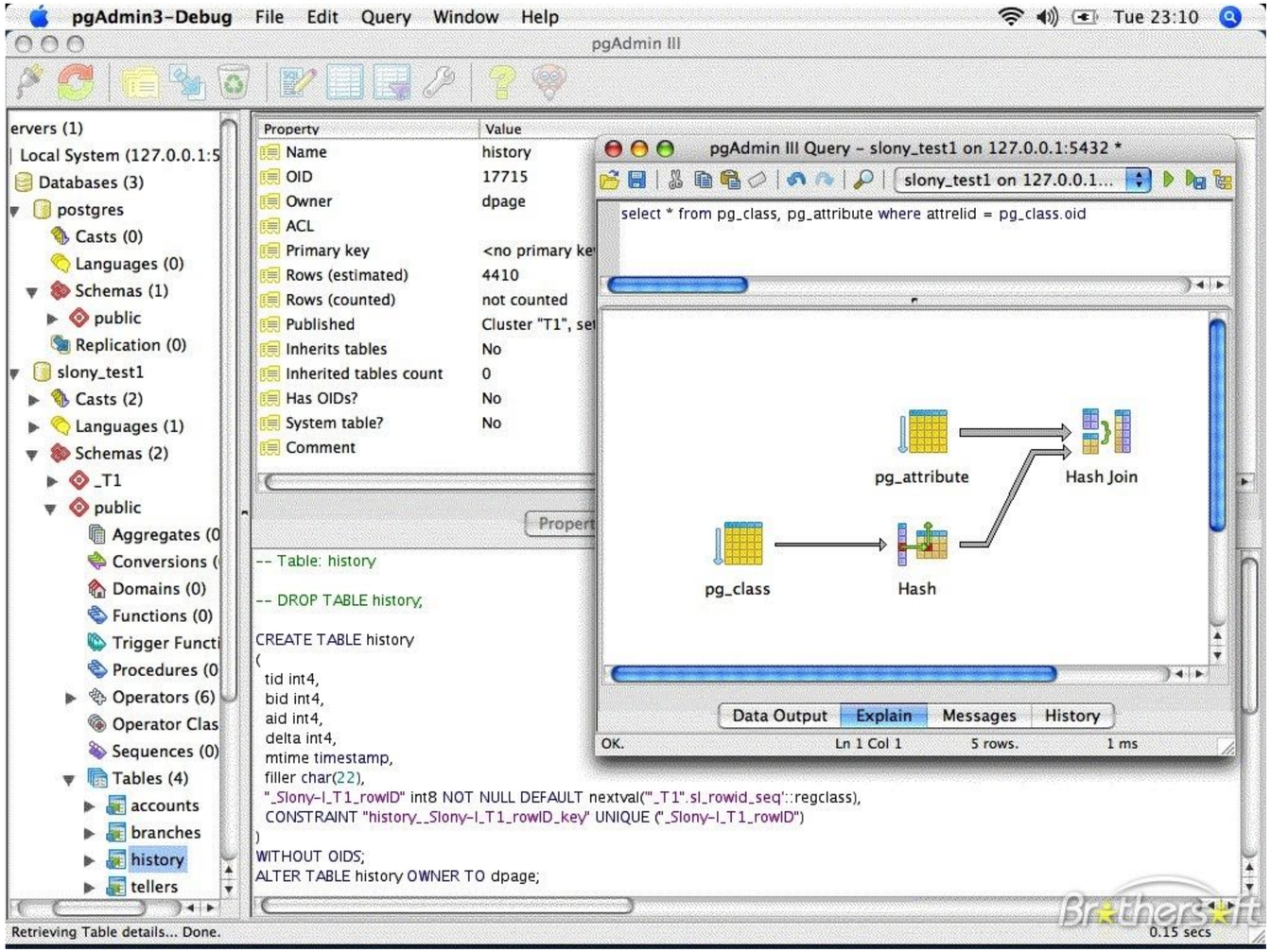
QUERY

SPATIAL RELATIONS
SPATIAL JOIN

Python for Spatial Computation

- Connect to a PostGIS database from a **Python script**
- Create and manipulate geometries with **Shapely**
- **Load** external data
- Load **ESRI shapefiles** with **Fiona**
- Load textual data in **WKT**
- **Spatial functions**
- **Indexing** and **prepared geometries**

Using PostGIS in a Desktop Environment



Using PostGIS in a Desktop Environment

QGIS is a cross-platform free and opensource desktop geographic information system (GIS) application that provides data viewing, editing, and analysis.

Exercise 1

- **In Exercise 1 you will learn how to**
  - Create a spatial database
  - Create a table with spatial columns
  - Create indexes on spatial columns
  - Load data into a table from
    - ESRI Shapefile
    - OSM metro extract
    - CSV
  - Transform SRID of a geometry column

Exercise 2

- **In Exercise 2 you will learn how to:**
  - Make simple spatial queries
  - Join spatial tables and different geometry types in complex queries
  - Connect a spatial database to a python script
  - Make spatial queries in a python script

Exercise 3

- **In Exercise 3 you will learn how to:**
  - Create a geometry in pure Python
  - Load data from a shapefile, CSV datasource
  - Perform transformations and basic spatial functions
  - Boost the performance: indexing and prepared geometries

@rschifan

schifane@di.unito.it

http://www.di.unito.it/~schifane