Lab 3
Spatial Predicates

Learning Objectives:

- ✓ Using non-spatial predicates
- ✓ Combining spatial operators with non-spatial predicates
- ✓ Executing queries using pure spatial predicates (ST_CONTAINS)
- ✓ Analyse data types in PostgreSQL

Revision:

Queries can be issued from either **SQL-Shell** , **pgAdmin**, or **QGIS**.
This lab **investigates the semantics of non-spatial and spatial predicates**. Predicates such as `=, < , LIKE,` and `IS NOT NULL` are often used in the `WHERE` clause of an SQL query.

For full details of the spatial predicates (and operations) you should consult the PostGIS manual and the OGC standard. This lab will cover the basic function of QGIS.

Queries with a purely textual result should be run in the **SQL-Shell**/**pgAdmin**. Queries with a graphical output, or without st_astext in front should be typed into the query window of **QGIS**, which can be accessed via the main tool bar.

If geometry does not display after a query in **QGIS** then use Full Zoom  , as your result might be out of focus. That happens particularly often if you have run multiple queries for a while.
There are also several mailing lists for QGIS users. They are searchable too:
https://qgis.org/en/site/forusers/support.html#mailing-lists

1) Some non-spatial predicated

Type the following select statements into the SQL-Shell in and study the output.

```
select (true = cast( 1 as boolean));
select (2<1);select 'Mary-Ellen' LIKE 'Mary%';
select 'Mary-Ellen' LIKE 'Mary_____';
select 'Mary'  ILIKE  'mary';
```

When using the `LIKE` predicate an underscore (_) in pattern stands for (matches) any single character; a percent sign (%) matches any string of zero or more characters. `LIKE` is case sensitive, `ILIKE` is case insensitive,

2) Using spatial operations with non-spatial predicates.

```
select (ST_distance(ST_GeomFromText('POINT(0 0)'),
ST_GeomFromText('POINT(10 10)'))) > 10;
```

Lab 3

Spatial Predicates

```
select (ST_distance(ST_GeomFromText('POINT(0 0)'),
ST_GeomFromText('POINT (5 5)'))) > 10;

select ST_AREA(ST_GeomFromEWKT('POLYGON((10 20, 40 20, 40 30, 20 30, 40
35, 20 40, 40 40, 40 50, 20 50, 20 60, 70 60, 70 70, 20 70, 10 70, 10
20))')) > 100;
```

3) Spatial Predicates `ST_Contains`

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. The signature of `ST_Contains` is:

-- this is only a signature, not to be run

```
boolean ST_Contains(geometry geomA, geometry geomB);
```

In order for a geometry to contain another `geometry (ST_Contains(A,B)=true)`, or for a geometry to be within another geometry `(ST_Within(B,A)=true)`, the interior of A and the interior of B must intersect, and no part of B can intersect the exterior of A.

Example demonstrating difference between contains and contains properly, do Points, Lines and Polygons contain themselves and their boundaries?

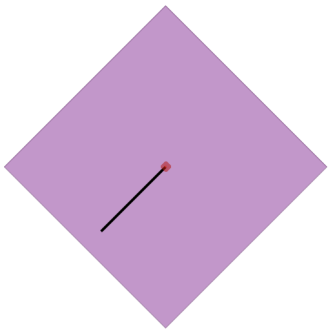
```
SELECT ST_GeometryType(geomA) As geomtype,
ST_Contains(geomA,geomA) AS "AcontainsA",
ST_ContainsProperly(geomA, geomA) AS "AcontainspropA",
ST_Contains(geomA, ST_Boundary(geomA)) As "AcontainsbdryA",
ST_ContainsProperly(geomA, ST_Boundary(geomA)) As "AcontainspropbdryA"
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
                  ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
                  ( ST_Point(1,1) )) As foo(geomA);
```

geomtype	AcontainsA	AcontainspropA	AcontainsbdryA	AcontainspropbdryA
ST_Polygon	t	f	f	f
ST_Line	t	f	f	f
ST_Point	t	t	f	f

(3 rows)

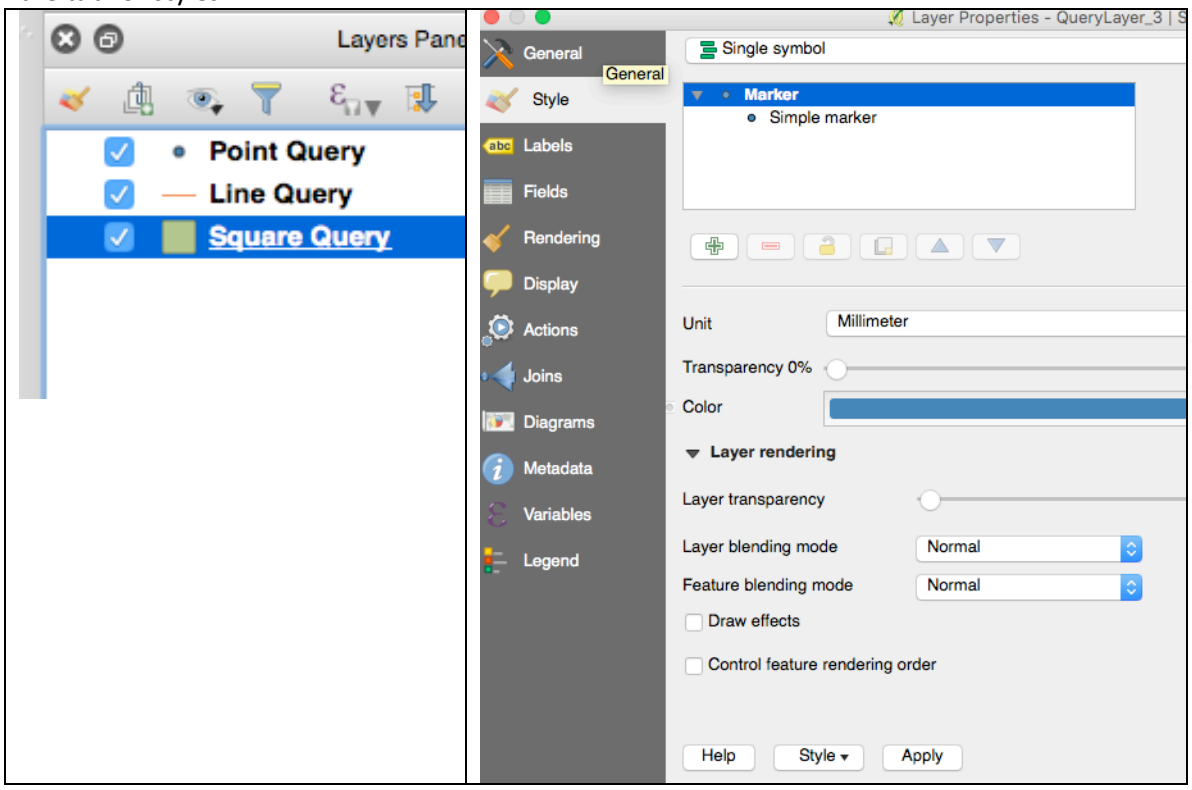
To draw the above shapes in QGIS.

```
SELECT ST_Buffer(ST_Point(1,1), 5,1);
SELECT ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1));
SELECT ST_Point(1,1);
```



Lab 3
Spatial Predicates

You will get 3 different layers in QGIS, which can be displayed in different colours. Right click or double click on the layer on the bottom left (unless you moved it somewhere else), and then use the tab for styles.



Next we look at a circle within a circle.

```
SELECT ST_Contains(smallc, bigc) As "smallConbig",
        ST_Contains(bigc,smallc) As "bigConsmall",
        ST_Contains(bigc, ST_Union(smallc, bigc)) as "bigConUnion",
        ST_Equals(bigc, ST_Union(smallc, bigc)) as "bigIsUnion",
        ST_Covers(bigc, ST_ExteriorRing(bigc)) As "bigCovExt",
        ST_Contains(bigc, ST_ExteriorRing(bigc)) As "bigConExt"
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
      ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
```

```
 smallConbig | bigConsmall | bigConUnion | bigIsUnion | bigCovExt | bigConExt
-------------+-------------+-------------+------------+-----------+-----------
 f           | t           |      t      | t          | t         | f
```

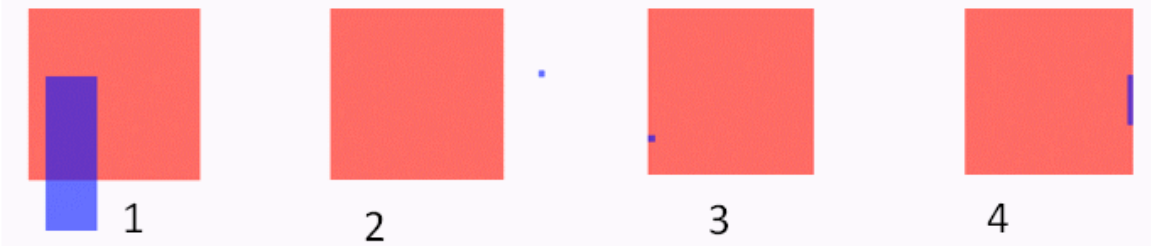
To draw the circles in **QGIS** use the following queries:

Lab 3
Spatial Predicates

```
SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) as smallc;
SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc;
```

You might have to move the layer with the small circle up to show both.

The following diagram represents some shapes that allow you to investigate how `ST_Contains` behaves.



For cases 1 and 2 the purple geometries are not completely within the red squares.
In cases 3 and 4 he purple geometries are only on the boundary of the red squares. ST_Contains does not consider the purple geometries to be contained by the red squares, even though they appear to be within them.

Using the following geometry simulate cases 1,2,3, and 4.

```
SELECT ST_MakeLine(ST_Point(20,12), ST_Point(20,15));
SELECT ST_MakePoint(5,7);
SELECT ST_MakePoint(23,17);
SELECT ST_PolygonFromText('POLYGON((5 5,20 5, 20 20, 5 20, 5 5))') as
red;
SELECT ST_PolygonFromText('POLYGON((10 0,15 0,15 15,10 15, 10 0))') as
purple;
```

For each case check the result of `ST_Contains` .

```
SELECT    ST_union(ST_GeomFromEWKT('LINESTRING(0    0,    10    10    )'),
ST_GeomFromEWKT('LINESTRING(0 9, 7 0)'));
```

Run the following two queries in the SQL Shell or pgAdmin:

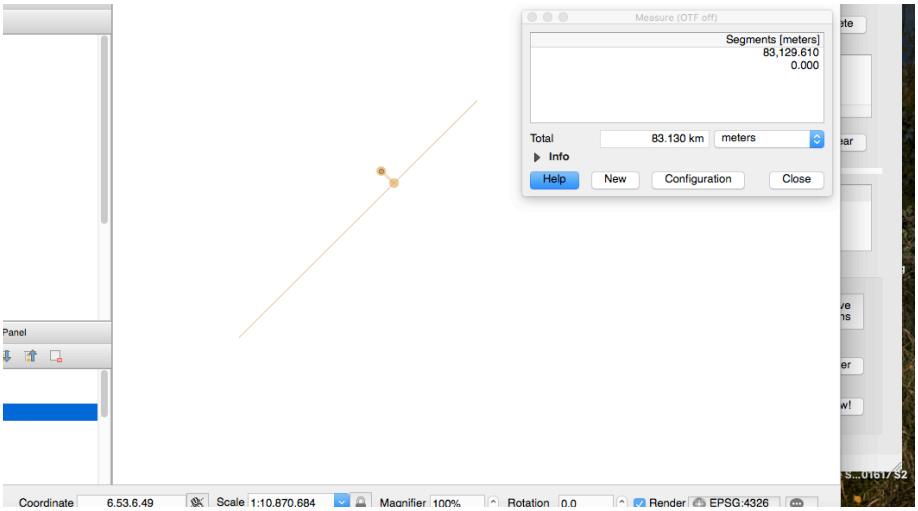
```
SELECT ST_DWithin(ST_GeometryFromText('LINESTRING(0 0, 10 10 )'),
ST_GeometryFromText('POINT(6 7)'), 1);

SELECT ST_DWithin(ST_GeometryFromText('LINESTRING(0 0, 10 10 )'),
ST_GeometryFromText('POINT(6 7)'), 0.5);
```

Using the PostGIS manual explain the results.
Confirm your explanation by viewing the data graphically in **QGIS**.

Lab 3
Spatial Predicates

```
SELECT ST_GeometryFromText('LINESTRING(0 0, 10 10 )');
SELECT ST_GeometryFromText('POINT(6 7)');
```

Use the **QGIS** measurement tool to check the distance between the point the line. You find the measuring tool under view > measure > measure line | measure area



Different orientation are considered equal.

```
select ST_Equals(
ST_GeomFromText('POLYGON ((0 0, 140 0, 140 140, 0 140, 0 0))'),
ST_GeomFromText('POLYGON ((140 0, 0 0, 0 140, 140 140, 140 0))')) AS
same;
```

We can force the orientation of the geometry .

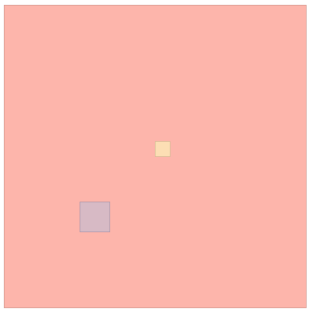
```
SELECT ST_AsEWKT(
  ST_ForceRHR(
      'POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1
2))'
  )
);
```

How do the input and output differ?

Polygons with holes are called **Rings** in PostGIS[i].

```
SELECT ST_AsText(b.the_geom) AS final_geom, ST_Area(b.the_geom) AS area
  FROM (SELECT (ST_DumpRings(a.geom)).geom AS the_geom
          FROM (
SELECT ST_PolyFromText(
'POLYGON((0 0,20 0,20 20,0 20,0 0),(10 10,10 11,11 11,11 10,10 10),(5
5,5 7,7 7,7 5,5 5))') AS geom ) a) b;
```

Lab 3
Spatial Predicates



4) Data types in PostgreSQL

SQL is a strongly typed language. This means that any piece of data represented by PostgreSQL has an associated data type, *even if it is not plainly obvious*. A data value's type both **defines and constrains the kinds of operations which may be performed on it**.

In general you can evaluate expressions using the `select` keyword.
For example, we can add two numbers as follows;

```
select 7 + 6 as result;
```

The string concatenation operator (||) combines two strings. The string concatenation operator (||) accepts non-string input, so long as at least one input is of a string type:
For full details of PostgeSQL string operations see:
http://www.postgresql.org/docs/current/static/functions-string.html

When the system changes a data type it is called automatic casting (aka autocasting or coercion).

This example shows how the system automatically casts the string '12' to the integer 12.

```
SELECT 1 + '12' AS add_one_to_twelve;
```

However the system will not automatically cast when both arguments of + are strings.

```
SELECT '1' + '12' AS add_one_to_twelve;
```

We can explicitly change a string to an integer and then add the converted result.

```
SELECT 1 + ('1' || '2')::integer AS add_one_to_twelve;
```

Which can also be written:

```
SELECT 1 + CAST('1' || '2' AS integer) AS add_on_to_twelve;
```

Explain the following statements and their outputs.

```
Select 162 / 1097 ;
select 1097 / 162;
select 1097::float / 162::float;
select cast(1097 as float) / cast(162 as float);
select 1097.0 / 162.0;
```

Lab 3
Spatial Predicates

Auto-casting in PostgreSQL/PostGIS
In general it is not a good idea to rely on automatic casting for spatial operations(auto-casting). The term text is often as a synonym for string. If a string representation of a geometry is used in a function that expects a geometry, then we rely on the system to do the correct conversion. While this is convenient, the system may not always produce the expected results. Below is a demonstration of such a case:

```
SELECT ST_Centroid('LINESTRING(1 2,3 4)');
SELECT ST_AsText(ST_Centroid('LINESTRING(1 2,3 4)'));
```

There is an autocast built into PostGIS that takes a string and converts it to a geometry data type automatically. It is better to use explicit casting using the function `ST_GeomFromText`.

```
SELECT ST_Centroid(ST_GeomFromText('LINESTRING(1 2,3 4)'));
```

The next example will fail because there is no auto-cast that will convert the text representation to actual geometry (in this case box3d).

```
SELECT ST_Xmin('LINESTRING(1 2, 3 4)');
SELECT ST_Xmin(ST_geomFromText('LINESTRING(1 2, 3 4)'));
```

Make a geometry collection:

```
SELECT AsEWKT(ST_Collect(Array[ST_MakePoint(0, 0), ST_MakePoint(10, 10),ST_MakePoint(15, 10)]));
```

Here we look at the difference between st_envelope (like a box) and convex hull (like rubber band enclosure). Run the queries in **QGIS**.

```
SELECT ST_Collect(
ST_GeomFromText('MULTILINESTRING((100 190,10 8),(150 10, 20 30))'),
ST_GeomFromText('MULTIPOINT(50 5, 150 30, 50 10, 10 10)'));

SELECT ST_ConvexHull(ST_Collect(
ST_GeomFromText('MULTILINESTRING((100 190,10 8),(150 10, 20 30))'),
ST_GeomFromText('MULTIPOINT(50 5, 150 30, 50 10, 10 10)')));

SELECT ST_envelope(ST_Collect(
ST_GeomFromText('MULTILINESTRING((100 190,10 8),(150 10, 20 30))'),
ST_GeomFromText('MULTIPOINT(50 5, 150 30, 50 10, 10 10)')));
```

[i] http://www.spatialdbadvisor.com/postgis_tips_tricks/92/filtering-rings-in-polygon-postgis
http://postgis.net/docs/ST_DumpRings.html