

Last week



- We have discussed spatial data models;
 - Spaghetti (without topological storage);
 - Network (for linear structures);
 - Topological (storing the topological elements; full 2D space);
- We have discussed how consistency enforcement can be achieved with software topology or structural topology implementations;
- We have discussed how topological relationships are evaluated in principle;
- We have quickly shown how this is implemented in SQL.
 - We will now extend this knowledge and use the relationships to query data...

Geo875 | FS24
University of Zürich

4. Lecture Spatial Databases

Spatial queries

Rolf Meile

Eidg. Forschungsanstalt für Wald, Schnee und Landschaft (WSL)
Swiss Federal Institute for Forest, Snow and Landscape Research

Zhiyong Zhou

Dept. of Geography, University of Zürich

Learning Objectives



- ✓ You get to know the concept for dealing with staging (temporary) and target (productive) database tables
- ✓ You will learn about types of queries applied in spatial databases
- ✓ You will learn how to query data based on spatial relationships
- ✓ You get an insight on query executing models in a client-server environment

Overview

- **1. Spatial Tables: definition and data loading**
- 2. Spatial queries
- 3. Where get queries executed?

Recall: spatial data types

	Postgis	Esri (ArcSDE)			Oracle Spatial
Abstract data type for Geometries	geometry	ST_GEOMETRY	SDO_GEOMETRY	BLOB	SDO_GEOMETRY
Database (underlying)	PostgreSQL	many possible	Oracle	Oracle	Oracle DB
Schema (owner)	public	SDE	MDSYS	Basic type of DB	MDSYS
Spatial table – data model	Attribute	Attribute	Attribute	in two tables	Attribute
Structure definition (DDL)	SQL	Esri SW, ±SQL	SQL, Esri SW	Esri SW	SQL
Data inputs (DML)	SQL, QGis, ...	Esri SW, SQL	SQL, Esri SW	Esri SW	SQL, ...
Spatial queries (DQL)	SQL, QGis, ...	Esri SW, SQL	SQL, Esri SW	Esri SW	SQL, ...



We cover this spatial datatype and setup;

Remember - additional Postgis types exist: geography, raster and topology;

Need data in spatial tables to show queries

- **Several steps and options**
 - import file data to temporary (=staging) spatial table
 - create target spatial table
 - transfer data to target spatial table
- **Important tasks in these steps**
 - a) Converting data types of tables and their data (geometry subtypes but also others; if needed)
 - b) Reprojecting
 - c) Applying transforming functions in SQL
 - d) SQL-inserting records in normal tables having no geometry type (e.g. lookup tables the spatial table depends on)
 - e) SQL-inserting records in spatial tables
 - f) Digitizing records in spatial tables with QGIS

Import data to staging table (also see project website)

1. Import dataset to a temporary, staging spatial table in database

- QGIS project [with appropriate projection] references file-based datasets
- Processing toolbox; search for postgres in the processing toolbox;
- Use 'Export to PostgreSQL' function to export (from QGIS perspective) local data to DB; table in the DB gets created "automatically";
- In the tool: change schema name from public to your schema e.g. project20
- Needs QGIS-connection to DB, first; see Browser window; [restart of QGIS required to use it in toolbox...]

2. Check the temporary spatial table

- What was automatically created? What was loaded? Further transformations needed? Errors but still imported? Analyze.
- Different perspectives!
Data centric -> DBeaver
Graphically, GIS centric -> QGIS

Staging table for biogeographic region geometries

Spatial DBeaver interface showing the structure and data of the `tmp_biogeo` table.

Table Structure:

Column Name	#	Data type	Length	Precision	Scale	Identity	Collation	Not Null	Default
id	1	int4		10				<input checked="" type="checkbox"/>	nextval('tmp_biogeo_id_seq')
geom	2	geometry						<input type="checkbox"/>	
area	3	float8	17	17				<input type="checkbox"/>	
perimeter	4	float8	17	17				<input type="checkbox"/>	
biogreg_	5	int4		10				<input type="checkbox"/>	
biogreg_id	6	int4		10				<input type="checkbox"/>	

SQL Query:

```
SELECT id, geom, area, perimeter, biogreg_, biogreg_id,
       biogreg_c1, biogreg_c6, biogreg_r6, biogreg_r1, biogreg_ve,
       shape_leng, shape_area
FROM tmp_biogeo;
```

Data View:

id	geom	area	perimeter	biogreg_
1	MULTIPOLYGON (((2689611.1781000 108439675.5, ...))	70246.753	2	
2	MULTIPOLYGON (((2689611.1781000 1330857641.11, ...))	626049.983	3	
3	MULTIPOLYGON (((2706995.95291 250867715.71, ...))	603499.847	4	
4	MULTIPOLYGON (((2634575.701000198648313.01, ...))	716393.255	5	
5	MULTIPOLYGON (((2751887.551100078255132.427, ...))	207549.016	6	
6	MULTIPOLYGON (((2645882.721700301875023.48, ...))	661018.809	7	
7	MULTIPOLYGON (((2703488.798099 268386654, ...))	150252.762	8	

Map View:

More on the staging table for biogeo region geometries

DB Manager

Database Schema Table Import Layer/File... Export to File...

Providers

- artwork
- biogeo
- canton
- classifies
- customer
- disch
- heightlevel
- likesartgroup
- likesartist
- mainriver
- measdaymean
- measfromto
- polyanalysis
- polyqual
- river
- scientist
- scotype
- slopetype
- soiltype
- statcomment
- station
- tmp_biogeo**
- tmp_canton
- tmp_measda...
- tmp_measfro...
- typofart
- zzz_tmp
- userdemo
- userdemo.geodb

tmp_biogeo

General info

Relation type: Table
Owner: user50
Pages: 1
Rows (estimation): 16
Rows (counted): 16
Privileges: select, insert, update, delete

PostGIS

Column: geom
Geometry: MULTIPOLYGON
Dimension: 2
Spatial ref: CH1903+ / LV95 (2056)
Estimated extent: 2485409.00000, 1075268.37500 - 2833842.00000, 1295934.75000
Extent: (unknown) ([find out](#))

Fields

#	Name	Type	Length	Null	Default
1	<u>id</u>	int4	4	N	{FUNCEXP :funcvaria ({FUNCEXP :funcvaria ((CONST :4 :constby 79 0 0 0
2	geom	geometry (MultiPolygon,2056)		Y	
3	area	float8	8	Y	
4	perimeter	float8	8	Y	
5	biogreg_	int4	4	Y	
6	biogreg_id	int4	4	Y	
7	biogreg_c1	int4	4	Y	
8	biogreg_c6	int4	4	Y	
9	biogreg_r6	varchar (35)		Y	
10	biogreg_r1	varchar (35)		Y	
11	biogreg_ve	varchar (10)		Y	
12	shape_leng	float8	8	Y	
13	shape_area	float8	8	Y	

Constraints

Name	Type	Column(s)
tmp_biogreg_pkey	Primary key	<u>id</u>

Indexes

Name	Column(s)
idx_tmp_biogreg_geom	geom

Structural and visual perspective from **QGIS**; even more info! see data types;

DB Manager

Database Schema Table Import Layer/File... Export to File...

Providers

- artwork
- biogeo
- canton
- classifies
- customer
- disch
- heightlevel
- likesartgroup
- likesartist
- mainriver
- measdaymean
- measfromto
- polyanalysis
- polyqual
- river
- scientist
- scotype
- slopetype
- soiltype
- statcomment
- station
- tmp_biogeo**
- tmp_canton
- tmp_measda...
- tmp_measfro...

Lookup tables 1: Create and fill with data

```
-- need to create a 'lookup' table with primary key
CREATE TABLE soilttype
(
    code          smallint,
    description   varchar(64)
);
ALTER TABLE soilttype ADD CONSTRAINT soilttype_pk
PRIMARY KEY (code); -- every table with primary key

-- we need to define and insert the lookupvalues
-- two records at a time in one SQL
INSERT INTO soilttype (code, desription)
VALUES
    (2, 'karbonatisch'),
    (1, 'silikatisch');
```

Lookup tables 2: Adjust target table to reference lookup table

```
/*
a) RIVER target table (created beforehand - no DDL sample
code here) needs a foreign key (ER model = N:1)
b) river.soiltype column was introduced as a
consequence of ER model N:1 [river vs. soiltype];
same type but different name in the lookup table:
soiltype.code column;
*/
-- if ER asked for a 'must', we would
ALTER TABLE river ALTER COLUMN soiltype SET NOT NULL;

-- creating foreign key on table river 'glueing' to
-- table soiltype
ALTER TABLE river ADD CONSTRAINT river_soiltype_fk
FOREIGN KEY (soiltype) REFERENCES soiltype(code);
```

The diagram illustrates the relationship between three entities: the river table, the soiltype table, and the code column. A red arrow points from the word 'column' under the river table entry to the 'soiltype' column in the ALTER TABLE command. Another red arrow points from the word 'table' under the river table entry to the 'soiltype' table in the same command. A third red arrow points from the word 'column' under the soiltype table entry to the 'code' column in the REFERENCES clause.

Create target spatial table biogeo according to my ER derivation

```
-- my destination/target table
```

```
CREATE TABLE biogeo
```

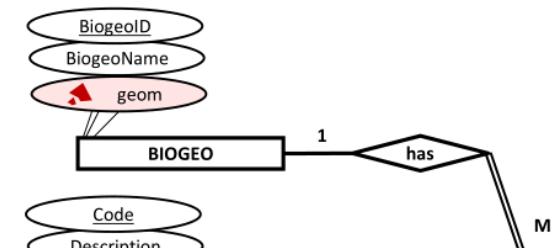
```
(
```

```
    biogeoid    int2,
```

```
    biogeoname varchar(512),
```

```
    geom        geometry(multipolygon,2056)
```

```
) ;
```



```
-- primary key - always
```

```
ALTER TABLE biogeo ADD CONSTRAINT biogeo_pk PRIMARY  
KEY (biogeoid);
```

```
-- spatial index - mandatory! for spatial columns
```

```
CREATE INDEX biogeo_geom_sidx ON biogeo USING  
GIST(geom);
```

Data input - transfer records from staging table

Transforming and transferring from one to another table

- Data types from staging tables need to fit the defined ones in target table
- We have to cast the types; e.g. from text to integer; e.g. from 3D to 2D
- We need modifications in the SELECT clause to 'fit' the records

```
INSERT INTO biogeo (biogeoid, biogeoname, geom)
SELECT
    ROW_NUMBER() OVER (ORDER BY 1) biogeoid, -- number up
    CASE WHEN q.derectionna ILIKE '%Zentralalpen%' THEN
        'Zentralalpen' ELSE q.derectionna END AS biogeoname,
    ST_Force2D(ST_multi(ST_union(q.geom))) geom -- grouping
FROM tmp_biogeo q
GROUP BY
    CASE WHEN q.derectionna ILIKE '%Zentralalpen%' THEN
        'Zentralalpen' ELSE q.derectionna END;
```

Data input - with geometry construction SQL

```
-- manually inserting a 'selfmade' polygon to the
polyanalysis spatial table

INSERT INTO polyanalysis (sciid, polyqual, patext,
geom)
VALUES (3, 1, 'an invented multipart analysis poly',
ST_GeomFromText('MULTIPOLYGON(
((2683846 1250116, 2755970 1246114, 2756725 1241991,
2683846 1250116)) ,
((2700001 1240199, 2346734 1771966, 2700001 1240199)) ,
)',2056));
```

More on transferring/inserting data

Things to keep in mind

- Recap <https://www.geo.uzh.ch/microsite/geo875/faq.html> with an example of reprojection of a complete table (subtype SRID and all records)
- ER-model -> Relations -> DB tables; this process leads to hierarchy in the resulting tables because foreign key columns are linked to primary key columns;
- 'Parent' tables have to be filled with records first; E.g. the lookup table soilttype is a parent table to river;
- Tables having no foreign keys pointing to other tables are **on top or at the end of branches [parents without parents]** of an ER-model hierarchy tree
- Use geometry editor functions – if needed; examples: ST_Force2D() or ST_Multi()
- Have a look at ST_Dump family of functions to find a way to e.g. create **singlepart** subtype from a **multipart** subtype aka 'exploding'

Validating data (geometries)

- There are **checking functions** for your geometries `ST_IsValid()`, `ST_IsValidReason()` and `ST_IsValidDetail()`

```
-- check consistency of
SELECT b.id, ST_IsValidDetail(b.geom) FROM tmp_biogeo
b;
```

id	st_isvaliddetail		
	valid	reason	location
1	[]	Self-intersection	POINT (2560100.639928136 1138436.6918350607)
2	[]	Self-intersection	POINT (2679075.7745317444 1121979.9632743374)
3	[]	Self-intersection	POINT (2771039.8912299313 1143390.6823539622)
4	[]	Self-intersection	POINT (2526910.0001321808 1168780.0000384077)
5	[v]	[NULL]	[NULL]
6	[v]	[NULL]	[NULL]

```
-- check: is it possible to make them valid? -> yes
SELECT ST_IsValidDetail(ST_MakeValid(b.geom)) FROM
tmp_biogeo b;
-- do replace the geometries with the fixed ones
UPDATE tmp_biogeo SET geom = ST_MakeValid(geom);
```

Overview

1. Spatial Tables: definition and data loading

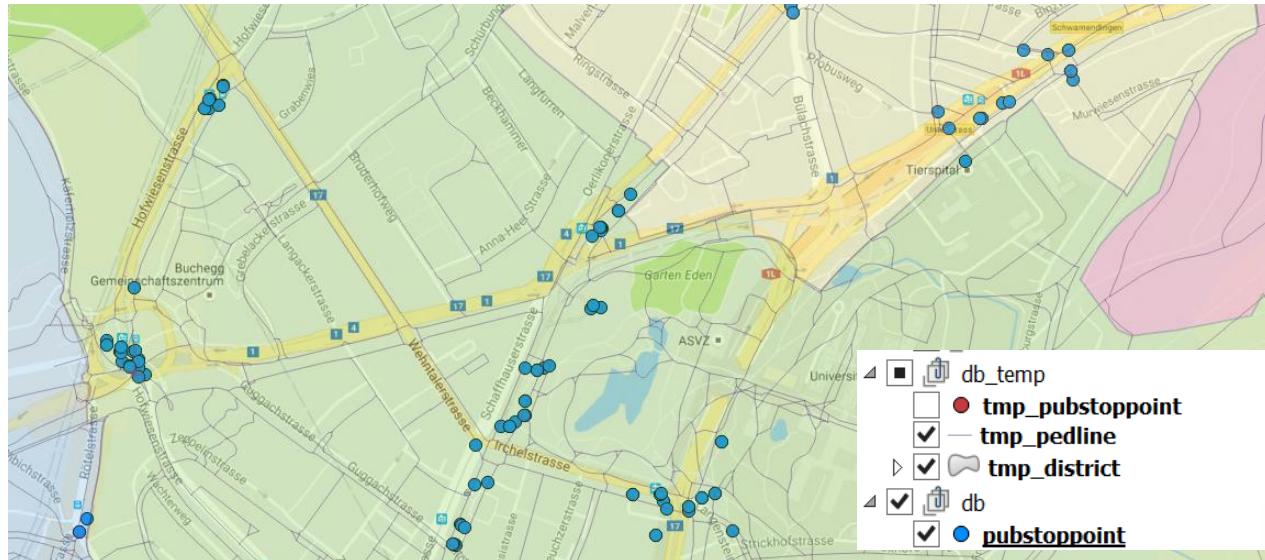
► 2. Spatial queries

3. Where get queries executed?

Examples used for spatial queries

The following spatial queries are based on these spatial tables

- **pubstoppoint** (**stop points** of VBZ public transport system in Zurich, point); target final table in my database model;
- **tmp_pedline** (**pedestrian road** network of Zurich, multipart line) staging table
- **tmp_district** (**districts** of Zurich, multipart polygon) staging table



Spatial queries

- Thematic queries
 - Selection and filtering of spatial objects based on attribute values (the same as in Geo874)
- Topological queries
 - Selection, filtering of objects based on qualitative spatial relationships (recall last lecture)
- Geometrical queries - includes geometry processing
 - Selection, creation and filtering of objects based on their shape, area, distance, direction,...

PostGIS Reference

<https://postgis.net/docs/reference.html>

Recall Geo243, see GITTA:

<http://www.gitta.info/SpatialQueries/en/html/index.html>

Thematic queries

Attribute value restrictions

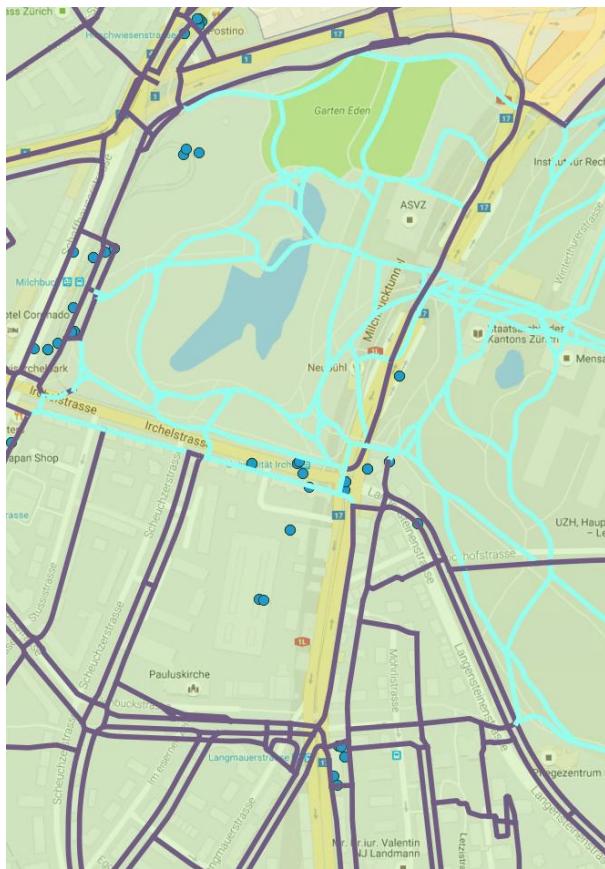
```
SELECT l.id, l.lokalisati, ST_AsText(l.geom) gtxt
FROM tmp_pedline l
WHERE l.lokalisati ILIKE '%Irchel%';

-- divisions of integers get truncated -> cast to numeric
SELECT l.id, CAST(l.id AS numeric)/2 AS liddiv,
       l.lokalisati,
       ST_AsText(l.geom) gtxt
FROM tmp_pedline l
WHERE CAST(l.id AS numeric)/2 = 12;

-- regexp expressions impact WHERE clause!!! See docs
SELECT l.id, l.lokalisati,
       regexp_matches(l.lokalisati, E'^..chel') AS pattfound,
       ST_AsText(l.geom) gtxt
FROM tmp_pedline l;
```

Thematic queries cont.

Attribute value restrictions in QGIS



Select by expression - tmp_pedline

Expression Function Editor

"lokalisati" ILIKE '%Irchel%'

Search

group Field

Double click to add field name to expression string.
Right-Click on field name to open context menu sample value loading options.

Notes

Loading field values from WFS layers isn't supported, before the layer is actually inserted, ie. when building queries.

Fields and Values

- id
- fahrrad
- NULL
- hoehe_anfa
- hoehe_ende
- lokalisati**
- velowege_m
- shape_leng

Fuzzy Matching

General

Geometry

Math

Operators

-
- %
- *
- /
- ^
- ||
- +
- <
- <=
- <>
- =
- >
- >=
- AND
- ILIKE
- IN

Output preview: 0

Load values all unique 10 samples

Select Close

Topological queries from last week

Returns objects satisfying a qualitative spatial relationship;

The single query geometry here is created on-the-fly with a constructor;

```
-----  
| SELECT l.id, l.lokalisati, ST_astext(l.geom) gtxt  
| FROM   tmp_pedline l  
| WHERE  ST_Intersects(l.geom,  
|                      ST_GeomFromText('LINESTRING(2683846 1250116,  
|                      2685846 1249812)',2056)  
| );  
  
| -- returns some multiline geometries  
| -----
```

Spatial join - base version

Topological queries can be applied between geometries of allowed dimensionalities (here two sets: 2D-multilines against 2D-points)

```
-----  
| SELECT l.id, l.lokalisati, p.pstopid, p.atycode  
| FROM     tmp_pedline l  
| JOIN    pubstoppoint p  
| ON      ST_Intersects(l.geom, p.geom) ;  
-----
```

What's exactly happening here?

- Similar joining as we know it from standard SQL
- Cartesian product for spatial comparison of **two spatial tables**
- Each geometry with each geometry - binary geometry comparing with regard to 'intersects'; others possible, e.g. 'contains'
- This query type is also called **spatial join** (with all 8+ kinds of topo relationships: intersects, contains, within, overlaps, ...)

Exercise 1



- What do you think happens in a database when we execute the previous query having 100'000 stops and 5'000'000 pedestrian roads? Debate with me.
- How long will such a query run in our database?
- How long will such a query run with two shape files?

Using QGIS for topological queries

Which stops lay on pedestrian roads (spatial join)?

Join attributes by location

Parameters Log Run as batch process... ? X

Target vector layer: pubstoppoint [EPSG:2056]

Join vector layer: tmp_pedline [EPSG:2056]

Geometric predicate:

- intersects
- contains
- disjoint
- equals
- touches
- overlaps
- within
- crosses

Precision: 0.000000

Attribute summary: Take attributes of the first located feature

Statistics for summary (comma separated) [optional]: sum,mean,min,max,median

Joined table: Only keep matching records

Joined layer: [Create temporary layer]

Open output file after running algorithm

0% Run Close

Join attributes by location

This algorithm takes an input vector layer and creates a new vector layer that is an extended version of the input one, with additional attributes in its attribute table.

The additional attributes and their values are taken from a second vector layer. A spatial criteria is applied to select the values from the second layer that are added to each feature from the first layer in the resulting one.

Spatial join cont.

```
SELECT l.id, l.lokalisati, p.pstopid, p.atycode
FROM tmp_pedline l
JOIN pubstoppoint p
ON ST_Intersects(l.geom, p.geom); -- no records out..... makes sense!
```

We should apply some buffer to our stop points; or any other method?

```
SELECT l.id, l.lokalisati, p.pstopid, p.atycode
FROM tmp_pedline l JOIN pubstoppoint p
ON ST_Intersects(l.geom, ST_Buffer(p.geom, 100));
-- long searching... Why?

SELECT l.id, l.lokalisati, p.pstopid, p.atycode
FROM tmp_pedline l JOIN pubstoppoint p
ON ST_DWithin(l.geom, p.geom, 100)
-- this is it!
```

Spatial join cont.

Testing with stop points and districts; more joining for more info;

```
SELECT d.id, d.distname, p.pstopid, p.atycode
FROM tmp_district d JOIN pubstoppoint p
ON ST_Intersects(d.geom, p.geom);
-- 140ms, perfectly fast, 10870 recs; ST_Contains;
```

```
SELECT d.id, d.distname, p.pstopid, p.atycode
FROM tmp_district d JOIN pubstoppoint p
ON ST_DWithin(d.geom, p.geom, 10);
-- same speed with more (why?) resulting records;
```

```
SELECT d.id, d.distname, p.pstopid, p.atycode,
lk.atydesc
FROM tmp_district d JOIN pubstoppoint p
ON ST_Intersects(d.geom, p.geom)
INNER JOIN soilttype lk ON p.atycode=lk.atycode
WHERE p.atycode=1; -- spatial join + join + where
```

Topological queries cont.

Topological queries can even be applied between geometries of the same dataset (self join...)

```
-----  
| SELECT d1.id, d1.distname, d2.id, d2.distname  
| FROM   tmp_district d1 JOIN tmp_district d2  
| ON     ST_Touches(d1.geom, d2.geom);  
  
-- with further filtering in the where clause  
SELECT d1.id, d1.distname, d2.id, d2.distname  
FROM   tmp_district d1 JOIN tmp_district d2  
ON     ST_Touches(d1.geom, d2.geom)  
WHERE  d1.distname = 'Kreis 1';  
-----
```

Geometrical queries

- **Distance:** rewrite slow buffer queries...
- Use ST_DWithin: distance within in the join/where clause; makes use of a spatial index (**topological operator!**)
- ST_Distance (**geometrical operator**) or <-> is to be used in SELECT and the ORDER BY part for nearest-neighbour Queries (also see exercises)

```
-- All stops/pedestrian path within 10cm distance
SELECT l.id, l.lokalisati, p.pstopid, p.atycode
FROM tmp_pedline l JOIN pubstoppoint p
ON ST_DWithin(l.geom, p.geom, 0.1);

SELECT l.id, l.lokalisati, p.pstopid, p.atycode,
-- show the distance
ST_Distance(l.geom, p.geom) AS dist
FROM tmp_pedline l JOIN pubstoppoint p
ON ST_DWithin(l.geom, p.geom, 0.1);
```

Exercise 2



What does this query do/return?

```
-----  
| SELECT l.id, l.lokalisati, p.pstopid, p.atycode,  
|       ST_Distance(l.geom, p.geom) AS dist  
| FROM   tmp_pedline l  JOIN  pubstoppoint p  
| ON     ST_DWithin(l.geom, p.geom, 25)  
| ORDER BY ST_Distance(l.geom, p.geom) ASC  
| LIMIT 5;  
-----
```

Geometrical queries cont.

Some further examples of geometrical functions

- Shortest Line calculation (between 2 geometries): creates a line as a result
- Centroid calculation (for one geometry): a point geometry
- Maximum distance (between 2 geometries): distance in SRID-unit
- Concave hull, shrink wrapping (for one geometry): a polygon geometry
- Difference (between 2 geometries): geometric difference

Geometrical functions normally go here

```
SELECT .....  
FROM   spatial_table1 JOIN spatial_table2  
ON     topological_function  
WHERE  standard_restriction_function  
ORDER BY .....
```

Geometrical queries cont.

- ST_UNION to **spatially aggregate**: building Zurich as one multipolygon record
- A spatial option for grouping geometries; similar and often used together with aggregate functions of standard SQL like max, min, avg with a **group by** expression

```
-- just the geometry
SELECT ST_AsText(ST_Union(d.geom)) AS zuerigeomtxt
FROM tmp_district d
GROUP BY 1; -- a constant, means: group all records;

-- with a more complex aggregate text function
SELECT string_agg(d.distname, ', ') AS aggrtxt,
round(CAST(ST_Area(ST_Union(d.geom)) / 1000 / 1000) AS numeric, 2) AS aera_km2
FROM tmp_district d GROUP BY 1;
```

Wrap up: Query types & PostGIS functions

- Tons of functionality to directly apply to your spatial data in a database
- Functionality & data transforming speed is high (few context switches)
- Some knowledge about data organization (ER diagram, tables diagram, indexes) are essential
- Gain clever prearranged data out of the honeypot by SQL; do smart things with this data using R, Phyton, Julia, Matlab...
- Let's quickly check the possibilities from the docs:
<https://postgis.net/docs/reference.html>

Overview

1. Spatial Tables: definition and data loading
2. Spatial queries
- **3. Where get queries executed?**

SQL-Client + Spatial DB

Client: DBeaver

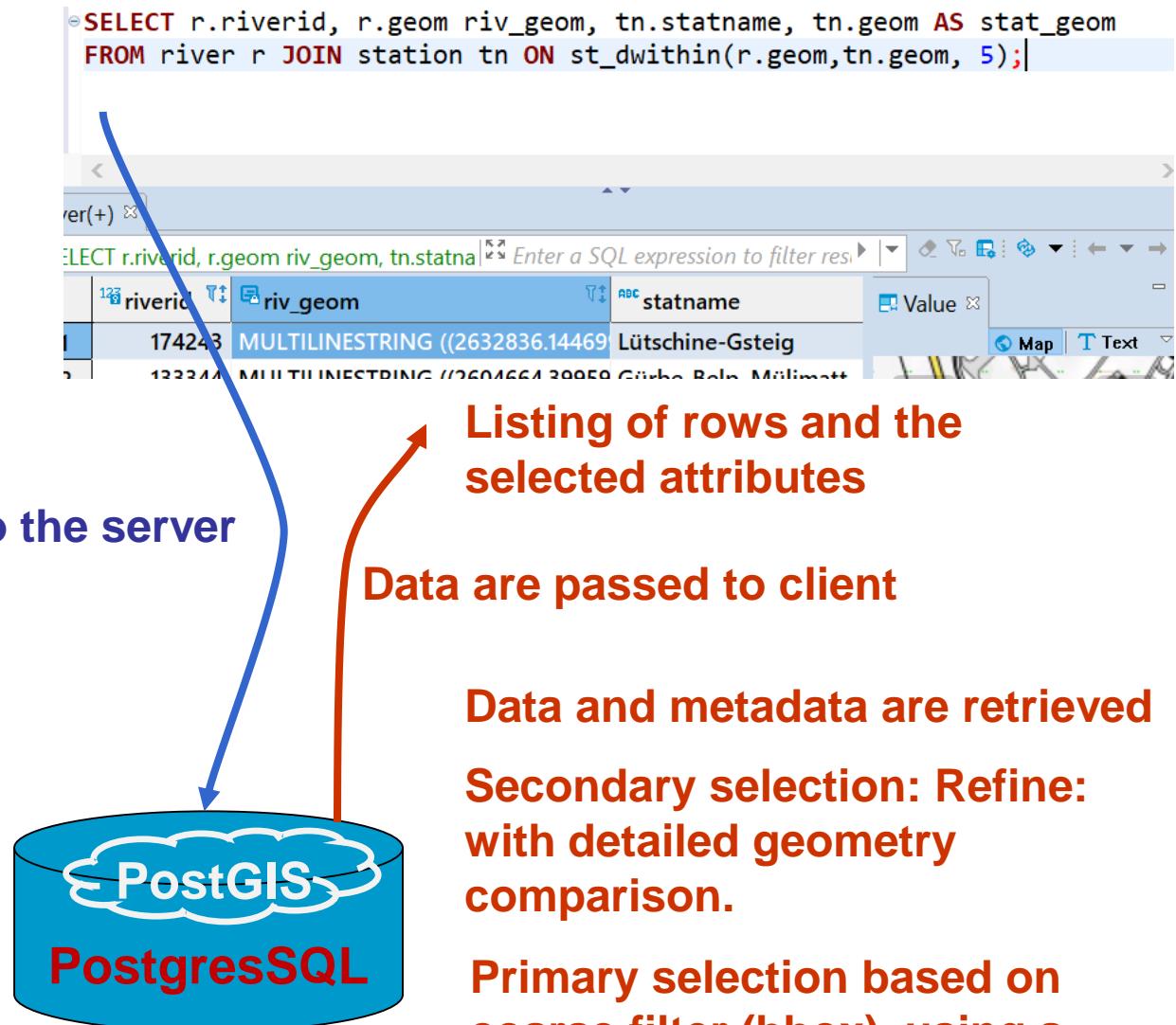
User generates a SQL-Query using a spatial operator on two spatial tables

Just text being sent to the server

Management

DB Explain Plan

SQLs executed



QGIS + Spatial DB

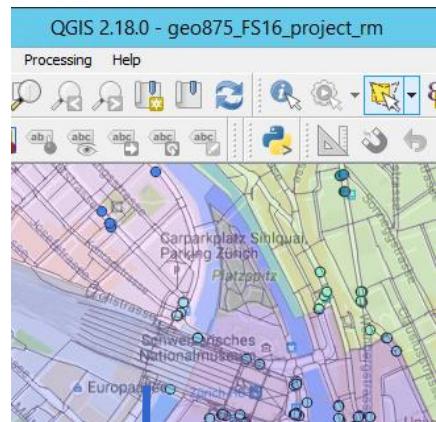
Client: QGIS

User performs a spatial function from the toolbox e.g. join attributes by location

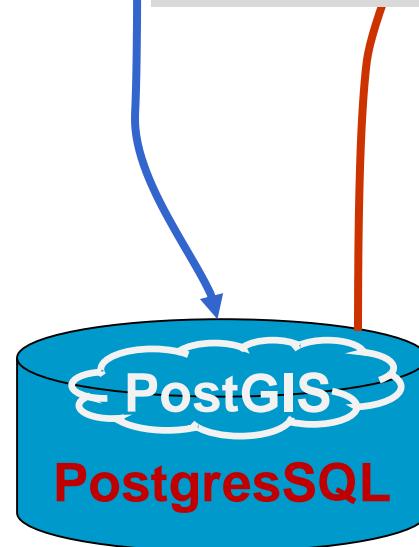
x,y coordinates of the bbox of geometries and metadata (IDs) are being sent to the server

this happens in the form of multiple SQLs - built on the client side

Management
DB Explain Plan
SQLs executed



Memory caching applies



Visualization or saving results (geometries)

Secondary selection: Refine with detailed geometry comparison.

Why here?

- Performance
- Same libraries local as on server!
- Distributed vs centralized

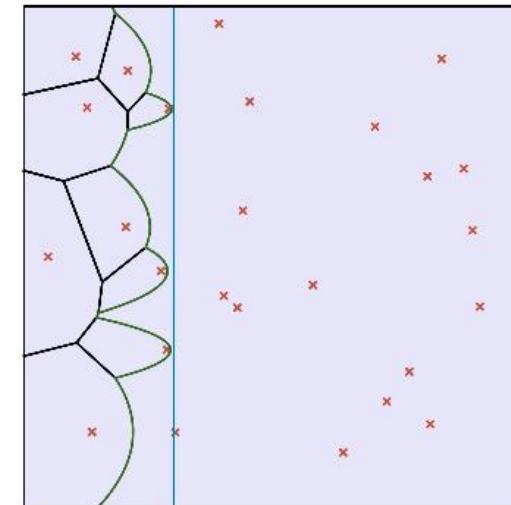
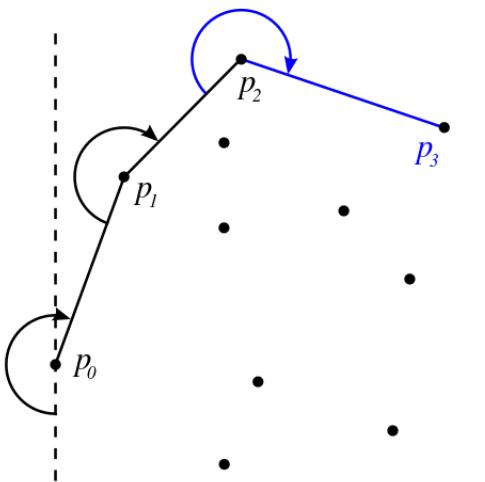
Preliminary data are passed to client

Data (IDs of potential features) and metadata are retrieved

Primary selection based on coarse filter (bbox), using the spatial index in the DB

Computational Geometry

- The discipline exploring algorithms for efficient execution of geometrical problems (see geo872, Ross Purves, Spatial Programming).
 - Intersections of geometries
 - Point in Polygon, Polygon in Polygon tests
 - Convex Hull (gift wrapping), Voronoi polygons (sweep line)
 - kNN (nearest neighbour)



https://en.wikipedia.org/wiki/Gift_wrapping_algorithm

<https://brilliant.org/wiki/intersections-line-segments/> <https://www.youtube.com/watch?v=k2P9yWSMaXE>

Summary



- We have seen the process of reading data to staging tables and then transforming to target tables.
- We have discussed querying (thematic, topological, geometrical) of spatial data in PostgreSQL database with PostGIS extension.
- We have explored where queries are executed.
- You got a glimpse of the Filter-Refine approach to efficient query execution (tbc, in spatial indexing).
- You understand that computational geometry algorithms underpin the execution of the queries in the DB.