CSCI 5980/8980: Spatial Enabled Artificial Intelligence

Assignment 1 (10 points)

Due Date: 2022/02/15 11:59:00 CST

1. Overview of the Assignment

With this assignment, you will learn to use existing spatial data to prepare feature vectors capturing geographic context to support spatial artificial intelligence tasks. You will use PostgreSQL with PostGIS and Apache Sedona (formerly GeoSpark) to perform spatial queries. You will compare their performance in terms of query efficiency using the same query tasks on both platforms. You can access the data and sample code on Google Drive.

2. Programming Requirements and Environment Settings

a. You must use **SQL** and **Python** to implement all tasks.

b. Programming Environment:

- JAVA version 1.8, Python 3.7, Pyspark 3.0.0, Sedona 1.1.1
- [Optional] You can use Conda to manage your programming environment.

```
$conda create --name [ENV] -y python=3.7

$conda activate [ENV]

$conda install -c conda-forge gdal==3.4.0

$conda install -c conda-forge pyspark==3.0.0

$pip install apache-sedona
```

Sedona Python requires two additional jar packages, **sedona-python-adapter** and **geotools-wrapper,** to work properly.[1] Specifically, you need to put two jar packages[2] under [YOUR PYTHON PATH]/site-packages/pyspark/jar/.

3. Assignment Datasets

This assignment will use two types of datasets.

a. Two sampled datasets of air quality sensor locations [download]

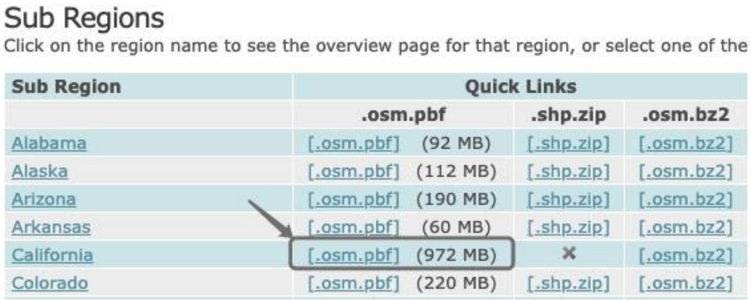
These datasets contain sensor IDs and locations (longitude and latitude in WGS84). One dataset contains more than 400 locations, and the other dataset has 10 locations. Both datasets are sampled PurpleAir sensor locations in California.

[1] See http://sedona.incubator.apache.org/setup/install-python/

[2] We provide sedona-python-adapter-3.0_2.12-1.1.1-incubating.jar and geotools-wrapper-1.1.0-25.2.jar

b. OpenStreetMap (OSM) data [download]

OpenStreetMap (OSM) is a crowd-sourced worldwide map dataset.[3] This assignment will use OSM data as a proxy to describe geographic context of air quality sensor locations. You need to download the file **[.osm.pbf] of California** (see the screenshot below), which is a compressed file of the OSM California data.



Sub Regions

Click on the region name to see the overview page for that region, or select one of the f

Sub Region	Quick Links		
	.osm.pbf	.shp.zip	.osm.bz2
Alabama	[.osm.pbf] (92 MB)	[.shp.zip]	[.osm.bz2]
Alaska	[.osm.pbf] (112 MB)	[.shp.zip]	[.osm.bz2]
Arizona	[.osm.pbf] (190 MB)	[.shp.zip]	[.osm.bz2]
Arkansas	[.osm.pbf] (60 MB)	[.shp.zip]	[.osm.bz2]
California	[.osm.pbf] (972 MB)	×	[.osm.bz2]
Colorado	[.osm.pbf] (220 MB)	[.shp.zip]	[.osm.bz2]

4. Deliverables and Tasks

You need to turn in a zip file, named as **[your_login_id]_assignment1.zip** (all lowercase),[4] e.g., lin00786_assignment1.zip, containing the following files:

a. [REQUIRED] Code scripts: one SQL script named as **main.sql**, one Python script named as **main.py**

We provide skeletons of the two scripts in [main.sql] and [main.py], and you will complete the required code block.

The SQL script will be graded manually. The Python script will be automatically graded by running

```
$python main.py --input_file [INPUT FILE] --osm_table [OSM TABLE] --out_path [OUT PATH]
```

b. [REQUIRED] A result file named as **geographic_features.csv**

c. [REQUIRED] A readme document in Word describing the results, named as **readme.docx**

d. [OPTIONAL] You can include other Python scripts to support your programs (e.g., callable functions).

4.1. Task: Importing OSM data to PostgreSQL (PostGIS)

You need to import the OSM data (i.e., california-latest.osm.pbf) to PostgreSQL using the ogr2ogr tool and re-organize the imported tables for later use. Here are the steps:

- Download and install PostgreSQL (and the PostGIS extension)[5] on your machine (see some installation screenshots in Appendix 1). Then you can create your own localhost server and database in PostgreSQL for this assignment. You can use PgAdmin to access and manage your PostgreSQL instance (Postico is recommended but only works for MacOS). You need to add the PostGIS extension by running **CREATE EXTENSION postgis**.[6]

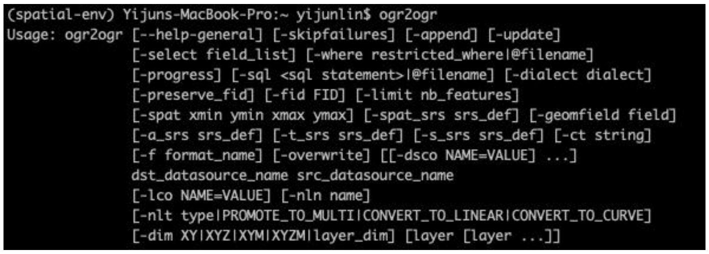
[3] See https://wiki.openstreetmap.org/wiki/Main_Page

[4] Also the name of your email address, e.g., if your email address is lin00786@umn.edu, your id would be lin00786

[5] PostgreSQL downloads: https://www.postgresql.org/download/

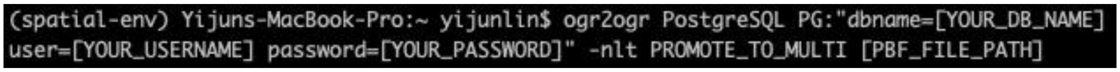
[6] https://postgis.net/install/

- Install GDAL[7] and verify if ogr2ogr works by running **$ogr2ogr.** You will see the following screenshot if it succeeds.



```
(spatial-env) Yijuns-MacBook-Pro:~ yijunlin$ ogr2ogr
Usage: ogr2ogr [--help-general] [-skipfailures] [-append] [-update]
               [-select field_list] [-where restricted_where|@filename]
               [-progress] [-sql <sql statement>|@filename] [-dialect dialect]
               [-preserve_fid] [-fid FID] [-limit nb_features]
               [-spat xmin ymin xmax ymax] [-spat_srs srs_def] [-geomfield field]
               [-a_srs srs_def] [-t_srs srs_def] [-s_srs srs_def] [-ct string]
               [-f format_name] [-overwrite] [[-dsco NAME=VALUE] ...]
               dst_datasource_name src_datasource_name
               [-lco NAME=VALUE] [-nln name]
               [-nlt type|PROMOTE_TO_MULTI|CONVERT_TO_LINEAR|CONVERT_TO_CURVE]
               [-dim XY|XYZ|XYM|XYZM|layer_dim] [layer [layer ...]]
```

- Use ogr2ogr in GDAL to import the pbf file into PostgreSQL by running the following command (it will take a few minutes):

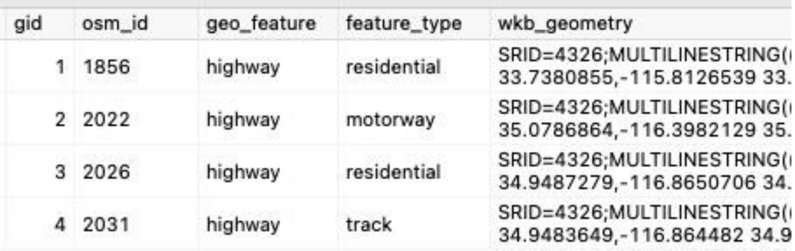


```
(spatial-env) Yijuns-MacBook-Pro:~ yijunlin$ ogr2ogr PostgreSQL PG:"dbname=[YOUR_DB_NAME]
user=[YOUR_USERNAME] password=[YOUR_PASSWORD]" -nlt PROMOTE_TO_MULTI [PBF_FILE_PATH]
```

  You will see five tables (e.g., lines) imported in your database if it succeeds.

- You will use these tables to create **three new tables** (named as line_features, polygon_features, and point_features) for later use. The SQL queries has been provided for you in osm.sql**.** If you are interested, we use gen_SQL_cmd.py to generate these SQL queries. You can also modify gen_SQL_cmd.py to generate your own queries for later assignments (e.g., modifying selected geo-features). The goal is to select some representative geo-features for each geometry type (points, lines, polygons). You can simply copy/paste and run the SQL queries in osm.sql in your PgAdmin (this step is time-consuming, taking around 20 minutes on a 2017 MacBook Pro).

- The SQL queries in osm.sql will generate tables with five columns (see the screenshot below). The geo_feature column describes the original categories in OSM data (e.g., highway and waterway). The feature_type column describes the subclass of these categories. For example, highway can be primary, motorway, residential, etc.[8]



gid	osm_id	geo_feature	feature_type	wkb_geometry
1	1856	highway	residential	SRID=4326;MULTILINESTRING((33.7380855,-115.8126539 33.
2	2022	highway	motorway	SRID=4326;MULTILINESTRING((35.0786864,-116.3982129 35.
3	2026	highway	residential	SRID=4326;MULTILINESTRING((34.9487279,-116.8650706 34.
4	2031	highway	track	SRID=4326;MULTILINESTRING((34.9483649,-116.864482 34.9

4.2. Task: Creating spatial buffers for air quality sensor locations (3 points)

You will create spatial buffers with varying sizes using the **ST_Buffer** function.[9] We will use spatial buffers with radii of 100, 500, and 1,000 meters in this assignment.

- Note that the given locations are in “epsg:4326”, so you need to transform their geometries to a coordinate system with a **meter** unit.

[7] https://gdal.org/

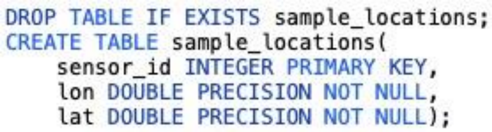
[8] See https://wiki.openstreetmap.org/wiki/Main_Page

[9] See ST_Buffer for PostGIS here: https://postgis.net/docs/ST_Buffer.html

- You need to implement this task using **BOTH PostgreSQL (section 4.2.1) and Apache Sedona (section 4.2.2).**

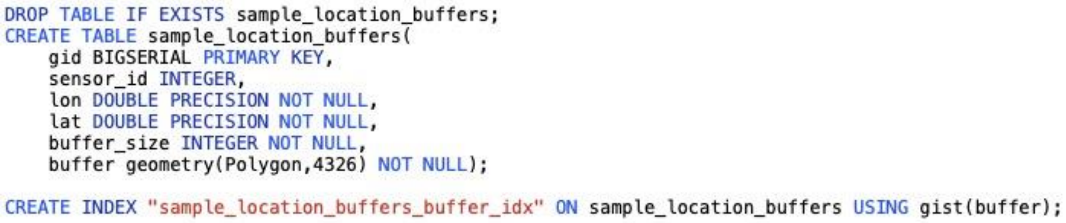
4.2.1. (1 point) Using PostgreSQL to create buffers. Here are the steps:

- Create a location table with the following schema:



```
DROP TABLE IF EXISTS sample_locations;
CREATE TABLE sample_locations(
    sensor_id INTEGER PRIMARY KEY,
    lon DOUBLE PRECISION NOT NULL,
    lat DOUBLE PRECISION NOT NULL);
```

- Import the sampled datasets of air quality sensor locations (from section 3.a)
- Create a table with the following schema:

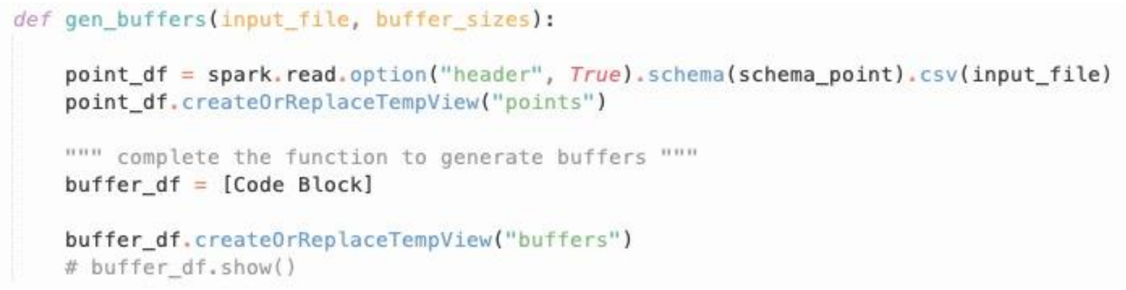


```
DROP TABLE IF EXISTS sample_location_buffers;
CREATE TABLE sample_location_buffers(
    gid BIGSERIAL PRIMARY KEY,
    sensor_id INTEGER,
    lon DOUBLE PRECISION NOT NULL,
    lat DOUBLE PRECISION NOT NULL,
    buffer_size INTEGER NOT NULL,
    buffer geometry(Polygon,4326) NOT NULL);

CREATE INDEX "sample_location_buffers_buffer_idx" ON sample_location_buffers USING gist(buffer);
```

- Compute and insert the buffers of size 100, 500 and 1,000 meters to the table

4.2.2. (2 points) Using GeoSpark to create buffers

- Complete the **gen_buffers** function in main.py



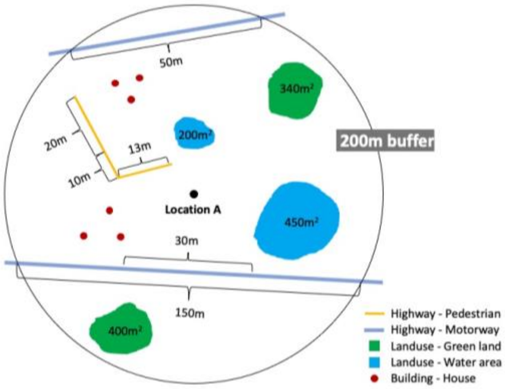
```
def gen_buffers(input_file, buffer_sizes):

    point_df = spark.read.option("header", True).schema(schema_point).csv(input_file)
    point_df.createOrReplaceTempView("points")

    """ complete the function to generate buffers """
    buffer_df = [Code Block]

    buffer_df.createOrReplaceTempView("buffers")
    # buffer_df.show()
```

4.3. Task: Computing geographic features (5 points)

You will compute feature vectors capturing geographic features by spatially joining the created buffers from Task 4.2 and the OSM tables generated from Task 4.1.

- Specifically, you need to join the geometries of the created buffers and OSM data to calculate the aggregated number/length/area of the OSM geographic features within each buffer. For example, if the geometry type is "MultiPolygon", the aggregation should be the summation of area sizes of all OSM geographic features within a buffer. You should use "meter" or "square meter" as your spatial units. For example, the left figure below shows a 200-meter buffer at Location A. The buffer intersects with five geographic features. The table below shows the desired output with the aggregated geographic features.

- You need to implement this task using **BOTH PostgreSQL (section 4.3.1) and Apache Sedona (section 4.3.2).**
-



GID	Sensor ID	Geometry Type	Buffer Size	Geo feature	Feature type	Value
1	A	line	200	Highway	Pedestrian	43
2	A	line	200	Highway	Motorway	200
3	A	polygon	200	Landuse	Green land	740
4	A	polygon	200	Landuse	Water area	650
5	A	point	200	Building	House	6

4.3.1. (1 point) Using PostgreSQL to compute geographic features

- Create a table with the following schema:

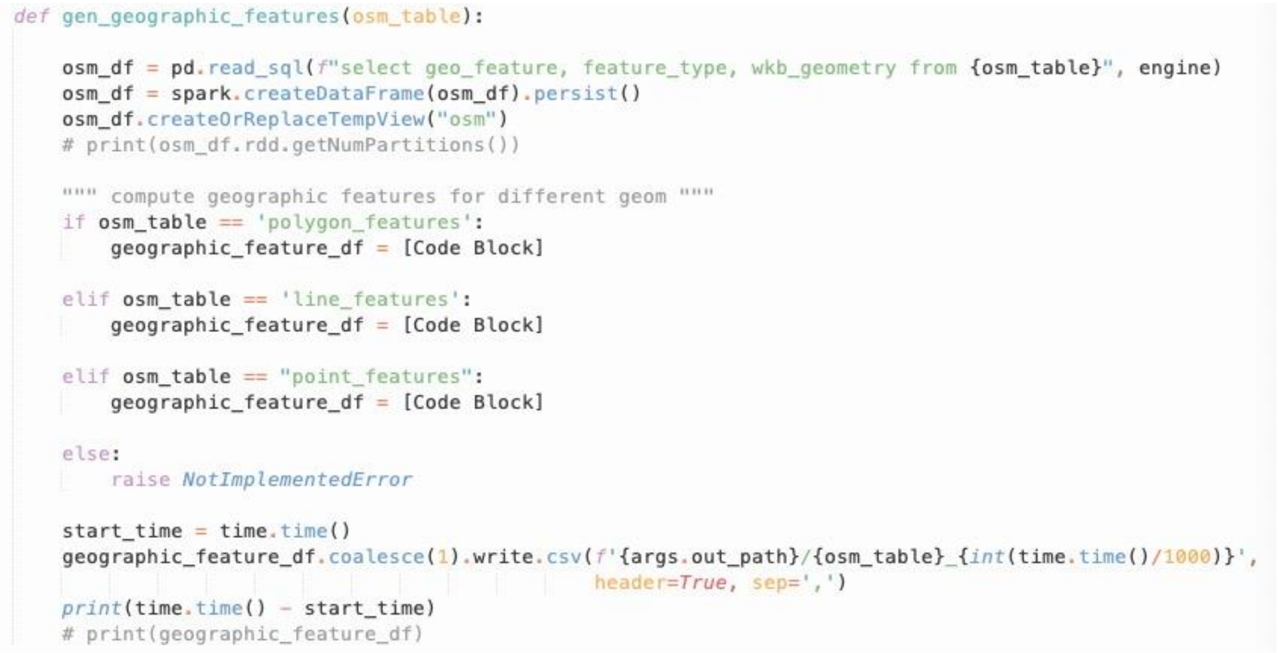
```
DROP TABLE IF EXISTS geographic_features;
CREATE TABLE geographic_features(
    gid BIGSERIAL PRIMARY KEY,
    sensor_id INTEGER NOT NULL,
    geom_type TEXT NOT NULL,
    geo_feature TEXT NOT NULL,
    feature_type TEXT NOT NULL,
    buffer_size INTEGER NOT NULL,
    value  DOUBLE PRECISION);
```

- Compute and insert all generated geographic features into the table. You need to write the SQL queries for line, polygon, and line features separately.

4.3.2. (4 points) Using GeoSpark to compute geographic features

- Load the OSM tables from PostgreSQL into Spark DataFrame. We provide an example solution in main.py using pandas.[10] To use the example solution, you need to create **an .env file** using the same format as env_example and add the database connection configuration (e.g., username and password). The .env file will be loaded using load_dotenv() (see common_db.py). You need to describe your own method in the readme document if you choose not to use the example solution.
- Complete the **gen_geographic_features** function in main.py. You need to call this function three times on the three OSM tables (polygon, line, and point features) generated from Task 4.1. Each call will generate a folder containing a CSV file by using the function [.coaleasce(1).write.csv()] in main.py. You will report the running time for each call.
- After running your scripts for the three tables (polygon, line, and point features), you need to merge the outputs of polygon, line, and point features to **geographic_features.csv** with the same schema in section 4.3.1. We provide merge.py for this purpose.

[10] https://pandas.pydata.org/



```
def gen_geographic_features(osm_table):

    osm_df = pd.read_sql(f"select geo_feature, feature_type, wkb_geometry from {osm_table}", engine)
    osm_df = spark.createDataFrame(osm_df).persist()
    osm_df.createOrReplaceTempView("osm")
    # print(osm_df.rdd.getNumPartitions())

    """ compute geographic features for different geom """
    if osm_table == 'polygon_features':
        geographic_feature_df = [Code Block]

    elif osm_table == 'line_features':
        geographic_feature_df = [Code Block]

    elif osm_table == "point_features":
        geographic_feature_df = [Code Block]

    else:
        raise NotImplementedError

    start_time = time.time()
    geographic_feature_df.coalesce(1).write.csv(f'{args.out_path}/{osm_table}_{int(time.time()/1000)}',
                                                header=True, sep=',')
    print(time.time() - start_time)
    # print(geographic_feature_df)
```

5. Grading criteria

a. Your code should be runnable and be able to generate results correctly (that is, generating tables with the submitted SQL queries and generating geographic_features.csv with the submitted Python scripts successfully) to receive full points on Task 4.2 and Task 4.3 (a total of 8 points).

The SQL script will be graded manually. The Python code will be automatically graded by running

$python main.py --input_file [INPUT FILE] --osm_table line_features --out_path [OUT PATH]

$python main.py --input_file [INPUT FILE] --osm_table point_features --out_path [OUT PATH]

$python main.py --input_file [INPUT FILE] --osm_table polygon_features --out_path [OUT PATH]

$python merge.py

b. In the readme document, you should report the running time of **completing Task 4.3 on each OSM table** by testing different location datasets as the input file in a table below (1 point). You need to explain the comparison results in several sentences (**no more than 300 words**) (1 point).

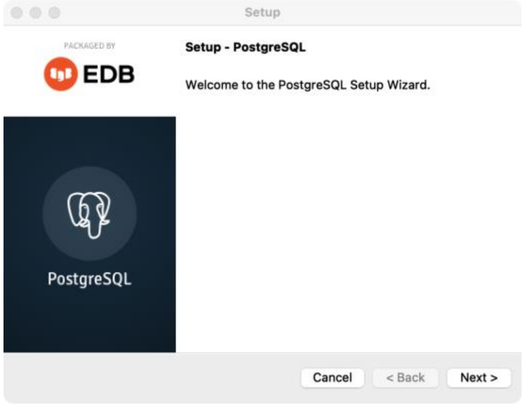
			#locations = 400	#locations = 10	
PostgreSQL	1 min 47 secs	Line	2 min 26 secs	3 secs 61 msec	1 secs 530 msec
	2 min 23 secs	Polygon	2 min 48 secs	3 secs 580 msec	2 secs 218 msec
	1 secs 262 msec	Point	2 secs 147 msec	222 msec	151 msec
Apache Sedona	1502.5403110980988	Line	880.5125770568848	481.0923671722412	948.214686870575
		Polygon		515.4751780033112	786.8028421401978
	37.0699200630188	Point	44.06004667282104	560.5235309600830	19.22355604171753

Table: Time comparison using PostgreSQL and Apache Sedona

c. You can put any external libraries that are necessary to execute your code or other instructions that can help TA run your assignment.

Appendix

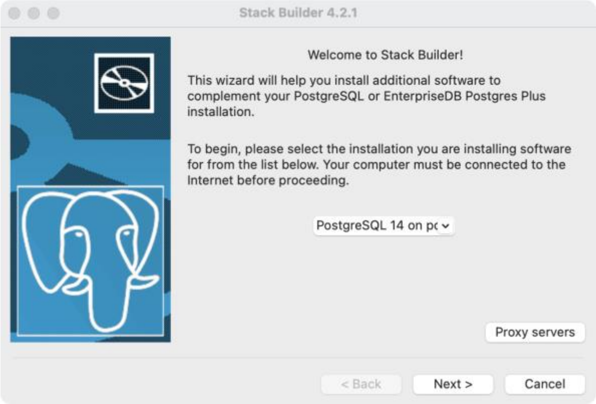
1. PostGIS extension installation via Stack Builder (you should see the following steps)



Setup

Setup - PostgreSQL

Welcome to the PostgreSQL Setup Wizard.



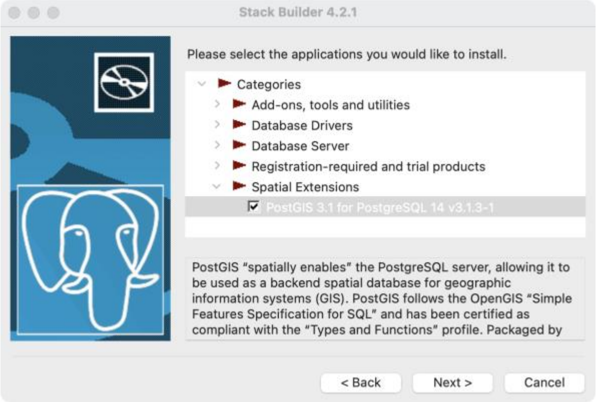
Stack Builder 4.2.1

Welcome to Stack Builder!

This wizard will help you install additional software to complement your PostgreSQL or EnterpriseDB Postgres Plus installation.

To begin, please select the installation you are installing software for from the list below. Your computer must be connected to the Internet before proceeding.

PostgreSQL 14 on pc

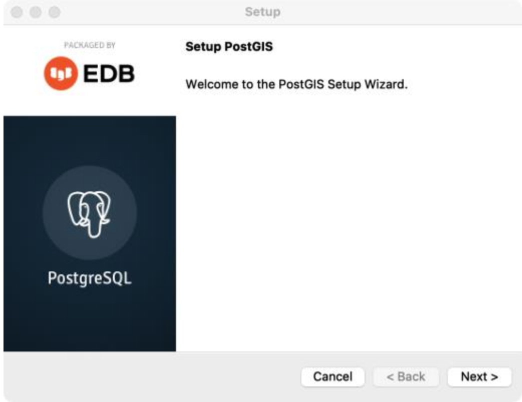


Stack Builder 4.2.1

Please select the applications you would like to install.

- Categories
  - Add-ons, tools and utilities
  - Database Drivers
  - Database Server
  - Registration-required and trial products
  - Spatial Extensions
    - ☑ PostGIS 3.1 for PostgreSQL 14 v3.1.3-1

PostGIS "spatially enables" the PostgreSQL server, allowing it to be used as a backend spatial database for geographic information systems (GIS). PostGIS follows the OpenGIS "Simple Features Specification for SQL" and has been certified as compliant with the "Types and Functions" profile. Packaged by



Setup

Setup PostGIS

Welcome to the PostGIS Setup Wizard.