Spatial Databases II

Advanced Topics in Databases

Spatial Relationships

- So far we have only used spatial functions that measure (ST_Area, ST_Length), serialize (ST_GeomFromText) or deserialize (ST_AsGML) geometries. What these functions have in common is that they only work on one geometry at a time.

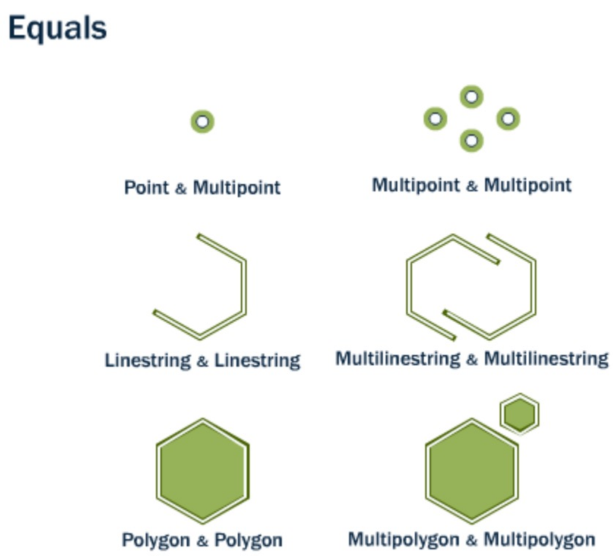
- Spatial databases are powerful because they not only store geometry, they also have the ability to compare relationships between geometries.

- Questions like “Which are the closest bike racks to a park?” or “Where are the intersections of subway lines and streets?” can only be answered by comparing geometries representing the bike racks, streets, and subway lines.

- The OGC standard defines several methods to compare geometries.

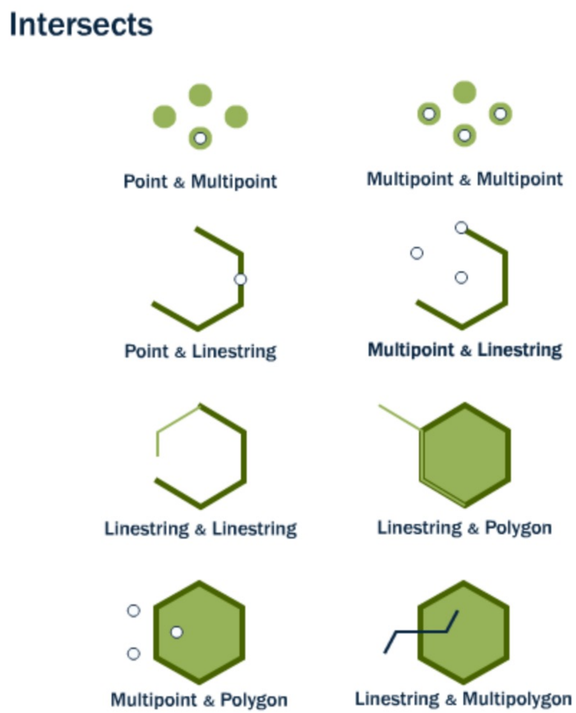
ST_Equals

- ST_Equals(geometry A, geometry B) tests the spatial equality of two geometries.
- ST_Equals returns TRUE if two geometries of the same type have identical x,y coordinate values, i.e. if the second shape is equal (identical) to the first shape.



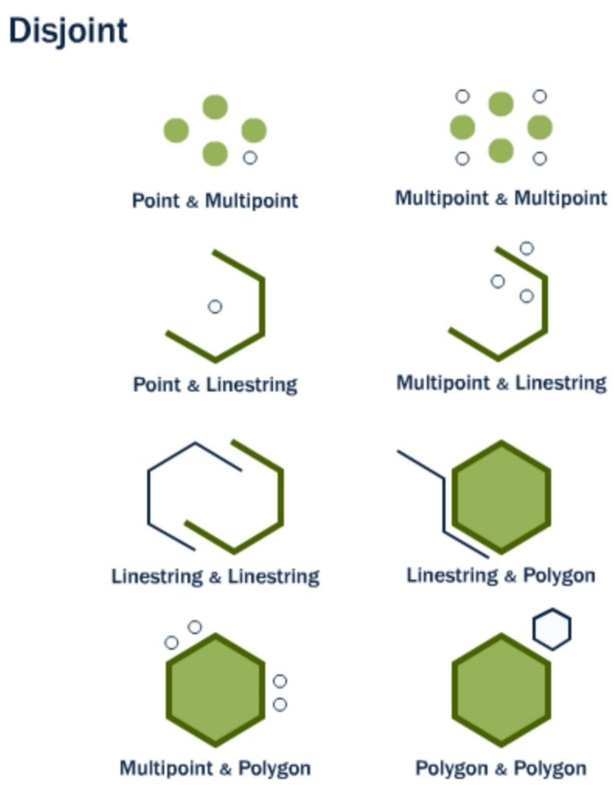
ST_Intersects, ST_Disjoint, ST_Crosses and ST_Overlaps

- ST_Intersects, ST_Crosses, and ST_Overlaps test whether the interiors of the geometries intersect.
- ST_Intersects(geometry A, geometry B) returns t (TRUE) if the two shapes have any space in common, i.e., if their boundaries or interiors intersect.



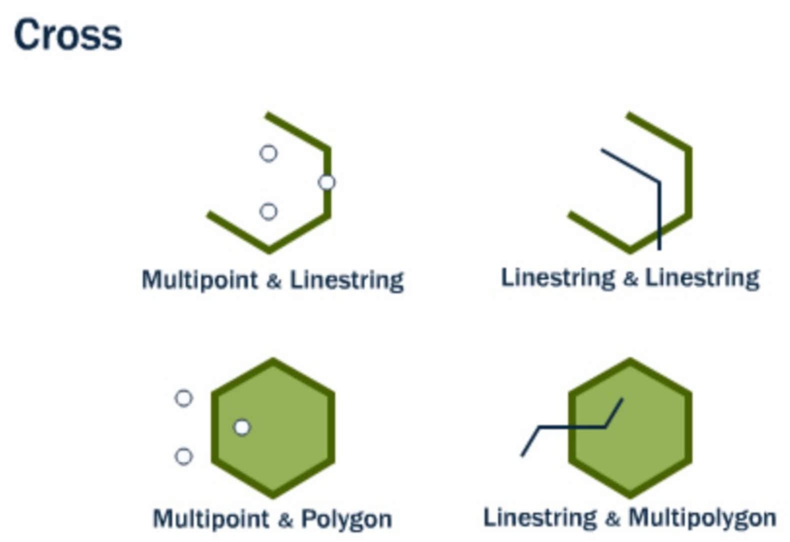
Disjoint

- The opposite of ST_Intersects is ST_Disjoint(geometry A , geometry B). If two geometries are disjoint, they do not intersect, and vice-versa.
- In fact, it is often more efficient to test "not intersects" than to test "disjoint" because the intersects tests can be spatially indexed, while the disjoint test cannot.



Cross

- For multipoint/polygon, multipoint/linestring, linestring/linestring, linestring/polygon, and linestring/multipolygon comparisons, ST_Crosses(geometry A, geometry B) returns t (TRUE) if the intersection results in a geometry whose dimension is one less than the maximum dimension of the two source geometries and the intersection set is interior to both source geometries.



Overlap

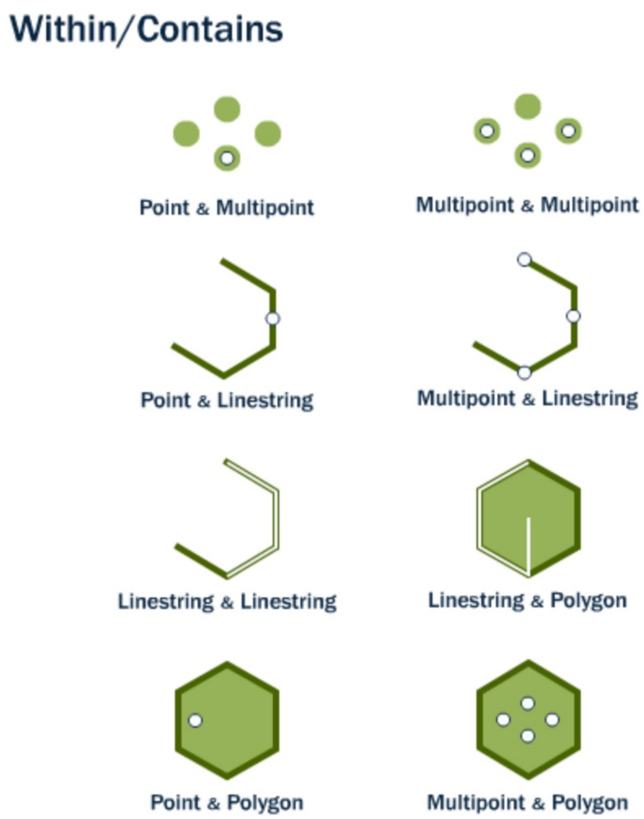
- ST_Overlaps(geometry A, geometry B) compares two geometries of the same dimension and returns TRUE if their intersection set results in a geometry different from both but of the same dimension.

Touch

- ST_Touches tests whether two geometries touch at their boundaries, but do not intersect in their interiors.
- ST_Touches(geometry A, geometry B) returns TRUE if either of the geometries' boundaries intersect or if only one of the geometry's interiors intersects the other's boundary.

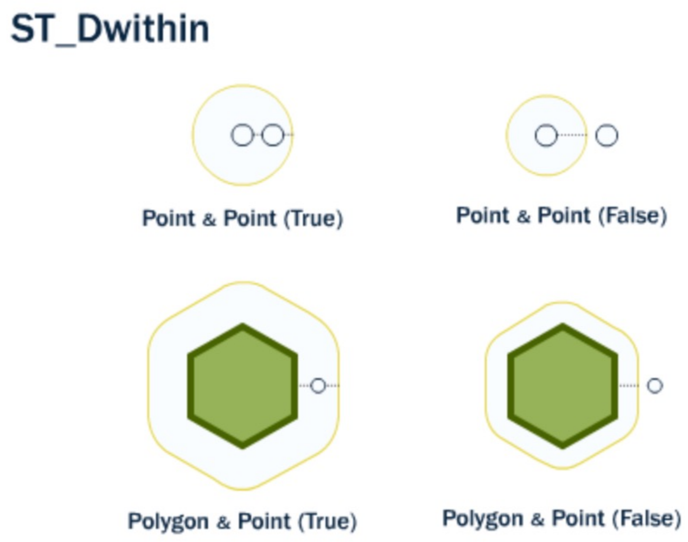
ST_Within and ST_Contains

- ST_Within and ST_Contains test whether one geometry is fully within the other.
- ST_Within(geometry A , geometry B) returns TRUE if the first geometry is completely within the second geometry. ST_Within tests for the exact opposite result of ST_Contains.
- ST_Contains(geometry A, geometry B) returns TRUE if the second geometry is completely contained by the first geometry.



ST_Distance and ST_DWithin

- The ST_Distance(geometry A, geometry B) calculates the shortest distance between two geometries and returns it as a float. This is useful for actually reporting back the distance between objects.
- For testing whether two objects are within a distance of one another, the ST_DWithin function provides an index-accelerated true/false test. This is useful for questions like “how many trees are within a 500 meter buffer of the road?”. You don’t have to calculate an actual buffer, you just have to test the distance relationship.



Function List

- **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.
- **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 Joins

- Spatial joins are a key concept in 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.

```
select name
from taxi_stands join cont_aad_caop2018
on st_within(proj_location,proj_boundary)
where freguesia='Ramalde';
```

The Taxi Dataset (adding 2 more tables)

- A table taxi_stands with the location of taxi stands in Porto.
- A table taxi_services with start and end data about taxi_services in Porto.
- A table tracks with Linestrings describing taxi trajectories in different states, having a second by second detail.
- A table cont_aad_caop2018 with Polygons, describing the administrative division of Portugal.

taxi_stands and taxi_services tables

```
mi=# \d taxi_stands;
                     Table "public.taxi_stands"
  Column  |          Type          | Collation | Nullable | Default
----------+------------------------+-----------+----------+---------
 id       | integer                |           | not null |
 name     | character varying(255) |           |          |
 location | geometry(Point,4326)   |           |          |
Indexes:
    "taxi_stands_pkey" PRIMARY KEY, btree (id)
```

```
mi=# \d taxi_services
                                         Table "public.taxi_services"
    Column     |         Type         | Collation | Nullable |                  Default
---------------+----------------------+-----------+----------+-------------------------------------------
 id            | integer              |           | not null | nextval('taxi_services_id_seq'::regclass)
 initial_ts    | integer              |           |          |
 final_ts      | integer              |           |          |
 taxi_id       | integer              |           |          |
 initial_point | geometry(Point,4326) |           |          |
 final_point   | geometry(Point,4326) |           |          |
Indexes:
    "taxi_services_pkey" PRIMARY KEY, btree (id)
```

tracks and cont_aad_caop2018 tables

```
mi=# \d tracks
                         Table "public.tracks"
   Column    |            Type             | Collation | Nullable | Default
-------------+-----------------------------+-----------+----------+---------
 id          | integer                     |           | not null |
 ts          | integer                     |           |          |
 taxi        | character(8)                |           |          |
 state       | character varying(6)        |           |          |
 track       | geometry(LineString,4326)   |           |          |
 proj_track  | geometry(LineString,3763)   |           |          |
Indexes:
    "tracks_pkey" PRIMARY KEY, btree (id)
```

```
mi=# \d cont_aad_caop2018
                                       Table "public.cont_aad_caop2018"
    Column     |          Type          | Collation | Nullable |                      Default
---------------+------------------------+-----------+----------+----------------------------------------------------
 gid           | integer                |           | not null | nextval('cont_aad_caop2018_gid_seq'::regclass)
 dicofre       | character varying(254) |           |          |
 freguesia     | character varying(254) |           |          |
 concelho      | character varying(254) |           |          |
 distrito      | character varying(254) |           |          |
 taa           | character varying(254) |           |          |
 area_ea_ha    | double precision       |           |          |
 area_t_ha     | double precision       |           |          |
 des_simpli    | character varying(254) |           |          |
 proj_boundary | geometry(Polygon,3763) |           |          |
Indexes:
    "cont_aad_caop2018_pkey" PRIMARY KEY, btree (gid)
    "cont_proj_boundary_idx" gist (proj_boundary)
```

Create table for tracks table

Note that tracks dataset requires that you previously create the table using the command:

```
create table tracks (
        id int primary key,
        ts int,
        taxi char(8),
        state varchar(6),
        track geometry(LINESTRING,4326));
```