

# PostGIS - Gérer des données géographiques dans PostgreSQL

Vandy Berten – Webinar 27/09/2022



**Innovation with  
new technologies**



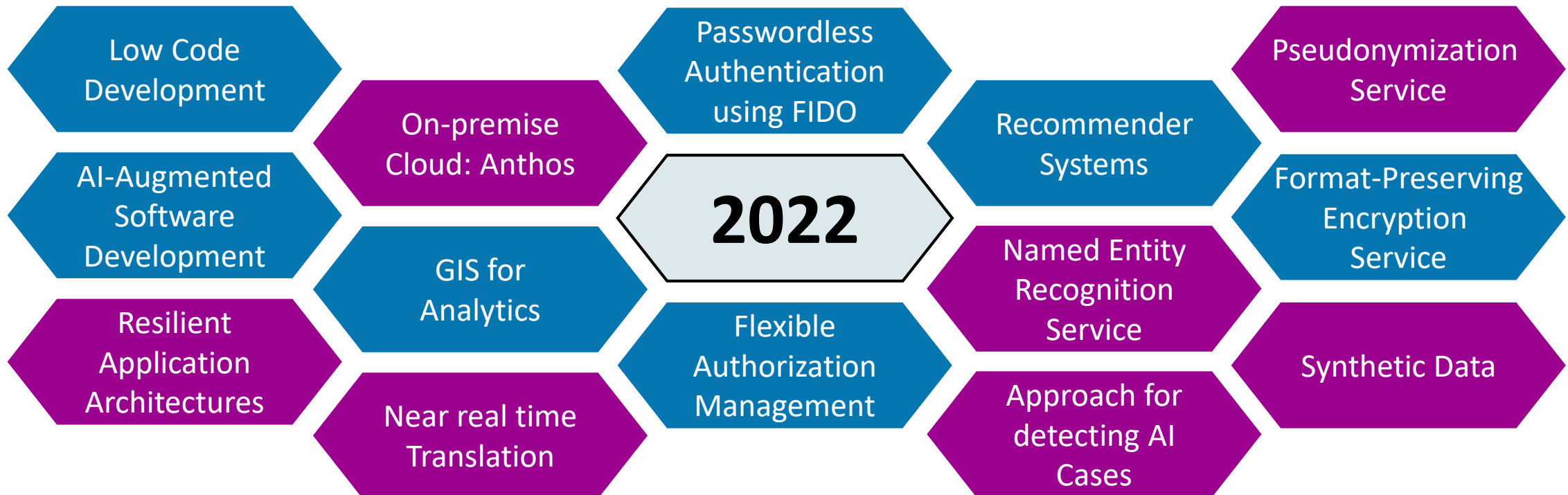
**Consultancy  
& expertise**



**Internal & external  
knowledge transfer**



**Support for  
going live**



# Content

## Context

- What is PostGIS, what kind of data
- Data source

## PostGIS queries

- Basic queries
- Spatial join

## Data load

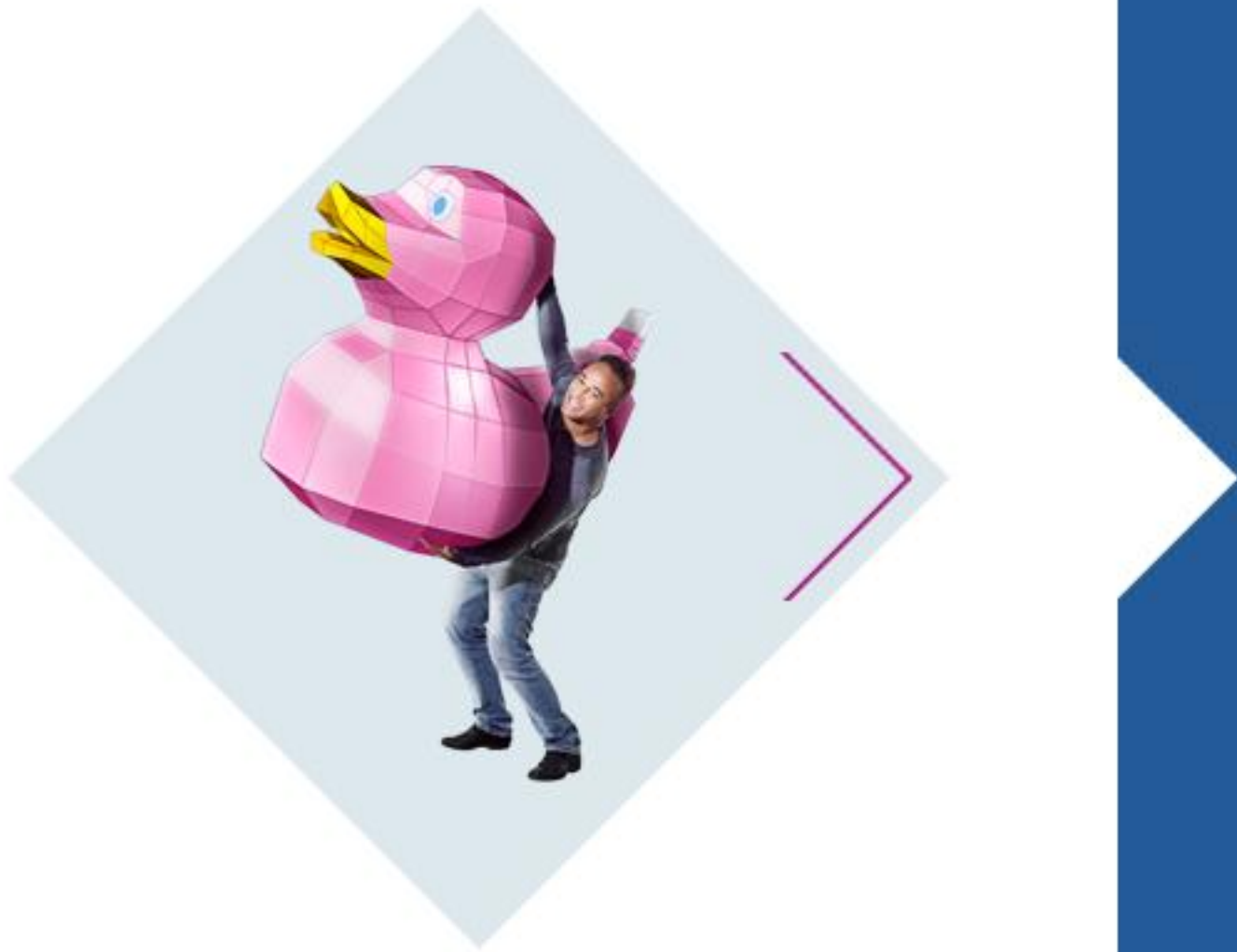
- How to feed PostGIS

## Indexing

- How to speed up queries

Previous webinar: <https://www.smalsresearch.be/webinar-gis-analytics-follow-up/> (05-05-2022)

Python notebook : <https://github.com/SmalsResearch/GISAnalytics/> → Webinar\_PostGIS



## Context & data

# PostGIS



- PostGIS (<https://postgis.net/>) is an extension (plugin) of PostgreSQL
- **P**ostgreSQL + **G**eographical **I**nformation **S**ystem

## • Add new (spatial) data types :

- POINT (+MULTIPOINT)



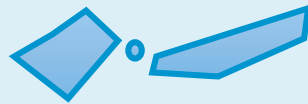
- LINESTRING (+MULTILINESTRING)



- POLYGON (+MULTIPOLYGON)



- GEOMETRYCOLLECTION



## • Add set of new (spatial) operations “ST\_” (spatial type):

- Comparison: ST\_Contains, ST\_Within, ST\_Intersects
- Aggregation: ST\_Union, ST\_Collect
- Combinations: ST\_Intersection, ST\_Difference
- Measurement: ST\_Distance, ST\_Area
- Projection handling: ST\_Transform
- ...

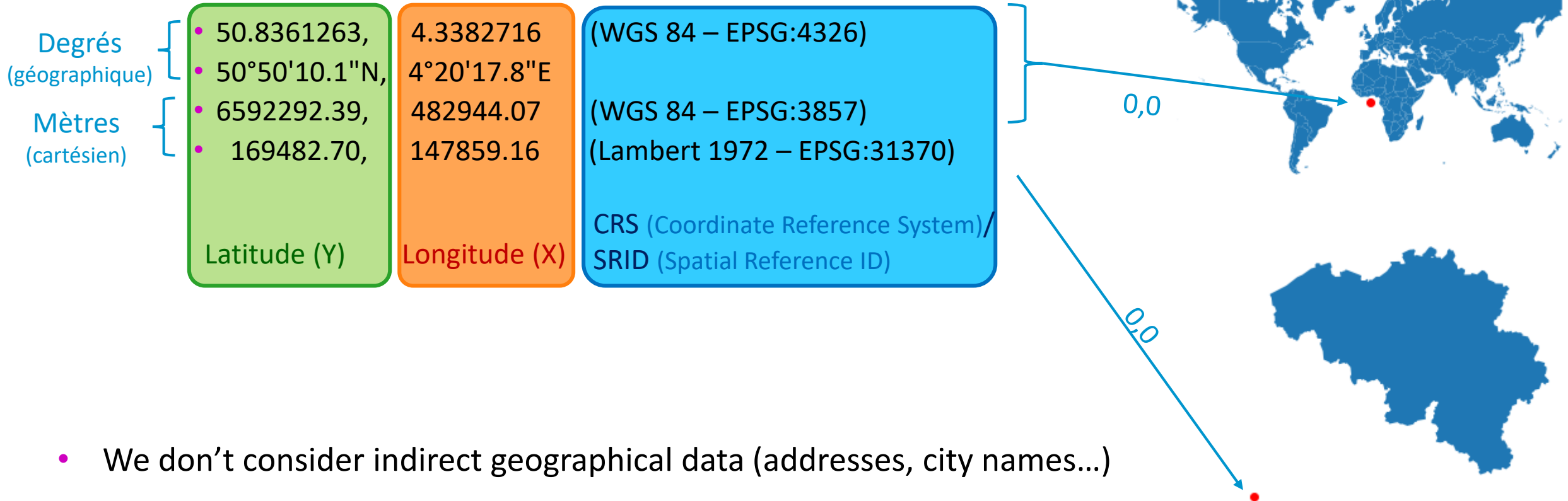
- Add a new (set of) spatial index type(s)

## (Minimal) Installation:

- Install PostgreSQL
- Install PostGIS binary (or build sources):  
sudo apt-get install postgresql-14-postgis-3
- **CREATE EXTENSION postgis;**

# Geographical data

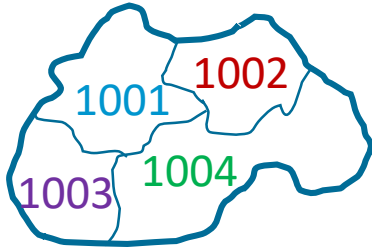
- Geographical point :



- We don't consider indirect geographical data (addresses, city names...)

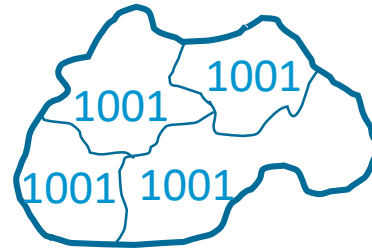
# Belgium division schemes – Postal scheme

- Postal schemes (BPost): main commune – localities



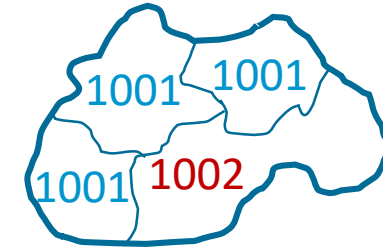
Main commune : Bruxelles(-Ville)

Zipcode	Locality
1000	Bruxelles
1020	Laeken
1120	Neder-Over-Hembeek
1130	Haren



Main commune : Jemeppe-Sur-Sambre

Zipcode	Locality
5190	Jemeppe-sur-Sambre
5190	Spy
5190	Onoz
5190	Balâtre
5190	...



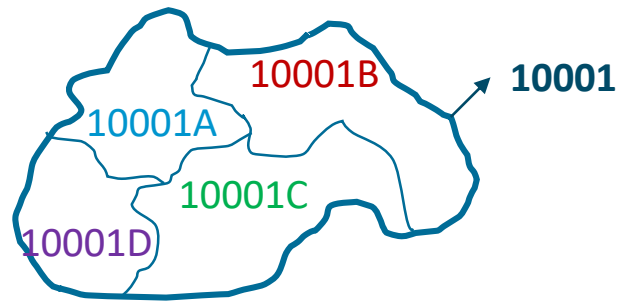
Main commune : Assesse

Zipcode	Locality
5330	Assesse
5330	Mailen
5330	Sart-Bernard
5332	Crupet
5333	...

- No (numerical) ID for a commune (only « commune name ») → 581 values
- Locality ID : Zipcode + locality name
- Lots of « special codes » (1110 NATO, 1044 RTBF, ...)

# Belgium division schemes – statistical scheme

- Statistical scheme (Statbel/Cadastre – SPF Economy):
  - Main commune (NIS5) → 581 values
  - Sections (NIS6)



- No 1-1 mapping Zipcode-NIS5 (or NIS6)
- No special code

Main commune : Bruxelles(-Ville)

NIS5		NIS6
21004	BRUXELLES PENTAGONE	21004A
21004	BRUXELLES-RUE DE LA LOI	21004B
21004	BRUXELLES-LOUISE	21004C
21004	BRUXELLES-CHAUSSEE D'ANVERS	21004D
21004	BRUXELLES-LAEKEN	21004E
21004	BRUXELLES-NEDER-OVERHEEMBEEK	21004F
21004	BRUXELLES-HAREN	21004G

- In our demo: do they agree at the commune level?

# Data sources

- We will load 4 OpenData sources:

Subject	Source	Shapes	Format	CRS	URL
Zipcode list (mapping zipcode-main commune)	BPost	Non geo	Excel	[NA]	<a href="https://www.bpost2.be/(...)/zipcodes_alpha_fr_new.xls">https://www.bpost2.be/(...)/zipcodes_alpha_fr_new.xls</a>
Zipcode boundaries	BPost	Polygons	Shapefile	3812	<a href="https://bgu.bpost.be/(...)_x-shapefile_3812.zip">https://bgu.bpost.be/(...)_x-shapefile_3812.zip</a>
Zipcode centers	Agence du Numérique	Points	GeoJSON	4326	<a href="https://www.odwb.be/(...)/code-postaux-belge/exports/geojson">https://www.odwb.be/(...)/code-postaux-belge/exports/geojson</a>
NIS code boundaries (statistical code level)	Statbel	Polygons	Shapefile	31370	<a href="https://statbel.fgov.be/(...)/sh_statbel_statistical_sectors_31370_20200101.shp.zip">https://statbel.fgov.be/(...)/sh_statbel_statistical_sectors_31370_20200101.shp.zip</a>

↳ To be standardized!!

- In DBEaver:

The screenshot shows the DBEaver interface. On the left, a table with columns 'zipcode' and 'geometry' is displayed. The 'geometry' column contains POLYGON coordinates. Row 45 is highlighted. On the right, a map of Belgium is shown with a blue polygon highlighting a specific area in the Brussels region, corresponding to the highlighted row in the table. The map includes labels for 'Audergem - Oudergem', 'Watermael - Watermaal', and 'Tervuren'. The status bar at the bottom indicates '200 Rows: 1 200 row(s) fetched - 2.658s (375ms fetch), on Aug 22, 14:56:23'.



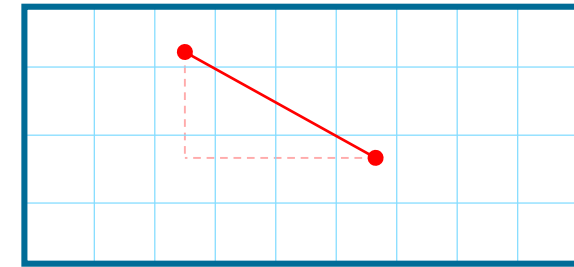
## Basic functions

# Simple distance

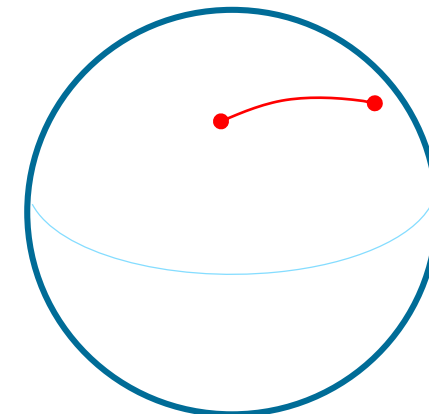
```
SELECT a.locality,  
       b.locality,  
       ST_Distance(a.geometry, b.geometry),  
       ST_DistanceSphere(a.geometry, b.geometry)  
FROM gistest.zipcodes_centers a,  
     gistest.zipcodes_centers b  
WHERE a.zipcode='1160' AND b.zipcode='5190'
```





Geometric Euclidian distance  
(ok only for metric CRS)

$$0,43783 = \sqrt{(4.433 - 4.995)^2 + (50.815 - 50.828)^2}$$



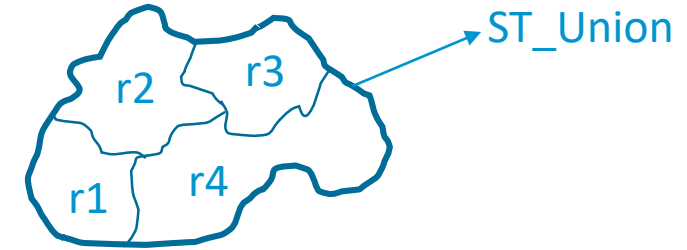
Geographical distance (in meters)



	ABC locality 	ABC locality 	123 st_distance 	123 st_distancesphere 
1	Auderghem	Moustier-Sur-Sambre	0.4378300821	43,089.00047741
2	Auderghem	Saint-Martin	0.3807369552	38,121.84729085
3	Auderghem	Balâtre	0.3778559061	38,134.34560066
4	Auderghem	Onoz	0.4006119939	39,630.80735948
5	Auderghem	Jemeppe-Sur-Sambre	0.4164304942	42,209.58949063
6	Auderghem	Ham-Sur-Sambre	0.4421927051	44,618.0699368
7	Auderghem	Mornimont	0.4508334992	44,379.13904389
8	Auderghem	Spy	0.4295900467	41,768.06793197

# Aggregation

```
SELECT c_provi,  
       t_provi_fr,  
       t_regio_fr,  
       ST_Union(geometry) as geometry  
FROM gistest.statistical_sectors  
GROUP BY c_provi, t_provi_fr, t_regio_fr
```



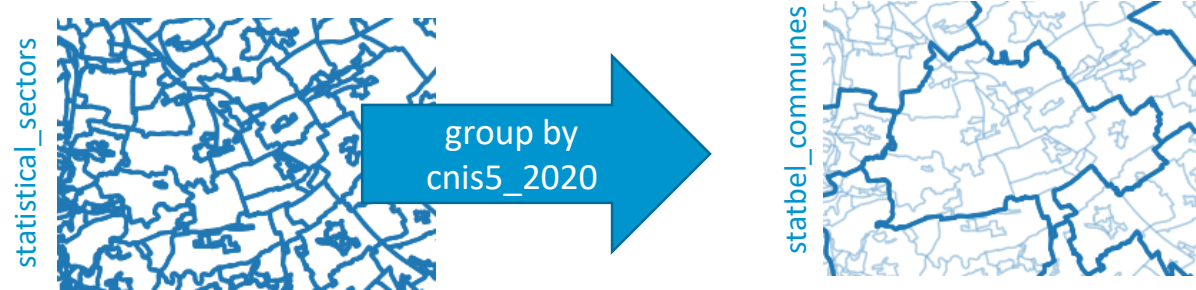
statistical\_sectors 1 x

SELECT c\_provi, t\_provi\_fr, t\_regio\_fr, ST\_Union(geometry) as geometry FROM

	ABC c_provi	ABC t_provi_fr	ABC t_regio_fr	geometry
1	10000	Province d'Anvers	Région flamande	MULTIPOLYGON Z(((4.935794074100193 51.409299441363345 0,
2	-	[NULL]	Région de Bruxelles-Capitale	POLYGON Z((4.297395339460635 50.81001442302577 0, 4.29735
3	90000	Province de Namur	Région wallonne	POLYGON Z((4.513943097267026 49.94891534289222 0, 4.51384
4	20001	Province du Brabant flamand	Région flamande	POLYGON Z((5.107842576285907 50.722848094043826 0, 5.1073
5	40000	Province de Flandre orientale	Région flamande	POLYGON Z((3.4916480229646765 50.757922055392285 0, 3.491
6	30000	Province de Flandre occidentale	Région flamande	POLYGON Z((2.842299598820434 50.73559228368382 0, 2.84249
7	50000	Province du Hainaut	Région wallonne	MULTIPOLYGON Z(((2.911669461934257 50.70399969450178 0,
8	80000	Province du Luxembourg	Région wallonne	POLYGON Z((5.0124933683789274 49.99737467979133 0, 5.0124
9	20002	Province du Brabant wallon	Région wallonne	POLYGON Z((4.145748827328134 50.636597982452905 0, 4.1457
10	60000	Province de Liège	Région wallonne	POLYGON Z((5.351476312487963 50.38838397246115 0, 5.35143
11	70000	Province du Limbourg	Région flamande	MULTIPOLYGON Z(((5.895636190850145 50.745452548013176 0,

# Aggregation – commune boundaries

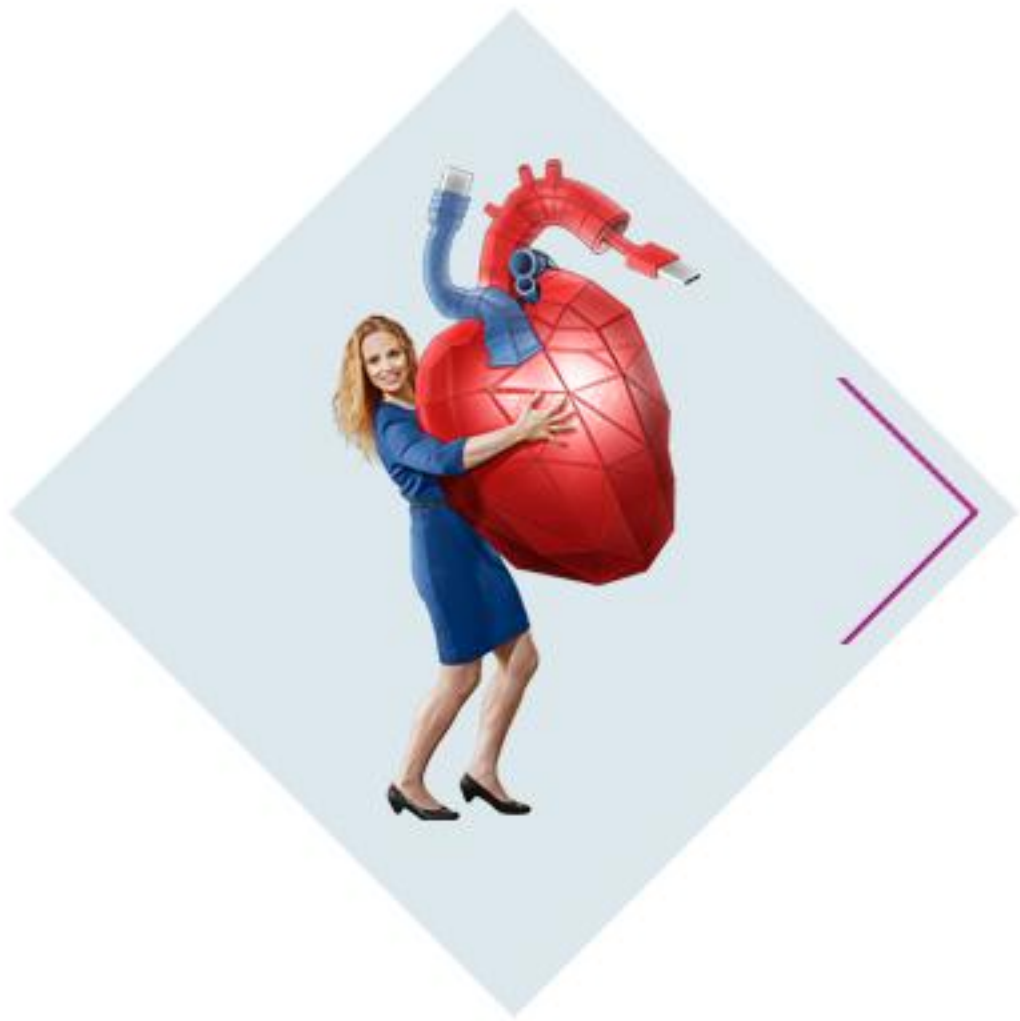
```
-- Create a table with NIS boundaries
CREATE TABLE gistest.statbel_communes AS
-- or: CREATE VIEW
SELECT cnis5_2020, -- Code NIS5
        t_mun_fr, -- Commune name
        ST_Union(geometry) AS geometry
FROM gistest.statistical_sectors
GROUP BY cnis5_2020, t_mun_fr, t_mun_nl
```



```
-- Create a table with postal boundaries
-- Assuming 'main_commune2' has been filled
-- for special zipcodes (see notebook for details)
```

```
CREATE TABLE gistest.bpost_communes AS
SELECT main_commune2 as main_commune,
        ST_Union(geometry) AS geometry
FROM gistest.zipcodes zp
JOIN gistest.zipcodes_boundaries zp_bnd
      ON zp.zipcode = zp_bnd.zipcode
GROUP BY main_commune2
```

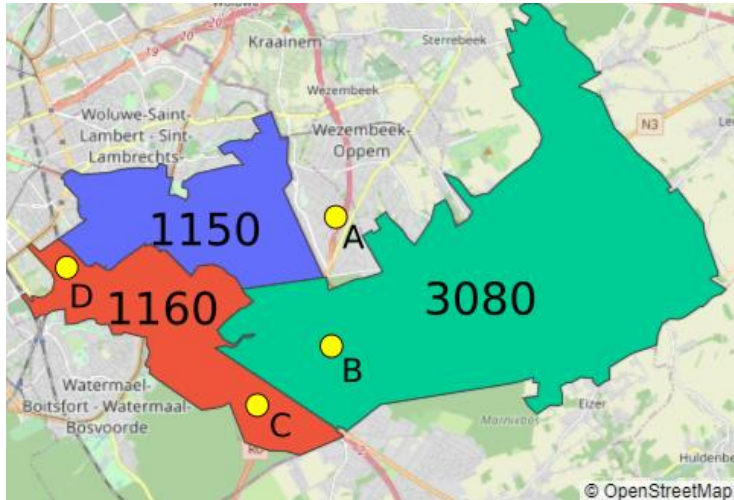




## Spatial join

# Spatial join

“Classical” join:



observations

ID	ZIPCODE
A	
B	3080
C	1160
D	1160
...	...

zipcodes

ZIPCODE	PROVINCE
1150	Bruxelles-Capitale
1160	Bruxelles-Capitale
3080	Brabant Flamand
...	...

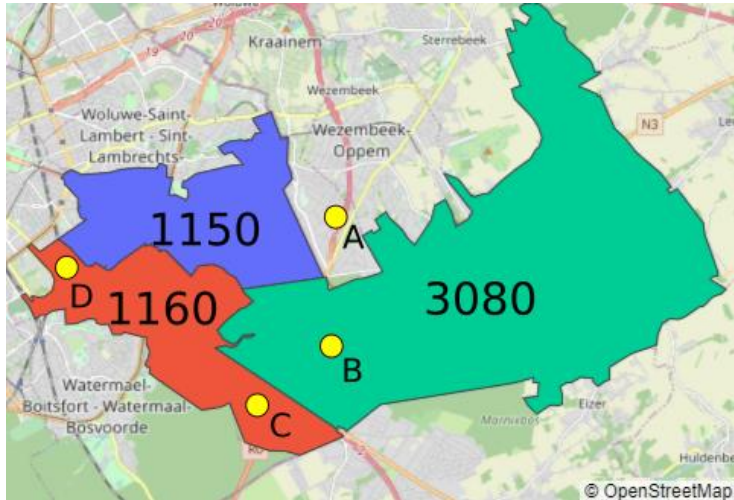
```
SELECT o.ID, o.zipcode, z.province
FROM observations o
LEFT JOIN zipcodes z
ON o.zipcode = z.zipcode
```



ID	ZIPCODE	PROVINCE
A		
B	3080	Brabant Flamand
C	1160	Bruxelles-Capitale
D	1160	Bruxelles-Capitale
...	...	...

# Spatial join

Spatial join:



observations

ID	GEOM
A	point( $x_a$ , $y_a$ )
B	point( $x_b$ , $y_b$ )
C	point( $x_c$ , $y_c$ )
D	point( $x_d$ , $y_d$ )
...	...

provinces

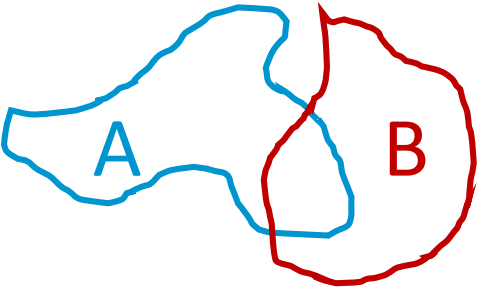







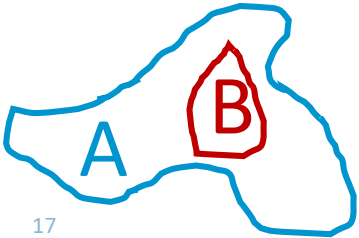



PROVINCE	GEOM
Bruxelles-Capitale	polygon( $pt_1$ , $pt_2$ , ...)
Brabant Flamand	polygon(...)
Brabant Wallon	polygon(...)
...	...

```
SELECT o.ID, p.province
FROM observations o
LEFT JOIN provinces p
ON ST_Contains (p.geom, o.geom)
```

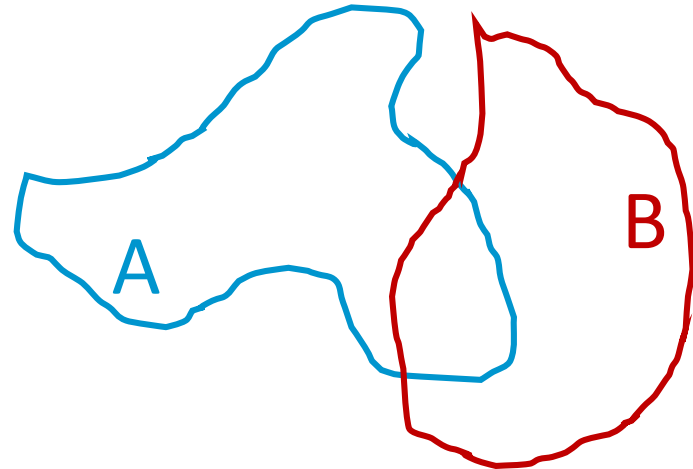


ID	PROVINCE
A	Brabant Flamand
B	Brabant Flamand
C	Bruxelles-Capitale
D	Bruxelles-Capitale
...	...

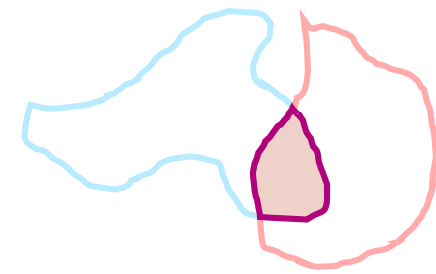
# Spatial operators (boolean)

	<code>ST_Intersects(A, B)</code>	<code>ST_Touches(A, B)</code>	<code>ST_Contains(A, B)</code> <code>= ST_Within(B, A)</code>
			
			
			

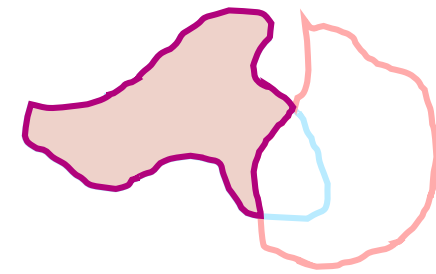
# Spatial operators (constructive)



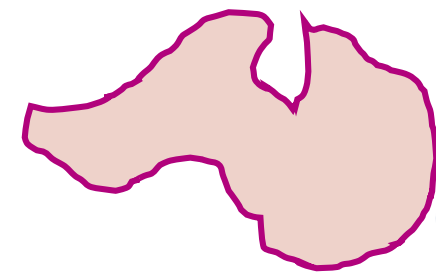
`ST_Intersection(A, B) :`



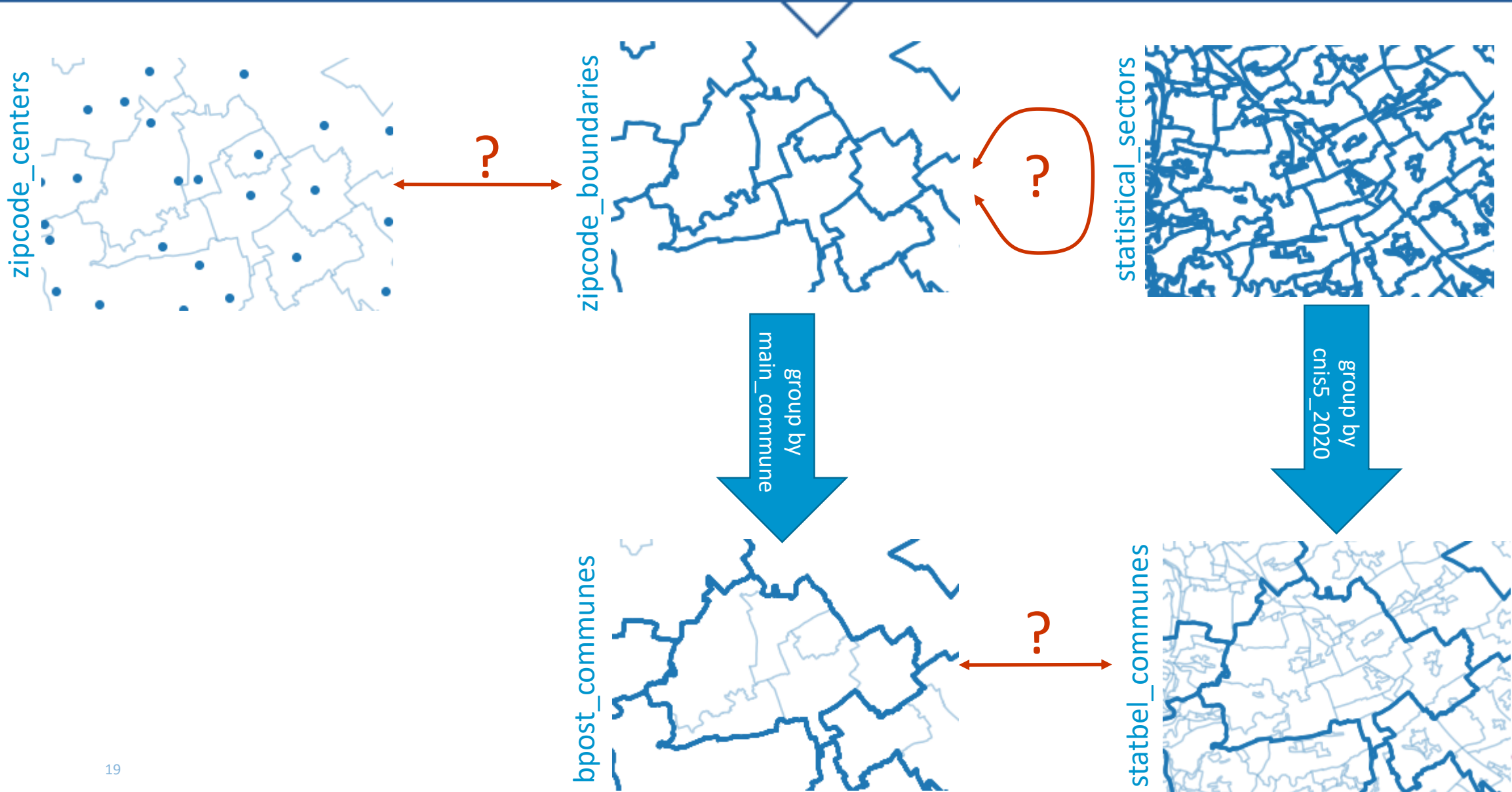
`ST_Difference(A, B) :`



`ST_Union(A, B) :`







# Data quality use cases




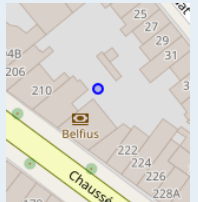
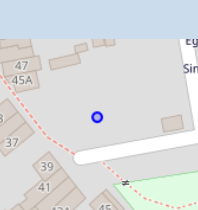

# Spatial join– point vs polygone

## Bpost – zipcode boundaries

zipcode	geom
1030	
1060	
1130	
1160	



## AdN – centre of zipcodes

zipcode	geom
1030	
1060	
1160	
7300	



Bp.zipcode	Bp.geom	AdN.zipcode	AdN.geom
1030	...	1030	...
1060	...	1060	...
1130	...	7300	...
1160	...	1160	...

53 “mismatches” (in AdN) :

- 1130 et 3700 Haren as same place
- “8600 Driekapellen” located “Driekapellenstraat, 1541 St. Pt Kap.”
- “6831 Noirfontaine” located “Rue de Noirfontaine, 4400 Flémalle”
- Commune name vs “lieu dit” (Laar, Schriek, Berg, Voort...)

# Spatial join: Mismatch zip centers – zip boundaries

```
SELECT bnd.zipcode AS bnd_zipcode,  
       ctr.zipcode AS ctr_zipcode,  
       ctr.locality AS ctr_locality,  
       bnd.geometry AS bnd_geometry,  
       ctr.geometry AS ctr_geometry
```

COLUMN SELECTION

```
FROM gistest.zipcodes_boundaries bnd  
JOIN gistest.zipcodes_centers ctr  
      ON ST_Contains(bnd.geometry, ctr.geometry)
```

SPATIAL JOIN

```
WHERE bnd.zipcode != ctr.zipcode
```

ROW SELECTION

	ABC bnd_zipcode	ABC ctr_zipcode	ABC ctr_locality	bnd_geometry	ctr_geometry
1	1130	3700	Haren	POLYGON Z((4.4211945903	POINT (4.4125708732 5
2	1541	8600	Driekapellen	POLYGON Z((3.9768305026	POINT (3.9855229 50.7
3	2000	2050	Antwerpen	POLYGON Z((4.4147477233	POINT (4.3997081 51.2
4	2000	2040	Antwerpen	POLYGON Z((4.4147477233	POINT (4.3997081 51.2
5	2000	2020	Antwerpen	POLYGON Z((4.4147477233	POINT (4.3997081 51.2
6	2000	2018	Antwerpen	POLYGON Z((4.4147477233	POINT (4.3997081 51.2
7	2000	2030	Antwerpen	POLYGON Z((4.4147477233	POINT (4.3997081 51.2
8	2000	2060	Antwerpen	POLYGON Z((4.4147477233	POINT (4.3997081 51.2
9	2180	3400	Laar	POLYGON Z((4.4609323026	POINT (4.4410406 51.2
10	2180	2223	Schriek	POLYGON Z((4.4609323026	POINT (4.4361161 51.2

Map interface showing a blue polygon and a blue point. The map is titled 'Leaflet | © OpenStreetMap contributors'. The status bar indicates '52 row(s) fetched - 406ms (+303ms)'.

# Spatial join – Internal mismatch @ BPost

```
SELECT zp1.zipcode, zp2.zipcode,  
       zp1.geometry, zp2.geometry,  
       ST_Intersection(zp1.geometry, zp2.geometry),  
       ST_Area(ST_Intersection(zp1.geometry, zp2.geometry)) / ST_Area(zp1.geometry)  
       AS common_ratio
```

COLUMN SELECTION

```
FROM gistest.zipcodes_boundaries zp1  
JOIN gistest.zipcodes_boundaries zp2  
     ON ST_Intersects(zp1.geometry, zp2.geometry)
```

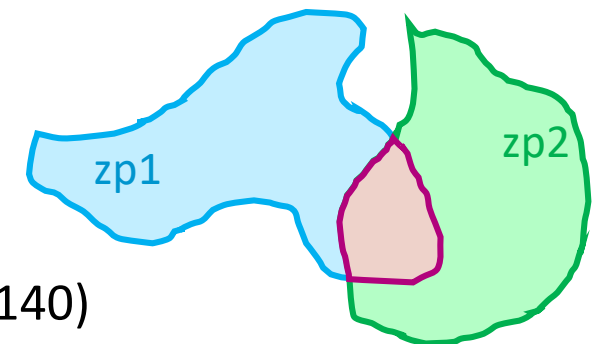
SPATIAL JOIN

```
WHERE zp1.zipcode < zp2.zipcode  
       AND ST_Area(ST_Intersection(zp1.geometry, zp2.geometry)) / ST_Area(zp1.geometry) > 0.01
```

ROW SELECTION




Only with special codes, we observe :

- Real overlaps (1830 or 1931 vs 1931 ; 7010 vs 7020)
- Missing “holes” for special codes (6075 or 6099 within 6022, 1049 within 1140)
- Several special codes with the same region (1043/1044, 1046/1049, 7510/7511...)





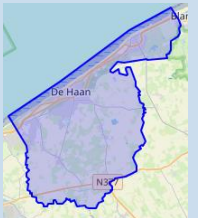
# Spatial join– polygone vs polygone

## Bpost – commune boundaries

commune	geom
VILVOORDE	
HAM-SUR-HEURE	
DE HAAN	



## Statbel – commune boundaries

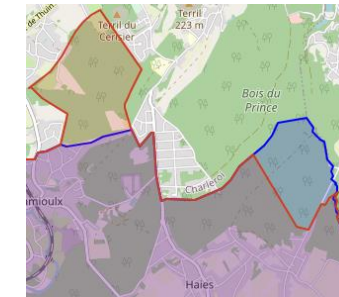
commune	geom
Vilvorde	
Ham-sur-Heure-Nalinnes	
De Haan	



AND

$$\frac{\text{Area}(\text{Intersection}(a, b))}{\text{Area}(\text{Union}(a, b))}$$

$$\in [50\%, 99.5\%]$$



- Coast: with (Statbel) or without (BPost) beach
- Known situations: 1040 = Etterbeek + Europe (Bxl) ; 1050 = Ixelles + Louise-Roosevelt (Bxl)
- Errors?

# Spatial join: Mismatch NIS – Zip boundaries

```
SELECT cnis5_2020, t_mun_fr, main_commune,  
        nis_bnd.geometry      AS nis_geom,  
        zip_bnd.geometry      AS zip_geom,
```

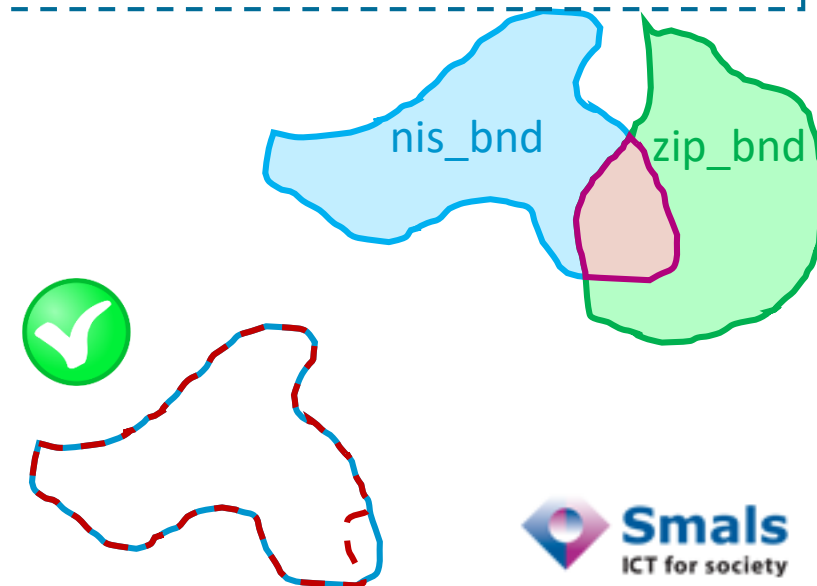
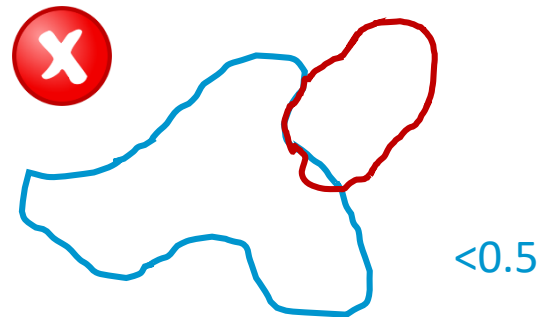
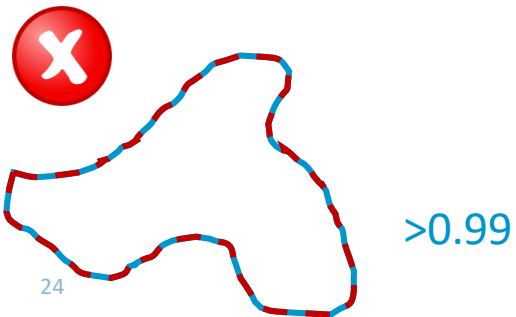
COLUMN SELECTION

```
FROM gistest.statbel_communes nis_bnd  
JOIN gistest.bpost_communes zip_bnd  
      ON ST_Intersects(nis_bnd.geometry, zip_bnd.geometry)
```

SPATIAL JOIN

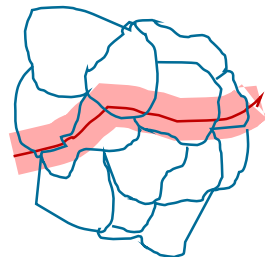
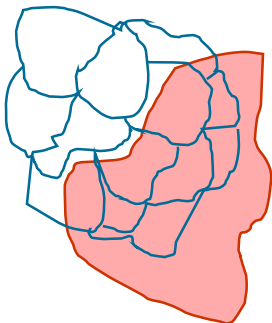
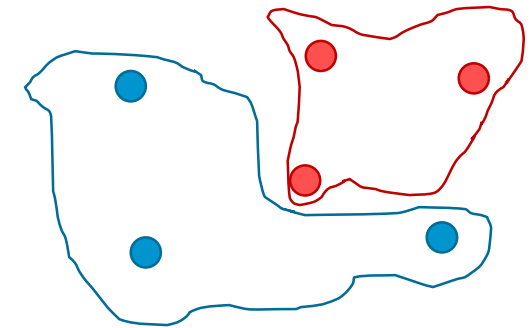
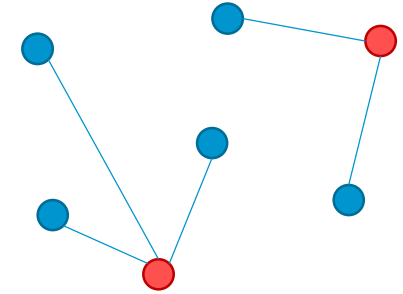
```
WHERE ST_Area(ST_Intersection(nis_bnd.geometry, zip_bnd.geometry)) /  
        ST_Area(nis_bnd.geometry) between 0.5 and 0.99
```

ROW SELECTION



# Spatial join: use cases

- Join **point >< point** (ST\_ClosestPoint):
  - Closest inspection antenna/inspector home for each worksite (as the crow flies)
- Join **point >< polygon** (ST\_Contains, ST\_Within):
  - Select a region (commune, province, neighborhood, business area) to a set of points (pharmacy, worksites...)
  - Aggregate (count, sum, mean...) points per region
- Join **polygon >< polygon** (ST\_Intersects, ST\_Contains, ST\_Touches):
  - Compute population of a given area
  - Get sectors within a buffer around a (set of) geometry





## Data import

- Using “WTK” (Well Known Text):

```
INSERT INTO app(p_id, geom)  
VALUES (2, ST_GeomFromText('POINT(-71.060316 48.432044)', 4326));
```

- Using ST\_MakePoint:

```
INSERT INTO app(p_id, geom)  
VALUES (2, ST_SetSRID(ST_MakePoint(-71.060316, 48.432044), 4326));
```

# Import shapefiles in Python

```
import geopandas as gpd # To handle shapefile (or geojson)
import sqlalchemy as sa # To push data on Postgres/PostGIS

# Download https://bgu.bpost.be/assets/9738c7c0-5255-11ea-8895-34e12d0f0423_x-
shapefile_3812.zip
# to "zipcode_boundaries.zip"

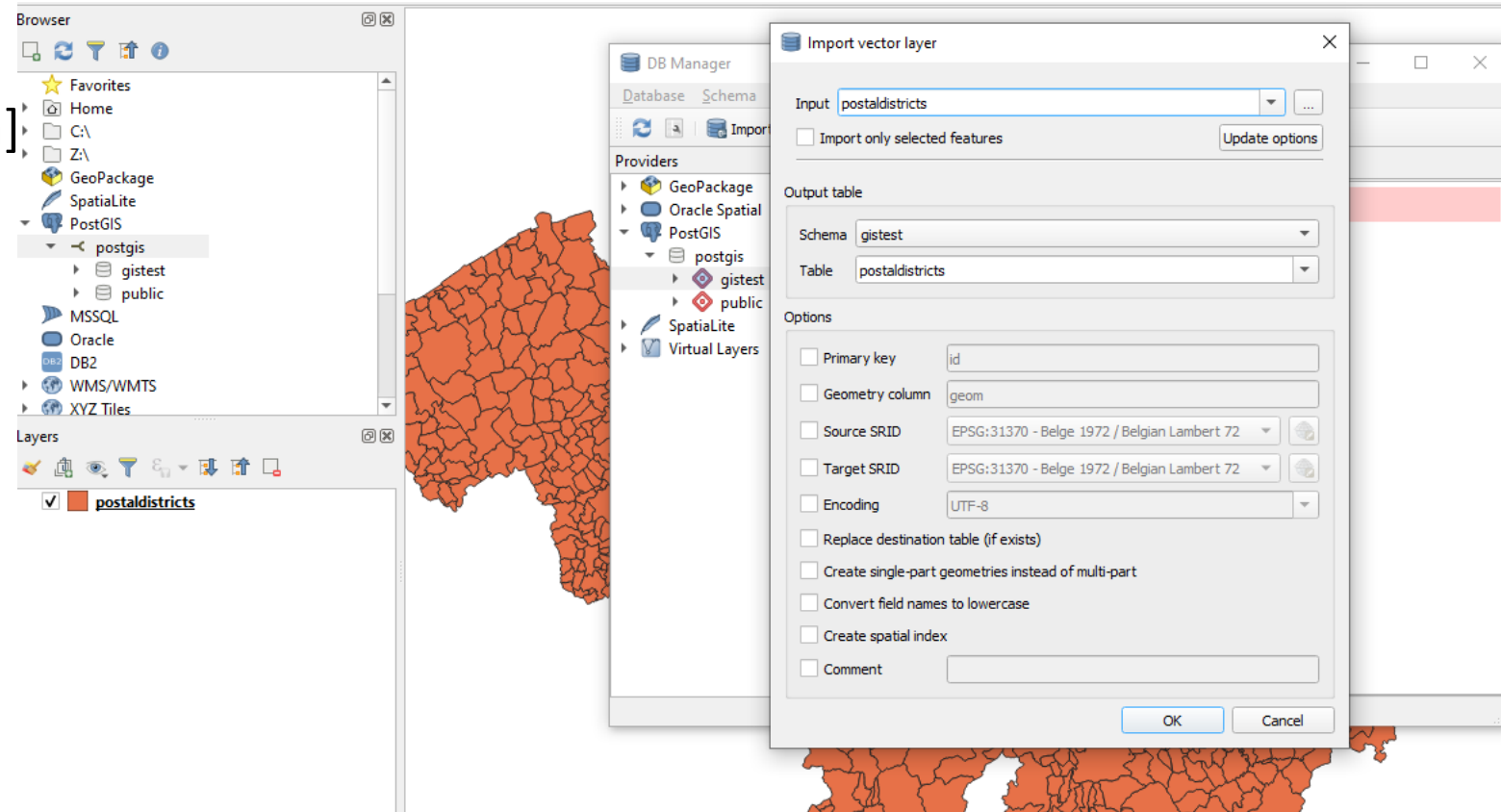
# Dataload
zipcode_boundaries = gpd.read_file(f"zip://zipcode_boundaries.zip/3812")
zipcode_boundaries = zipcode_boundaries.to_crs("epsg:4326")

# Create DB connection
engine = sa.create_engine(f"postgresql+psycopg2://{usr}:{pwd}@{host}:{port}/{db}")

# Push data into DB
zipcode_boundaries.to_postgis(name="zipcode_boundaries",
                              con=engine,
                              if_exists="replace",
                              schema="gistest")
```

# Import shapefiles via QGIS

- On [www.geo.be](http://www.geo.be), search “Cantons postaux”
- Select dataset “Cantons postaux”, download “projection = 31370”
- Open QGIS, drag&drop zipfile to “Layers” tab
- In Browser > PostGIS > new connection
- Database > DB Manager
- Select PostGIS > postgis > [schema]
- Import Layer/File...



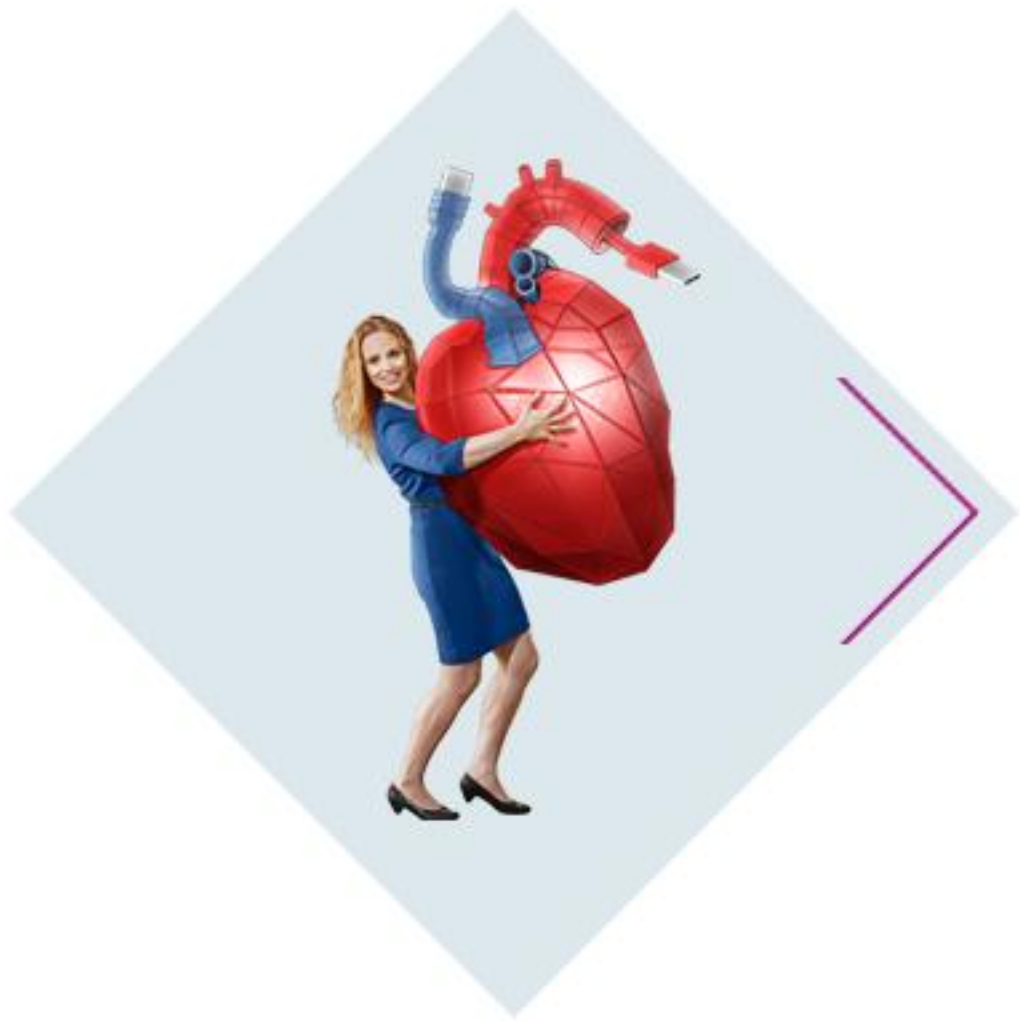
# Command line

```
export PGPASSWORD=mydatabasepassword
```

```
ogr2ogr \  
  -nln zipcode_boundaries \  
  -nlt PROMOTE_TO_MULTI \  
  -lco GEOMETRY_NAME=geom \  
  -lco FID=gid \  
  -lco PRECISION=NO \  
  Pg:'dbname=gistest host=localhost user=pramsey port=5432' \  
  zipcode_boundaries.shp
```

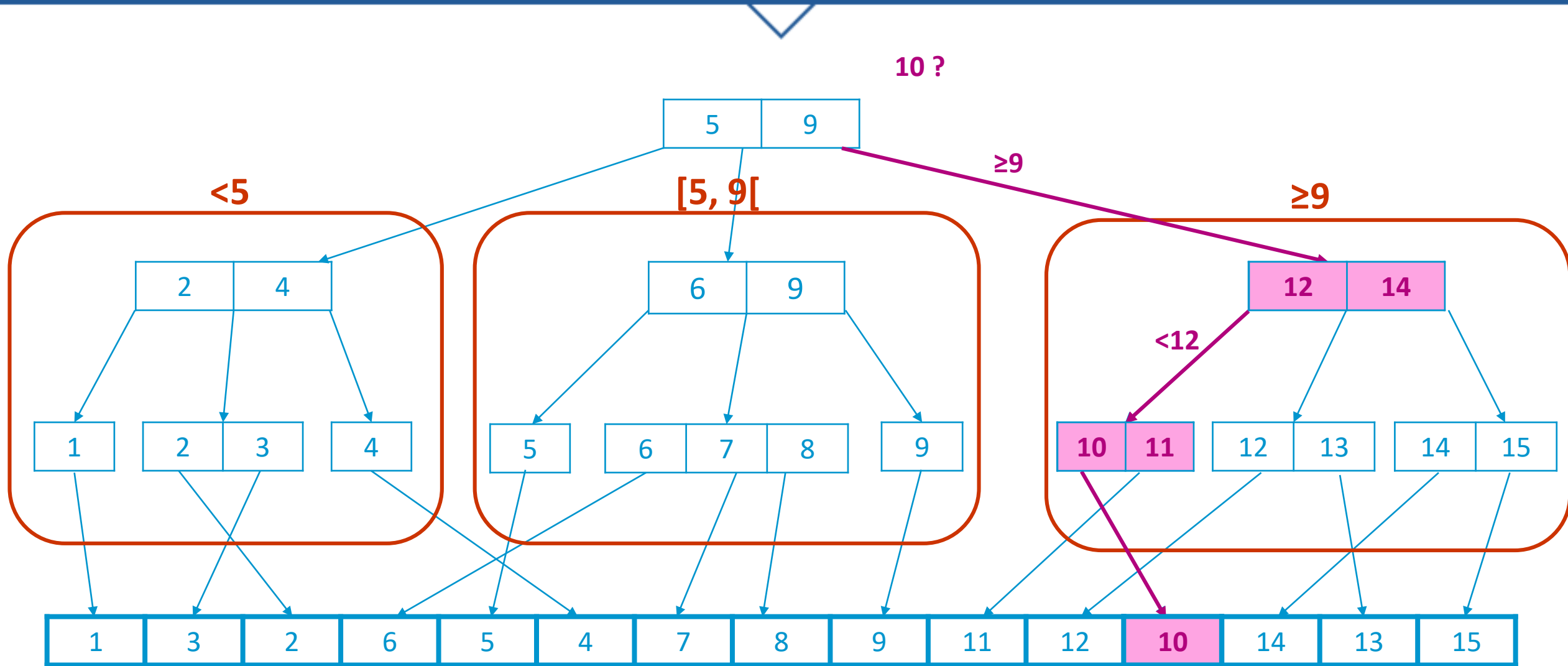
Alternative:

```
shp2pgsql -D -I -s 3812 zipcode_boundaries.shp zipcode_boundaries \  
  | psql dbname=gistest user=postgres host=localhost
```



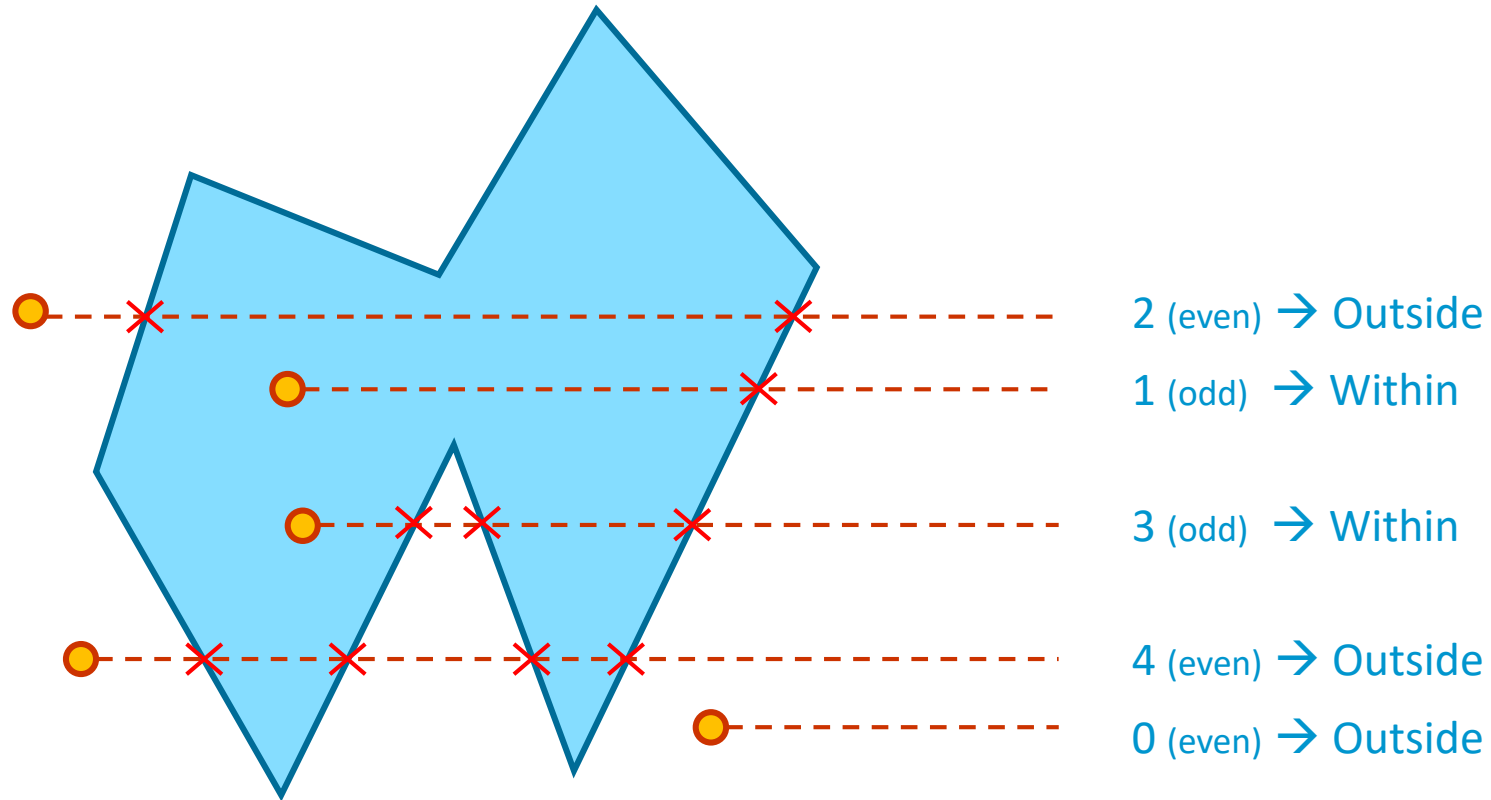
## Spatial index

# Basic indexing: B-Tree index



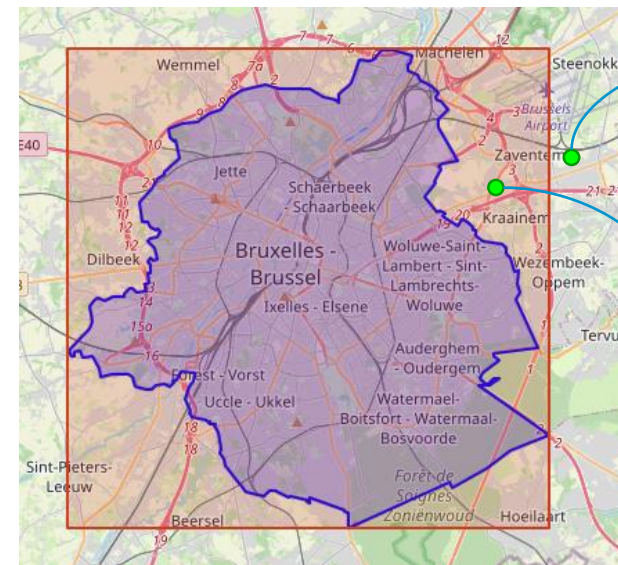
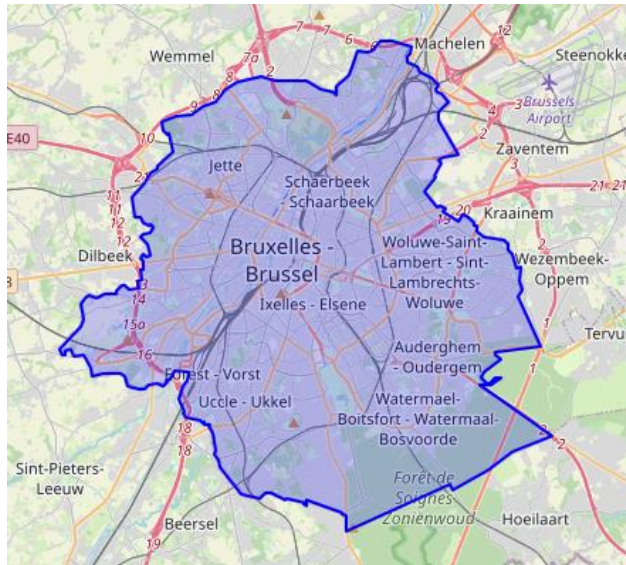
# Spatial index

- With geometrical shapes, “join” question is “Is a point within a polygon?”
- Requires a **complex geometric operation** :  $O(n \text{ points}(\text{polygon}))$

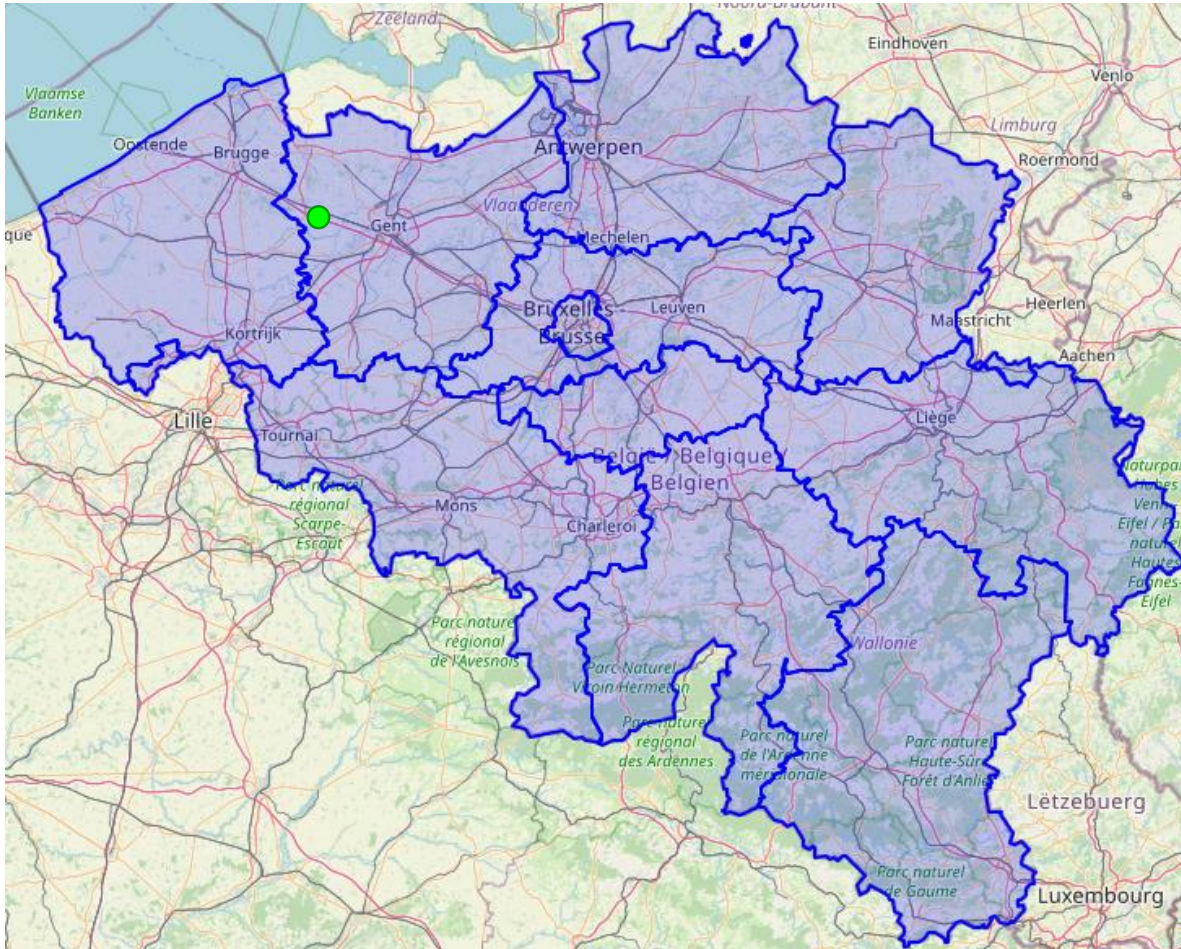


# Spatial index

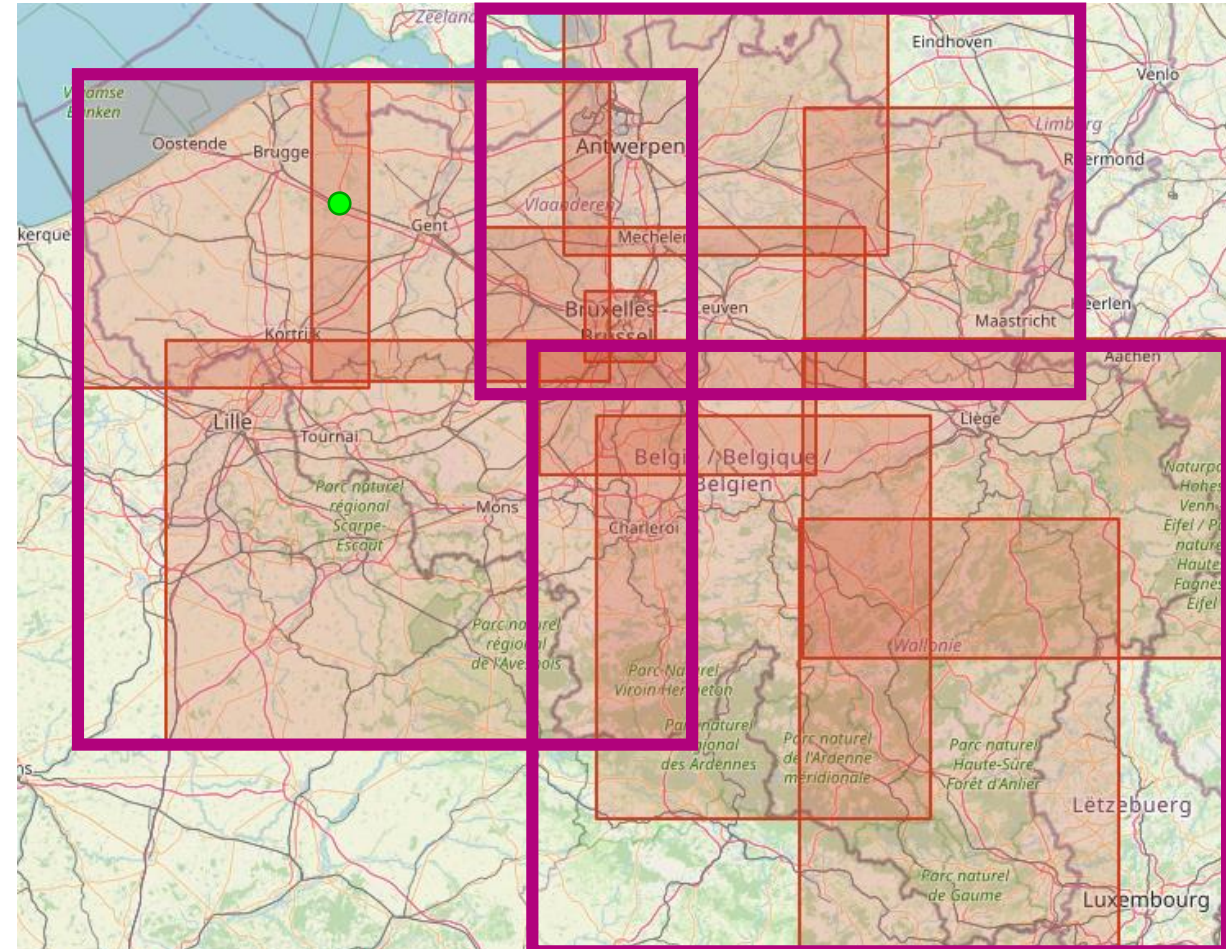
- Without indexing, spatial join between a list of N points and a list of M polygons requires  $N \times M$  “is within a polygon” operations
- But: “Is a point within a horizontal rectangle?”: very fast (using B-tree)
- Basic concept for spatial index: first consider bounding boxes (or envelope) for polygons, and then exact polygon only for those matching with bounding box



# R-tree



11 x “is ● within polygon”



11 x “is ● within rectangle” + 2 x “is ● within polygon”  
(3 + 3) x “is ● within rectangle” + 2 x “is ● within polygon”

# R-tree: performance

```
DROP INDEX IF EXISTS gistest.idx_zipcodes_boundaries_geometry;  
DROP INDEX IF EXISTS gistest.idx_zipcodes_centers_geometry;
```

```
SELECT *  
FROM gistest.zipcodes_boundaries bnd  
JOIN gistest.zipcodes_centers ctr  
ON ST_Contains(bnd.geometry, ctr.geometry);
```

```
SELECT *  
FROM gistest.statistical_sectors bnd  
JOIN gistest.zipcodes_centers ctr  
ON ST_Contains(bnd.geometry, ctr.geometry);
```

```
CREATE INDEX idx_zipcodes_boundaries_geometry  
ON gistest.zipcodes_boundaries  
USING GIST (geometry);
```

```
CREATE INDEX idx_zipcodes_centers_geometry  
ON gistest.zipcodes_centers  
USING GIST (geometry);
```

	Zipcode	Statistical sectors
Number of geometries	1 286 x 2 575	19 794 x 2 575
Number of points	1,14M	3,5M
Avg. points per geom.	900	180
Timing without index	9,3 seconds	48 seconds
Timing with index	0,45 seconds	0,3 seconds
Gain	X 20	X 160

## Index: a few remarks

- Works for comparison (**polygon, point**), but also (**polygon, polygon**)
- Index “**lost**” if any **operation is performed** before join: `ST_Transform`, `ST_Buffer`, ...
  - Either build the appropriate column and index it
  - Or use “`a.geom && b.geom`” = true if bounding boxes intersect
- **Alternatives** to GIST: SPGIST, BRIN...
- After some data updates: **VACUUM ANALYZE <table\_name>**
- As for “classical” PostgreSQL: indexes takes **space and time**.  
For statistical sectors (~20K records, 3.5M points): ~ 1MB - 0,5sec.



# Conclusion

# Conclusion

- **Not more difficult** than “classical RDBMS”, but with powerful **additional functionalities**
- Added value:
  - To just plot simple points (lat, lon) onto a map → No added value, PostgreSQL + leaflet.js (or other) is enough
  - **Distance** computation (even for simple points)
  - **Projection** handling
  - Line, Polygon handling: **spatial join, aggregation/combination...**
  - **Buffer**: area 1km away from Seveso sites, river, pipeline, highway...
- Alternatives:
  - Oracle Spatial , Microsoft SQL Server Spatial
  - Similar (but less advanced) functionalities in **ElasticSearch** (geoshapes)

## Conclusion – Use cases

- Clear potential for many **eGov applications** (see first webinar for inspiration)
- Can be used on Smals PostgreSQL DBs (ask DBAs... conditions TBD)... but no project so far?
- Known usages:
  - **BestAddress** (BOSA)
  - NGI/IGN & partners
  - (OpenStreetMap)
- Potential usages:
  - **Quality** checks
  - **Enrichment**: assign a zone (commune, province, business zone...) to a set of points
  - **Selection**: select addresses within a given area (i.e., companies within a known neighborhood)
  - **Disaster management**: area 10 km away from potential risk
  - **Impact analysis**: population living in a given area

Vandy Berten  
Smals Research

## Smals, ICT for society

02 787 57 11

Fonsnylaan 20 / Avenue Fonsny 20

1060 Brussel / 1060 Bruxelles

From your side?

Happy to get feedback and discuss  
potential use cases!