Aalto University
School of Engineering

YYT-C3002
Application Programming in Engineering

GIS II

Anas Altartouri
Otaniemi 24.1.2019

Recap

In the previous lecture, we introduced and discussed:

- What GIS and GIS applications are
- Geoprocessing with software libraries and command-line programs
- Extending a desktop GIS with plugins

GIS Application Development

Desktop

Extending a GIS
- ArcGIS add-in
- QGIS plugin

Web-based

Web map application
- Database/spatial DB
- Server-side application, geospatial web services
- Client-side application

Geoprocessing
- Reading/writing libraries
- Vector data geoprocessing libraries
- Raster data geoprocessing libraries

Overview

In today's lecture,

Databases (geospatial data queries)

Server-side applications (geospatial web services)

Client-side applications (geospatial data loading and rendering)

In today's exercise,

Perform geoprocessing in *PostGIS*

Set geospatial data services using *Geoserver*

GIS Application Development

Desktop

Extending a GIS
- ArcGIS add-in
- QGIS plugin

Web-based

Web map application
- Database/spatial DB
- Server-side application, geospatial web services
- Client-side application

Geoprocessing
- Reading/writing libraries
- Vector data geoprocessing libraries
- Raster data geoprocessing libraries

Software and information systems

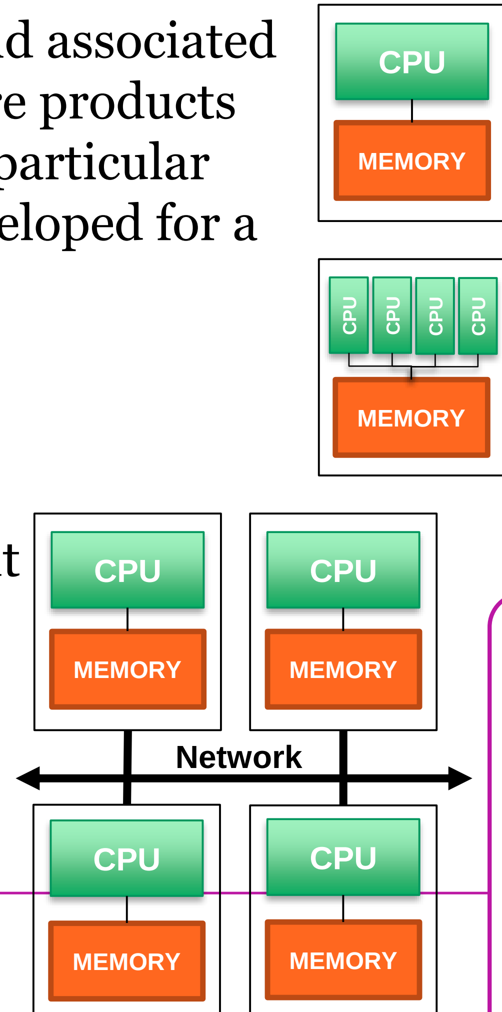
Software

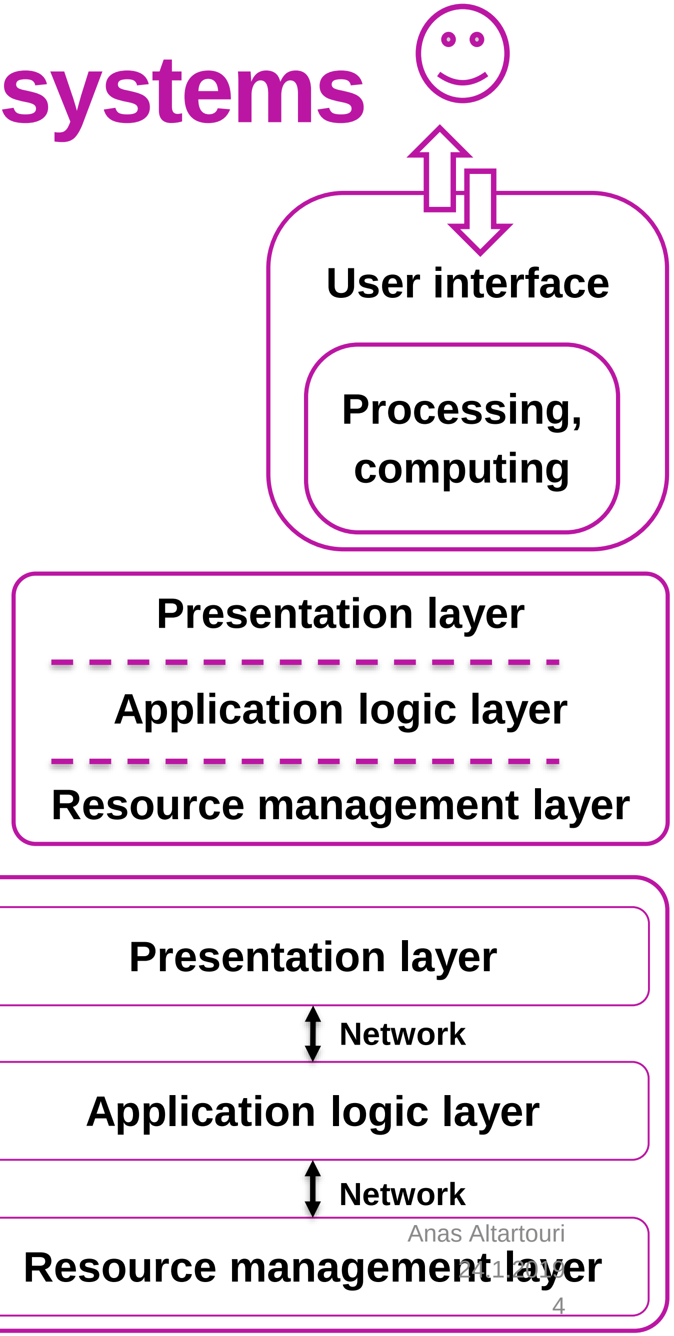
"Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market."[1]

Information system

"An information system is a piece of software that manages facts about some aspect of the state of the real world for a specific purpose"[2]

Distribution

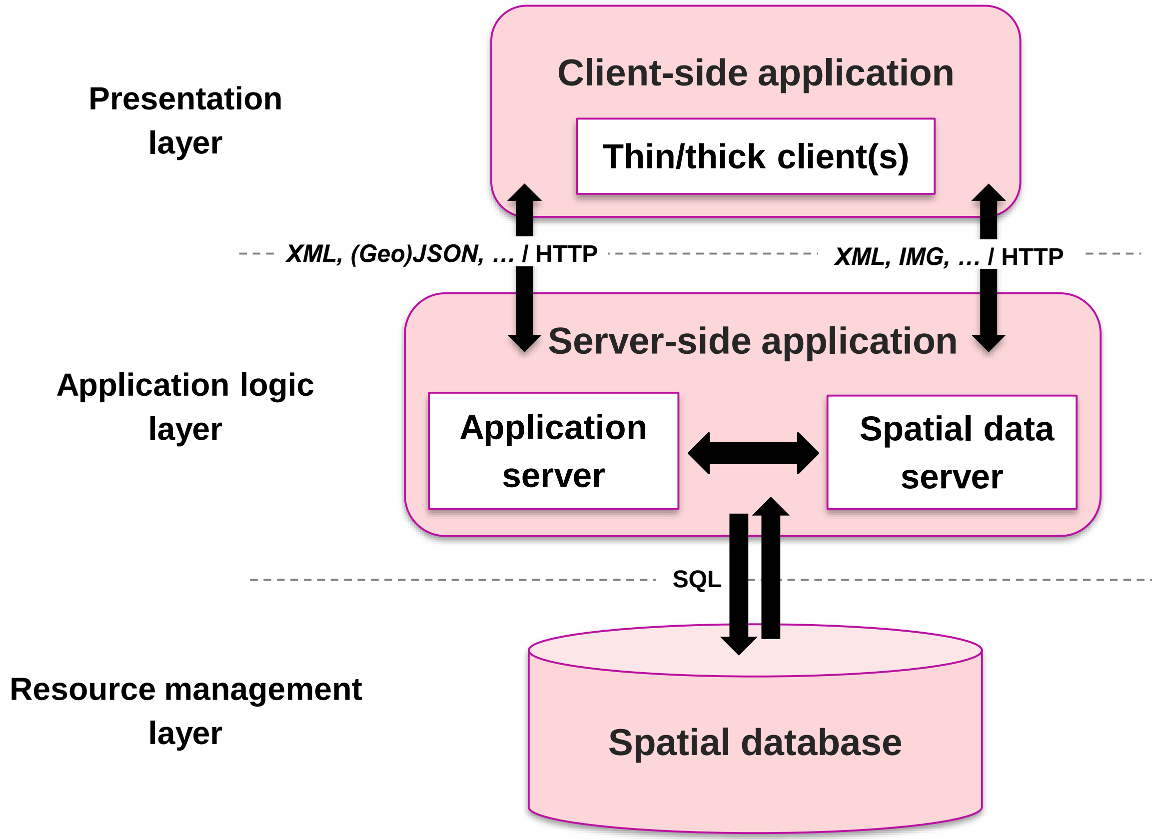




Aalto University
School of Engineering

[1] Sommerville, Ian (2015). *Software Engineering* (10th ed.). Pearson.
[2] Karl Aberer (2006-07), Distributed Information System,EPFLIC, Laboratoirede systèmesd'informationsrépartis

Web map applications – generic system architecture



Alonso, G., Casati, F., Kuno, H., Machiraju, V. (2003). Web services: Concepts, Architectures and Applications. Springer, Berlin, 354 p. Chapter 1: Distributed Information Systems.

Spatial databases
The need for a database

Data is key in any information system

- deriving information
- providing services

Why databases?

- Data consistency
- Efficient data processing
- Web interfaces to data

A database allows us to create, read, update, and delete data using Structured Query Language (SQL)

Spatial databases
Why 'spatial'?

Many data sets in various domains (e.g., natural and built environments) have a geographic aspect

An 'ordinary' database can store numbers, strings, and dates

It can also analyze them

- Mathematical operations on numbers
- Concatenating strings
- Deriving information from dates

But the geographic space is ignored in an ordinary DB

Spatial databases
Basics

Important concepts of spatial data in a database:

Data types

Indexes

Functions

Spatial databases
Vector feature types

Representation of objects in the real world

- Points
  - *E.g., bus stops*
- Lines
  - *E.g., roads, railways*
- Polygons
  - *E.g., lakes*

Source: NLS orthophotos WMS

Note: Spatial databases (such as PostGIS) can also deal with raster data

Spatial databases
Spatial indexes

Numbers, strings, and dates can be easily sorted;

Any value is:

less than,

greater than, or

equal to every

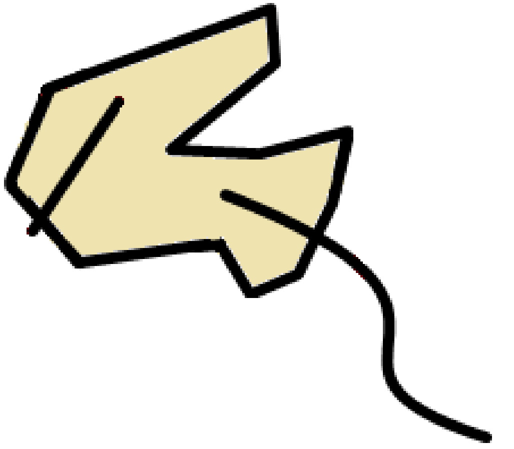
another value

But how to sort spatial objects?

Spatial databases
Spatial indexes

Spatial relations

Objects can overlap, be contained, etc.



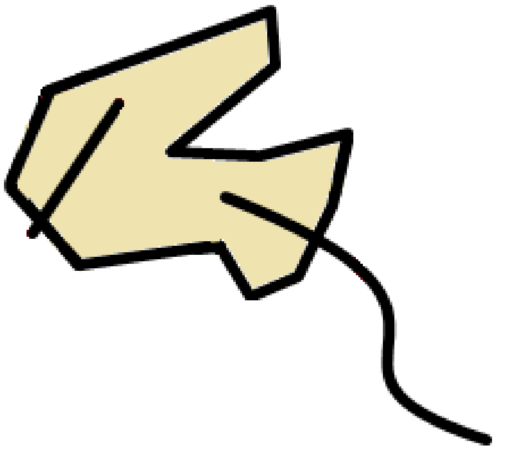
Spatial databases
Spatial indexes

Bounding boxes

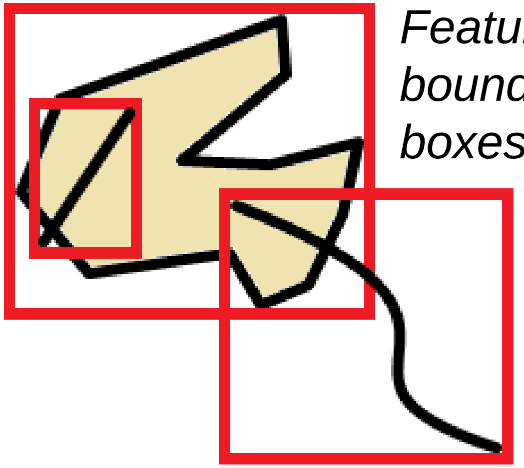
the smallest size rectangle containing a given feature

An example query:

"what lines are inside this polygon?"



Spatial databases
Spatial indexes



Bounding boxes

the smallest size rectangle containing a given feature

An example query:

"what lines are inside this polygon?"

How is it evaluated?

Spatial databases
Spatial indexes

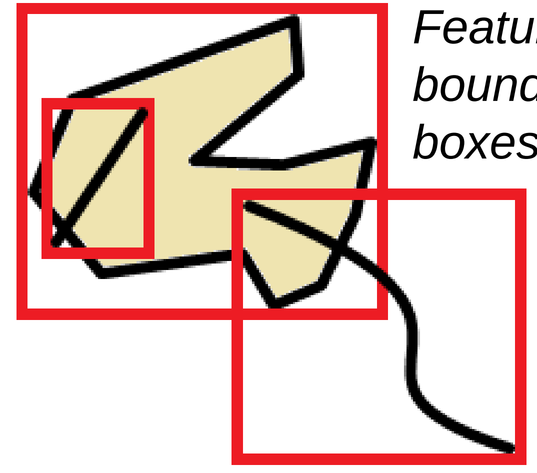
Bounding boxes

the smallest size rectangle containing a given feature

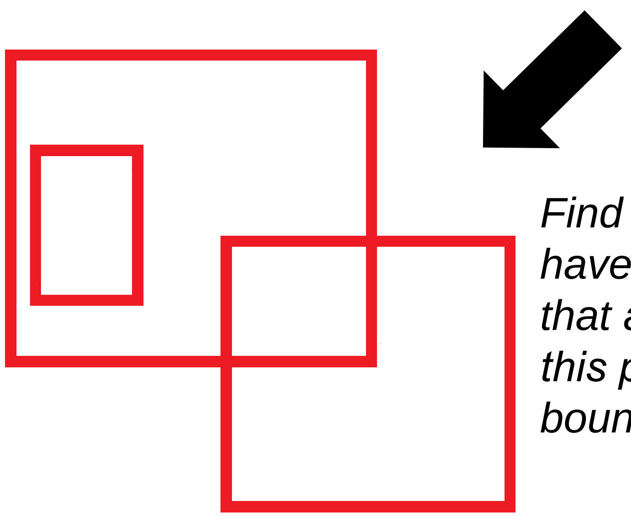
An example query:

"what lines are inside this polygon?"

How is it evaluated?



Feature bounding boxes



Find the lines that have bounding boxes that are contained in this polygon's bounding box

Spatial databases
Spatial indexes

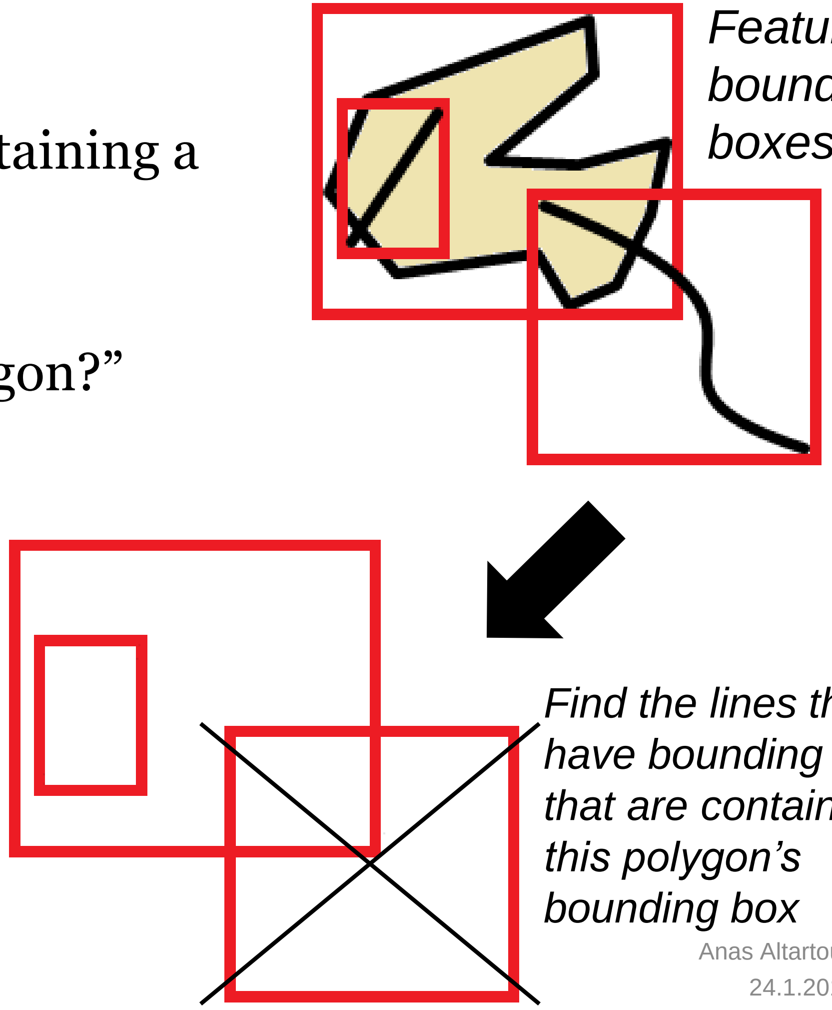
Bounding boxes

the smallest size rectangle containing a given feature

An example query:

“what lines are inside this polygon?”

How is it evaluated?



Spatial databases
Spatial indexes

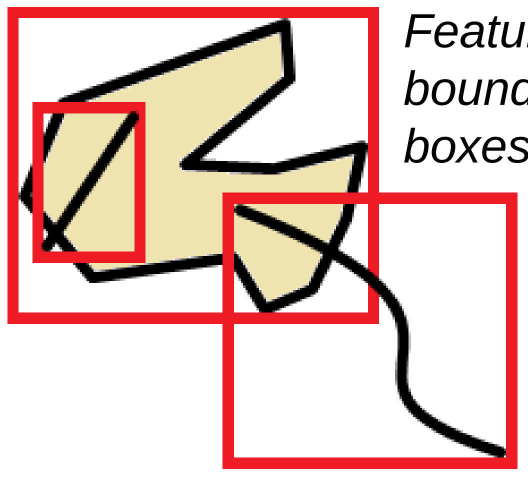
Bounding boxes

the smallest size rectangle containing a given feature

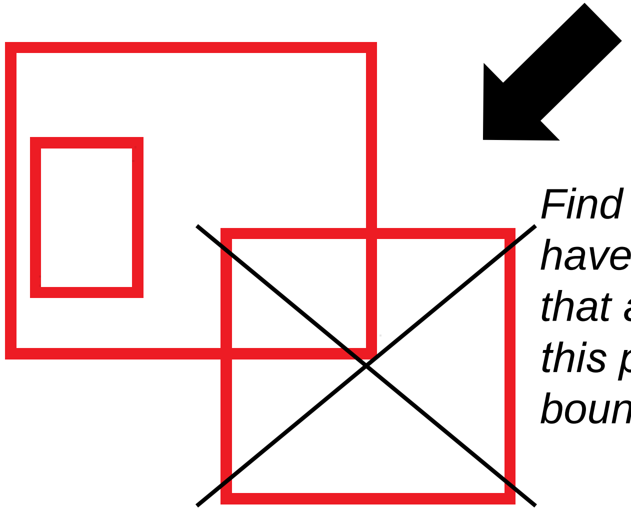
An example query:

“what lines are inside this polygon?”

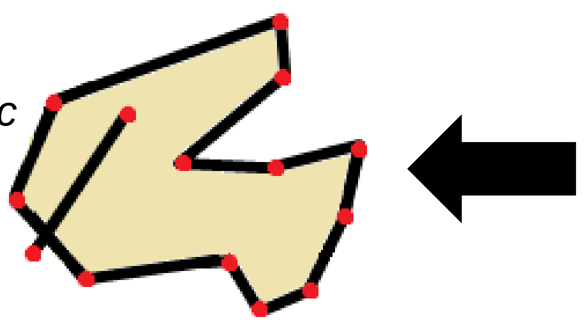
How is it evaluated?



Feature bounding boxes



Find the lines that have bounding boxes that are contained in this polygon’s bounding box



Then, test all those lines for exact geometric containment inside this polygon

Spatial databases
Spatial functions

“Conversion

Functions that convert between geometries and external data formats

Management

Functions that manage information about spatial tables and PostGIS administration

Retrieval

Functions that retrieve properties and measurements of a Geometry

Comparison

Functions that compare two geometries with respect to their spatial relation

Generation

Functions that generate new geometries from others”[1]

[1] Source: https://postgis.net/workshops/postgis-intro/introduction.html

Server-side application

Web map applications consume web services created by programs running on servers

Standards of the OGC Web Mapping Framework

Allows for interoperable data and processing services

An application server provides an interface to the functionality of the system

Web services

“A software system designed to support interoperable machine-to-machine interaction over a network” (W3C, 2004).

Geospatial web services
Web map service (WMS)

Web Map Service Interface Standard (WMS)

- "A simple HTTP interface for requesting geo-registered map images from one or more distributed geospatial databases"[1]
- "A WMS request defines the geographic layer(s) and area of interest to be processed" [1]
- "The response to the request is one or more geo-registered map images (returned as JPEG, PNG, etc.) that can be displayed in a browser application" [1]
- WMS supports a 'time' dimension
- WMS supports simple queries

[1] Source: http://www.opengeospatial.org/standards/wms

Geospatial web services
Web feature service (WFS)

"Web Feature Service allows a client to retrieve and update geospatial data encoded in Geography Markup Language (GML) from multiple Web Feature Services"[1]

- WFS allows vector data querying and retrieval
- Transactional WFS (WFS-T) allows clients to create, delete, and update features
- WFS allows more freedom for clients in using data

[1] Source: https://portal.opengeospatial.org/files/?artifact_id=8339

Server-side application

Geospatial data servers

E.g., GeoServer, MapServer, Deegree

Web frameworks

E.g., GeoDjango (Python), NodeJS (JavaScript)

Client-side application

A client-side application of an information system

- Renders contextualized information
- Allows the user's control of the application
- Allows the user to interact with the information system

Being the front-end with which the users interact makes it a primary determinant of the usability of the system

- Map applications can run in clients of varying *hardware* and *software* platforms
- Design and implementation consideration vary for different platforms

Thin or thick

- Based on analysis of the purpose and requirements of the system
- Software installation needed for thick clients
- Usually limited functionality in thin clients

Client-side application

OpenLayers

An open source JavaScript library that allows retrieving and rendering maps from multiple sources (web services) on the web

Leaflet

An open source JavaScript library for web mapping

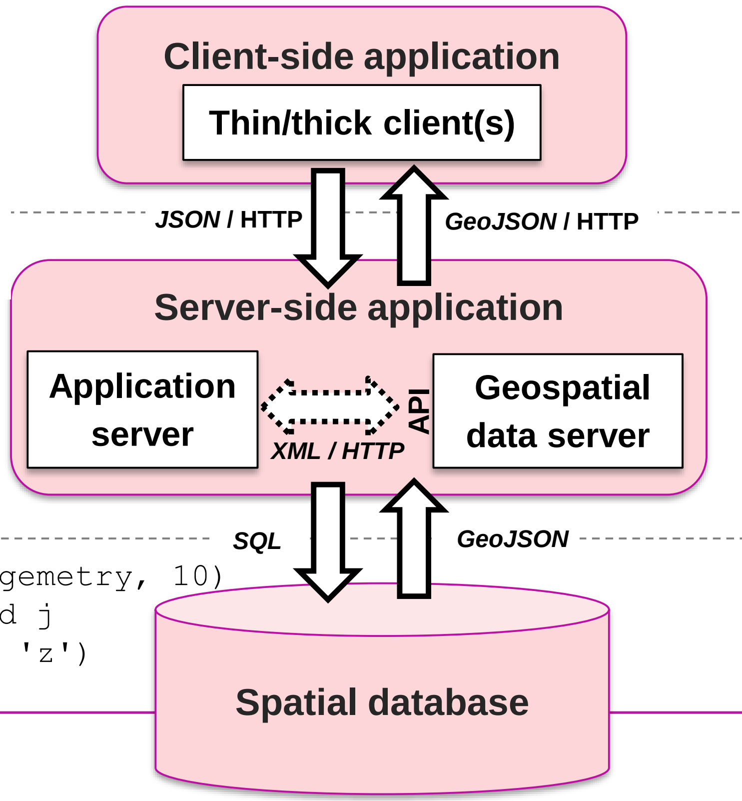
Mobile-friendly interactive maps

Example and demo…

Example client-server request-response


```
{"type": "FeatureCollection",
  "features": [
    {"type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [[x1, y1], [x2, y2]]
      },
      "properties": {
        "pipeType": "wastewater", "length": 80
      }
    }, …
  ]
}
```


http://hostIP/query?operation=intersection&layer1=pipes&layer2=roads&buffer=10&attributes=<*JSONObject_as_an_encoded_URL_component*>




```
SELECT row_to_json(MyGeoJSON)
FROM
(
  SELECT a.*
  FROM pipes AS a, roads AS b
  WHERE ST_DWithin(a.geometry, b.gemetry, 10)
    AND a.attributeX BETWEEN i and j
    AND b.attributeY IN('x', 'y', 'z')
)
AS MyGeoJSON
```

User selects two layers for checking intersecting features.

Backend script formulates a spatial query as defined by the user.

Database performs the query.

Client-side application renders the query result on the map.

Backend script sends query results to the client-side (*e.g.* as a WFS).

Database returns the query result.

Exercise walkthrough

In today's exercise, we will work with *PostGIS* and perform spatial queries. We will also set geospatial data services using *Geoserver*

- Database
  - *Create a spatial database in PostGIS*
  - *Load data into the DB*
  - *Perform spatial and non-spatial queries & continue the openness calculations*
  - *Visulaize query results using QGIS (a desktop GIS)*
- Geospatial web services
  - *Publish a query result as WMS and WFS using Geoserver*
  - *Retrieve published WMS and WFS in QGIS*

SQL

Structured Query Language

A standard language for storing, manipulating and retrieving data in databases

```
SELECT some columns                          ← Relational projection
  FROM some tables                           ← A DB relation
 WHERE ST_Contains() AND                     ← Spatial predicate
       columnX BETRWEEN value1 AND value2    ← Range predicate
```

Table: cities

id	name_fi	name_se	population	foundation_year
1	Helsinki	Helsingfors	642045	1550
2	Espoo	Esbo	277375	1972
3	Lahti	Lahtis	119395	1905
4	Vantaa	Vanda	221821	1974
5	Tampere	Tammerfors	230537	1779
6	Oulu	Uleåborg	201124	1605
…	…	…	…	…

Source: en.wikipedia.org/wiki/List_of_cities_and_towns_in_Finland

Geoprocessing

Procedure

- **Download the topographic data**
- **Create polygons of the sea area**
- **Create a grid of points**
- **Create radiating lines**

Prepare the data
- **Command-line tools**
- **Software library**

- **Import data into the database**
- **Clip and clean the radiating lines**
- **Aggregate the fetch lengths around each grid point**

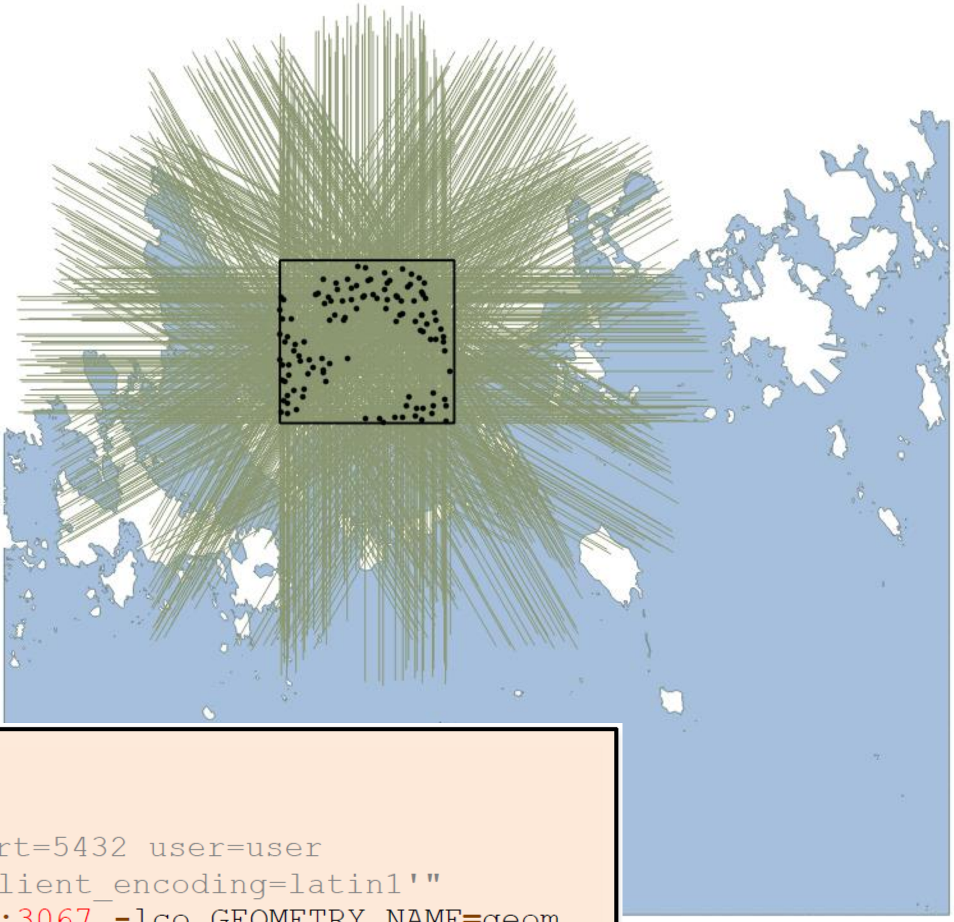
Do the computations
- **Spatial database**

Geoprocessing

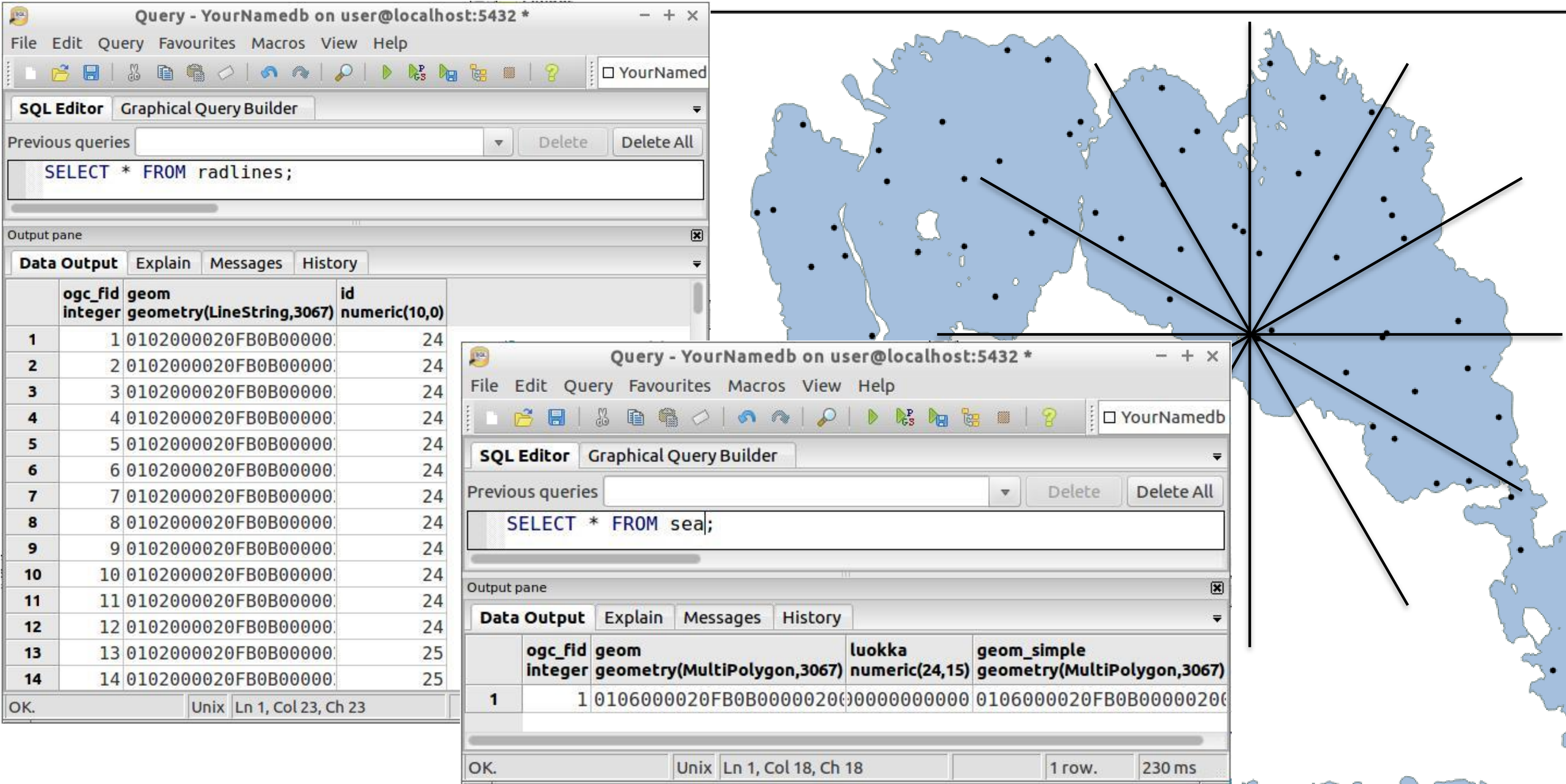
Procedure

Data to import into the DB

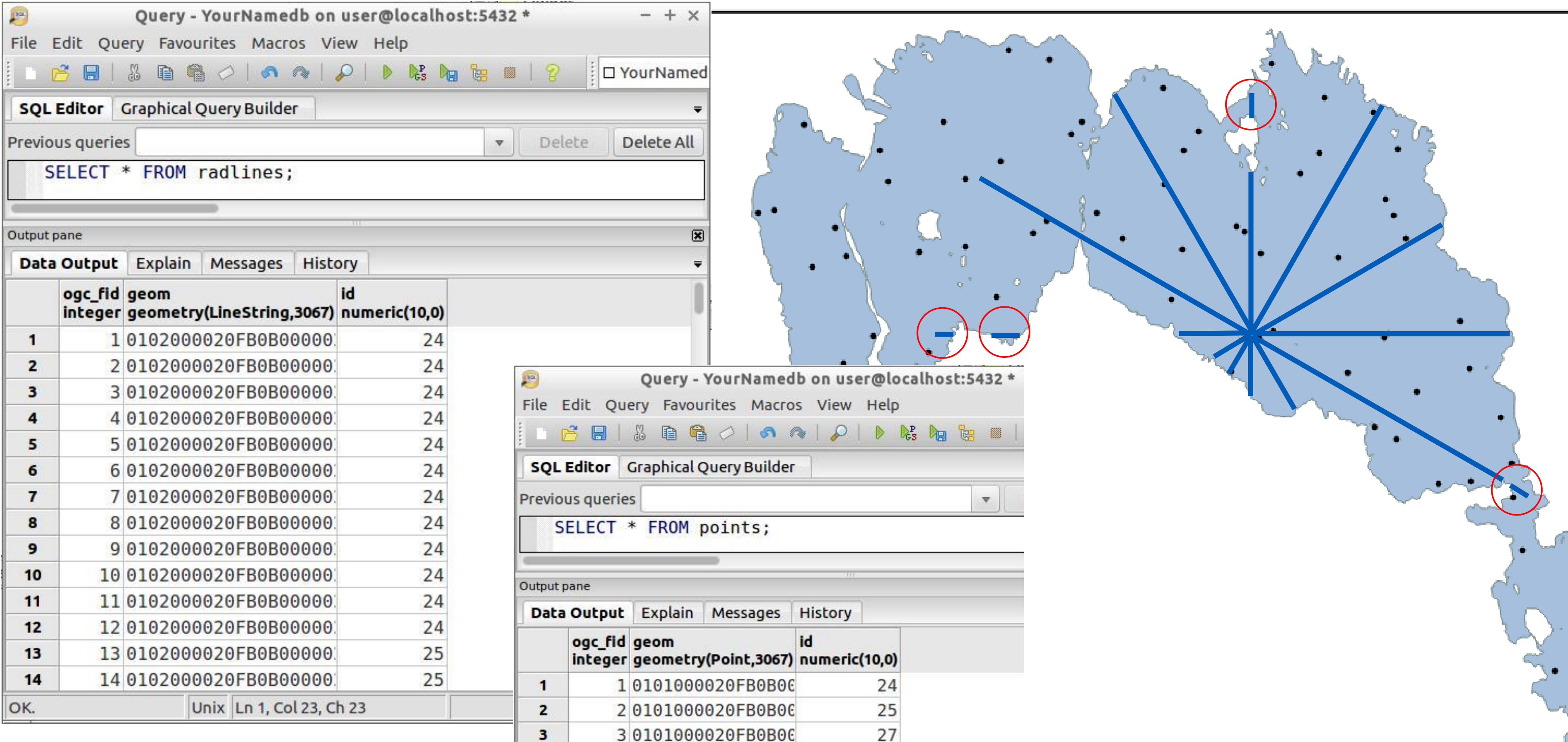
- Sea areas
- Grid of points
- Radiating lines



```
$ cd /home/user/YourDir/gis/data/input/

$ ogr2ogr -f "PostgreSQL" PG:"host=localhost port=5432 user=user
  dbname=YourNamedb password=user options='-c client_encoding=latin1'"
  points.shp -nln points -nlt POINT -a_srs EPSG:3067 -lco GEOMETRY_NAME=geom

$ cd /home/user/YourDir/gis/data/output/

$ ogr2ogr -f "PostgreSQL" PG:"host=localhost port=5432 user=user
  dbname=YourNamedb password=user options='-c client_encoding=latin1'"
  sea.shp -nln sea -nlt POLYGON -a_srs EPSG:3067 -lco GEOMETRY_NAME=geom
  -explodecollections

$ ogr2ogr -f "PostgreSQL" PG:"host=localhost port=5432 user=user
  dbname=YourNamedb password=user options='-c client_encoding=latin1'"
  rad_lines.shp -nln radlines -nlt LINESTRING -a_srs EPSG:3067 -lco
  GEOMETRY_NAME=geom
```



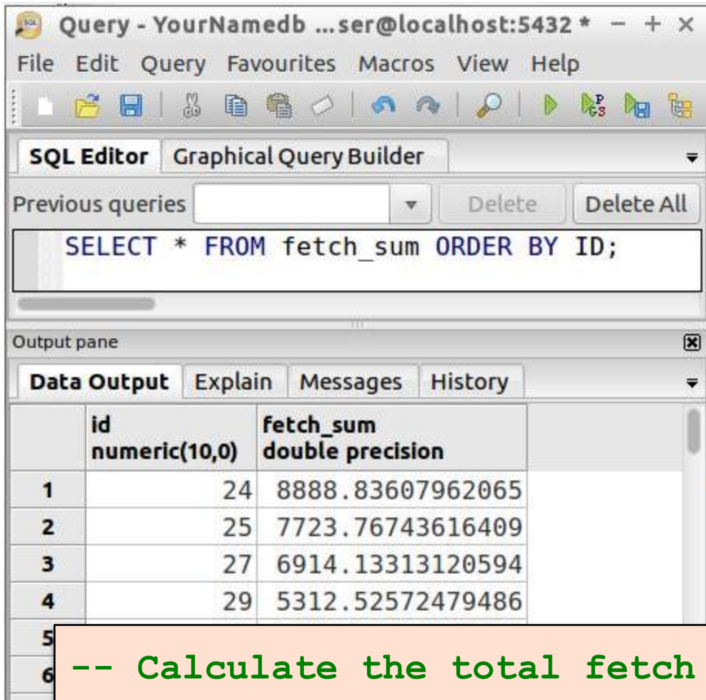

```sql
-- Clip the radiating lines with the sea polygons
CREATE TABLE radlines_clp AS
  SELECT (ST_Dump(ST_Intersection(s.geom, r.geom))).geom AS geom, r.ID
  FROM sea s, radlines r;
```

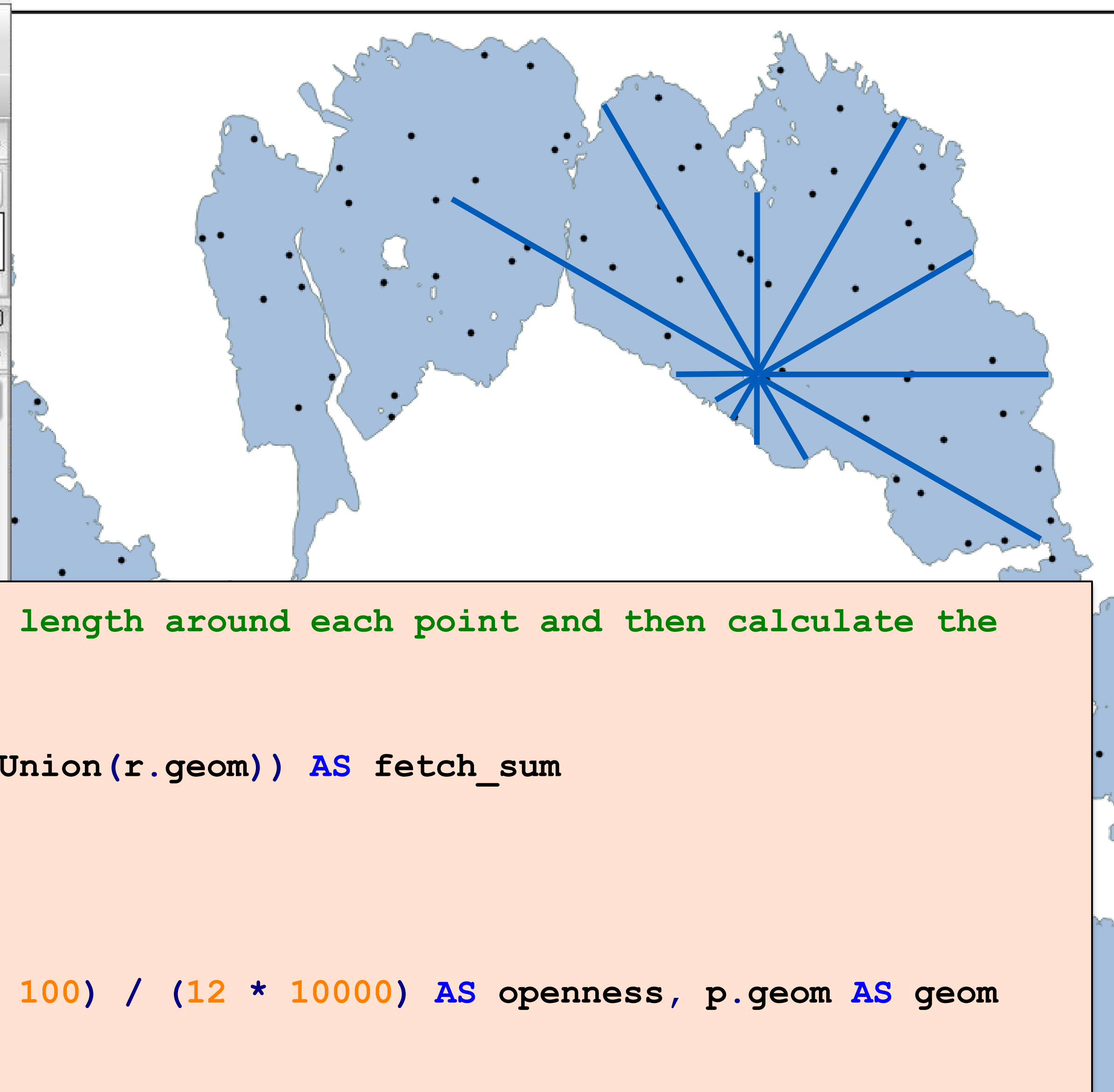


	ogc_fid integer	geom geometry(LineString,3067)	id numeric(10,0)
1	1	0102000020FB0B00000	24
2	2	0102000020FB0B00000	24
3	3	0102000020FB0B00000	24
4	4	0102000020FB0B00000	24
5	5	0102000020FB0B00000	24
6	6	0102000020FB0B00000	24
7	7	0102000020FB0B00000	24
8	8	0102000020FB0B00000	24
9	9	0102000020FB0B00000	24
10	10	0102000020FB0B00000	24
11	11	0102000020FB0B00000	24
12	12	0102000020FB0B00000	24
13	13	0102000020FB0B00000	25
14	14	0102000020FB0B00000	25

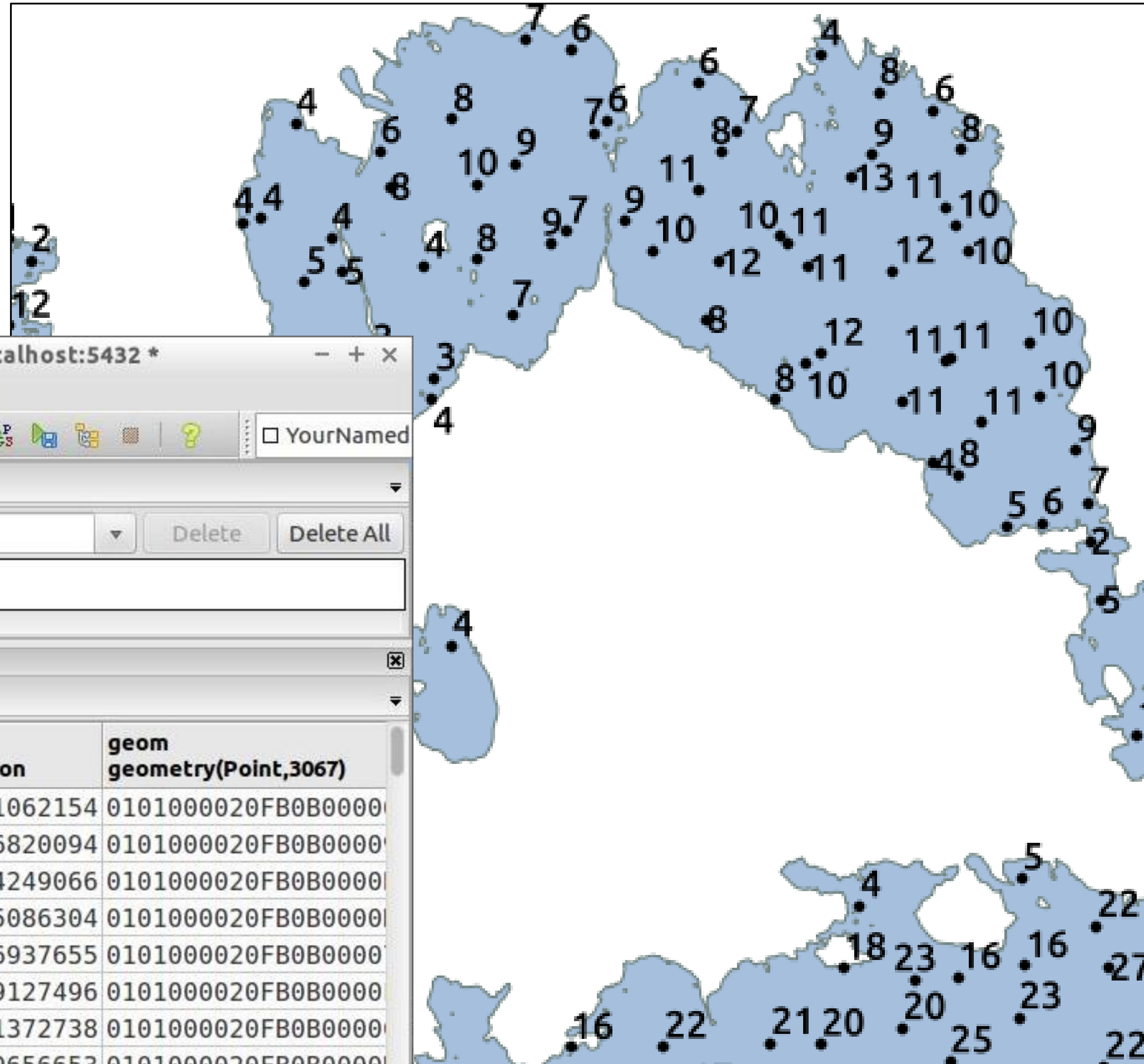
	ogc_fid integer	geom geometry(Point,3067)	id numeric(10,0)
1	1	0101000020FB0B00	24
2	2	0101000020FB0B00	25
3	3	0101000020FB0B00	27

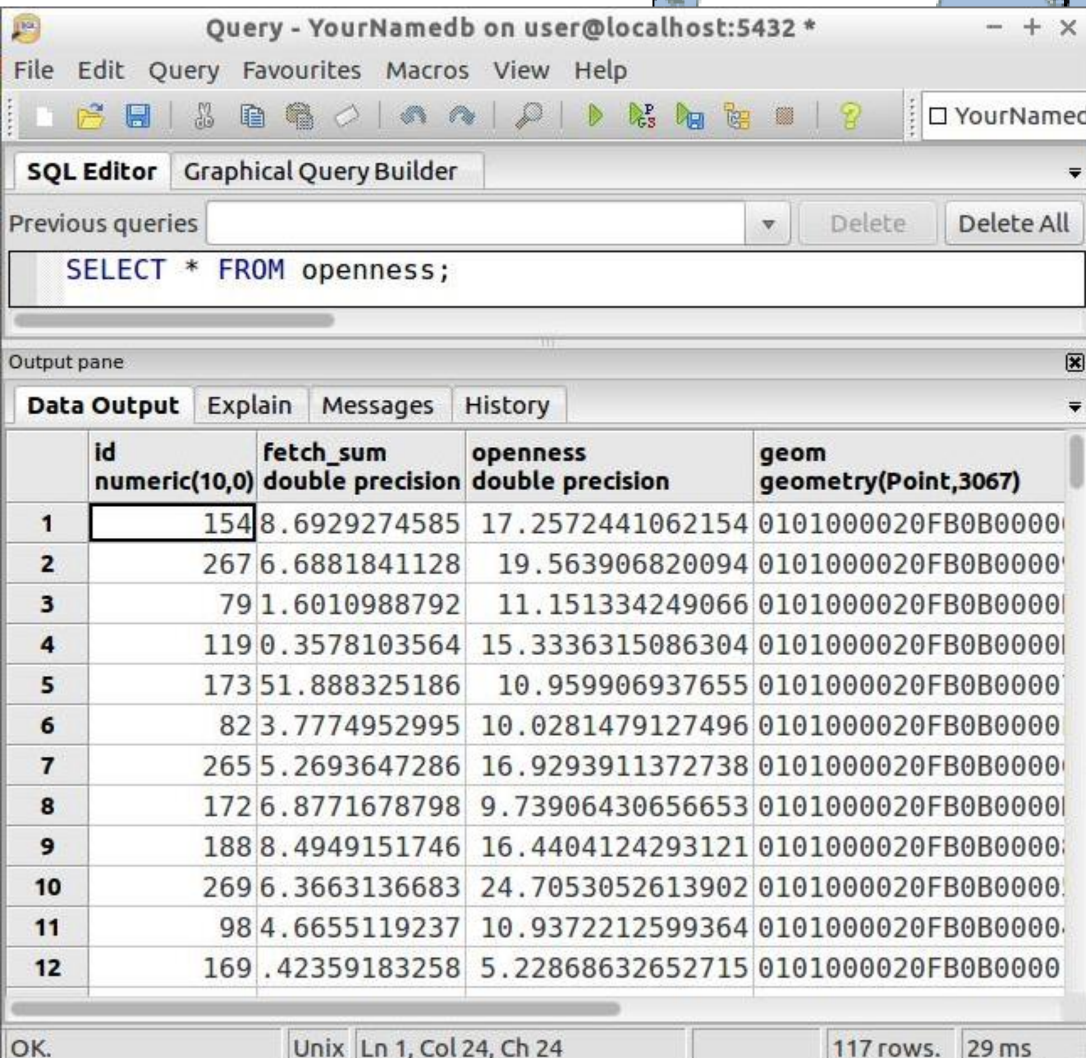
```
-- Remove all segments that don't intersect with their points of origin
CREATE TABLE radlines_cln AS
  SELECT r.ID, r.geom
  FROM radlines_clp r, points p
  WHERE ST_Intersects(r.geom, p.geom) AND r.ID = p.ID;
```





```
-- Calculate the total fetch length around each point and then calculate the
-- normalized openness
CREATE TABLE fetch_sum AS
  SELECT r.ID, ST_Length(ST_Union(r.geom)) AS fetch_sum
  FROM radlines_cln r
  GROUP BY r.ID;

CREATE TABLE openness AS
  SELECT f.*, (f.fetch_sum * 100) / (12 * 10000) AS openness, p.geom AS geom
  FROM fetch_sum f, points p
  WHERE f.ID = p.ID;
```





	id numeric(10,0)	fetch_sum double precision	openness double precision	geom geometry(Point,3067)
1	154	8.6929274585	17.2572441062154	0101000020FB0B0000
2	267	6.6881841128	19.563906820094	0101000020FB0B0000
3	79	1.6010988792	11.151334249066	0101000020FB0B0000
4	119	0.3578103564	15.3336315086304	0101000020FB0B0000
5	173	51.888325186	10.959906937655	0101000020FB0B0000
6	82	3.7774952995	10.0281479127496	0101000020FB0B0000
7	265	5.2693647286	16.9293911372738	0101000020FB0B0000
8	172	6.8771678798	9.73906430656653	0101000020FB0B0000
9	188	8.4949151746	16.4404124293121	0101000020FB0B0000
10	269	6.3663136683	24.7053052613902	0101000020FB0B0000
11	98	4.6655119237	10.9372212599364	0101000020FB0B0000
12	169	.42359183258	5.22868632652715	0101000020FB0B0000

Bonus exercise question…

Can we create the radiating lines (from previous lecture) using PostGIS functions instead of Python and PySAL?

→ Yes

How?

→ This is on you ☺

Hint: use `cross join`, `ST_MakeLine()`, `ST_MakePoint()`, and basic trigonometry