Université de Liège
Faculté des Sciences Appliquées

Tutorial on PostGIS
A Support for Geospatial Data Storage, Processing & Analysis.

INFO9016-1 - Advanced Databases

Authors
ABDELALEEM Aly - s206111
DELPORTE Guillaume - s191981
HOORELBEKE Jordi - s175615

June 4, 2023

Contents

1 Introduction

PostGIS is an extension to the PostgreSQL object-relational database that allows you to work with Geospatial data, which is a type of data that conveys information about coordinates and locations such as Maps and GPS. What differentiates PostGIS from other databases is the following three key features:

1. Spatial data types, that reference geometric shapes such as points, lines, and polygons.
2. Spatial indexing, which is utilized to make spatial queries fast.
3. Spatial functions, written in SQL to query based on attributes and relationships.

In addition, PostGIS contains a lot of other features and spatial functions that we will dig deeper into in the next sections. Moreover, the fact that it is built on top of PostgreSQL combining both features gives us a powerful geographic information system (GIS), which is a software tool designed to capture, store, manipulate, interpret, and visualize Geospatial data. As PostGIS deals with spatial data it has to have types to support such data types, in fact, PostGIS has the following four spatial data types:

1. Geometry, a data type for Cartesian coordinates such as Planar geometric shapes.
2. Geography, a data type for spherical coordinates such as Spherical geometric shapes.
3. Raster, a data type for grid data such as Satellite images.
4. Topology, a data type for connected nodes and edges such as Train Networks.

PostGIS is open source under GPL v2 license, meaning that it is free to download and one is able to check and change the source code, which allows for customization along with a lot of add-ons and free updates from the community. The fact it is open source also means that it is free and relatively cheap compared to other GIS such as Esri's ArcGIS and Oracle's Spatial GIS.

PostGIS loads data in the form of a shapefile, which is the default format for saving spatial data. A shapefile is normally composed of four files: $.shp$, $.shx$, $.dbf$, and $.prj$, which respectively contain the shape geometry, index, attributes, and coordinate system information. It also supports importing/exporting data with standard textual formats such as KML, GML, GeoJSON, GeoHash and WKT using SQL. It can also include a Spatial reference systems (SRID) which is a unique identifier that reference a particular coordinate system, tolerance, and resolution. In other words, it summarizes the coordinate system measurements and units into a number that can be referenced.

Our tutorial is structured as follows:

- We will discuss PostGIS's aforementioned three key features in detail as well as highlight the others 2.
- Then we will compare the benefits and drawbacks of PostGIS 3.
- Afterwards, we will present real-world examples 4.
- Followed by a conclusion 5.
- And last but not least, a beginner's tutorial on how to use MongoDB 6.

2 Features

In this section we will focus on two features PostGIS add on top of the already existing features available with the underlying database, such as PostgreSQL: The spatial data types a spatial database needs and the functions made available to make computations on these data types.

2.1 Spatial data types

In an ordinary database, we have data types to represent simple objects such as numbers and text, but we have no way of representing space properties[1]. Therefore, a spatial database such as PostgreSQL with the PostGIS extension provides supplementary data types for representing points, lines, shapes, dimensions, etc. Spatial data types are hierarchical in their structure, where each sub-type inherits the shape and the behaviour of its parents [See figure 1]. Now we will discuss the four spatial data types that exist in a PostGIS database.

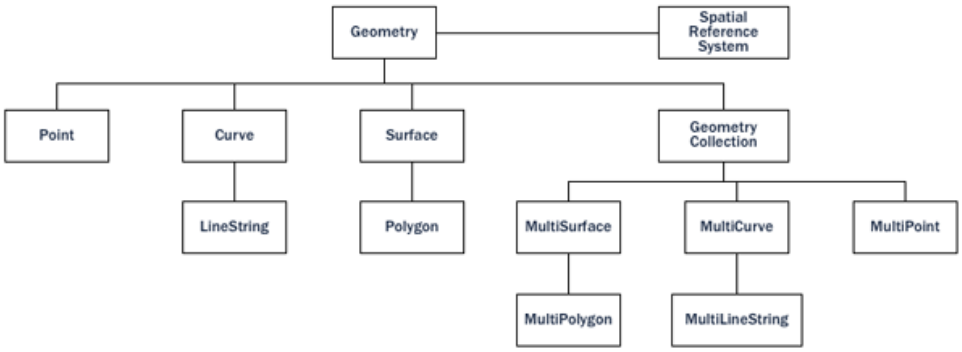


Figure 1: Retrieved from The official PostGIS documentation.

2.1.1 Geometry

The geometry data type is there to help you store basically any 2D shapes such as Points, Lines, Polygons and many others. The geometry data type uses only Cartesian coordinates x-axis and y-axis and that is enough as it only deals with 2D shapes. In the above figure 1, we can see the hierarchy and all the 2D shapes that can be constructed in PostGIS.

2.1.2 Geography

To map real-world points we use spherical coordinates also known as latitude and longitude, hence there is a need for the geography data type as it is able to store any coordinates and gives more accurate descriptions and measurements for points on the surface of the earth. On the other hand, it is computationally very heavy to do calculations with it and some of the spatial functions do not even support this data type, so one will have to cast them into the geometry data type, which is a supported functionality of PostGIS.

[1] At least not in a trivial way.

2.1.3 Raster

The advancement in satellite imagery technology allowed us to take high-quality images of the earth. The raster data type takes advantage of these images and maps them digitally. This of course saves a huge amount of time rather than entering the data manually but also allows us to convert them to the geometry data type to apply further processing on them if needed.

2.1.4 Topology

This data type is used for the storage of typologies such as Train underground networks, city streets, etc. The idea here is that sometimes we need to store the relations between points as well as the points themselves, it doesn't care about the measurements only, the relationship. This data type gives you the ability to do so.

2.2 Spatial functions

In addition to providing various data types, it offers a range of functions that can be grouped into five categories. **Creation** and **retrieval** functions in order to be able to add new spatial data to the database as well as retrieve information about existing data. **Conversion** functions in order to be able to import and export data. **Management** functions for PostGIS administration. **Comparison** functions to perform all sorts of comparisons between the spatial data such as size, distance, and localization. Finally, **generation** functions to generate new geometries from others. All the former functions are defined by the Open Geospatial Consortium (OGC) and implemented by PostGIS.

3 Benefits & Drawbacks of PostGIS

3.1 Is PostGIS what you need?

As we have seen before, PostGIS is an open-source, object-relational database system that adds support for geographic objects, allowing for the storage, querying, and manipulation of spatial data. In this section, we are going to list the main benefits and drawbacks of using PostGIS compared to traditional relational databases. We think that by understanding the strengths and weaknesses of PostGIS, one can evaluate whether it is the right fit for their specific use case, budget, and technical requirements.

3.2 Main benefits

The first benefit is that PostGIS is made to handle spatial data. As such, it will be very robust for spatial geographic queries. This makes it ideal for mapping and location-based applications.
PostGIS is also stocked with advanced spatial functions: PostGIS provides a rich set of spatial functions and operators that allow for advanced spatial analysis and processing as a function to compute distances `ST_Distance` in meters, miles or kilometres between two geographic objects, such as points, lines, or polygons.
Furthermore, PostGIS is open source. Even if this has some drawbacks, this means that it is free to use and can make it really appealing for small companies that cannot afford high licensing costs. It can also seamlessly integrate with other open-source applications such as GeoServer. As such, PostGIS supports a wide range of industry-standard data formats, including shapefiles and GeoJSON that really ease the importation and exportation of data to other GIS applications.
Another benefit of PostGIS is that PostGIS is efficient indexing for spatial data. It implements Spatial Indexing that greatly improves spatial query efficiency. It is based on an R-Tree spatial data structure that recursively breaks up data into rectangles, subrectangles, etc. That creates a tree that can be efficiently checked before making some exact calculations. When there is a lot of data, this two steps algorithm can radically reduce the amount of computation necessary to answer a query.
An important benefit is that PostGIS is built on top of the PostgreSQL relational database, which means that it inherits all of its features and flexibility, including the use of classical SQL queries and advanced data types.

3.3 Main Drawbacks

Every system has its drawbacks, but PostGIS does not break the rule. It has several drawbacks which could be considered a deal-breaker.
Firstly, even though PostGIS has its implementation of spatial indexing, it can be a real computational hog and can face some performance issues in certain situations. This means that PostGIS needs some pretty good hardware to run on if you plan on using large data sets. This can particularly be a deal-breaker for someone or some organization with limited funds.
Secondly, PostGIS requires specialized knowledge to set up and use effectively. Because of that, it has a pretty steep learning curve. In fact, it uses a specialized set of functions and operators, making it hard to tame for developers who are not familiar with GIS concepts or terminology. Our tutorial will however help in the start of learning about GIS.

Thirdly, Because of PostGIS's special data structures, communication with a more classical relational database can be a bit difficult and one might need to develop some data handlers. For example, some applications may not be designed to work with spatial data or may have limitations on the size or complexity of the spatial data they can handle. In these cases, there may be a need to build custom data connectors or use specialized tools to integrate PostGIS with these applications. Last but not least, PostGIS is open-source software. As we said before, even if this has some benefits, one of the main drawbacks is that, unlike paid software, it has no vendor support. This can be a real problem to troubleshoot any inconvenience. However, the community is very active and might help through troubleshooting.

3.4 Side note

In this section, We claimed the main drawbacks and benefits that we thought were important. There might be more benefits or inconveniences depending on you're use cases. Some more niche benefits can be expressed. For example, PostGIS is capable of Time-series Analysis. By indexing some spatial data with time stamps, it becomes possible to retrieve and analyze data at a certain place for a certain time period. This can be really useful for weather forecasting and climate modelling for example.

4 Real World Examples

The fact that most of the earth's surface is now mapped digitally and available as well as the locality is now seen as a very important commodity this opens the door to unlimited potential in different industries with regard to the value that PostGIS can add. We will explore two real-world examples of how PostGIS can be used and how to exploit its features to create value across multiple different industries and gain competitive advantage, also bear in mind that the idea here is not only finding new ways for which PostGIS can be used because the upcoming examples can be implemented using normal databases but to make those implementations more efficient, easy and inter-operable with other systems, as we saw in 3 one of the benefits of PostGIS is spatial indexing which enables a fast spatial query compared to a relational database, so the examples might not be groundbreaking or something new but implementing them in PostGIS could be the start.

4.1 Retail industry

In this example we will show how PostGIS can be useful in the retail industry we can take for example a clothing store/fast food chain with multiple stores and points of sale, and then such stores can use PostGIS spatial functions to calculate the demands hot-spots or even can be used by the customer to find the nearest point of sale, also using QGIS which a tool that enables the visualization of spatial data, can give decision makers a clearer picture of the current situation and improve the decision-making process, for example, taking advantage of the topology data type in PostGIS can enable them to take a decision on where the next store should be opened to ensure connectivity and fulfil demands in an efficient way, also mapping the locations of the warehouses or distributors against the stores' locations along with calculating the distance between them can help assign which store can use which warehouse/distributor leading to more efficient logistical operations. That being said we can see that the sky is the limit with regard to the use of PostGIS in the retail industry.

4.2 Fitness Industry

In this example we will show how PostGIS can be useful in the fitness industry we can take for example the GPS in our phones/fitness watches and we can store these coordinates in the PostGIS database and apply some spatial functions that can be useful to the user, for example, we can calculate the following:

- Average speed
- Distance walked
- Step count
- Flights climbed
- Visited locations

These data can help track the fitness level and can even give some indication on some health issues as well as it can be used to calculate other fitness components such as walking asymmetry, burnt calories and many more. Again as we saw from the retail industry example 4.1 the potential is unlimited depending on the business needs and goals.

5 Conclusion

In conclusion, PostGIS is a powerful open-source extension to the PostgreSQL database that allows for the storage, querying, and manipulation of spatial data. Its key features include spatial data types, spatial indexing, and spatial functions, making it ideal for mapping and location-based applications. PostGIS also provides advanced spatial functions and supports a wide range of industry-standard data formats. While its integration with PostgreSQL and open-source nature provides significant benefits, its performance can be a potential drawback and may require good hardware to run on in certain situations.

Now, let us discover the power of spatial databases with PostGIS by following our tutorial and gaining valuable hands-on experience working with geographic data.

6 Tutorial

This tutorial will guide you through the process of working with spatial data in PostGIS using two popular tools: pgAdmin and QGIS. We will be using the World Countries (Generalized) shapefile dataset, which contains generalized boundaries for the countries of the world as of August 2022. You will learn how to set up a PostGIS environment using a Docker image, import the shapefile into the database, and perform spatial queries and analysis using SQL in pgAdmin and QGIS. To demonstrate the capabilities of PostGIS, we will add an imaginary country as well as cities to the dataset and query the database to obtain information on world metrics. This exercise will enable you to apply your new skills in a practical setting, gaining hands-on experience in creating new geometries, adding new data to the database, and querying the data to obtain relevant information. By the end of this tutorial, you will have a basic understanding of how to work with spatial data in PostGIS, and how to use pgAdmin and QGIS to interact with a PostGIS database.

6.1 Prerequisites

To follow this tutorial, it is recommended that you have a basic understanding of relational databases, SQL, and the PostgreSQL relational database management system. If you are new to these concepts, we recommend that you familiarize yourself with them before proceeding. In addition, you will need to have Docker and pgAdmin[2] installed on your computer. It is also optional but recommended to install QGIS for visualizing spatial data.

6.2 Setting up PostGIS

6.2.1 Deploy the database server

To simplify the process of setting up PostGIS and remove unnecessary overhead[3], we will deploy the database server inside a Docker container using a pre-built Docker image.

Follow the steps below to set up the environment:

1. Download the Docker image from the Docker Hub repository to your local machine

```
docker pull jordih2o/postgis-tutorial
```

2. Run the container on port 54320 in detached mode

```
docker run -p 54320:5432 -v postgis-tutorial:/var/lib/
    postgresql/data -d --name postgis-tutorial jordih2o/
    postgis-tutorial
```

   This command will create a new container named "postgis-tutorial" running on port 54320 and map it to the default Postgres port 5432. It will also create a new volume named "postgis-tutorial" to store the data.

[2]This tutorial assumes that you are using pgAdmin on your local computer. However, if necessary, you can also run pgAdmin inside a Docker container and connect it to the database container via a Docker network. For more information, please consult the relevant documentation.

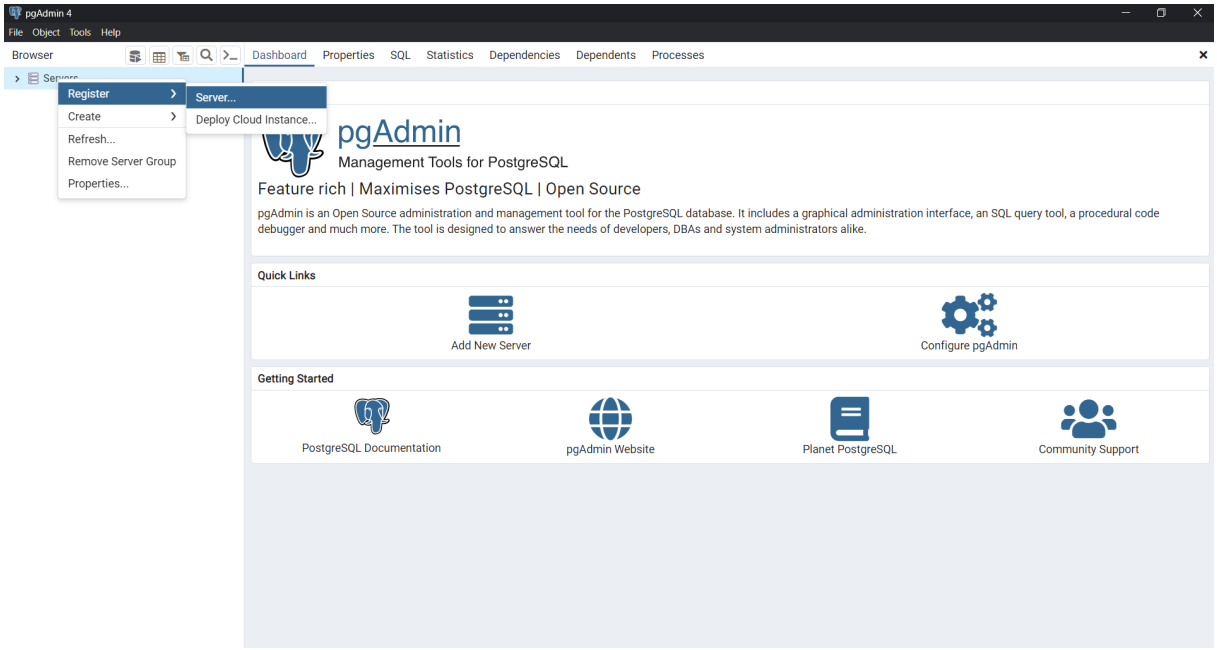
[3]If you want to set up PostGIS on your local computer, please refer to the following documentation.

6.2.2 Connect to the server with pgAdmin

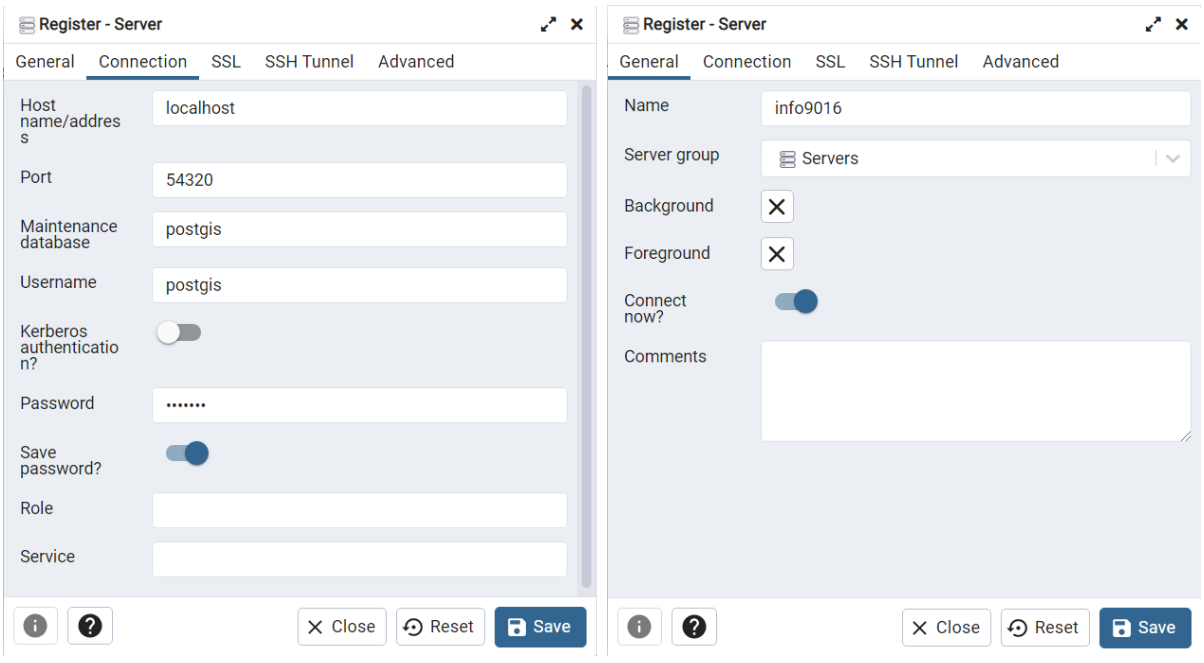
To interact with the database, there are two primary options available: utilizing psql, which is an interactive terminal allowing users to type in queries manually, or using pdAdmin, a visual interface that enables users to interact with psql commands. In this tutorial, we will be opting for the latter option as it offers a more user-friendly tool for querying PostgreSQL databases, making it more accessible to beginners.

Follow the steps below in order to connect to the database server deployed inside the docker container:

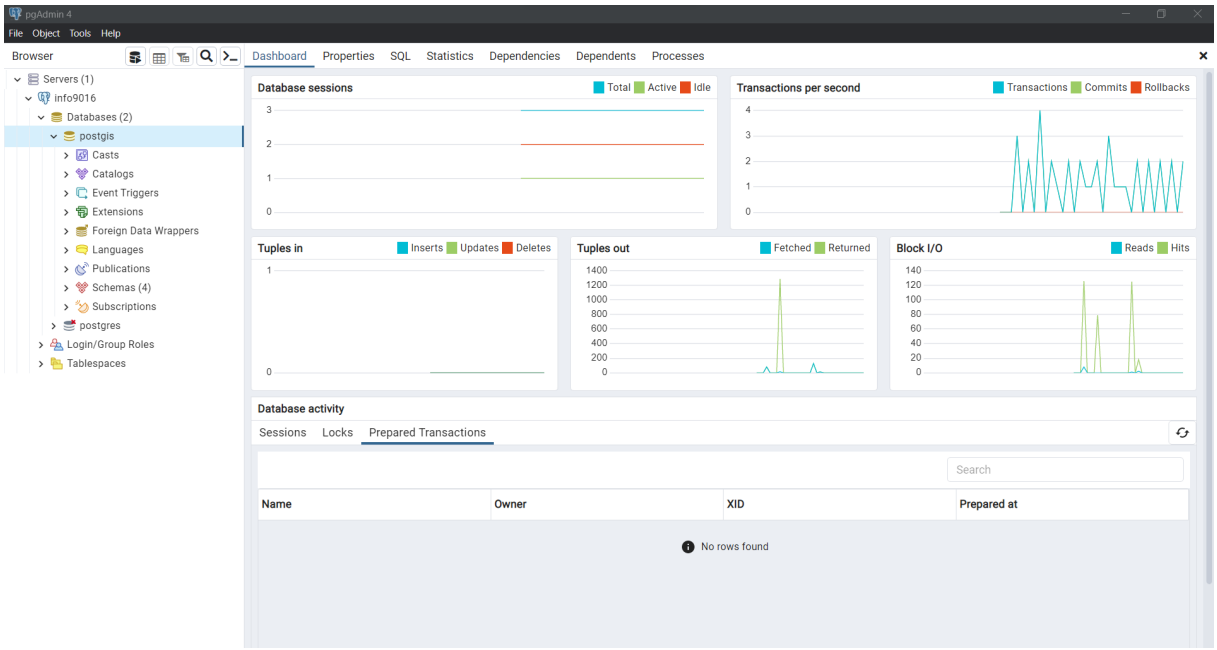
1. Start up pgAdmin and right-click on Server and register to a new database



2. Enter the following credentials:



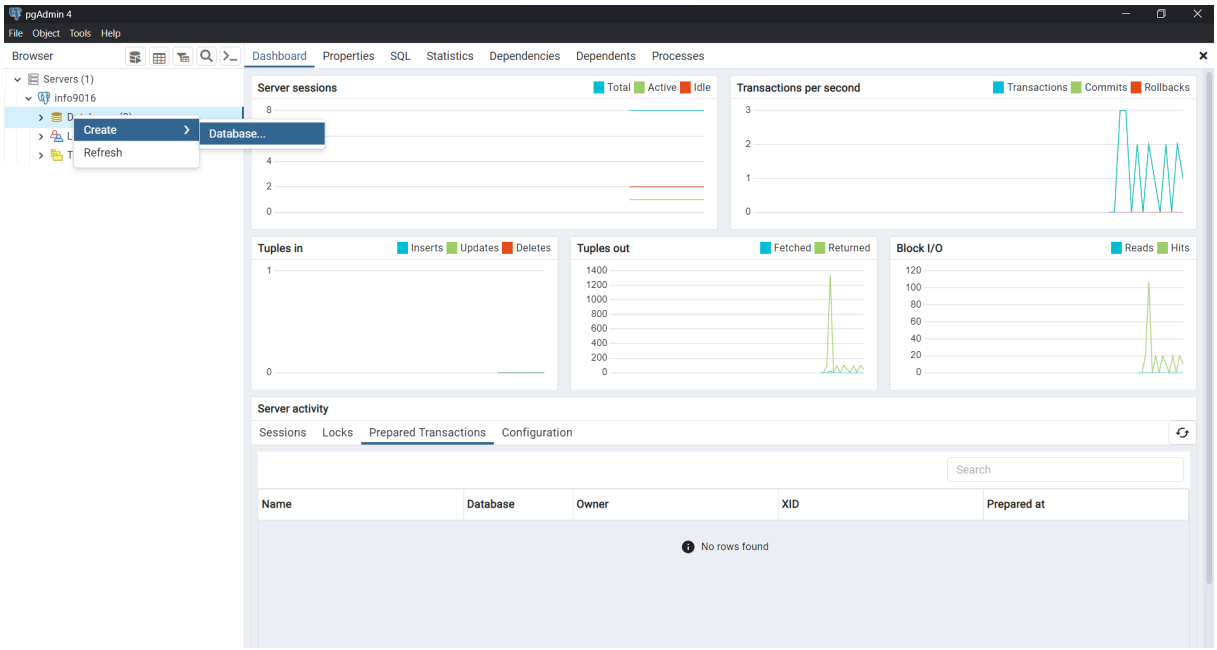
3. You are now connected to the database server and should see the following:



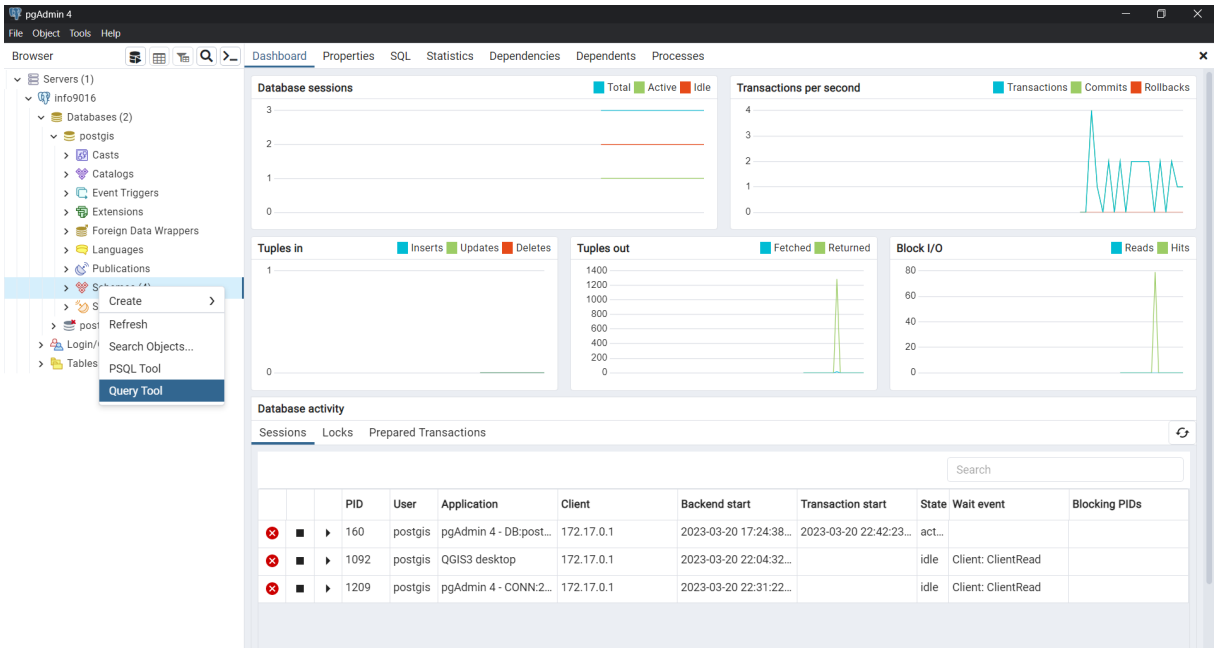
6.2.3 Create a Spatial Database

We will now create a Spatial Database with pgAdmin.

1. Inside pgAdmin, right-click on Databases > create > database. Enter a name for your database and click on save.



2. Select the new database in the sidebar to display the tree of objects. Right-click on Schemas and select Query tool.



3. Enter the following query into the query text field to load the PostGIS spatial extension:

```
CREATE EXTENSION postgis;
```

   and click on the Execute button 

4. Confirm that PostGIS is installed by running:

```
SELECT postgis_full_version();
```

Congratulations, you have successfully created your first PostGIS spatial database. But for the purpose of this tutorial, we will be using our World Countries (Generalized) shapefile.

5. Inside a terminal, open an interactive terminal session to the container:

```
docker exec -it postgis-tutorial bash
```

6. Import the World Countries (Generalized) shapefile into the container:

```
./import.sh
```

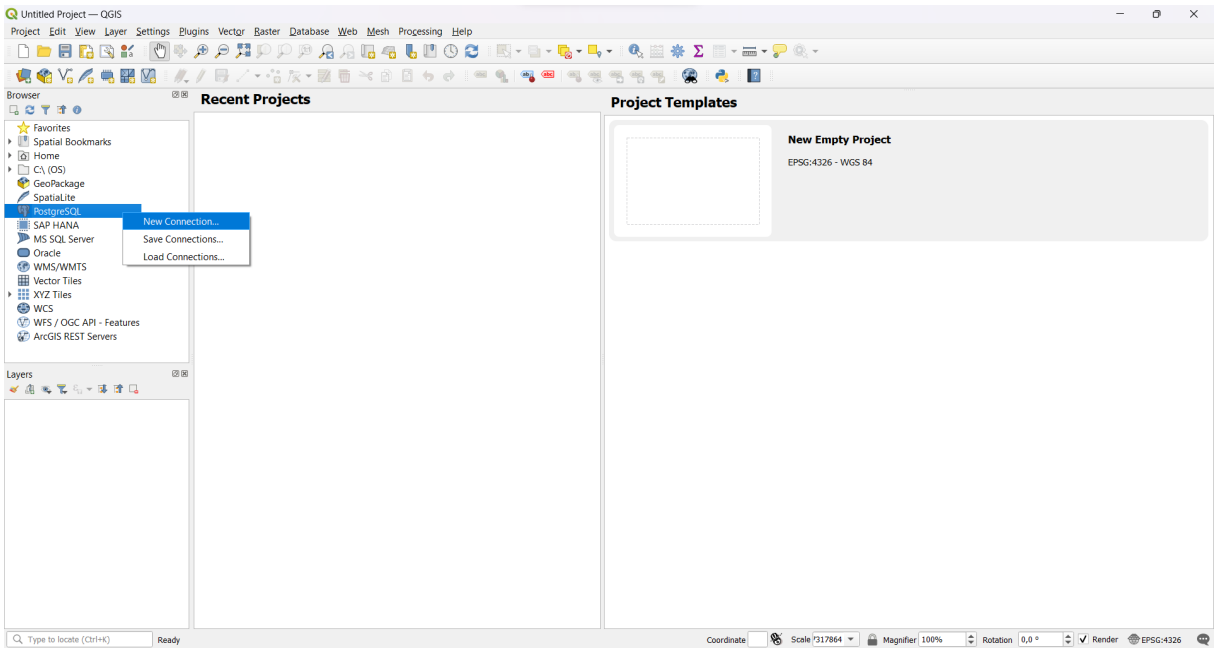
   This command will run the import.sh script inside the container, which will import the Shapefile into the database.

You now have a working environment with PostGIS installed and a sample dataset imported, ready to be used for the tutorial.

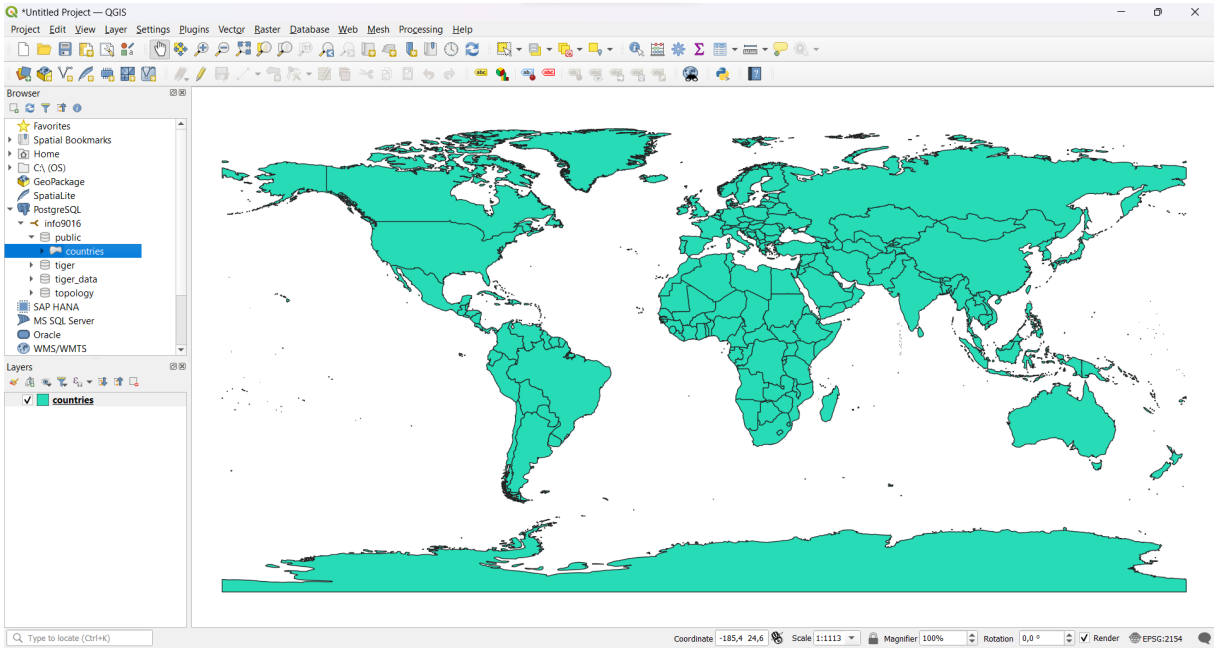
6.3 Get a visual representation of the data with QGIS

For this tutorial, we will primarily use pgAdmin to communicate with the database. However, if you want to fully leverage the capabilities of PostGIS and visualize the data we'll be working with, QGIS is a recommended tool to have.

1. Start up QGIS and create a connection to your database with the same credentials you used with pgAdmin.



2. Inside the connection, double-click on countries and you should see the world map generated from the data imported with the shapefile.



As a high-level GIS software, QGIS relies on several underlying SQL queries to interact with the database. However, these queries may obscure some of the fundamental SQL commands that are crucial for learning PostGIS. Therefore, in this tutorial that focuses on learning PostGIS, we will use pgAdmin to directly query the database, enabling us to develop a comprehensive understanding of its functionalities.

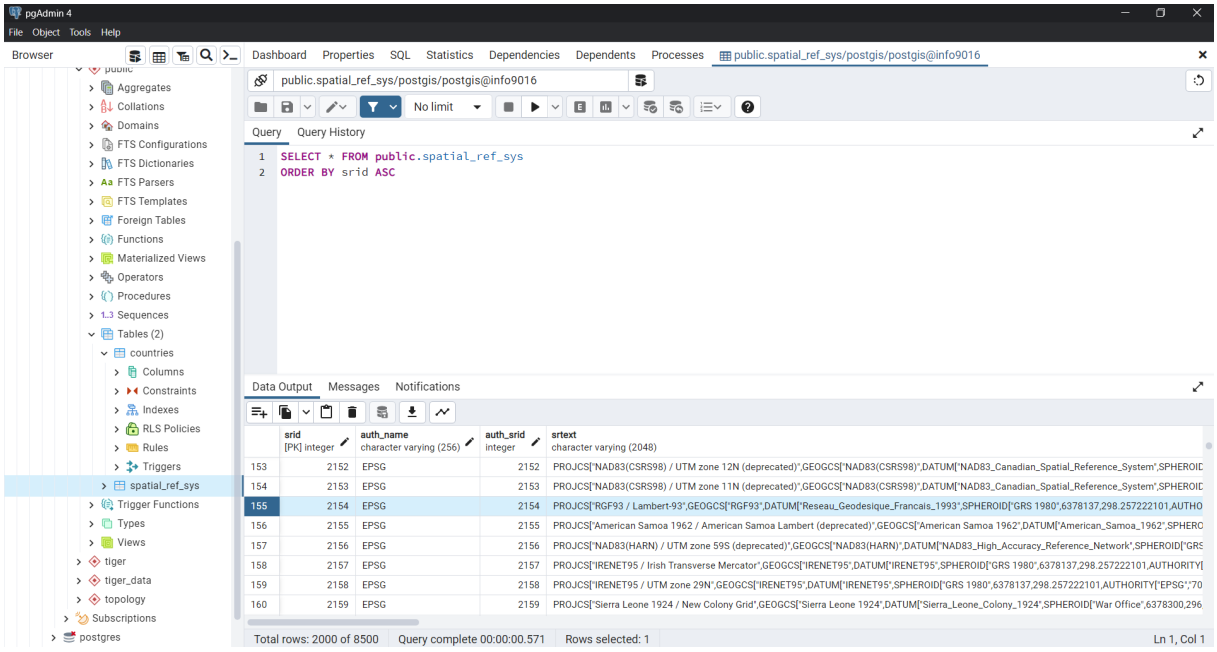
6.4 Working with PostGIS

PostGIS is an extension of PostgreSQL that provides spatial data types to represent geometric or geographic objects such as points, lines, polygons, and collections. These data types differ from standard data types such as integers and strings, as they have properties and characteristics that are specific to the representation of spatial information. Spatial data types allow you to perform spatial queries and analysis on your data. As an example in our imported dataset, spatial data types are used to represent the locations and the boundaries of countries.

PostGIS provides a range of spatial data types, including point, line, polygon, multipoint, multiline string, multi-polygon, collection, and geography. Each data type represents a different kind of spatial object and has its own set of properties and characteristics that define how it behaves and can be manipulated.

For example, the point data type represents a single point in space, defined by its X, Y, and optionally Z coordinates. The line data type represents a set of connected points that define a path in space, while the polygon data type represents a closed shape made up of one or more connected lines. The geography data type, which will not be covered in this tutorial, is a special type of geometry that represents data on a geodetic (curved) surface, such as the earth and provides built-in functions for calculating distances and areas on the surface.

In addition to the spatial data types, an important property is their spatial reference system (SRS). This property defines the coordinate system used to represent the data. When working with spatial data, it's important to use the appropriate SRS for the data being represented in order to ensure accurate results. In our case, we will be working exclusively with SRS 2154, which represents the RGF93 / Lambert-93 coordinate system commonly used in France. This system is based on the geodetic reference system RGF93 (Réseau Géodésique Français 1993) and the Lambert-93 projection and uses meters as its metric. It's important to note that the choice of SRS will vary depending on the specific dataset and analysis requirements. SRSes are listed inside the *spacial_ref_sys* table.



Before we explore some useful functions that PostGIS provides for querying and analyzing data, let's take a moment to learn how to create some of the spatial data types we discussed earlier. These data types will be essential for creating our new country and our cities.

6.4.1 Points

A spatial point is represented by its coordinates. We use points when the shape and size of the object we want to represent are not important. Later on, we will use points to represent cities on our world map.

Some important functions for working with points are:

- $ST\_MakePoint(x, y)$ or $ST\_Point(x, y)$ create a new point.
  - $ST\_MakePoint()$ can take up to 4 coordinates.
- $ST\_X(geometry)$ returns the X coordinate of a point geometry.
- $ST\_Y(geometry)$ returns the Y coordinate of a point geometry.

6.4.2 Linestrings

A linestring is a straight line that connects two or more points and represents a path between locations (e.g. roads, underground map).

Some important functions for working with linestrings are:

- $ST\_MakeLine(p1, p2)$ or $ST\_MakeLine(ARRAY[p1, p2, \ldots, pn])$ connect 2 or n points.
- $ST\_StartPoint(geometry)$ returns the first coordinate as a point.
- $ST\_EndPoint(geometry)$ returns the last coordinate as a point.
- $ST_NPoints(geometry)$ returns the number of coordinates in the linestring.

6.4.3 Polygons

A polygon is a type of geometry that represents an area made of a ring, which is a closed linestring. If needed, a polygon can be made of multiple linestrings to represent holes. Unlike a point, polygons are used for objects where shape and size are important.

Some important functions for working with polygons are:

- $ST\_MakePolygon(l)$ creates a polygon with linestrings.
- $ST\_MakePolygon(l, ARRAY[l1, l2, \ldots, ln])$ creates a polygon with a linestring and n interior linestrings.
- $ST\_Area(geometry)$ returns the area of the polygon.
- $ST\_Perimeter(geometry)$ returns the length of all the rings.

6.4.4 Collections

Last but not least, we can define collections containing the spacial data types we covered. Hence, we have MultiPoints, MultiLineStrings, MultiPolygons, and GeometryCollections (heterogeneous collections).

Some important functions for working with collections are:

- *ST_NumGeometries(geometry)* returns the number of parts in the collection.
- *ST_Area(geometry)* returns the total area of all polygonal parts.
- *ST_Length(geometry)* returns the total length of all linear parts.

When creating a country, we could use either a polygon or a Multypolygon. In our case, we will be using the latter to represent our countries for the following two reasons:

- The data imported from the World Countries (Generalized) shapefile represents the countries with this geometry type. This is important as we need to stay consistent on our data types inside the database table. Of course, using this spacial data type was not a trivial choice for the following reason.
- A country is often made of multiple disconnected regions or islands, in sometimes distant parts of the world. For example, Indonesia, which is made up of thousands of islands, or France, which has regions like Reunion Island east of Madagascar or Saint-Martin, an island in the northeast Caribbean Sea.

6.4.5 Some useful functions

- With our current knowledge, if we want to create a polygon with a whole inside, we would need to execute a query as in the following:

```
ST_MakePolygon(ST_MakeLine(ARRAY[ST_Point(0, 0), ST_Point(0,
    1), ST_Point(1, 1), ST_Point(1, 0), ST_Point(0, 0)]),
  ARRAY[ST_MakeLine(ARRAY[ST_Point(0, 0), ST_Point(1, 0),
  ST_Point(1, 1), ST_Point(0, 1), ST_Point(0, 0)])])
```

  Which is without a doubt a big overhead knowing that this only creates the outline of a square. But PostGIS provides *ST_GeomFromText()*, which constructs a geometry object from the OGC Well-Known text representation. Our query becomes a lot more simple:

```
ST_GeomFromText('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0), (0.25
  0.25, 0.75 0.25, 0.75 0.75, 0.25 0.75, 0.25 0.25))')
```

- *ST_SetSRID(geometry, srid)* sets the SRID (SRS) on geometry to a particular integer value.
- In the database, the geometry is stored encrypted. We use *ST_AsText(geometry)* to display the geometry in human-readable text.
- *ST_GeometryType(geometry)* returns the type of a given geometry.

6.5 Creation of a country

With all the knowledge we just acquired, let us dive into some code.

To create a new country in PostGIS, we need to understand how the system works. In practice, the data used to create the geometry for a country's boundary is obtained from authoritative sources, such as national mapping agencies. However, for the sake of simplicity, we will create an imaginary simple country. Our new country will be called *Saint Tilman* and will have an ISO code of *DB*. It will be affiliated with *Belgium*. As said previously, we will use Multipolygons to represent our country.

In the following, we execute our queries using the query tool on our tables  , followed by an explanation.

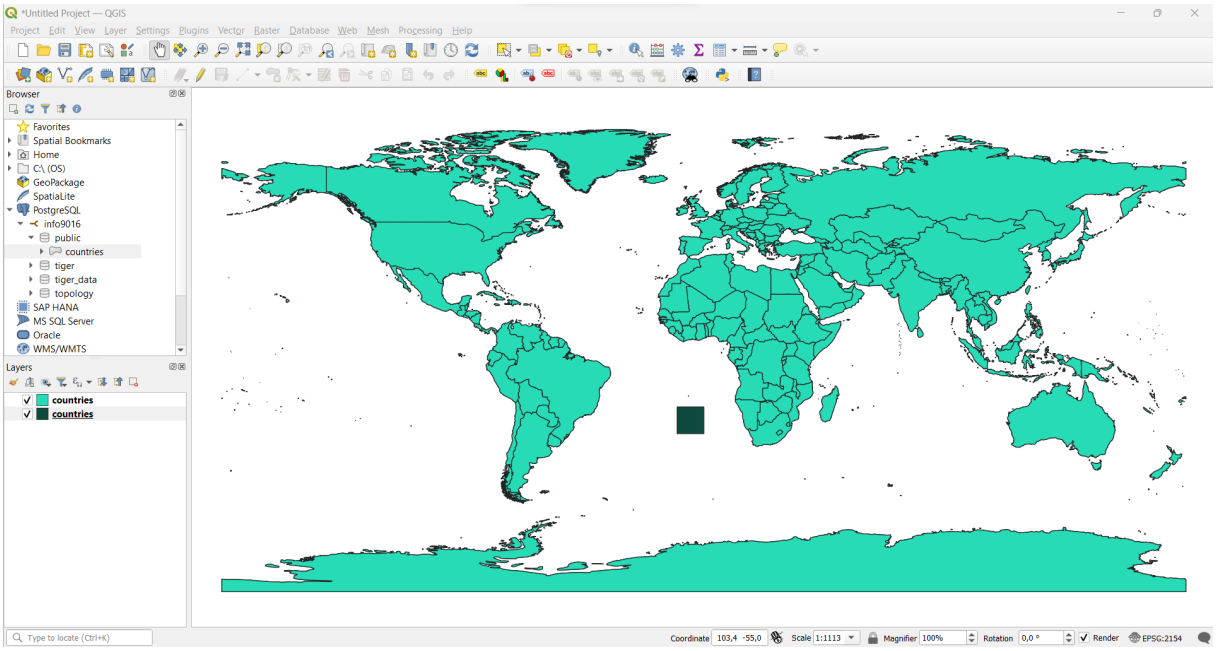
```
INSERT INTO countries (gid, fid, country, iso, countryaff,
    aff_iso, shape_leng, shape_area, geom)

VALUES(252, 252, 'Saint Tilman', 'DB', 'Belgium', 'BE', 40, 100,
    ST_SetSRID(ST_Multi('POLYGON ((-10 -20, -10 -30, 0 -30, 0
    -20, -10 -20))'), 2154));

SELECT ST_Area(geom), ST_Perimeter(geom), ST_AsText(geom),
    ST_GeometryType(geom)
FROM countries
WHERE gid = 252;
```

Part 1: select the *countries* table imported from the shapefile and selects every column name to insert our new values.

Part 2: Insert an entry for each value. We use $ST\_SetSRID$ to set the SRS and $ST\_Multi()$ the function used to create a Multipolygon. We can directly define the Multipolygon with an implicit $ST\_GeomFromText$ call.

Part 3: Retrieve from the DB some information about the entry we inserted. Take note that we are using $ST\_Perimeter()$ instead of ST_Length as our Multipolynom has been transformed into a simple Polynom.

After creating *Saint Tilman*, you should see a new geometry in the QGIS view upon refreshing the table.



6.6 Creation of two cities

In the following, we execute our queries using the query tool on our tables , followed by an explanation.

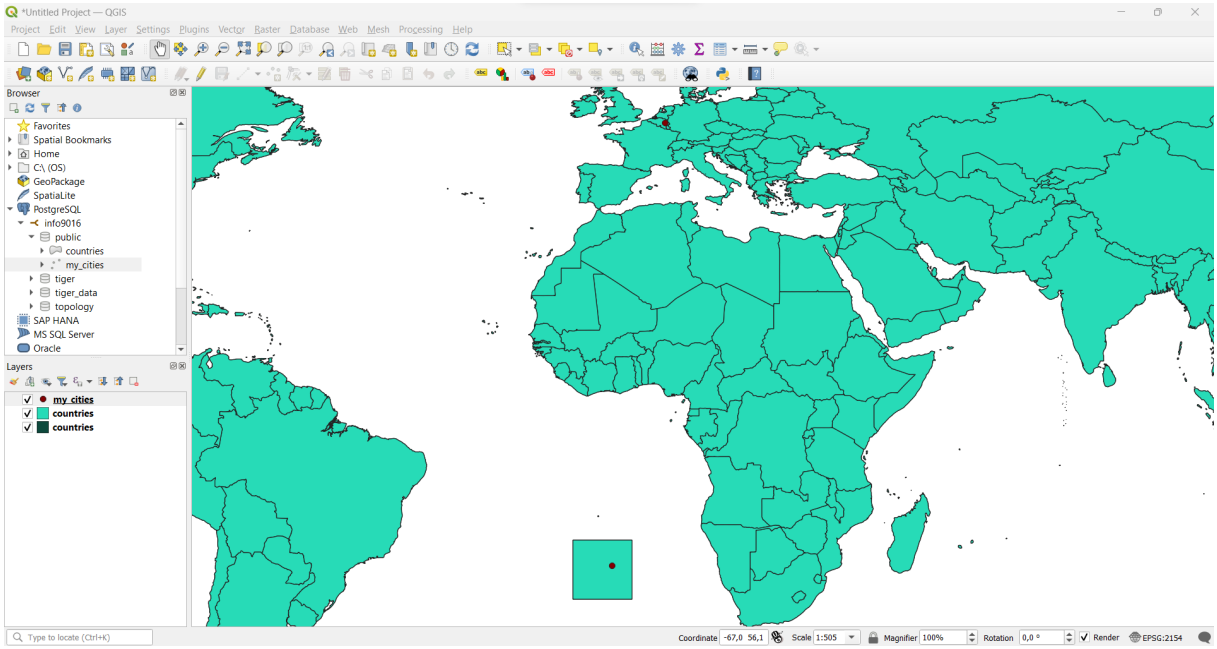
```
CREATE TABLE my_cities (
    id SERIAL PRIMARY KEY,
    name character varying (40),
    geom GEOMETRY(Point, 2154)
);

INSERT INTO my_cities (id, name, geom)
VALUES
    (1, 'Liege', ST_SetSRID(ST_Point(5.6103,50.5926), 2154)),
    (2, 'Mont Efiore', ST_SetSRID(ST_Point(-3.332,-24.394),
        2154));

WITH subq AS (
    SELECT geom FROM my_cities
)

SELECT ST_X(geom), ST_Y(geom), ST_AsText(geom), ST_GeometryType(
    geom)
FROM subq;
```

Part 1: Creates a new table with geometry-type data.

Part 2: Insert a new entry in the table. We use *ST_SetSRID* to set the SRS and *ST_Point* to define the coordinates of the cities

Part 3 & 4: Display some info about the entries we created.

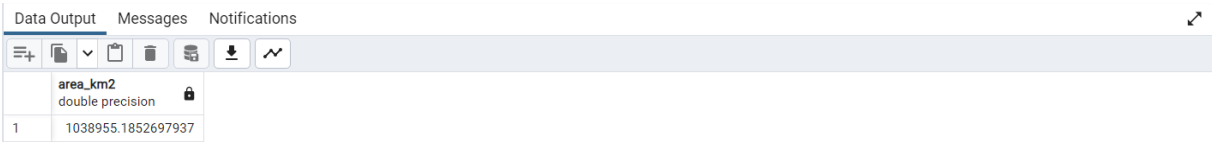
After creating *Liege* and *Mont Efiore*, you should see two new geometries (dots) in the QGIS view upon refreshing the table.



6.7 Compute some metrics

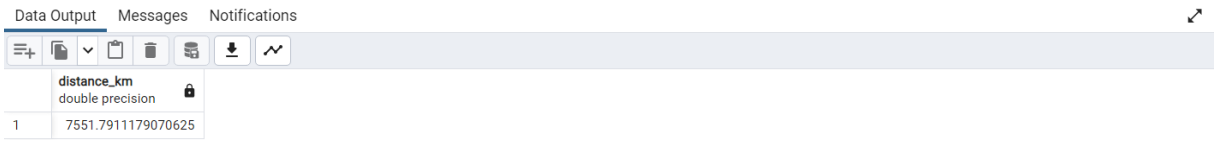
In the following, we execute our queries using the query tool on our tables , followed by an explanation.

```
SELECT (ST_Area(ST_Union(geom))/1000000) * (100000^2) AS
    area_km2 FROM (
     SELECT geom FROM countries
     WHERE iso IN ('BE', 'DB')
) subquery;
```



We use *ST_Area*() to calculate the area of the shape in the parameter. The parameter we give it is a *ST_Union*() entry, which combines the list of geometries it is given. We then select the countries we want to combine. Take note that our SRS metric is in meters and that our imported world map is at the scale of 100000 : 1. Hence, we need to scale up the result by a factor of 100000^2 to get the real-world area in m^2 and divide it by 1000000 to get to km^2.

```
SELECT ST_Distance(
   (SELECT geom FROM my_cities WHERE id = 1),
   (SELECT geom FROM my_cities WHERE id = 2)
) * 100000 / 1000 as distance_km;
```



To compute the distance between the two cities, we simply use *ST_Distance()*. We do not forget to multiply the result by $\frac{100000}{1000}$ to get up to scale and convert the result in kilometres.

6.8 What we have learned

We have explored several of the powerful functionalities that PostGIS offers, including the essential spatial data types and useful functions to manipulate them. By putting our knowledge into practice on a real dataset, we have gained valuable hands-on experience with PostGIS. However, there is still much more to discover in the world of geospatial databases with PostGIS, and this tutorial only scratches the surface.

If you're looking to expand your knowledge of PostGIS, there are several excellent resources available:

- The official PostGIS documentation offers in-depth information on all aspects of the software.
- AWS's Spatial functions page provides clear and concise documentation on all PostGIS functions, along with examples of how to use them effectively. PostGIS workshop is a comprehensive tutorial that covers a wide range of topics.
- ArcGIS Hub and Eurostat offer several valuable Shapefile, including Eurostat's country map, which is particularly complete.

By exploring these resources, you can gain a deeper understanding of PostGIS and learn how to use it to its fullest potential.

References

[1] *Introduction to PostGIS - 15. Spatial Indexing*. URL: http://postgis.net/workshops/postgis-intro/indexing.html.

[2] Antonin Guttman. *R-Trees: A Dynamic Index Structure for Spatial Indexing*. URL: http://postgis.net/docs/support/rtree.pdf.

[3] Regina Obe & Leo Hsu. *PostGIS in action*. URL: http://159.69.3.96/ebooks/IT/PostgreSQL/PostGIS_In_Action.pdf.

[4] Felix Kunde. *Maps with PostgreSQL and PostGIS*. URL: https://engineering.zalando.com/posts/2021/12/maps-with-postgresql-and-postgis.html.

[5] Branislav Stojkovic. *PostGIS - A Real-World Example*. URL: https://symphony.is/blog/postgis---a-real-world-example.

[6] Boundless. *PostGIS Workshop Geometries*. URL: http://postgis.net/workshops/postgis-intro/geometries.html.

[7] Abdullah Alger. *GeoFile: pgAdmin 4 and the Geometry Viewer*. URL: https://www.compose.com/articles/geofile-pgadmin-4-and-the-geometry-viewer/.

[8] Abdishakur. *Spatial Data Science with PostgreSQL: Geometries*. URL: https://towardsdatascience.com/spatial-data-science-with-postgresql-geometries-c00387755700.

[9] *QGIS*. URL: https://www.qgis.org/fr/site/.

[10] ArcGIS. *ArcGIS Hub*. URL: https://hub.arcgis.com/.

[11] ArcGIS. *World Countries (Generalized) dataset*. URL: https://hub.arcgis.com/datasets/esri::world-countries-generalized/about.

[12] Eurostat. *Eurostat*. URL: https://ec.europa.eu/eurostat/web/main/home.

[13] Eurostat. *Eurostat countries*. URL: https://ec.europa.eu/eurostat/web/gisco/geodata/reference-data/administrative-units-statistical-units/countries.

[14] *Docker*. URL: https://www.docker.com/.

[15] *pgAdmin*. URL: https://www.pgadmin.org/.

[16] *PostGIS*. URL: https://postgis.net/.

[17] *psql*. URL: https://www.postgresql.org/docs/current/app-psql.html.

[18] *PostGIS Documentation*. URL: https://postgis.net/docs/.

[19] esri. *PostgreSQL data types supported in ArcGIS*. URL: https://pro.arcgis.com/en/pro-app/latest/help/data/geodatabases/manage-postgresql/data-types-postgresql.htm.

[20] pgAdmin. *pgAdmin Container Deployment*. URL: https://www.pgadmin.org/docs/pgadmin4/latest/container_deployment.html.

[21] *PostGIS installation*. URL: https://postgis.net/install/.

[22] epsg.io. *EPSG:2154 RGF93 v1 / Lambert-93 – France*. URL: https://epsg.io/2154.

[23] AWS. *Special Functions*. URL: https://docs.aws.amazon.com/redshift/latest/dg/geospatial-functions.html.