

## Exercise # 5

Postgres was first developed in the year 1972 and revived in the year 1992 or around the same period. However, with the advent of web, satellite remote sensing and GIS data types, it has come a long way in being able to handle these datasets well. And has for long served as the go to solution for many GIS db administrators and web developers.

The fact that it can handle a variety of types of spatial data is encouraging with several extensions and utilities in the GeoFOSS arena to make the process seamless. Two such utilities are,

1. shp2pgsql
2. raster2pgsql

You can try typing these two commands in your linux terminal windows. This will print the following information on your screens, respectively.

```
user@user-virtualbox:~$ shp2pgsql
RELEASE: 3.2.0 (c3e3cc0)
USAGE: shp2pgsql [<options>] <shapefile> [[<schema>.]<table>]
OPTIONS:
  -s [<from>:]<srid> Set the SRID field. Defaults to 0.
    Optionally reprojects from given SRID.
  (-d|-a|-c|-p) These are mutually exclusive options:
    -d Drops the table, then recreates it and populates
        it with current shape file data.
    -a Appends shape file into current table, must be
        exactly the same table schema.
    -c Creates a new table and populates it, this is the
        default if you do not specify any options.
    -p Prepare mode, only creates the table.
  -g <geocolumn> Specify the name of the geometry/geography column
        (mostly useful in append mode).
  -D Use postgresql dump format (defaults to SQL insert statements).
```

```
user@user-virtualbox:~$ raster2pgsql
RELEASE: 3.2.0 GDAL_VERSION=34 (c3e3cc0)
USAGE: raster2pgsql [<options>] <raster>[ <raster>[ ...]] [[<schema>.]<table>]
Multiple rasters can also be specified using wildcards (*,?).
OPTIONS:
  -s <srid> Set the SRID field. Defaults to 0. If SRID not
    provided or is 0, raster's metadata will be checked to
    determine an appropriate SRID.
  -b <band> Index (1-based) of band to extract from raster. For more
    than one band index, separate with comma (,). Ranges can be
    defined by separating with dash (-). If unspecified, all bands
    of raster will be extracted.
  -t <tile size> Cut raster into tiles to be inserted one per
    table row. tile size is expressed as WIDTHxHEIGHT
```

These outputs contain the arguments, that one can pass to the shp2pgsql and raster2pgsql commands and get the desired effect on data when converting it. Considering that sufficient information is available, with each of these commands, if you can go through each of these options and find out the options combinations for.

1. The flags you will use when you must generate simple geometries, while dropping a table from your shapefile?

2. The flags you will use when you must create a GIST index and specifically extract a range of bands from a multiband raster file.

Based on our understanding of your previous training, can you convert the “pak\_river\_ocha.shp” file into a SQL script?

Kindly share your output with one of the coordinators in the lab.

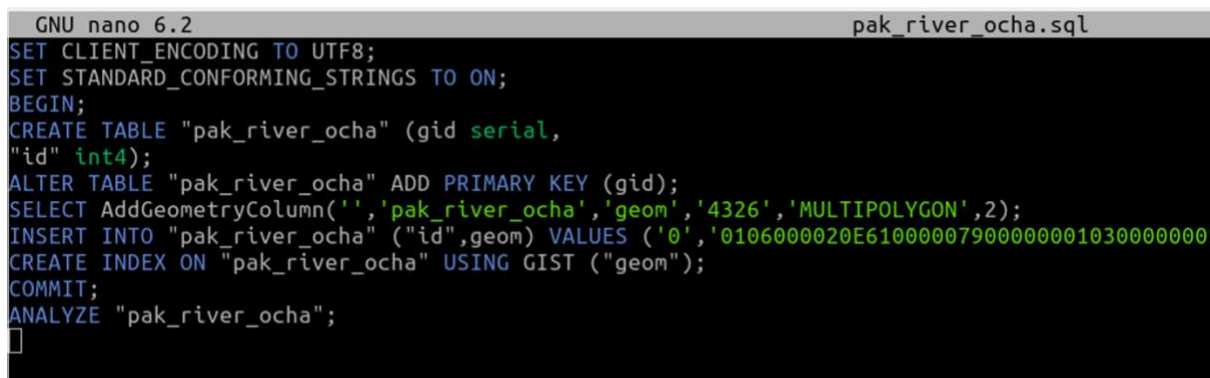
For those who still need to find the solution and do not already know how to do it, might take hints from the following syntax.

```
shp2pgsql -s 4326 -I pak_river_ocha.shp > pak_river_ocha.sql
```

If you run

```
nano pak_river_ocha.sql
```

You will see the following output.



```
GNU nano 6.2 pak_river_ocha.sql
SET CLIENT_ENCODING TO UTF8;
SET STANDARD_CONFORMING_STRINGS TO ON;
BEGIN;
CREATE TABLE "pak_river_ocha" (gid serial,
"id" int4);
ALTER TABLE "pak_river_ocha" ADD PRIMARY KEY (gid);
SELECT AddGeometryColumn('','pak_river_ocha','geom','4326','MULTIPOLYGON',2);
INSERT INTO "pak_river_ocha" ("id",geom) VALUES ('0','0106000020E610000079000000010300000000
CREATE INDEX ON "pak_river_ocha" USING GIST ("geom");
COMMIT;
ANALYZE "pak_river_ocha";

```

You may also want to run the following command in your terminal window, and you will get a different output of the same.

```
shp2pgsql -s 4326 -I pak_river_ocha.shp
```

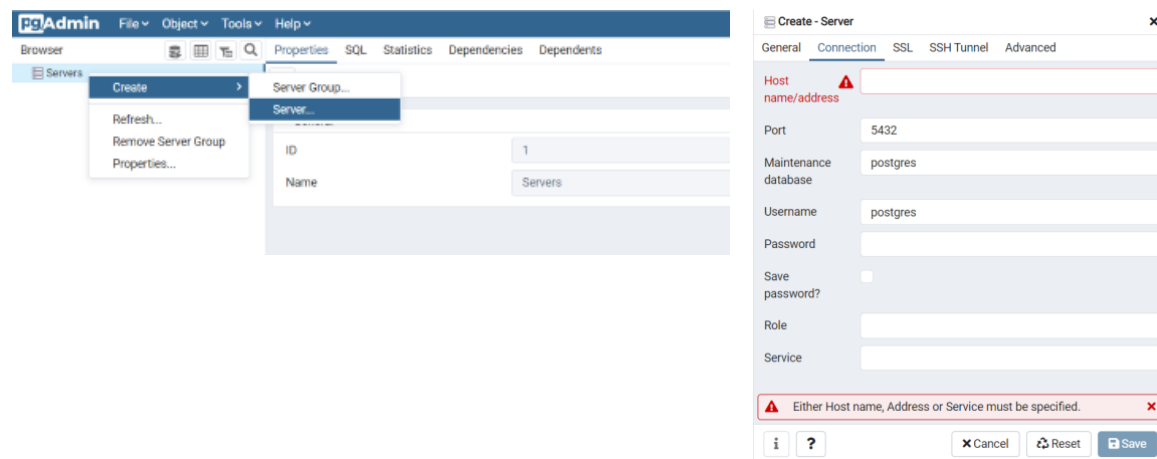
This directly prints the sql version of the file into your terminal window, but that is something you will rarely need.

You can also load the same into your PostGIS database, but for that you will need to create a database, for this purpose first. If you want you can use the traditional method of doing the same with, pgAdmin, that is appreciate, however for the sake, a more flexible approach and to initiate advanced, learning, it is recommended that you use, the command line interface in linux. It does not different, however once, you know the relevant scripts and commands, you can easily combine these lines of code into a complete script and run multiple tasks in one go. This is commonly referred to as a processing pipeline and does not require one to be a programmer.

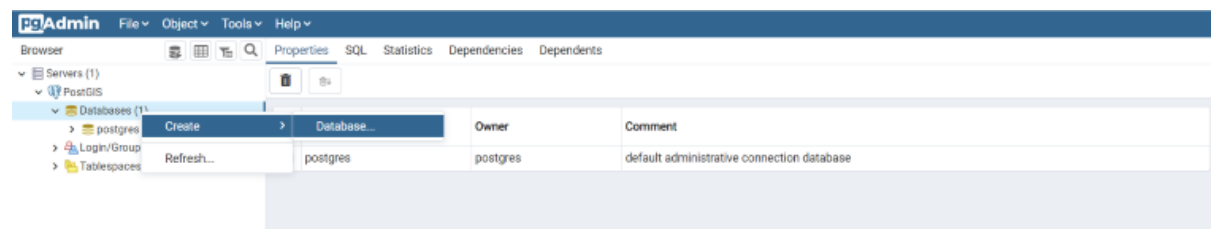
First, we will go through the traditional methods, and then we will go with the script-based command line approach.

Traditionally, you can open pgAdmin and connect to your database, after you have done so, you can create a database, using the following approach. Since, you have already made the database connection, you can skip the first two steps mentioned below.

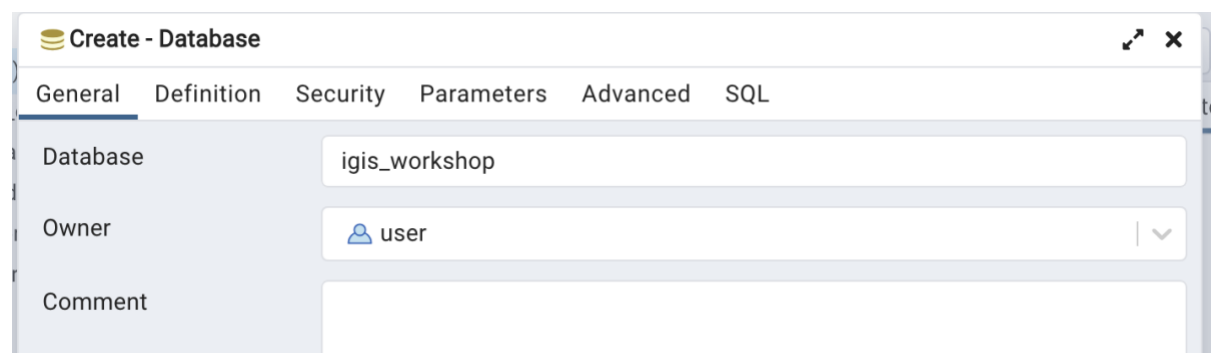
Create a new server and connect to it,



The you can create your new database by right clicking your server connection.



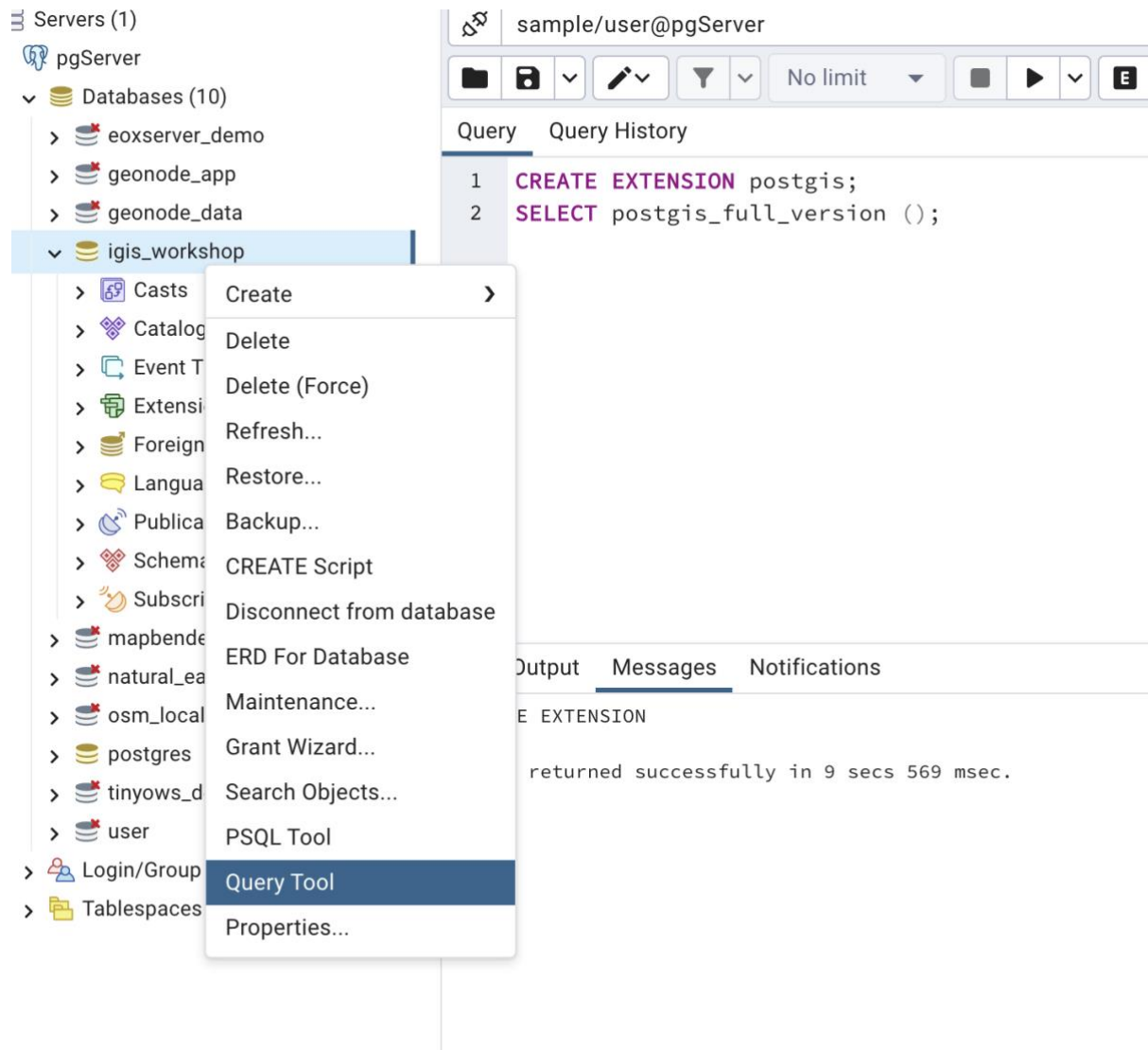
And add the relevant information as desired, as this stage we are leaving it as it is for the user, since there are no requirements, of ensuring rights and privileges. However, in a deployment level setting we might need to improve over that.



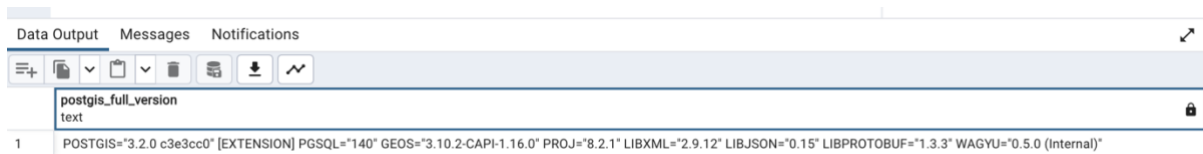
Once, your database is created, you can now make it spatial by adding the PostGIS extension and checking the extension installation through the following commands.

```
CREATE EXTENSION postgis;  
SELECT postgis_full_version ();
```

To do the above, you will need to right click on the database, and run the query tool as shown below.



If you can run the two seamlessly, you have gotten everything right. Your screen at the bottom will show the following output and this means, that the installation of postgis as well as all its required components is working fine. It is important to mention here, that if you are preparing a linux system from scratch, this might be daunting and can even, produce errors. But we believe that this exercise, will allow you to get a hand on how to start. And maybe in a future workshop we can work on how to create a personal deployable machine with postgres and postgis configured for deployment.



Here, it would be important to mention, that there are many providers online that create pre-packaged deployable solutions for full scale systems. One such provider is the OSGeoLive itself, others are Oracle and Bitnami.

Let's now create the same type of database, using command line utilities in our linux machine. For that, you can type the following command to get into your psql default shell.

First you will need to login in to your psql shell, since your username is user, it should not be much of a hassle, and you will not need to use advanced privileges. If that is not the case, things might get a little tricky, but can be managed.

Type the following command in your terminal

`psql -U user`

and you will see the following on your screen

```
user@user-virtualbox:~$ psql -U user
psql (14.4 (Ubuntu 14.4-0ubuntu0.22.04.1), server 14.9 (Ubuntu 14.9-0ubuntu0.22.04.1))
Type "help" for help.

user=#
```

To get a list of the database, you will need to type,

`\l`

(above is a short form of the world list) You will get the following output

```
user=# \l
```

List of databases					
Name	Owner	Encoding	Collate	Ctype	Access privileges
eooserver_demo	user	UTF8	en_US.UTF-8	en_US.UTF-8	
geonode_app	user	UTF8	en_US.UTF-8	en_US.UTF-8	
geonode_data	user	UTF8	en_US.UTF-8	en_US.UTF-8	
mapbender3.3.1	user	UTF8	en_US.UTF-8	en_US.UTF-8	
natural_earth2	user	UTF8	en_US.UTF-8	en_US.UTF-8	
osm_local	user	UTF8	en_US.UTF-8	en_US.UTF-8	
postgres	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	
template0	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres +
					postgres=CTc/postgres
template1	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres +
					postgres=CTc/postgres
tinyows_demo	user	UTF8	en_US.UTF-8	en_US.UTF-8	
user	user	UTF8	en_US.UTF-8	en_US.UTF-8	

(11 rows)

To be able to connect and import files to a specific database, you can use the following command.

You can do the following two things, either type

`\c <name of the database>`

You will see the prompt change on the left from user to the dbname

```
user=# \c osm_local
psql (14.4 (Ubuntu 14.4-0ubuntu0.22.04.1), server 14.9 (Ubuntu 14.9-0ubuntu0.22.04.1))
You are now connected to database "osm_local" as user "user".
```

To list the tables contained in it, you can type the

`\dt`

And the entire list of tables will appear on your screen.

```
osm_local=# \dt
          List of relations
Schema |      Name      | Type | Owner
-----+-----+-----+-----
public | planet_osm_line | table | user
public | planet_osm_point | table | user
public | planet_osm_polygon | table | user
public | planet_osm_roads | table | user
public | spatial_ref_sys | table | user
(5 rows)

osm_local=#
```

Some of the other commands you should try, once you are in your psql shell are

Task	PSQL COMMAND
Close connection	<code>\q</code>
Connect to a database	<code>\c &lt;database&gt;</code>
List databases	<code>\l</code>
Show table definition including triggers	<code>\d &lt;table&gt;</code>
List Schemas	<code>\dn</code>
List functions	<code>\df</code>
List views	<code>\dv</code>
Pretty-format	<code>\x</code>

Or use the argument when connecting to the psql shell as

`psql -U <user_name-database_name>`

It is usually a good approach, if you know which database you already must connect to, or else you get a list of your databases and table and that is also fine. To quit the psql shell you can simply type.

\q

and you will exit the shell prompt, but let's first create a database and then import some shapefiles and rasters. This can be done using the following workflow.

```
psql -u user
```

```
CREATE DATABASE exercise_db;  
CREATE EXTENSION postgis;  
SELECT PostGIS_Full_Version();
```

You will get the following output,

```
postgres_full_version  
-----  
PostGIS="3.2.0 c3e3cc0" [EXTENSION] PGSQL="140" GEOS="3.10.2-CAPI-1.16.0" PROJ="8.2.1" LIBXML="2.9.12" LIBJSON="0.15" LIBPROTOBUF="1.3.3" WAGYU="0.5.0 (Internal)"  
(1 row)  
(END)
```

To get out of this screen, you will need to press the button.

q

to list and see if your database is created you can again type the command.

\l

```
user=# \l  
List of databases  
+-----+-----+-----+-----+-----+  
Name | Owner | Encoding | Collate | Ctype | Access privileges  
+-----+-----+-----+-----+-----+  
eoxserver_demo | user | UTF8 | en_US.UTF-8 | en_US.UTF-8 |  
exercise_db | user | UTF8 | en_US.UTF-8 | en_US.UTF-8 |
```

Now connect your database and print the database tables and your screen should show you the following.

There is something wrong that we did, maybe if you do not see the following on your screen, can you correct it?

```
exercise_db=# \dt  
List of relations  
+-----+-----+-----+-----+  
Schema | Name | Type | Owner  
+-----+-----+-----+-----+  
public | spatial_ref_sys | table | user  
(1 row)  
exercise_db=#
```