Karlsruhe Institute of Technology

PostGIS
Spatial Indices

Workshop

GEODETIC INSTITUTE, DEPARTMENT OF CIVIL ENGINEERING, GEO AND ENVIRONMENTAL SCIENCES, CHAIR IN GEOINFORMATICS

Spatial Indices

- In PostGIS, there are two possible ways to formulate spatial queries:
  - as *Operators* or
  - as *Functions*
- Whether two geometries e.g. *intersect* can be determined in two ways:
  - Operator: && or
  - Function: *st_intersects()*
  - The return value of *Operations* and *Functions* is a BOOLean variable

Operations and functions can be used in PostGIS

- The *operations* calculate the result based on the *Minimal Bounding Boxes* of the single geometries
  - Thus, these return only rough results of a query (*coarse filter*)
- The *functions* otherwise determine the results in two steps:

  1. The coarse filter is applied
  2. A fine filter performs an exact calculation of the query on the actual geometries

- *Operators* and *functions* work especially efficient if a *spatial index* on the spatial data has been created in advance

on "spatial index" see "Guttman 1984. R-Trees A Dynamic Index Structure for Spatial Searching."

- A spatial index can be created with the following SQL command:

```
CREATE INDEX index_name ON table_name
            USING gist(geometry_column);
```

- Even if a spatial index is created, it does not mean that PostgreSQL uses it in every search query

- PostgreSQL decides by itself in each case which search strategy probably is the fastest to provide the result

- For very small amounts of data, it is often not worth to use the index, since the administrative costs of the index are comparable high and a sequential search would lead to faster results

- However, for very large databases, the cost of searching usually lowers by a logarithmic factor with the assistance of an index

Excursus: Unit of costs [Page Size]

- The estimated cost of an SQL query in PostgreSQL is expressed in the number of accesses to disk pages
- A disk page typically consists of multiple physical blocks of the hard drive (or other external memory)
- The costs therefore indicate the number of disk page requests of the DBMS to the hard disk (the actual number of block accesses is much higher)
- To display the cost, it suffices to insert the EXPLAIN command before the actual request, e.g.:

```
EXPLAIN SELECT * FROM TABLE data WHERE id=42;
```

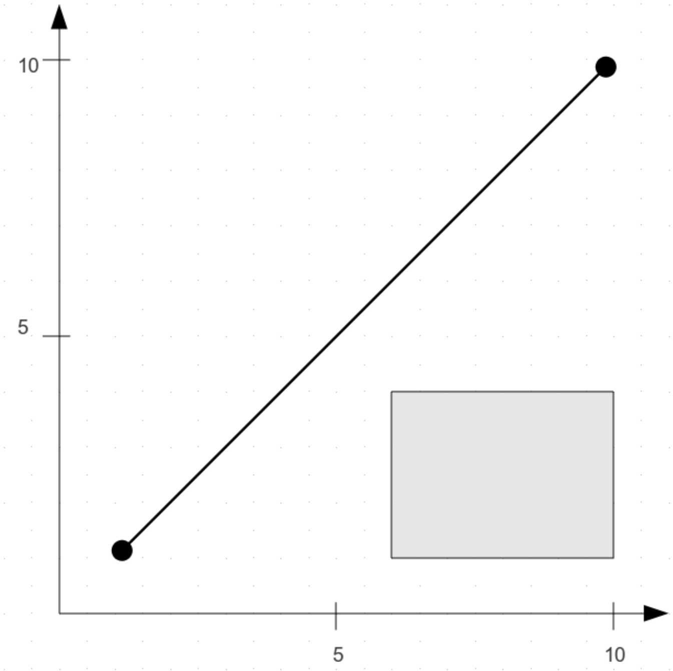
```
QUERY PLAN
-----------------------------------------------------------------
Seq Scan on table (cost=0.00..20.80 rows=1080 width=41) (1 Zeile)
```

- Then a "query plan" is returned, which contains two cost indicators, e.g. *cost = 0.00 .. 20.80*. The first value shows the *start-up/overhead* costs, the second is the *total cost*

Spatial Indices

Exercise 1: The figure on right shows the following geometries:

- *Line: [1 1], [10 10]*
- *Polygon: [6 1], [6 4], [10 4], [10 1]*



- Check the output of PostGIS on the question whether the two geometries *intersect*. Initially with an operator and subsequently with a function. How can the result be interpreted?

Exercise 2: Create a table *polygons2* that can accommodate features with type *POLYGON* geometry (with local coordinate system). Add the following four polygons into the table:

- Polygon *A*: *(0 0, 0 4, 4 4, 4 0, 0 0)*
- Polygon *B*: *(6 0, 6 4, 10 4, 10 0, 6 0)*
- Polygon *C*: *(0 6, 0 10, 4 10, 4 6, 0 6)*
- Polygon *D*: *(6 6, 6 10, 10 10, 10 6, 6 6)*

- Examine the table properties with the command `\d polygons2`. Create a *spatial index* on the polygons. Prompt again for `\d polygons2`. What changed, compared to the first invocation of the command?
- Query with the help of an operator for the polygon that *intersects* with the line *(6 6, 10 10)*. Also examine the cost of the query with the `EXPLAIN` command of PostgreSQL. What is the *total cost* of the query (in number of accesses on DB pages)? Which *search strategy* does PostgreSQL use?

Exercise 3: Prevent the PostgreSQL DBMS from using a *sequential search strategy* by prompting the PostgreSQL statement

```
SET enable_seqscan=off;
```

Now review the *total cost* of the same *intersection operation* (again by an *operator*). Which search strategy is chosen by PostgreSQL this time? Which of both search strategies is more efficient, considering this number of polygons?

Exercise 4: Create a new table from the *db_dump.sql* table dump. The DB now contains a table with the name *polygone3*. This table includes more than *10,000* polygons.

Let PostgreSQL calculate the costs for the following query: which polygon intersects the line *(-3861 11863, -3136 12976)*? Which *search strategy* is used by PostgreSQL automatically?

Now create a *spatial index* on the table and review the costs of the query. Which *search strategy* is used by PostgreSQL now? Compare the costs.