Vamos a ver cómo te puedes a PostgreSQL desde Python. Con ello podrás ejecutar sentencias y consultas SQL sobre la base de datos. Sin embargo, antes de poder conectarte, necesitas configurar tu sistema.

Contenidos

* [**1 Instalación de los paquetes necesarios**](https://www.neoguias.com/como-conectarse-postgresql-python/#Instalacion_de_los_paquetes_necesarios)
* [**2 Cómo conectarte a una base de datos**](https://www.neoguias.com/como-conectarse-postgresql-python/#Como_conectarte_a_una_base_de_datos)
  + [**2.1 Conexión a PostgreSQL con Python mediante psycopg2**](https://www.neoguias.com/como-conectarse-postgresql-python/#Conexion_a_PostgreSQL_con_Python_mediante_psycopg2)
    - [**2.1.1 Función de conexión a PostgreSQL con psycopg2**](https://www.neoguias.com/como-conectarse-postgresql-python/#Funcion_de_conexion_a_PostgreSQL_con_psycopg2)
    - [**2.1.2 Conexión avanzada a PostgreSQL con psycopg2**](https://www.neoguias.com/como-conectarse-postgresql-python/#Conexion_avanzada_a_PostgreSQL_con_psycopg2)
  + [**2.2 Conexión a PostgreSQL con Python mediante PyGreSQL**](https://www.neoguias.com/como-conectarse-postgresql-python/#Conexion_a_PostgreSQL_con_Python_mediante_PyGreSQL)
    - [**2.2.1 Función de conexión a PostgreSQL con PyGreSQL**](https://www.neoguias.com/como-conectarse-postgresql-python/#Funcion_de_conexion_a_PostgreSQL_con_PyGreSQL)
    - [**2.2.2 Conexión avanzada a PostgreSQL con PyGreSQL**](https://www.neoguias.com/como-conectarse-postgresql-python/#Conexion_avanzada_a_PostgreSQL_con_PyGreSQL)
    - [**2.2.3 Conexión a PostgreSQL con el módulo Legacy de PyGreSQL**](https://www.neoguias.com/como-conectarse-postgresql-python/#Conexion_a_PostgreSQL_con_el_modulo_Legacy_de_PyGreSQL)
* [**3 Solución de problemas**](https://www.neoguias.com/como-conectarse-postgresql-python/#Solucion_de_problemas)

Instalación de los paquetes necesarios

Antes de poder conectarte a PostgreSQL utilizando Python, necesitas instalar una serie de paquetes, ya sea en tu sistema o en una máquina virtual. Decir que la instalación de los paquetes se realiza mediante PIP, el gestor de paquetes de Python. Si no tienes PIP instalado en tu sistema, consulta esta [**guía de instalación de PIP para Python**](https://www.neoguias.com/como-instalar-pip-python/). Solamente necesitas instalar uno de estos dos paquetes, dependiendo del que prefieras utilizar:

* **psycopg2**: Este paquete contiene el módulo psycopg2. Para instalar el paquete psycopg2 utiliza el siguiente comando:

pip install psycopg2

* **PyGreSQL**: Se trata de un paquete que contiene el módulo pgdb. Para instalar el paquete PyGreSQL utiliza el siguiente comando:

pip install pygresql

Estos dos paquetes soportan la **API de bases de datos de Python**, por lo que si decides cambiar de módulo en el futuro, podrás reutilizar la mayor parte del código que hayas escrito.

Cómo conectarte a una base de datos

Vamos a ver cómo te puedes conectar a una base de datos PostgreSQL existente utilizando tanto la función **psycopg2.connect** del módulo **psycopg2** como  la función **pgdb.connect** del módulo **PyGreSQL**, que hemos visto en el apartado anterior.

Conexión a PostgreSQL con Python mediante psycopg2

Vamos a ver cómo puedes crear una nueva sesión en al base de datos de un modo sencillo. Luego veremos un modo más adecuado para conectarnos, que se aproxima más a lo que se usa en el mundo real. Todo esto nos permitirá cosas como crear un cursor para ejecutar consultas o ejecutar y confirmar transacciones.

Función de conexión a PostgreSQL con psycopg2

Puedes especificar los **parámetros de la conexión** como una cadena de texto y pasársela a la función **connect** del siguiente modo:

conexion = psycopg2.connect("dbname=empleados user=neoguias password=pimientos44")

Debes reemplazar los datos del ejemplo por la base de datos a la que te quieres conectar junto a tus credenciales de acceso.

También te puedes utilizar una **lista de argumentos** y pasárselos a la función **connect** del siguiente modo:

conexion = psycopg2.connect(host="localhost", database="empleados", user="neoguias", password="pimientos44")

En estos ejemplos tenemos los siguientes parámetros:

* **host**: El nombre o la dirección IP del servidor al que te quieres conectar.
* **database/ dbname**: El nombre de la base de datos a la que te quieres conectar. En nuestro ejemplo, la base de datos se llama *«empleados»*.
* **user**: El nombre de usuario de PostgreSQL. En nuestro ejemplo, el nombre de usuario que utilizaremos es *«neoguias»*.
* **password**: La contraseña de acceso establecida para el usuario especificado. En nuestro ejemplo, la contraseña establecida es *«pimientos44»*.
* **port**: El número del puerto al que te quieres conectar. Si no introduces ningún valor, se utilizará el nombre del puerto por defecto, que es el **5432**.

Para ejecutar una consulta, primero debes crear un cursor utilizando el objeto que contiene la conexión que hemos creado, y luego introducir una consulta SQL:

# Creamos el cursor con el objeto conexion

cur = conexion.cursor()

# Ejecutamos una consulta

cur.execute( "SELECT nombre, apellidos FROM empleados" )

# Recorremos los resultados y los mostramos

for nombre, apellidos in cur.fetchall() :

print nombre, apellidos

# Cerramos la conexión

conexion.close()

Lo que hemos hecho es crear una conexión con los datos de acceso a la base de datos y, seguidamente, crear un cursor, que es lo que nos permitirá ejecutar sentencias SQL mediante la función **execute()**.

Veamos ahora cómo almacenar estos datos en un [**archivo**](https://www.neoguias.com/archivo/) separado.

Conexión avanzada a PostgreSQL con psycopg2

Lo más habitual es almacenar la configuración de acceso a la base de datos en un archivo **.ini**, u otro tipo de archivo cualquiera utilizado para almacenar configuraciones:

[postgresql]

host=localhost

database=empleados

user=neoguias

password=pimientos44

En el archivo anterior no hemos introducido el puerto, por lo que se utilizará el puerto por defecto, que es el puerto 5432. En caso de usar uno deferente, como el puerto 5433, agrega esta línea:

port=5433

Una vez introducidos todos los datos, guarda el archivo con el nombre de ***base\_de\_datos.ini***, puesto que lo vamos a utilizar en el siguiente ejemplo.

Vamos a crear una función llamada ***config()*** que lea los archivos de un archivo llamado ***base\_de\_datos.ini*** y devuelva los parámetros de conexión a la base de datos almacenados en el mismo.

Crea un archivo llamado **config.py** e introduce este código en su interior:

#!/usr/bin/python

from configparser import ConfigParser

def config(archivo='base\_de\_datos.ini', seccion='postgresql'):

# Crear el parser y leer el archivo

parser = ConfigParser()

parser.read(archivo)

# Obtener la sección de conexión a la base de datos

db = {}

if parser.has\_section(seccion):

params = parser.items(seccion)

for param in params:

db[param[0]] = param[1]

else:

raise Exception('Secccion {0} no encontrada en el archivo {1}'.format(seccion, archivo))

Desde ahora, para conectarnos a la base de datos de nombre ***empleados*** de nuestro ejemplo, podemos usar este código. Vamos a ver un ejemplo en el que seguimos los siguientes pasos (el código correspondiente está justo debajo):

1. Lo primero que hacemos en el ejemplo es leer los parámetros de configuración del archivo ***base\_de\_datos.ini***.
2. Seguidamente creamos una nueva conexión con la base de datos mediante el método ***psycopg2.connect()***, al que le pasamos los parámetros de conexión.
3. Seguidamente creamos un cursor al que llamamos ***cur*** mediante el método ***conexion.cursor()***, siendo ***conexion*** la instancia de conexión creada anteriormente.
4. Ejecutamos una consulta para, por ejemplo, obtener la versión de PostgreSQL. Para ello utilizamos la función **cur.execute()**, siendo ***cur*** el cursor.
5. Tras la lectura, leemos el resultado con el método ***cur.fetchone()***, siendo ***cur*** el cursor creado, que obtendrá le primer registro del resultado o el único que exista, en caso de haber solamente uno.
6. Luego imprimimos el resultado en pantalla.
7. Luego cerramos la comunicación con el cursor creado para liberar los recursos mediante la función ***cur.close()***.
8. Ya solamente nos queda cerrar la conexión a la base de datos con el método ***conexion.close()***, siendo ***conexion*** nuestra instancia de conexión.

Este es el código completo:

#!/usr/bin/python

import psycopg2

from config import config

def conectar():

""" Conexión al servidor de pases de datos PostgreSQL """

conexion = None

try:

# Lectura de los parámetros de conexion

params = config()

# Conexion al servidor de PostgreSQL

print('Conectando a la base de datos PostgreSQL...')

conexion = psycopg2.connect(\*\*params)

# creación del cursor

cur = conexion.cursor()

# Ejecución de una consulta con la version de PostgreSQL

print('La version de PostgreSQL es la:')

cur.execute('SELECT version()')

# Ahora mostramos la version

version = cur.fetchone()

print(version)

# Cierre de la comunicación con PostgreSQL

cur.close()

except (Exception, psycopg2.DatabaseError) as error:

print(error)

finally:

if conexion is not None:

conexion.close()

print('Conexión finalizada.')

if \_\_name\_\_ == '\_\_main\_\_':

conectar()

Y esto es todo. En el apartado de solución de problemas verás como solucionar algunos de los problemas más comunes.

Conexión a PostgreSQL con Python mediante *PyGreSQL*

Primero veremos cómo te puedes conectar del modo más sencillo, y luego mediante un archivo de configuración, que es lo que te encontrarás habitualmente.

Función de conexión a PostgreSQL con PyGreSQL

En este caso especificaremos los **parámetros de la conexión** como una cadena de texto y pasársela a la función **connect** del siguiente modo, reemplazando los datos del ejemplo por la base de datos a la que te quieres conectar y los datos de acceso correspondientes:

conexion = pgdb.connect(host="localhost",database="empleados", user="neoguias", password="pimientos44")

Estos son los parámetros que puedes configurar:

* **host**: El nombre o la dirección IP del servidor al que te quieres conectar.
* **database**: Nombre de la base de datos a la que te quieres conectar, que en nuestro ejemplo es la base de datos *empleados*.
* **user**: El nombre de usuario de la base de datos, que en nuestro ejemplo es *neoguias*.
* **password**: La contraseña que tenga el usuario especificado, que en nuestro ejemplo es *pimientos44*.
* **port**: Número de puerto de PostgreSQL. Es un parámetro opcional. Si no introduces ningún valor, se usará el **5432**.

Vamos a utilizar el objeto de conexión creado para ejecutar una consulta sobre al base de datos:

# Crea del cursor

cur = conexion.cursor()

# Ejecuta una consulta

cur.execute( "SELECT nombre, telefono FROM empleados" )

# Muestra los resultados

for nombre, telefono in cur.fetchall() :

print nombre, telefono

# Cierra la conexión

conexion.close()

Este ejemplo crea un objeto de conexión y utiliza la API de SQL para ejecutar una consulta. Cuando tienes un objeto de conexión asociado a una base de datos, puedes crear un cursor, que permite ejecutar consultas SQL mediante la sentencia **execute()**.

Veamos ahora cómo almacenar estos datos en un archivo separado.

Conexión avanzada a PostgreSQL con PyGreSQL

Al igual que hemos hecho con el otro módulo, también podemos crear un archivo **.ini** con los datos de acceso a PyGreSQL. Veamos cómo hacerlo:

Crea un archivo nuevo e introduce los datos de acceso a la base de datos con este formato. Guarda el archivo como **base\_de\_datos.ini**.

[postgresql]

host=localhost

database=empleados

user=neoguias

password=pimientos44

Crea un archivo llamado ***config.py*** e introduce este código, que se encargará de parsear el archivo de configuración **base\_de\_datos.ini** que acabas de crear, obteniendo los datos de la conexión:

#!/usr/bin/python

from configparser import ConfigParser

def config(archivo='base\_de\_datos.ini', seccion='postgresql'):

# Parsear el archivo

parser = ConfigParser()

parser.read(archivo)

# Ir a la sección de postgresql y extraer los parámetros

db = {}

if parser.has\_section(seccion):

params = parser.items(seccion)

for param in params:

db[param[0]] = param[1]

else:

raise Exception('Secccion {0} no encontrada en el archivo {1}'.format(seccion, archivo))

Ahora, debemos crear el script que cumpla nuestro cometido. En este ejemplo simplemente imprimiremos al versión de PostgreSQL. Utiliza este código:

#!/usr/bin/python

import pgdb

from config import config

def conectar():

""" Conexión al servidor de pases de datos PostgreSQL """

conexion = None

try:

# Lectura de los parámetros de conexion

params = config()

# Conexion al servidor de PostgreSQL

print('Conectando a la base de datos PostgreSQL...')

conexion = pgdb.connect(\*\*params)

# creación del cursor

cur = conexion.cursor()

# Ejecución la consulta para obtener la conexión

print('La version de PostgreSQL es la:')

cur.execute('SELECT version()')

# Se obtienen los resultados

db\_version = cur.fetchone()

# Se muestra la versión por pantalla

print(db\_version)

# Cerremos el cursor

cur.close()

except (Exception, psycopg2.DatabaseError) as error:

print(error)

finally:

if conexion is not None:

conexion.close()

print('Conexión finalizada.')

if \_\_name\_\_ == '\_\_main\_\_':

conectar()

Lo que hemos hechos es lo siguiente:

1. Obtenemos los datos del archivo ***base\_de\_datos.ini*** importando el archivo **config.py**.
2. Creamos una conexión con la base de datos utilizando ***pgdb.connect()***.
3. Seguidamente creamos el cursor ***cur*** con el método ***conexion.cursor()***.
4. Ejecutamos la consulta con **cur.execute()**.
5. Obtenemos el primer resultado con el método ***cur.fetchone()*** y lo mostramos por pantalla.
6. Finalmente cerramos la comunicación con ***cur.close()*** y la conexión con ***conexion.close()***.

Y esto es todo, aunque aún podemos conectarnos de otra forma, utilizando el módulo que viene por defecto para conectarse a PostgreSQL.

Conexión a PostgreSQL con el módulo Legacy de PyGreSQL

El paquete de **PyGreSQL** también incluye un módulo legacy llamado **pg** que puedes utilizar para conectarte a PostgreSQL. Con est módulo nos podremos conectar de un modo más sencillo, pero no se implementa la API portable de SQL para Python.

En este ejemplo, nos conectaremos a una base de datos PostgreSQL usando el módulo **pg**, que importamos al comienzo del script:

#!/usr/bin/python

import pg

# Creamos la conexión

conexion = pg.DB(host="localhost", user="neoguias", passwd="pimientos44", dbname="empleados")

# Ejecutamos una consulta y guaramos los resultados

resultados = conexion.query("SELECT nombre, telefono FROM empleados")

# Recorremos los resultados y los mostramos

for nombre, telefono in resultados.getresult() :

print nombre, telefono

# Cerramos la conexión

conexion.close()

Lo que hemos hecho es conectarnos a la base de datos de nombre ***empleados*** con el nombre de usuario ***neoguias*** y la contraseña ***pimientos44***. Luego obtenemos el nombre y el teléfono de los empleados ejecutando una consulta y obteniendo los resultados con **getresult()**. Finalmente, mostramos los resultados por pantalla y cerramos la conexión con **close()**.

Veamos ahora algunos problemas que pueden surgir.

Solución de problemas

Estos son algunos de los problemas que podrías tener.

* Si se muestra el error ***«FATAL: database «base\_de\_datos» does not exist»***, significará que la base de datos de nombre ***«base\_de\_datos»***, ***«empleados»***, o la que hayas introducido, no existe.  
  Para crearla, inicia una conexión con PostgreSQL desde la terminal y usa el siguiente comando, siendo **base\_de\_datos** el nombre de la base de datos, que en nuestro ejemplo sería ***«empleados»***.

CREATE DATABASE empleados;

* Si se muestra el error ***«FATAL: password authentication failed for user «usuario»***, significará que los datos de acceso no son los correctos, ya sea el nombre de usuario, la contraseña o ambos. Revisa los datos y cambia los parámetros.
* Si se muestra el error: ***«could not translate host name «servidor» to address: Unknown host»***, significará que el servidor de bases de datos introducido no es el correcto. habitualmente éste suele ser ***localhost*** si te conectas a un servidor local. Si se trata de un servidor remoto, asegúrate de introducir el nombre de servidor correcto.

Y esto ha sido todo.

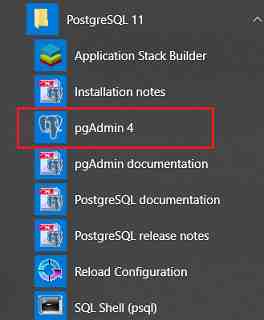
La tercer base de datos que veremos su acceso desde Python es PostgreSQL.

Se encuentra invitado para desarrollar un curso completo de [PostgreSQL Ya!](https://www.tutorialesprogramacionya.com/postgresqlya/)

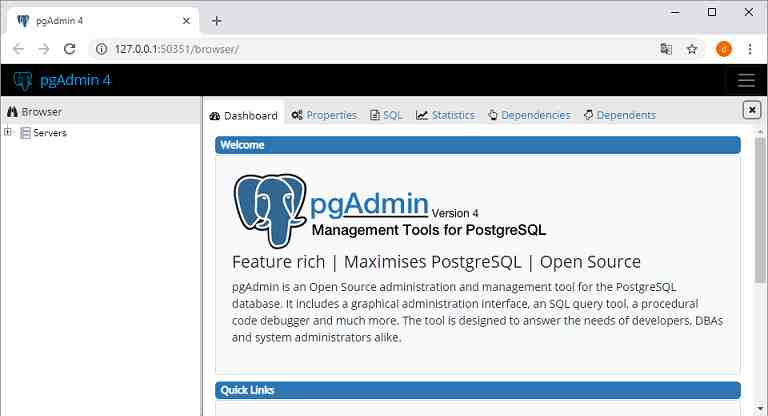
Tenga en cuenta que debe tener instalado el motor de base de datos PostgreSQL para poder seguir este concepto. En el tutorial de PostgreSQL están todos los pasos para su instalación.

### Creación de una base de datos y una tabla desde el programa pgAdmin.

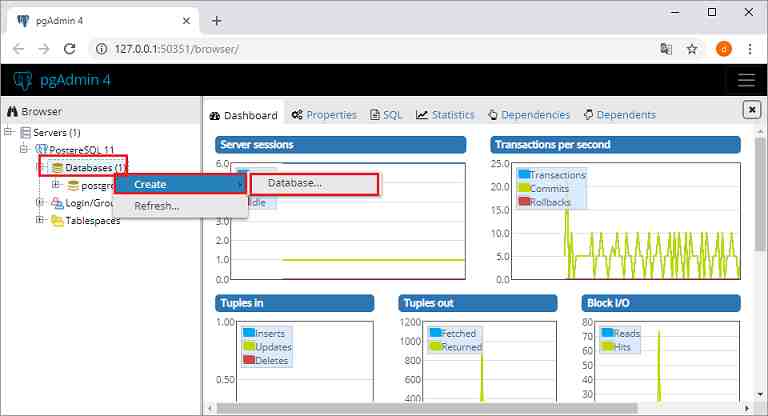
Ejecutemos el programa pgAdmin accediendo desde el menú de Windows:



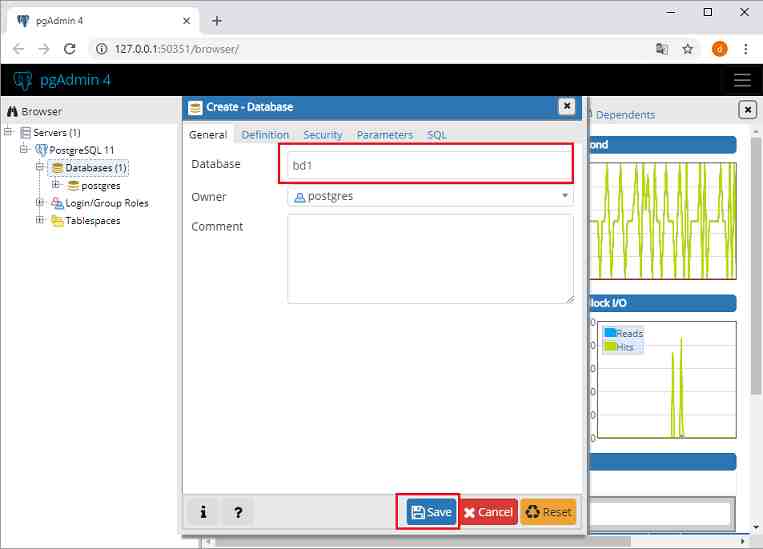
Se abre el navegador con la aplicación:



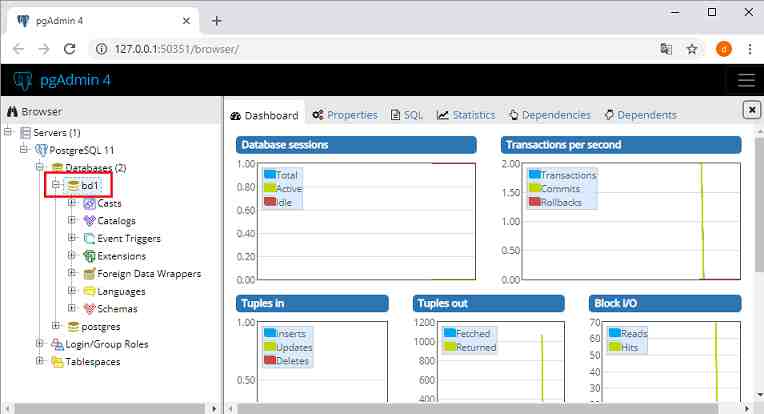
Para crear una base de datos presionamos el botón derecho del mouse donde dice "Databases" y seleccionamos la opción "Create"->"Database...":



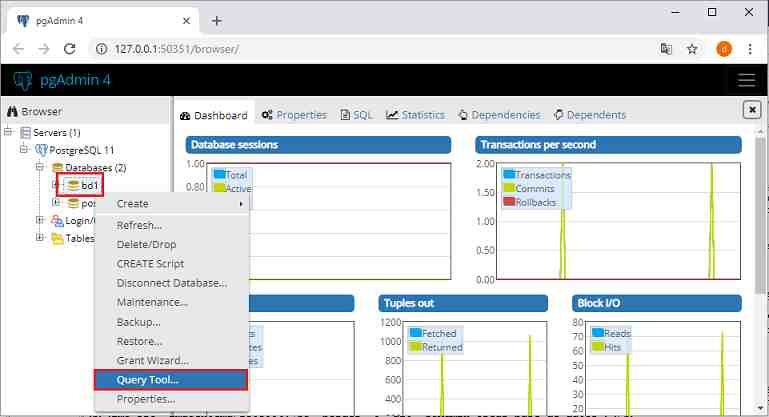
Aparece un diálogo donde debemos ingresar el nombre de la base de datos a crear, la llamaremos "bd1":



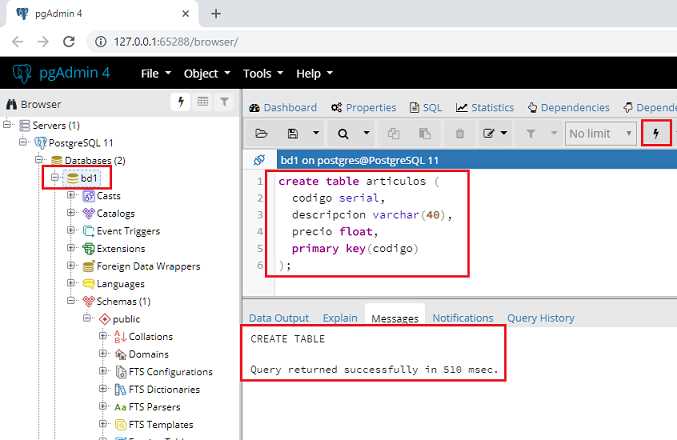
Ahora podemos seleccionar en la ventana de la izquierda la base de datos "bd1" que acabamos de crear:



Para poder ejecutar comandos SQL debemos presionar el botón derecho del mouse sobre el nombre de la base de datos "bd1" y seleccionar la opción "Query Tool..":



Ahora crearemos la tabla articulos en la base de datos "bd1":



### Paquete de Python necesario para conectarnos a PostgreSQL.

Utilizaremos el programa 'pip' que vimos anteriormente para instalar el paquete necesario para interconectar 'Python' y 'PostgreSQL'.

Desde la línea de comandos ejecutamos el programa pip con el siguiente paquete a instalar:

pip install psycopg2

Luego de ejecutar el programa pip podemos ver que nos informa que la instalación del paquete se efectuó correctamente:



### Conexión con el servidor de PostgreSQL.

El primer programa que implementaremos nos conectaremos con el servidor de PostgreSQL e insertaremos un par de filas en la tabla 'articulos' que creamos desde el programa 'pgAdmin'.

#### Programa: ejercicio333.py

import psycopg2

conexion1 = psycopg2.connect(database="bd1", user="postgres", password="heladera")

cursor1=conexion1.cursor()

sql="insert into articulos(descripcion, precio) values (%s,%s)"

datos=("naranjas", 23.50)

cursor1.execute(sql, datos)

datos=("peras", 34)

cursor1.execute(sql, datos)

datos=("bananas", 25)

cursor1.execute(sql, datos)

conexion1.commit()

conexion1.close()

Ejecutemos este programa para que se efectúe la carga de las tres filas en la tabla 'articulos' de la base de datos 'bd1'.

Por el momento si queremos controlar que se han cargado las tres filas en la tabla 'articulos' podemos abrir el 'pgAdmin' que viene com PostgreSQL y ver el contenido de la tabla:



Lo primero que hacemos es importar el módulo que nos permite conectarnos con PostgreSQL:

import psycopg2

Del módulo importado llamamos a la función connect pasando la ubicación con el nombre de la base de datos, nombre de usuario y la clave de dicho usuario:

conexion1 = psycopg2.connect(database="bd1", user="postgres", password="heladera")

Si por ejemplo el servidor de PostgreSQL no se encuentra en ejecución el programa se detendrá en esta línea informando un error.

Luego a partir del objeto 'conexion1' llamamos al método 'cursor':

cursor1=conexion1.cursor()

Definimos un string con el comando SQL insert disponiendo la máscara %s donde queremos que se sustituya por un valor que le pasaremos al método execute:

sql="insert into articulos(descripcion, precio) values (%s,%s)"

La variable datos es una tupla que contiene los datos que se utilizarán en la sustitución %s:

datos=("naranjas", 23.50)

Finalmente llamamos al método 'execute' y le pasamos las dos variables que acabamos de crear:

cursor1.execute(sql, datos)

Es fundamental llamar al final al método 'commit' para que queden firmes los comandos SQL 'insert':

conexion1.commit()

### Recuperar todas las filas de una tabla.

Implementaremos un programa que solicite ejecutar un 'select' en la tabla 'articulos' de la base de datos 'bd1' y nos retorne todas sus filas.

#### Programa: ejercicio334.py

import psycopg2

conexion1 = psycopg2.connect(database="bd1", user="postgres", password="123456")

cursor1=conexion1.cursor()

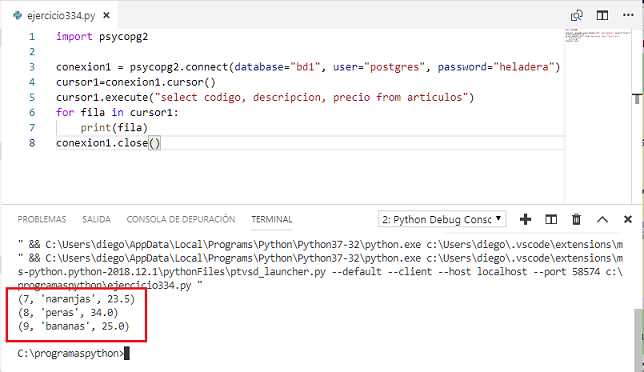
cursor1.execute("select codigo, descripcion, precio from articulos")

for fila in cursor1:

print(fila)

conexion1.close()

Cuando ejecutamos el programa podemos ver que se recuperan todas las filas de la tabla 'articulos':



Luego de conectarnos y crear un cursor procedemos a ejecutar el comando 'select', recorremos con un for el 'cursor1':

cursor1=conexion1.cursor()

cursor1.execute("select codigo, descripcion, precio from articulos")

for fila in cursor1:

print(fila)

### Borrado y modificación de filas.

Las otras dos actividades fundamentales que podemos hacer con una tabla es borrar filas y modificar datos.  
Desarrollaremos un pequeño programa que borre el artículo cuyo código sea el 1 y modifique el precio del artículo cuyo código sea 3.

#### Programa: ejercicio335.py

import psycopg2

conexion1 = psycopg2.connect(database="bd1", user="postgres", password="123456")

cursor1=conexion1.cursor()

cursor1.execute("delete from articulos where codigo=1")

cursor1.execute("update articulos set precio=50 where codigo=3")

conexion1.commit()

cursor1.execute("select codigo, descripcion, precio from articulos")

for fila in cursor1:

print(fila)

conexion1.close()

Cuando ejecutamos el programa podemos ver que se eliminó el artículo cuyo código es 1 y se modificó el precio del artículo con código 3:



Luego de crear el cursor podemos llamar al método 'execute' varias veces y pasar distintos comando SQL:

cursor1=conexion1.cursor()

cursor1.execute("delete from articulos where codigo=1")

cursor1.execute("update articulos set precio=50 where codigo=3")

Siempre que pasemos un comando SQL: insert, delete o update debemos llamar al método commit para que quede firme los cambios en la base de datos:

conexion1.commit()

Ejecutamos finalmente un 'select' para comprobar los cambios efectuados en la tabla 'articulos':

cursor1.execute("select codigo, descripcion, precio from articulos")

for fila in cursor1:

print(fila)

# Empezando con Django

## Observaciones[#](https://riptutorial.com/es/django#observaciones)

Django se anuncia a sí mismo como "el marco web para perfeccionistas con fechas límite" y "Django facilita la creación de mejores aplicaciones web más rápidamente y con menos código". Puede verse como una arquitectura MVC. En su núcleo tiene:

* Un servidor web ligero y autónomo para desarrollo y pruebas.
* un sistema de validación y serialización de formularios que se puede traducir entre formularios HTML y valores adecuados para el almacenamiento en la base de datos
* Un sistema de plantillas que utiliza el concepto de herencia prestada de la programación orientada a objetos.
* un marco de almacenamiento en caché que puede usar cualquiera de los varios métodos de caché compatibles con las clases de middleware que pueden intervenir en varias etapas del procesamiento de solicitudes y llevar a cabo funciones personalizadas
* un sistema de despacho interno que permite que los componentes de una aplicación se comuniquen entre sí mediante señales predefinidas
* un sistema de internacionalización, que incluye traducciones de los componentes propios de Django a una variedad de idiomas
* un sistema de serialización que puede producir y leer representaciones XML y / o JSON de instancias del modelo Django
* Un sistema para ampliar las capacidades del motor de plantillas.
* una interfaz para el marco de prueba de unidad incorporado de Python

## Versiones[#](https://riptutorial.com/es/django#versiones)

| **Versión** | **Fecha de lanzamiento** |
| --- | --- |
| [1.11](https://docs.djangoproject.com/en/stable/releases/1.11/) | 2017-04-04 |
| [1.10](https://docs.djangoproject.com/en/stable/releases/1.10/) | 2016-08-01 |
| [1.9](https://docs.djangoproject.com/en/stable/releases/1.9/) | 2015-12-01 |
| [1.8](https://docs.djangoproject.com/en/stable/releases/1.8/) | 2015-04-01 |
| [1.7](https://docs.djangoproject.com/en/stable/releases/1.7/) | 2014-09-02 |
| [1.6](https://docs.djangoproject.com/en/stable/releases/1.6/) | 2013-11-06 |
| [1.5](https://docs.djangoproject.com/en/stable/releases/1.5/) | 2013-02-26 |
| [1.4](https://docs.djangoproject.com/en/stable/releases/1.4/) | 2012-03-23 |
| [1.3](https://docs.djangoproject.com/en/stable/releases/1.3/) | 2011-03-23 |
| [1.2](https://docs.djangoproject.com/en/stable/releases/1.2/) | 2010-05-17 |
| [1.1](https://docs.djangoproject.com/en/stable/releases/1.1/) | 2009-07-29 |
| [1.0](https://docs.djangoproject.com/en/stable/releases/1.0/) | 2008-09-03 |

## Un ejemplo completo de hola mundo.[#](https://riptutorial.com/es/django#un-ejemplo-completo-de-hola-mundo-)

**Paso 1** Si ya tienes Django instalado, puedes omitir este paso.

pip install Django

**Paso 2** Crear un nuevo proyecto

django-admin startproject hello

Eso creará una carpeta llamada hello que contendrá los siguientes archivos:

hello/

├── hello/

│ ├── \_\_init\_\_.py

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

└── manage.py

**Paso 3** Dentro del módulo de hello (la carpeta que contiene el \_\_init.py\_\_ ) cree un archivo llamado views.py :

hello/

├── hello/

│ ├── \_\_init\_\_.py

│ ├── settings.py

│ ├── urls.py

│ ├── views.py <- here

│ └── wsgi.py

└── manage.py

y poner en el siguiente contenido:

from django.http import HttpResponse

def hello(request):

return HttpResponse('Hello, World')

Esto se llama una función de vista.

**Paso 4** Edita hello/urls.py como sigue:

from django.conf.urls import url

from django.contrib import admin

from hello import views

urlpatterns = [

url(r'^admin/', admin.site.urls),

url(r'^$', views.hello)

]

que enlaza la función de vista hello() a una URL.

**Paso 5** Inicia el servidor.

python manage.py runserver

**Paso 6**

Vaya a http://localhost:8000/ en un navegador y verá:

Hola Mundo

## Proyecto de implementación amigable con soporte Docker.[#](https://riptutorial.com/es/django#proyecto-de-implementacion-amigable-con-soporte-docker-)

La plantilla de proyecto Django predeterminada está bien, pero una vez que implementas tu código y, por ejemplo, los desarrolladores ponen sus manos en el proyecto, las cosas se complican. Lo que puede hacer es separar su código fuente del resto que se requiere para estar en su repositorio.

Puede encontrar una plantilla de proyecto Django utilizable en [GitHub](https://github.com/pkucmus/django-project-template) .

# Estructura del proyecto

PROJECT\_ROOT

├── devel.dockerfile

├── docker-compose.yml

├── nginx

│   └── project\_name.conf

├── README.md

├── setup.py

└── src

├── manage.py

└── project\_name

├── \_\_init\_\_.py

└── service

├── \_\_init\_\_.py

├── settings

│   ├── common.py

│   ├── development.py

│   ├── \_\_init\_\_.py

│   └── staging.py

├── urls.py

└── wsgi.py

Me gusta mantener el directorio de service denominado service para cada proyecto gracias a que puedo usar el mismo Dockerfile en todos mis proyectos. La división de requisitos y configuraciones ya está bien documentada aquí:  
[Usando múltiples archivos de requerimientos](https://riptutorial.com/fr/django/example/8561/using-multiple-requirements-files)  
[Usando múltiples configuraciones](https://riptutorial.com/fr/django/example/4567/using-multiple-settings)

# Dockerfile

Con el supuesto de que solo los desarrolladores hacen uso de Docker (no todos los desarrolladores de operaciones confían en ello en estos días). Esto podría ser un entorno de desarrollo devel.dockerfile :

FROM python:2.7

ENV PYTHONUNBUFFERED 1

RUN mkdir /run/service

ADD . /run/service

WORKDIR /run/service

RUN pip install -U pip

RUN pip install -I -e .[develop] --process-dependency-links

WORKDIR /run/service/src

ENTRYPOINT ["python", "manage.py"]

CMD ["runserver", "0.0.0.0:8000"]

Agregar solo los requisitos aprovechará la memoria caché de Docker mientras se construye, solo necesita reconstruir el cambio de requisitos.

# Componer

Docker compose es muy útil, especialmente cuando tiene múltiples servicios para ejecutar localmente. docker-compose.yml :

version: '2'

services:

web:

build:

context: .

dockerfile: devel.dockerfile

volumes:

- "./src/{{ project\_name }}:/run/service/src/{{ project\_name }}"

- "./media:/run/service/media"

ports:

- "8000:8000"

depends\_on:

- db

db:

image: mysql:5.6

environment:

- MYSQL\_ROOT\_PASSWORD=root

- MYSQL\_DATABASE={{ project\_name }}

nginx:

image: nginx

ports:

- "80:80"

volumes:

- "./nginx:/etc/nginx/conf.d"

- "./media:/var/media"

depends\_on:

- web

# Nginx

Su entorno de desarrollo debe ser lo más cercano posible al entorno prod, por lo que me gusta usar Nginx desde el principio. Aquí hay un ejemplo de archivo de configuración nginx:

server {

listen 80;

client\_max\_body\_size 4G;

keepalive\_timeout 5;

location /media/ {

autoindex on;

alias /var/media/;

}

location / {

proxy\_pass\_header Server;

proxy\_set\_header Host $http\_host;

proxy\_redirect off;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Scheme $scheme;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Ssl on;

proxy\_connect\_timeout 600;

proxy\_read\_timeout 600;

proxy\_pass http://web:8000/;

}

}

# Uso

$ cd PROJECT\_ROOT

$ docker-compose build web # build the image - first-time and after requirements change

$ docker-compose up # to run the project

$ docker-compose run --rm --service-ports --no-deps # to run the project - and be able to use PDB

$ docker-compose run --rm --no-deps <management\_command> # to use other than runserver commands, like makemigrations

$ docker exec -ti web bash # For accessing django container shell, using it you will be inside /run/service directory, where you can run ./manage shell, or other stuff

$ docker-compose start # Starting docker containers

$ docker-compose stop # Stopping docker containers

## Conceptos Django[#](https://riptutorial.com/es/django#conceptos-django)

**django-admin** es una herramienta de línea de comandos que se envía con Django. Viene con [varios comandos útiles](https://docs.djangoproject.com/en/stable/ref/django-admin/) para comenzar y administrar un proyecto Django. El comando es el mismo que ./manage.py , con la diferencia de que no es necesario que esté en el directorio del proyecto. La variable de entorno DJANGO\_SETTINGS\_MODULE debe configurarse.

Un **proyecto de Django** es un código base de Python que contiene un archivo de configuración de Django. El administrador de Django puede crear un proyecto mediante el comando [django-admin startproject NAME](https://docs.djangoproject.com/en/stable/ref/django-admin/#startproject). El proyecto normalmente tiene un archivo llamado manage.py en el nivel superior y un archivo URL raíz llamado urls.py manage.py es una versión específica del proyecto de django-admin , y le permite ejecutar comandos de administración en ese proyecto. Por ejemplo, para ejecutar su proyecto localmente, use python manage.py runserver . Un proyecto se compone de aplicaciones Django.

Una **aplicación de Django** es un paquete de Python que contiene un archivo de modelos ( models.py de forma predeterminada) y otros archivos, como urls y vistas específicas de la aplicación. Se puede crear una aplicación a través del comando [django-admin startapp NAME](https://docs.djangoproject.com/en/stable/ref/django-admin/#startapp)(este comando debe ejecutarse desde el directorio de su proyecto). Para que una aplicación sea parte de un proyecto, debe incluirse en la lista de [INSTALLED\_APPS](https://docs.djangoproject.com/en/stable/ref/settings/#installed-apps)en settings.py . Si usó la configuración estándar, Django viene con varias aplicaciones de sus propias aplicaciones preinstaladas que manejarán cosas como la [autenticación](https://docs.djangoproject.com/en/stable/topics/auth/) para usted. Las aplicaciones se pueden utilizar en múltiples proyectos de Django.

El **ORM de Django** recopila todos los modelos de base de datos definidos en models.py y crea tablas de base de datos basadas en esas clases de modelos. Para hacer esto, primero, configure su base de datos modificando la configuración de [DATABASES](https://docs.djangoproject.com/en/stable/ref/settings/#databases)en settings.py . Luego, una vez que haya definido sus [modelos de base de datos](https://docs.djangoproject.com/en/stable/topics/db/models/) , ejecute [python manage.py makemigrations](https://docs.djangoproject.com/en/stable/ref/django-admin/#makemigrations)seguido de [python manage.py migrate](https://docs.djangoproject.com/en/stable/ref/django-admin/#migrate)para crear o actualizar el esquema de su base de datos según sus modelos.

## Un solo archivo Hello World Ejemplo[#](https://riptutorial.com/es/django#un-solo-archivo-hello-world-ejemplo)

Este ejemplo muestra una forma mínima de crear una página Hello World en Django. Esto le ayudará a darse cuenta de que el comando de django-admin startproject example crea básicamente un montón de carpetas y archivos y que no necesariamente necesita esa estructura para ejecutar su proyecto.

1. Crea un archivo llamado file.py
2. Copia y pega el siguiente código en ese archivo.
3. import sys
5. from django.conf import settings
7. settings.configure(
8. DEBUG=True,
9. SECRET\_KEY='thisisthesecretkey',
10. ROOT\_URLCONF=\_\_name\_\_,
11. MIDDLEWARE\_CLASSES=(
12. 'django.middleware.common.CommonMiddleware',
13. 'django.middleware.csrf.CsrfViewMiddleware',
14. 'django.middleware.clickjacking.XFrameOptionsMiddleware',
15. ),
16. )
18. from django.conf.urls import url
19. from django.http import HttpResponse
21. # Your code goes below this line.
22. def index(request):
23. return HttpResponse('Hello, World!')
24. urlpatterns = [
25. url(r'^$', index),
26. ]
27. # Your code goes above this line
29. if \_\_name\_\_ == "\_\_main\_\_":
30. from django.core.management import execute\_from\_command\_line
32. execute\_from\_command\_line(sys.argv)

1. Vaya a la terminal y ejecute el archivo con este comando python file.py runserver .
2. Abra su navegador y vaya a [127.0.0.1:8000](http://127.0.0.1:8000/) .

## Comenzando un proyecto[#](https://riptutorial.com/es/django#comenzando-un-proyecto)

Django es un framework de desarrollo web basado en Python. Django **1.11** (la última versión estable) requiere la instalación de Python **2.7** , **3.4** , **3.5** o **3.6** . Suponiendo que pip está disponible, la instalación es tan simple como ejecutar el siguiente comando. Tenga en cuenta que omitir la versión como se muestra a continuación instalará la última versión de django:

$ pip install django

Para instalar una versión específica de django, supongamos que la versión es django **1.10.5** , ejecute el siguiente comando:

$ pip install django==1.10.5

Las aplicaciones web creadas con Django deben residir dentro de un proyecto de Django. Puede usar el comando django-admin para iniciar un nuevo proyecto en el directorio actual:

$ django-admin startproject myproject

donde myproject es un nombre que identifica de forma única el proyecto y puede constar de **números** , **letras** y **guiones bajos** .

Esto creará la siguiente estructura de proyecto:

myproject/

manage.py

myproject/

\_\_init\_\_.py

settings.py

urls.py

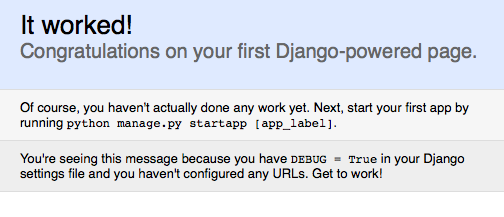
wsgi.py

Para ejecutar la aplicación, inicie el servidor de desarrollo.

$ cd myproject

$ python manage.py runserver

Ahora que el servidor está funcionando, visite http://127.0.0.1:8000/ con su navegador web. Verás la siguiente página:

[](https://i.stack.imgur.com/sABAE.png)

De forma predeterminada, el comando runserver inicia el servidor de desarrollo en la IP interna en el puerto 8000 . Este servidor se reiniciará automáticamente a medida que realice cambios en su código. Pero en caso de que agregue nuevos archivos, deberá reiniciar manualmente el servidor.

Si desea cambiar el puerto del servidor, páselo como un argumento de línea de comandos.

$ python manage.py runserver 8080

Si desea cambiar la IP del servidor, pásela junto con el puerto.

$ python manage.py runserver 0.0.0.0:8000

Tenga en cuenta que *runserver* es solo para compilaciones de depuración y pruebas locales. Los programas de servidor especializados (como Apache) siempre deben usarse en producción.

**Añadiendo una aplicación Django**

Un proyecto de Django usualmente contiene múltiples apps . Esta es simplemente una forma de estructurar su proyecto en módulos más pequeños y mantenibles. Para crear una aplicación, vaya a su carpeta de proyecto (donde manage.py es), y ejecute el comando startapp (cambie myapp a lo que quiera):

python manage.py startapp myapp

Esto generará la carpeta myapp y algunos archivos necesarios para usted, como models.py y views.py .

Para que Django sea consciente de myapp , agréguelo a su settings.py :

# myproject/settings.py

# Application definition

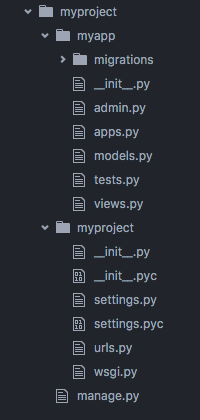
INSTALLED\_APPS = [

...

'myapp',

]

La estructura de carpetas de un proyecto de Django se puede cambiar para que se ajuste a sus preferencias. A veces, la carpeta del proyecto cambia de nombre a /src para evitar repetir los nombres de las carpetas. Una estructura de carpetas típica se ve así:

[](https://i.stack.imgur.com/LRsRO.png)

## Ambiente virtual[#](https://riptutorial.com/es/django#ambiente-virtual)

Aunque no es estrictamente necesario, se recomienda iniciar su proyecto en un "entorno virtual". Un entorno virtual es un **contenedor** (un directorio) que contiene una versión específica de Python y un conjunto de módulos (dependencias), y que no interfiere con el Python nativo del sistema operativo u otros proyectos en la misma computadora.

Al configurar un entorno virtual diferente para cada proyecto en el que trabaja, varios proyectos de Django pueden ejecutarse en diferentes versiones de Python y pueden mantener sus propios conjuntos de dependencias, sin riesgo de conflicto.

# Python 3.3+

Python 3.3+ ya incluye un módulo venv estándar, al que normalmente puedes llamar como pyvenv . En entornos donde el comando pyvenv no está disponible, puede acceder a la misma funcionalidad invocando directamente el módulo como python3 -m venv .

Para crear el entorno virtual:

$ pyvenv <env-folder>

# Or, if pyvenv is not available

$ python3 -m venv <env-folder>

# Python 2

Si usa Python 2, primero puede instalarlo como un módulo separado de pip:

$ pip install virtualenv

Y luego cree el entorno usando el comando virtualenv lugar:

$ virtualenv <env-folder>

# Activar (cualquier versión)

El entorno virtual ya está configurado. Para poder utilizarlo, debe estar activado en el terminal que desea utilizar.

Para 'activar' el entorno virtual (cualquier versión de Python)

Linux como:

$ source <env-folder>/bin/activate

Windows como:

<env-folder>\Scripts\activate.bat

Esto cambia su indicador para indicar que el entorno virtual está activo. (<env-folder>) $

De ahora en adelante, todo lo que se instale usando pip se instalará en su carpeta de env virtual, no en todo el sistema.

Para salir del entorno virtual utilice deactivate :

(<env-folder>) $ deactivate

# Alternativamente: use virtualenvwrapper

También puede considerar el uso de [virtualenvwrapper,](http://virtualenvwrapper.readthedocs.io/) que hace que la creación y activación de virtualenv sea muy útil, así como la separación de su código:

# Create a virtualenv

mkvirtualenv my\_virtualenv

# Activate a virtualenv

workon my\_virtualenv

# Deactivate the current virtualenv

deactivate

# Alternativamente: use pyenv + pyenv-viritualenv

En entornos en los que necesita manejar varias versiones de Python, puede beneficiarse de virtualenv junto con pyenv-virtualenv:

# Create a virtualenv for specific Python version

pyenv virtualenv 2.7.10 my-virtual-env-2.7.10

# Create a vritualenv for active python verion

pyenv virtualenv venv34

# Activate, deactivate virtualenv

pyenv activate <name>

pyenv deactivate

Cuando se utiliza virtualenvs, a menudo es útil configurar PYTHONPATH y DJANGO\_SETTINGS\_MODULE en el [script postactivate](http://virtualenvwrapper.readthedocs.io/en/latest/scripts.html#postactivate).

#!/bin/sh

# This hook is sourced after this virtualenv is activated

# Set PYTHONPATH to isolate the virtualenv so that only modules installed

# in the virtualenv are available

export PYTHONPATH="/home/me/path/to/your/project\_root:$VIRTUAL\_ENV/lib/python3.4"

# Set DJANGO\_SETTINGS\_MODULE if you don't use the default `myproject.settings`

# or if you use `django-admin` rather than `manage.py`

export DJANGO\_SETTINGS\_MODULE="myproject.settings.dev"

# Establece la ruta de tu proyecto

A menudo también es útil configurar la ruta de su proyecto dentro de un archivo .project especial ubicado en su <env-folder> . Al hacer esto, cada vez que active su entorno virtual, cambiará el directorio activo a la ruta especificada.

Cree un nuevo archivo llamado <env-folder>/.project . El contenido del archivo SOLO debe ser la ruta del directorio del proyecto.

/path/to/project/directory

Ahora, inicie su entorno virtual (ya sea utilizando source <env-folder>/bin/activate workon my\_virtualenv o workon my\_virtualenv ) y su terminal cambiará los directorios a /path/to/project/directory .

# How to import CSV file data into a PostgreSQL table?

Solution paraphrased here:

**Create your table:**

CREATE TABLE zip\_codes

(ZIP char(5), LATITUDE double precision, LONGITUDE double precision,

CITY varchar, STATE char(2), COUNTY varchar, ZIP\_CLASS varchar);

**Copy data from your CSV file to the table:**

COPY zip\_codes FROM '/path/to/csv/ZIP\_CODES.txt' WITH (FORMAT csv);

[Fuente](https://stackoverflow.com/q/2987451) [Compartir](https://stackoverrun.com/es/q/656737#2987451)

Crear 07 jun. 10 [Bozhidar Batsov](https://stackoverflow.com/users/291550/)

27

You could also use pgAdmin, which offers a GUI to do the import. That's shown in this [SO thread](https://stackoverflow.com/questions/19400173/how-should-i-import-data-from-csv-into-a-postgres-table-using-pgadmin-3). The advantage of using pgAdmin is that it also works for remote databases.

Much like the previous solutions though, you would need to have your table on the database already. Each person has his own solution but what I usually do is open the CSV in Excel, copy the headers, paste special with transposition on a different worksheet, place the corresponding data type on the next column then just copy and paste that to a text editor together with the appropriate SQL table creation query like so:

CREATE TABLE my\_table (

/\*paste data from Excel here for example ... \*/

col\_1 bigint,

col\_2 bigint,

/\* ... \*/

col\_n bigint

)

[Fuente](https://stackoverflow.com/q/26721945) [Compartir](https://stackoverrun.com/es/q/656737#26721945)

Crear 03 nov. 14 [Paul](https://stackoverflow.com/users/112044/)

50

One quick way of doing this is with the Python pandas library (version 0.15 or above works best). This will handle creating the columns for you - although obviously the choices it makes for data types might not be what you want. If it doesn't quite do what you want you can always use the 'create table' code generated as a template.

Here's a simple example:

import pandas as pd

df = pd.read\_csv('mypath.csv')

df.columns = [c.lower() for c in df.columns] #postgres doesn't like capitals or spaces

from sqlalchemy import create\_engine

engine = create\_engine('postgresql://username:password@localhost:5432/dbname')

df.to\_sql("my\_table\_name", engine)

And here's some code that shows you how to set various options:

#Set is so the raw sql output is logged

import logging

logging.basicConfig()

logging.getLogger('sqlalchemy.engine').setLevel(logging.INFO)

df.to\_sql("my\_table\_name2",

engine,

if\_exists="append", #options are ‘fail’, ‘replace’, ‘append’, default ‘fail’

index=False, #Do not output the index of the dataframe

dtype={'col1': sqlalchemy.types.NUMERIC,

'col2': sqlalchemy.types.String}) #Datatypes should be [sqlalchemy types][1]

[Fuente](https://stackoverflow.com/q/29722393) [Compartir](https://stackoverrun.com/es/q/656737#29722393)

Crear 18 abr. 15 [RobinL](https://stackoverflow.com/users/1779128/)

13

Most other solutions here require that you create the table in advance/manually. This may not be practical in some cases (e.g., if you have a lot of columns in the destination table). So, the approach below may come handy.

Providing the path and column count of your csv file, you can use the following function to load your table to a temp table that will be named as target\_table:

The top row is assumed to have the column names.

create or replace function data.load\_csv\_file

(

target\_table text,

csv\_path text,

col\_count integer

)

returns void as $$

declare

iter integer; -- dummy integer to iterate columns with

col text; -- variable to keep the column name at each iteration

col\_first text; -- first column name, e.g., top left corner on a csv file or spreadsheet

begin

set schema 'your-schema';

create table temp\_table ();

-- add just enough number of columns

for iter in 1..col\_count

loop

execute format('alter table temp\_table add column col\_%s text;', iter);

end loop;

-- copy the data from csv file

execute format('copy temp\_table from %L with delimiter '','' quote ''"'' csv ', csv\_path);

iter := 1;

col\_first := (select col\_1 from temp\_table limit 1);

-- update the column names based on the first row which has the column names

for col in execute format('select unnest(string\_to\_array(trim(temp\_table::text, ''()''), '','')) from temp\_table where col\_1 = %L', col\_first)

loop

execute format('alter table temp\_table rename column col\_%s to %s', iter, col);

iter := iter + 1;

end loop;

-- delete the columns row

execute format('delete from temp\_table where %s = %L', col\_first, col\_first);

-- change the temp table name to the name given as parameter, if not blank

if length(target\_table) > 0 then

execute format('alter table temp\_table rename to %I', target\_table);

end if;

end;

$$ language plpgsql;

[Fuente](https://stackoverflow.com/q/30083588) [Compartir](https://stackoverrun.com/es/q/656737#30083588)

Crear 06 may. 15 [mehmet](https://stackoverflow.com/users/2863603/)

127

If you don't have permission to use COPY (which work on the db server), you can use \copy instead (which works in the db client). Using the same example as Bozhidar Batsov:

**Create your table:**

CREATE TABLE zip\_codes

(ZIP char(5), LATITUDE double precision, LONGITUDE double precision,

CITY varchar, STATE char(2), COUNTY varchar, ZIP\_CLASS varchar);

**Copy data from your CSV file to the table:**

\copy zip\_codes FROM '/path/to/csv/ZIP\_CODES.txt' DELIMITER ',' CSV

You can also specify the columns to read:

\copy zip\_codes(ZIP,CITY,STATE) FROM '/path/to/csv/ZIP\_CODES.txt' DELIMITER ',' CSV

[Fuente](https://stackoverflow.com/q/30951435) [Compartir](https://stackoverrun.com/es/q/656737#30951435)

Crear 20 jun. 15 [bjelli](https://stackoverflow.com/users/153574/)

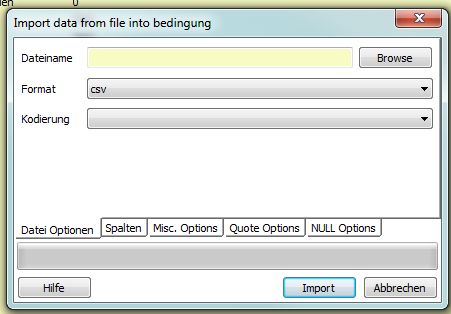
18

As Paul mentioned, import works in pgAdmin:

right click on table -> import

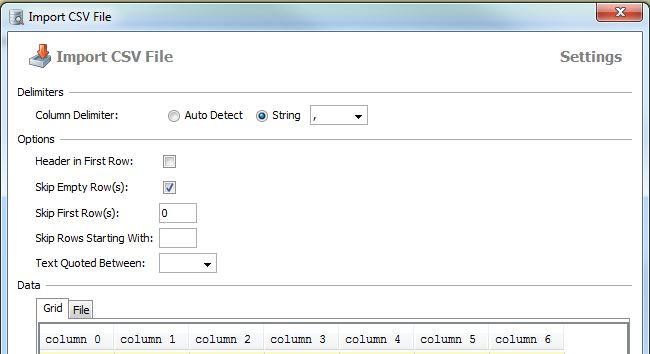
select local file, format and coding

here is a german pgAdmin GUI screenshot:

[](https://i.stack.imgur.com/z9M5o.jpg)

similar thing you can do with DbVisualizer (I have a license, not sure about free version)

right click on a table -> Import Table Data...

[](https://i.stack.imgur.com/107St.jpg)

[Fuente](https://stackoverflow.com/q/32626274) [Compartir](https://stackoverrun.com/es/q/656737#32626274)

Crear 17 sep. 15 [Andreas L.](https://stackoverflow.com/users/2583733/)

3

IMHO, the most convenient way is to follow "[Import CSV data into postgresql, the comfortable way ;-)](http://blog.aplikate.eu/2015/01/03/import-csv-data-into-postgresql-the-comfortable-way/)", using [csvsql](https://csvkit.readthedocs.org/en/0.9.1/scripts/csvsql.html) from [csvkit](https://csvkit.readthedocs.org/), which is a python package installable via pip.

[Fuente](https://stackoverflow.com/q/33581389) [Compartir](https://stackoverrun.com/es/q/656737#33581389)

Crear 07 nov. 15 [sal](https://stackoverflow.com/users/2772305/)

12

COPY table\_name FROM 'path/to/data.csv' DELIMITER ',' CSV HEADER;

[Fuente](https://stackoverflow.com/q/40640306) [Compartir](https://stackoverrun.com/es/q/656737#40640306)

Crear 16 nov. 16 [Tim Siwula](https://stackoverflow.com/users/1482105/)

4

Use this SQL code

copy table\_name(atribute1,attribute2,attribute3...)

from 'E:\test.csv' delimiter ',' csv header

the header keyword lets the DBMS know that the csv file have a header with attributes

for more visit <http://www.postgresqltutorial.com/import-csv-file-into-posgresql-table/>

[Fuente](https://stackoverflow.com/q/41607362) [Compartir](https://stackoverrun.com/es/q/656737#41607362)

Crear 12 ene. 17 [djdere](https://stackoverflow.com/users/7404825/)

0

Create table and have required columns that are used for creating table in csv file.

1. Open postgres and right click on target table which you want to load & select import and Update the following steps in **file options** section
2. Now browse your file in filename
3. Select csv in format
4. Encoding as ISO\_8859\_5

Now goto **Misc. options** and check header and click on import.

[Fuente](https://stackoverflow.com/q/44900705) [Compartir](https://stackoverrun.com/es/q/656737#44900705)

Crear 04 jul. 17 [suriruler](https://stackoverflow.com/users/8135480/)

4

Personal experience with PostgreSQL, still waiting for a faster way.

**1. Create table skeleton first if the file is stored locally:**

drop table if exists ur\_table;

CREATE TABLE ur\_table

(

id serial NOT NULL,

log\_id numeric,

proc\_code numeric,

date timestamp,

qty int,

name varchar,

price money

);

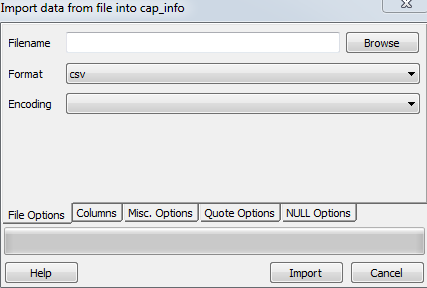
COPY

ur\_table(id, log\_id, proc\_code, date, qty, name, price)

FROM '\path\xxx.csv' DELIMITER ',' CSV HEADER;

**2. When the \path\xxx.csv is on the server, postgreSQL doesn't have the permission to access the server, you will have to import the .csv file through the pgAdmin built in functionality.**

Right click the table name choose import.

[](https://i.stack.imgur.com/qeQ6V.png)

If you still have problem, please refer this tutorial. <http://www.postgresqltutorial.com/import-csv-file-into-posgresql-table/>

[Fuente](https://stackoverflow.com/q/45333329) [Compartir](https://stackoverrun.com/es/q/656737#45333329)

Crear 26 jul. 17 [flowera](https://stackoverflow.com/users/5667269/)

0

1) create a table first

2) Then use copy command to copy the table details:

**copy** table\_name (C1,C2,C3....) **from** 'path to your csv file' delimiter ',' csv header;

Thanks

# Cómo exportar una tabla de PostgreSQL – PostGIS a CSV

por [Aurelio Morales](https://mappinggis.com/author/aurelio/)

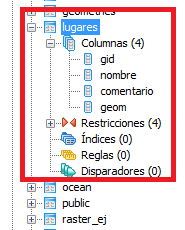
[](http://twitter.com/share?url=https://mappinggis.com/2013/12/como-exportar-una-tabla-de-postgresql-csv/&text=C%C3%B3mo%20exportar%20una%20tabla%20de%20PostgreSQL%20%E2%80%93%20PostGIS%20a%20CSV%20)[](http://www.linkedin.com/shareArticle?mini=true&url=https://mappinggis.com/2013/12/como-exportar-una-tabla-de-postgresql-csv/)[](http://www.facebook.com/sharer.php?u=https://mappinggis.com/2013/12/como-exportar-una-tabla-de-postgresql-csv/)[](mailto:?subject=C%C3%B3mo%20exportar%20una%20tabla%20de%20PostgreSQL%20%26%238211;%20PostGIS%20a%20CSV&body=%20https://mappinggis.com/2013/12/como-exportar-una-tabla-de-postgresql-csv/)

En esta entrada vamos ver qué pasos hay que seguir para exportar el contenido de una tabla de PostgreSQL a formato CSV, para poder trabajar con ella en una hoja de cálculo.

Exportar una tabla de PostgreSQL es muy sencillo usando el comando COPY que nos permite copiar tablas entre un archivo y una tabla. Tan solo tenemos que definir:

* La tabla a exportar.
* Las columnas que queremos exportar (es opcional, si no especificamos ninguna se exportarán todas).
* La ubicación y el nombre del archivo DBF (como rutas absolutas).
* El delimitador.
* Definir la cabecera.

En primer lugar localizamos la tabla a exportar:



Podemos ejecutar la siguiente consulta SQL:

COPY lugares(«gid», «nombre», «comentario») TO ‘C:\Temp\lugares.csv’ delimiters ‘;’ WITH CSV HEADER;

El **delimitador ‘;’** es el carácter ASCII único que separa las columnas de cada fila (línea) del archivo. El valor predeterminado en modo texto es un carácter de tabulación o una coma en el modo CSV.

Incluimos la palabra **HEADER** para especificar que el archivo creado contenga una línea de cabecera con los nombres de las columnas de las tablas en el archivo.



Al ejecutar la consulta se generará un archivo .csv en la ruta indicada con este aspecto:

[](http://mappinggis.com/wp-content/uploads/2013/12/csv-resultado.png)

Más info: [http://www.postgresql.org/docs/8.2/static/sql-copy.ht](http://www.postgresql.org/docs/8.2/static/sql-copy.html)

# 10 motivos para utilizar PostGIS

por [Aurelio Morales](https://mappinggis.com/author/aurelio/)

[](http://twitter.com/share?url=https://mappinggis.com/2012/09/por-que-utilizar-postgis/&text=10%20motivos%20para%20utilizar%20PostGIS%20)[](http://www.linkedin.com/shareArticle?mini=true&url=https://mappinggis.com/2012/09/por-que-utilizar-postgis/)[](http://www.facebook.com/sharer.php?u=https://mappinggis.com/2012/09/por-que-utilizar-postgis/)[](mailto:?subject=10%20motivos%20para%20utilizar%20PostGIS&body=%20https://mappinggis.com/2012/09/por-que-utilizar-postgis/)

utilizar PostGISEn esta entrada veremos los motivos y las ventajas que tiene utilizar PostGIS frente a otras opciones.

PostGIS es una extensión que convierte el sistema de base de datos PostgreSQL en una base de datos espacial. La combinación de ambos es una solución perfecta para el almacenamiento, gestión y mantenimiento de datos espaciales. En una entrada anterior hemos visto [por qué no debemos utilizar shapefiles](https://mappinggis.com/2012/08/02/shapefiles-vs-bases-de-datos-espaciales/) para almacenar datos espaciales por las desventajas que conlleva.

PostGIS is as powerful (or more) than many desktop GIS systems.

Paul Ramsey

Actualmente es la base de datos espacial de código abierto más ámpliamente utilizada. Muchas y muy variadas organizaciones de todo el mundo usan PostGIS, incluyendo agencias gubernamentales de riesgos adversos y organizaciones que almacenan terabytes de datos y sirven millones de peticiones web al día.

Debido a que está construido sobre PostgreSQL, PostGIS hereda automáticamente sus características, así como los estándares abiertos. Algunas de las características que le hacen único:

**Índice**[[cerrar](https://mappinggis.com/2012/09/por-que-utilizar-postgis/)]

* [1. PostGIS es software libre y es compatible con los estándares de Open Geospatial Consortium (OGC)](https://mappinggis.com/2012/09/por-que-utilizar-postgis/#1_PostGIS_es_software_libre_y_es_compatible_con_los_estandares_de_Open_Geospatial_Consortium_OGC)
* [2. Permite el acceso de usuarios simultáneos](https://mappinggis.com/2012/09/por-que-utilizar-postgis/#2_Permite_el_acceso_de_usuarios_simultaneos)
* [3. Funciones espaciales mediante SQL y trabajar con topología](https://mappinggis.com/2012/09/por-que-utilizar-postgis/#3_Funciones_espaciales_mediante_SQL_y_trabajar_con_topologia)
* [4. PostGIS incluye un visor de geometrías integrado en pgAdmin 4](https://mappinggis.com/2012/09/por-que-utilizar-postgis/#4_PostGIS_incluye_un_visor_de_geometrias_integrado_en_pgAdmin_4)
* [5. Permite trabajar con triggers](https://mappinggis.com/2012/09/por-que-utilizar-postgis/#5_Permite_trabajar_con_triggers)
* [6. Enrutamiento](https://mappinggis.com/2012/09/por-que-utilizar-postgis/#6_Enrutamiento)
* [7. Tipos de geometría](https://mappinggis.com/2012/09/por-que-utilizar-postgis/#7_Tipos_de_geometria)
* [8. Es rápido y seguro](https://mappinggis.com/2012/09/por-que-utilizar-postgis/#8_Es_rapido_y_seguro)
* [9. Sencilla importación y exportación de datos](https://mappinggis.com/2012/09/por-que-utilizar-postgis/#9_Sencilla_importacion_y_exportacion_de_datos)
* [10. Existen decenas de aplicaciones que trabajan bien con PostGIS](https://mappinggis.com/2012/09/por-que-utilizar-postgis/#10_Existen_decenas_de_aplicaciones_que_trabajan_bien_con_PostGIS)

## 1. PostGIS es software libre y es compatible con los estándares de Open Geospatial Consortium (OGC)

Tiene licencia GNU General Public License (GPL). Es pues gratuito y una alternativa real al software propietario superándole en estabilidad y rapidez.

Es compatible con los estándares de [***Open Geospatial Consortium***](https://www.opengeospatial.org/) (OGC), con el objetivo de facilitar el intercambio de información geográfica.

## 2. Permite el acceso de usuarios simultáneos

Por otro lado, QGIS fue desarrollado originalmente para ser un cliente SIG capaz de visualizar los datos almacenados en PostGIS, por lo que tiene capacidad de edición multiusuario integrada en su núcleo. La base de datos maneja todo lo relacionado con el almacenamiento de datos, que es muy bueno, mientras que QGIS se centra en lo que es bueno, que es la edición y visualización de datos espaciales.

## 3. Funciones espaciales mediante SQL y trabajar con topología

Soporta tipos de datos espaciales, índices espaciales y tiene cientos de funciones espaciales (+ 1000). Estas funciones espaciales incluyen herramientas para convertir sistemas de coordenadas, medición (distancia y área), pruebas de relaciones (superposiciones, toques, etc.) y creación de nuevas geometrías (búffer, intersección, diferencia, etc.).

Además, tiene capacidad para almacenar información topológica.

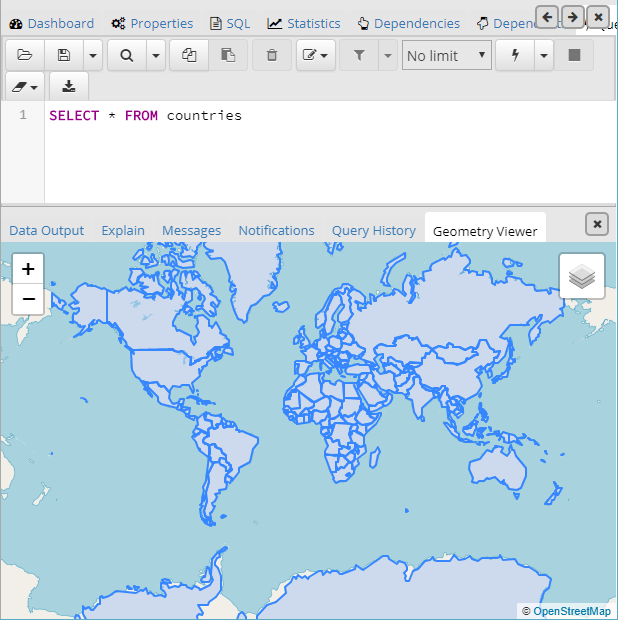
## 4. PostGIS incluye un visor de geometrías integrado en pgAdmin 4

[pgAdmin](https://www.pgadmin.org/) es una herramienta indispensable para gestionar y administrar PostgreSQL, la base de datos de código abierto más avanzada del mundo. Por lo tanto pgAdmin es la herramienta para gestionar nuestras bases de datos espaciales [PostGIS](https://mappinggis.com/2012/09/por-que-utilizar-postgis/). Además, pgAdmin 4 aporta [numerosas ventajas](https://mappinggis.com/2017/11/descubre-el-nuevo-pgadmin-4-para-trabajar-con-postgis/).

Pues bien, la versión 3.3 de [pgAdmin 4](https://mappinggis.com/2017/11/descubre-el-nuevo-pgadmin-4-para-trabajar-con-postgis/) lanzada el 6 de septiembre viene con una importante novedad: un visor de geometrías integrado.

Con este visor integrado podremos ver los resultados de nuestras consultas directamente en pgAdmin, siempre y cuando el resultado de la consulta genere una columna de geometría o geografía (columnas geom o geog).

Si la geometría se encuentra en el [SRID](https://mappinggis.com/2016/04/los-codigos-epsg-srid-vinculacion-postgis/) 4326 (WGS 84 lon / lat), pgAdmin las mostrará automáticamente en un mapa con un mapa base de OpenStreetMap.



En [esta entrada](https://mappinggis.com/2018/09/por-fin-postgis-ya-incluye-un-visor-de-geometrias-integrado/) tienes más información sobre esta utilidad.

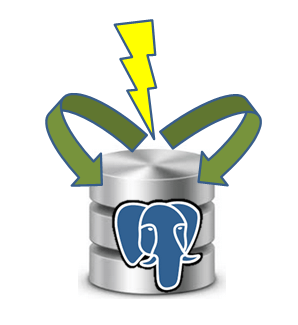
## 5. Permite trabajar con triggers

En el manejo recurrente de actualización de datos en tablas en una base de datos espacial PostGIS, en ocasiones necesitamos realizar tareas repetitivas y tediosas que, si se automatizasen, nos liberarían de esfuerzo innecesario con la consiguiente optimización del tiempo de trabajo. Por ejemplo:

* Cuando es necesario controlar los valores de los campos, evitando que se tomen valores concretos.
* Duplicar valores de una tabla en tiempo real.
* Introducir restricciones en determinados campos con determinados valores.
* Crear un auto id o identificador.
* Controlar la modificación de los valores de los campos.
* Ejecutar actualizaciones en cascada.

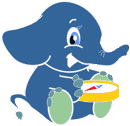
Estas rutinas periódicas y recurrentes realmente sí pueden automatizarse mediante la creación de un disparador o trigger.

[En este tutorial](https://mappinggis.com/2016/06/crear-ejecutar-disparador-trigger-postgis/) te explicamos qué es un trigger y cómo crearlo y ejecutarlo aplicado a PostGIS.



## 6. Enrutamiento

Tiene capacidades de enrutamiento mediante [pgrouting](https://mappinggis.com/2015/01/como-configurar-una-base-de-datos-postgis-para-pgrouting/). pgRouting es una extensión que añade enrutamiento y funcionalidad de análisis de redes a las bases de datos PostGIS/PostgreSQL.



## 7. Tipos de geometría

Permite almacenar varios tipos de geometría en una única tabla.

## 8. Es rápido y seguro

Gracias a la indexación espacial se ahorran segundos en cada consulta. En PostGIS se pueden otorgar permisos hasta el nivel de individuo. Cada usuario tiene que iniciar sesión en la base de datos con un nombre y una contraseña, y eso determina qué tipos de cambios pueden realizar. De esta forma, se puede evitar que los usuarios vean, cambien o eliminen información a la que no deberían tener acceso.

## 9. Sencilla importación y exportación de datos

Permite importar y exportar datos fácilmente a través de  varias herramientas conversoras (shp2pgsql, pgsql2shp, ogr2ogr, dxf2postgis).

## 10. Existen decenas de aplicaciones que trabajan bien con PostGIS

Existe un gran número de clientes SIG de escritorio y servidores de mapas web que pueden trabajar con PostGIS:

* uDig
* [QGIS](https://mappinggis.com/2012/09/13/como-conectar-qgis-a-postgis/)
* mezoGIS
* OpenJUMP
* SpatialKit para ArcGIS
* [gvSIG](https://mappinggis.com/2012/08/22/cinco-razones-para-comenzar-a-usar-gvsig/)
* GRASS
* ArcGIS
* Manifold
* GeoConcept
* MapInfo
* AutoCAD Map 3D
* Mapserver
* [GeoServer](https://mappinggis.com/2015/09/geoserver-para-novatos-crear-un-wms-a-partir-de-un-shapefile/#Que_es_GeoServer)
* MapGuide
* ArcGIS Server

Aprende a manejar esta base de datos espacial en nuestro [curso online de bases de datos espaciales: PostGIS](http://mappinggis.com/cursos/postgis/)

[Cómo conectar con PostGIS desde QGIS 3.x en 7 pasos](https://mappinggis.com/2012/09/como-conectar-qgis-a-postgis/)

[Las mejores aplicaciones GIS open source](https://mappinggis.com/2012/09/aplicaciones-gis-open-source/)

### 1 comentario en «10 motivos para utilizar PostGIS»

# ¿Qué es PostGIS? Extensión GIS para PostgreSQL

 por [**Raúl Estévez**](http://www.geomapik.com/author/raul-estevez/)

[26 septiembre, 2019](http://www.geomapik.com/desarrollo-programacion-gis/que-es-postgis/)

en [**Desarrollo GIS**](http://www.geomapik.com/category/desarrollo-programacion-gis/)

[Home](http://www.geomapik.com/)  [Desarrollo GIS](http://www.geomapik.com/category/desarrollo-programacion-gis/)

1.6k

Vistas

[Compartir](http://www.facebook.com/sharer.php?u=http%3A%2F%2Fwww.geomapik.com%2Fdesarrollo-programacion-gis%2Fque-es-postgis%2F)[Twitter](https://twitter.com/intent/tweet?text=%C2%BFQu%C3%A9+es+PostGIS%3F+Extensi%C3%B3n+GIS+para+PostgreSQL+via+%40%40jegtheme&url=http%3A%2F%2Fwww.geomapik.com%2Fdesarrollo-programacion-gis%2Fque-es-postgis%2F)

En este artículo se definen las ventajas que ofrece **PostgreSQL** como **base de datos** open source, así como **qué es PostGIS** y qué capacidades ofrece para dotar de **soporte espacial a PostgreSQL**.

La unión de **PostgreSQL y PostGIS** es la solución de almacenamiento de datos espaciales más habitual, extendida y potente en el ámbito**GIS open source**.

## La base de datos PostgreSQL

[**PostgreSQL**](https://www.postgresql.org/) es la **base de datos relacional**de tipo **open source** más popular y famosa del mercado. De hecho, está considerada la base de datos más avanzada y completa del momento.

Está disponible para la mayoría de Sistemas Operativos, desde MacOS y Windows a sistemas Linux RedHat, Solaris, SuSe u OpenSuSe, Debian y, como no, para Ubuntu.



La **base de datos PostgreSQL** aporta una gran flexibilidad dado que soporta múltiples lenguajes para programar funciones como PLpgSQL, PLPython, PLPHP o PLJava, entre otros.

Destacan también su seguridad, consistencia e integridad, así como el amplio espectro de tipos de datos soportados, entre los que destacan:

* Datos primitivos como enteros, numéricos, cadenas de texto, booleanos.
* Datos Estructurados como datos de fecha y hora, arrays o identificadores únicos.
* Documentos como JSON, XML, Key-value.
* Datos Geométricos: puntos, líneas, círculos y polígonos.
* Formatos personalizados

## PostGIS: la extensión para datos espaciales

Uno de los puntos más relevantes es la capacidad de almacenar y trabajar con datos de tipo geométrico. Para ello surge **PostGIS**: la extensión espacial para PostgreSQL, una herramienta impulsada por OsGEO.

Se trata de un módulo de ampliación indispensable para **PostgreSQL** a la hora de trabajar en proyectos **GIS**.

La extensión **PostGIS** permite dotar a la base de datos relacional PostgreSQL de una serie de ventajas. Entre otros, destacan:

* soporte para archivos GIS ráster y vectoriales.
* provee funciones de análisis, transformación y consulta espaciales.
* velocidad de procesamiento gracias a índices espaciales.
* herramientas de geocodificación, 3D, topología, cálculo de rutas…

## ¿Qué es PostGIS y para qué sirve?

La extensión **PostGIS** habilita el soporte para trabajar con objetos geográficos localizados en el espacio.

En otras palabras: **convierte PostgreSQL en una base de datos espacial**, que en la práctica funciona (quitando el apartado gráfico) exactamente como un auténtico **Sistema de Información Geográfica** de escritorio.



Eso es posible porque, además de permitir el almacenamiento de información geográfica, el módulo en cuestión añade un enorme paquete de funciones para consultar, procesar, transformar y analizar información espacial almacenada.

No sólo eso, los **tiempos de ejecución** de análisis GIS basados en **PostGIS** son **notablemente más cortos y eficientes** que cualquier otro Sistema de Información Geográfica de escritorio.

Aquí tienes un magnífico [artículo de Topi Tjukanov](https://medium.com/@tjukanov/why-should-you-care-about-postgis-a-gentle-introduction-to-spatial-databases-9eccd26bc42b) hablando sobre PostGIS y sus bondades. A parte de la gran calidad de la información que contiene el artículo, expone una figura donde demuestra la comparativa en la velocidad de procesamiento entre **PostGIS y QGIS**.

Por ello, aprender a programar en SQL para gestionar y analizar conjuntos de datos geográficos es un factor clave.

## ¿Qué permite hacer PostGIS?

**PostGIS** permite trabajar tanto con **información geográfica** vectorial como ráster de múltiples formatos distintos.

Así, trabajando con formatos vectoriales permite:

* crear y editar geometrías
* establecer relaciones espaciales entre elementos geométricos
* realizar análisis de distancia y enrutamiento con pgRouting
* realizar correcciones topológicas de elementos, etc.

Por su parte, al trabajar con formatos ráster pueden realizarse, entre otros:

* análisis y combinaciones de bandas
* editar y reclasificar píxeles de capas ráster
* llevar a cabo operaciones de álgebra de mapas
* generar modelos de elevaciones, etc.

Otro punto relevante es la capacidad de vincular**PostGIS** fácilmente con Geoserver y Mapserver para ser servir información geográfica espacial.



En este sentido, se trata de una pieza clave a la hora de integrar piezas de la arquitectura web GIS para proyectos de webmapping y para compartir información geográfica en base a estándares OGC.

Como comentábamos al principio, PostgreSQL admite el trabajo con lenguajes de programación para incluir **funciones**. PostGIS permite utilizar y programar triggers (disparadores) para realizar acciones automáticas que aceleran el trabajo en tareas repetitivas.

Te puede resultar interesante conocer las [25 funciones de PostGIS para análisis espacial](http://www.geomapik.com/analisis-gis/postgis-analisis-espacial-funciones/), resumidas en un artículo, explicadas y con ejemplos.

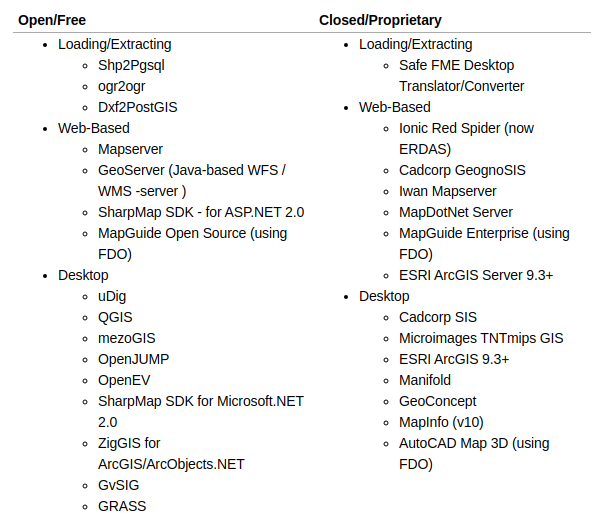
También permite implementar **índices espaciales** para acelerar considerablemente los tiempos de consulta y análisis de la información espacial almacenada e indexada.

## Integración con otro software GIS

Existe un amplio número de programas de **software GIS** tanto Open Source como propietario que ofrecen soporte y facilitan el trabajo con **PostGIS**.

La [**integración de PostGIS con QGIS**](http://www.geomapik.com/desarrollo-programacion-gis/como-conectar-postgis-qgis-importar-capas-espaciales/) es perfecta y fácil, como mostramos en otro post acerca de la conexión del GIS de escritorio **QGIS** con la **base de datos espacial PostgreSQL – PostGIS.**

En la imagen se muestra el listado completo facilitado por PostGIS donde se listan todos los programas que permiten el soporte de PostGIS.

Integración de PostGIS con otro software GIS. Fuente: [web de PostGIS](https://postgis.net/workshops/postgis-intro/introduction.html)

Por todas esas razones, la combinación de **PostgreSQL y PostGIS** se ha convertido en la piedra angular de muchos proyectos de información geográfica. En cierto modo su uso ha pasado a convertirse en un estándar dentro del sector GIS en el ámbito Open Source.

Si deseas empezar a utilizar estas herramientas en tus proyectos **GIS**, aprende a instalarlas fácilmente paso a paso en nuestro artículo que realizamos acerca de la [instalación de **PostgreSQL, PostGIS**](http://www.geomapik.com/desarrollo-programacion-gis/como-instalar-postgis-en-ubuntu/)y [cómo instalar**pgAdmin**.](http://www.geomapik.com/tutoriales-gis/como-instalar-pgadmin4-en-ubuntu/)

**Etiquetas:** [Desarrollo GIS](http://www.geomapik.com/tag/desarrollo-gis/)[GIS](http://www.geomapik.com/tag/gis/)[PostGIS](http://www.geomapik.com/tag/postgis/)[PostgreSQL](http://www.geomapik.com/tag/postgresql/)

# Librerías Python GIS para manipular y analizar datos espaciales

 por [**Raúl Estévez**](http://www.geomapik.com/author/raul-estevez/)

[16 septiembre, 2019](http://www.geomapik.com/desarrollo-programacion-gis/librerias-python-gis/)

en [**Análisis GIS**](http://www.geomapik.com/category/analisis-gis/), [**Desarrollo GIS**](http://www.geomapik.com/category/desarrollo-programacion-gis/)

[Home](http://www.geomapik.com/)  [Desarrollo GIS](http://www.geomapik.com/category/desarrollo-programacion-gis/)

2.7k

Vistas

[Compartir](http://www.facebook.com/sharer.php?u=http%3A%2F%2Fwww.geomapik.com%2Fdesarrollo-programacion-gis%2Flibrerias-python-gis%2F)[Twitter](https://twitter.com/intent/tweet?text=Librer%C3%ADas+Python+GIS+para+manipular+y+analizar+datos+espaciales+via+%40%40jegtheme&url=http%3A%2F%2Fwww.geomapik.com%2Fdesarrollo-programacion-gis%2Flibrerias-python-gis%2F)

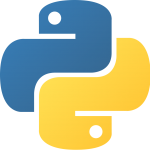
En este articulo se pretende recoger, describir y diferenciar las **principales librerías de Python para GIS** más utilizadas para trabajar con geoinformación y para realizar **análisis espacial**.

El **lenguaje de programación Python** goza de una tremenda popularidad. Su vinculación con los **GIS** es innegable: aporta grandes ventajas en la gestión y el análisis de información espacial.

## Python: un aliado de los GIS

[**Python**](https://www.python.org/) es uno de los lenguajes de programación más extendidos dentro del desarrollo de software y del análisis de datos.

En clave **GIS**, el lenguaje **Python** puede utilizarse como una potente herramienta para manipular y realizar **análisis espacial** avanzado mediante una serie de **librerías** específicamente desarrolladas para tales fines.



En cierto modo, **Python** rivaliza en popularidad e importancia con lenguajes como SQL para trabajar con Bases de Datos Espaciales u otros lenguajes vinculados específicamente a la programación web como JavaScript, utilizado habitualmente para realizar aplicaciones y visores de mapas web.

**Python** es uno de los lenguajes principales y fundamentales dentro del ámbito **GIS**, tal y como destacamos en nuestro [artículo sobre los principales lenguajes de programación GIS](http://www.geomapik.com/desarrollo-programacion-gis/5-lenguajes-programacion-gis-imprescindibles/).

La clave en el uso de **Python**frente a un**GIS** de escritorio convencional radica en la facilidad para **visualizar y manipular**datos mediante scripting, adaptar los geoprocesos de análisis o diseñar nuevos, establecer rutinas de trabajo con los**datos espaciales** que ahorren tiempo y reducir los tiempos de ejecución de tareas, entre otros.

## Fortalezas y aplicaciones de Python como lenguaje de programación

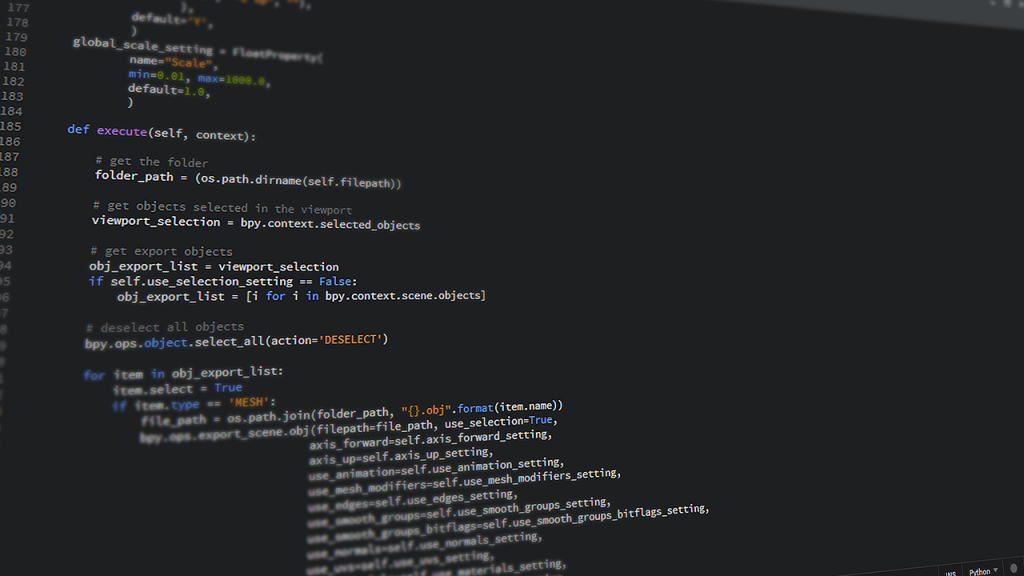
La **popularidad** creciente de **Python** no es casualidad: existen ciertos aspectos del lenguaje que lo apuntalan como uno de los lenguajes más usados después de JavaScript y Java.

Entre esos **aspectos** que hacen fuerte a **Python** destacan:

* su relativa facilidad de aprendizaje debido a la sencillez de su sintaxis y a las estructuras de datos que utiliza.
* ser multiparadigma, pero especialmente por el hecho de poder adaptarse a la programación orientada a objetos.
* el carácter modular: es fácilmente ampliable mediante paquetes o librerías.
* la capacidad de ejercer como lenguaje «pegamento» o «puente», es decir, por tratarse de un lenguaje apto para trabajar como conexión entre componentes de software distinto o entre plataformas.
* un gran soporte, popularidad y comunidad activa que lo respalda, lo aprende y lo utiliza a diario.

Además, a diferencia de otros lenguajes de programación, **Python** es claramente **multipropósito**: cubre y es susceptible de ser aplicado a un enorme rango de escenarios, proyectos y tareas precisamente derivado de su naturaleza modular.

Su **versatilidad**, es decir, la capacidad de ser aplicado a prácticamente cualquier campo y ante cualquier propósito, es claramente otro de sus principales puntos fuertes.

Código Python. Fuente: imagen de Johnson Martin de Pixabay.

**Python** permite ser utilizado en **programación web** en el lado servidor, se utiliza para acceder y gestionar **bases de datos**, para programar aplicaciones de escritorio y desarrollar interfaces.

Además, tiene enormes aplicaciones en estadística y **análisis de datos**, es imprescindible en Big Data y Machine Learning, se utiliza también para el desarrollo de aplicaciones científicas e incluso en el ámbito de los videojuegos.

## Principales aplicaciones y usos de Python para GIS

Cuando hablamos de las **aplicaciones de Python dentro del ámbito GIS**o geoespacial, cabe destacar su utilidad ante dos propósitos generales:

* **Automatizar tareas**repetitivas mediante scripts o plugins, centrándose en la propia gestión y el tratamiento de la geoinformación.
* **Realizar análisis espacial avanzado**, expandiendo las herramientas de análisis convencionales de los GIS de escritorio mediante librerías específicas.

## Librerías Python GIS para análisis espacial

A continuación listamos las principales (que no únicas) **librerías de Python**especialmente diseñadas y utilizadas para tratar con **datos GIS** tanto vectoriales como ráster, así como para realizar**análisis espacial.**

### GDAL/OGR: la librería de cabecera

[**GDAL**](https://gdal.org/) es el acrónimo de la librería Geospatial Data Abstraction Library. Fue creada en los años 90 por Frank Warmerdam y publicada originalmente en el año 2000.

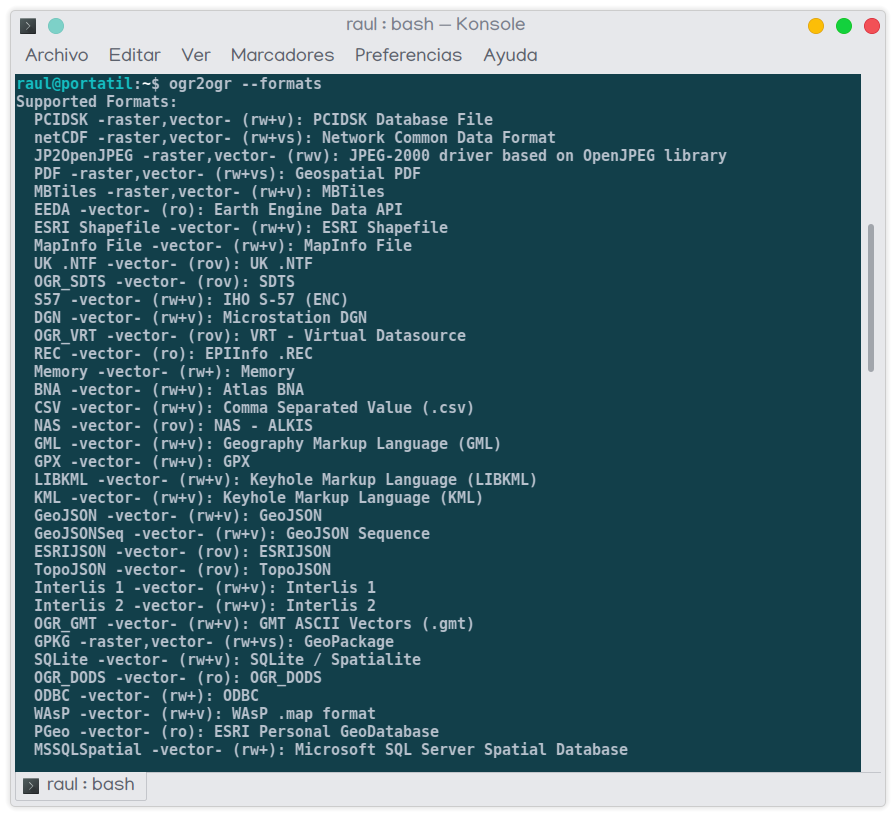
Por su parte, el nombre [**OGR**](https://live.osgeo.org/es/overview/gdal_overview.html) proviene de OpenGIS Simple Features Reference Implementation, nombre olvidado y adaptado posteriormente a OGR Simple Features Library.



Las principales diferencias entre ambas librerías se deben a su aplicación: **GDAL** se orienta al procesamiento de **datos de tipo ráster** provenientes de más de 200 formatos distintos, mientras que **OGR** se utiliza para leer, escribir, proyectar y convertir **datos GIS de carácter vectorial**, soportando también una gran cantidad de formatos distintos.

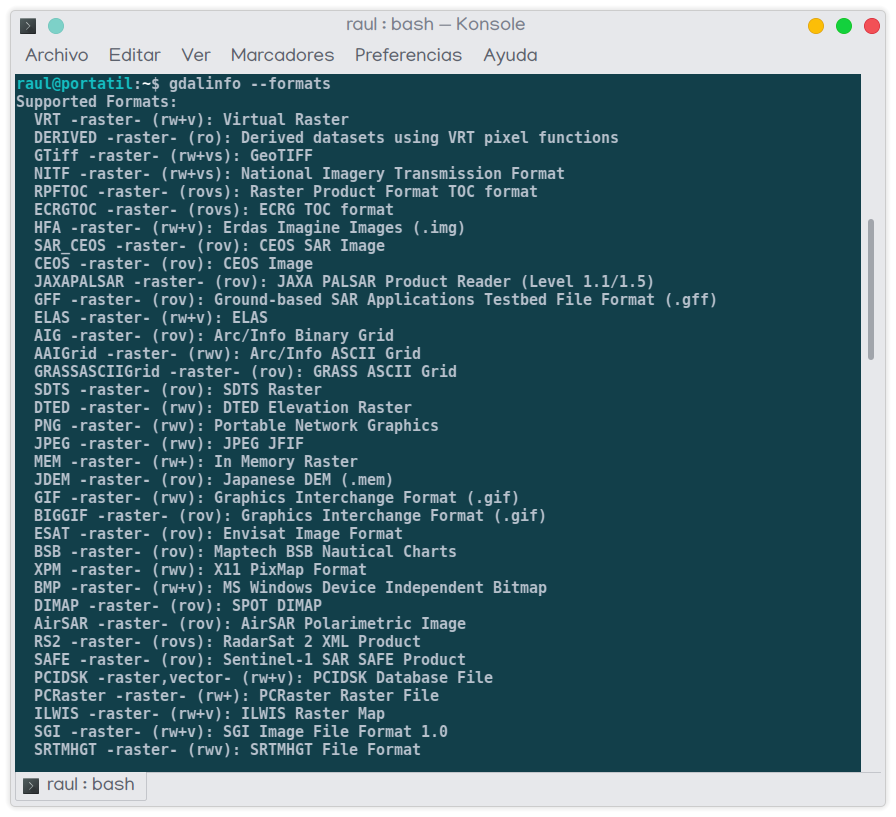
Se puede acceder al listado de todos los formatos GIS ráster soportados por GDAL introduciendo en la consola:

|  |  |
| --- | --- |
| 1 | ogr2ogr --formats |

Algunos de los formatos GIS vectoriales soportados por la librería OGR. Fuente: elaboración propia.

Por su parte, podemos conocer el listado de formatos GIS ráster utilizados en OGR haciendo uso del comando:

|  |  |
| --- | --- |
| 1 | gdalinfo --formats |

Formatos soportados por GDAL para trabajara con archivos GIS de tipo ráster. Fuente: elaboración propia.

El hecho de hacer referencia a ambas librerías como una única se debe a que la instalación de GDAL lleva implícita o da acceso a OGR, al depender en ciertos aspectos de la primera. En cierto modo podríamos decir que OGR es la «sublibrería» de GDAL dedicada al trabajo con datasets vectoriales.

**GDAL/OGR** es una de las principales **librerías** de acceso a datos de carácter espacial y de las más utilizadas, no sólo por programadores, sino también por otras librerías de **Python** y aplicaciones de software diverso.

Entre las distintas aplicaciones que se nutren de esta librería se sitúan: MapServer, GRASS, QGIS, ArcGIS, FME…

### Shapely: para manipular datos vectoriales

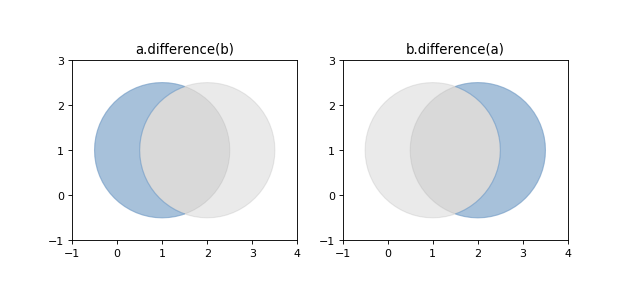
La [**librería Shapely para Python**](https://pypi.org/project/Shapely/) está enfocada en la manipulación y el análisis de datos vectoriales. Se basa en [GEOS](https://trac.osgeo.org/geos/), una librería escrita en C++ que provee de algoritmos a [PostGIS](https://postgis.net/), además de a otras librerías como [sf para el lenguaje R](https://github.com/r-spatial/sf).

**Shapely** trabaja básicamente con **elementos geométricos** de tipo **Punto**, **Línea** y **Polígono**.

Asimismo, también admite geometrías basadas en colecciones de elementos de carácter Multipunto, Multilínea y Multipolígono.

Permite, entre otros, realizar **análisis geométricos**:

* basados en las **características y la posición**de los elementos involucrados (calcular coordenadas, centroides, distancias, áreas, perímetros, buffers o áreas de influencia…).
* basados en la **relación espacial** entre los elementos (intersección, cruce, inclusión, exclusión…) en base a **predicados booleanos** (verdadero o falso).

Ejemplo de método de análisis espacial, referido a la diferencia entre dos objetos geométricos distintos, contenido en la librería de Shapely de Python. Fuente: Documentación oficial de Shapely.

Tiene un [manual de usuario](https://shapely.readthedocs.io/en/latest/manual.html) completo y de fácil comprensión, muy recomendable para familiarizarse con la librería y consultar ante dudas o para visualizar ejemplos. En [GIS StackExchange](https://gis.stackexchange.com/questions/tagged/shapely) pueden resolverse dudas siguiendo la etiqueta «shapely».

### Fiona: para leer y escribir archivos vectoriales

El propósito principal de la **librería de Python**[**Fiona**](https://pypi.org/project/Fiona/) es la lectura y escritura de archivos geométricos de tipo vectorial.

Generalmente se utiliza en combinación con otras librerías como **Shapely** para el procesado y la manipulación de información geográfica vectorial.

Para el trabajo en exclusiva con archivos en formato shapefile de ESRI puede utilizarse la **librería pyshp**.

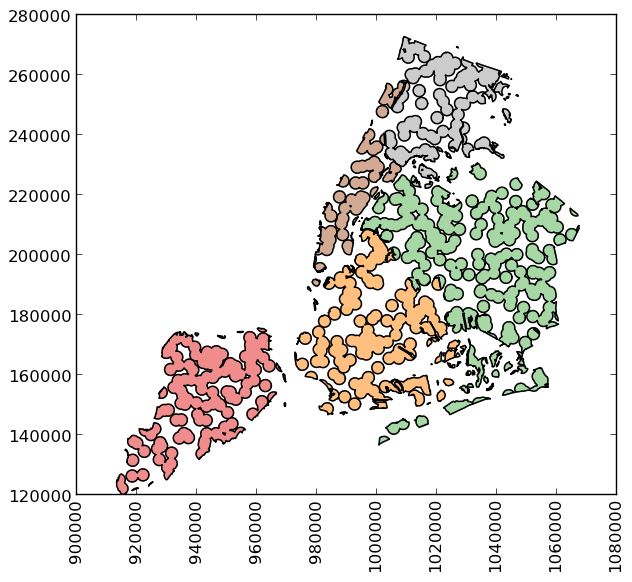
La **librería para Python**[**pyshp**](https://pypi.org/project/pyshp/) es tan simple como su propósito: está escrita estrictamente para leer y/o escribir archivos shapefile de ESRI, corriendo directamente sobre Python.

### GeoPandas: facilitando el trabajo con datos espaciales

La [**librería GeoPandas**](http://geopandas.org/) es la extensión de la librería [Pandas](https://pandas.pydata.org/) de Python, que permite acceder y trabajar con las estructuras y tipos de datos de esta librería en operaciones de tipo espacial.

**Geopandas**, no obstante, depende de las otras dos librerías vectoriales mencionadas: para acceder a archivos depende de **Fiona** y para realizar operaciones geométricas depende de **Shapely**.

Así pues, los objetos geométricos válidos son exactamente los mismos que usa Shapely: punto, línea y polígono además de las colecciones de objetos. No obstante, GeoPandas introduce dos estructuras de datos distintas, heredadas y adaptadas a las correspondientes referencias espaciales de la librería original Pandas: **GeoSeries** y **GeoDataFrame**.

Ejemplo de mapa construido con GeoPandas mostrando buffers generados con la misma librería alrededor de puntos. Fuente: Documentación Oficial de la librería GeoPandas.

Podría decirse que ambas estructuras funcionan análogamente a una base de datos. GeoSeries es un elemento vectorial con capacidad de reunir conjuntos de elementos geométricos como un único objeto. GeoDataFrame es la estructura tabular de los datos contenidos en GeoSeries.

GeoPandas También permite graficar datos y mostrar visualizaciones de mapas, en este caso mediante matplotlib. Puedes echar un vistazo a nuestro[artículo sobre generación de mapas con GeoPandas y matplotlib](http://www.geomapik.com/desarrollo-programacion-gis/mapas-con-python-geopandas-matplotlib/) para conocer más acerca de éstas y otras librerías para visualización de datos espaciales y confección de mapas en Python.

### Pyproj: para trabajar con proyecciones

La **librería**[**pyproj**](https://pypi.org/project/pyproj/)consiste en la interfaz de la librería [PROJ4](https://live.osgeo.org/es/overview/proj4_overview.html) de OSGeo traída a **Python** desde C y su uso está centrado básicamente en la **proyección** y **conversión** de geometrías entre sistemas de referencia de coordenadas.

A continuación, un simple ejemplo de la transformación de las coordenadas de un punto de un SRC a otro:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | # Importar la librería pyproj  from pyproj import Proj, transform    # Definimos los SRC involucrados  etrs89 = Proj(init='epsg:25830') # SRC origen  wgs84 = Proj(init='epsg:4326') # SRC destino    # Indicamos coordenadas X e Y del punto a transformar  x, y = (209811.26, 4374736.23)    # Aplicamos transformación, pasando como parámetros  # las 3 variables definidas anteriormente  coordenadas = transform(etrs89, wgs84, x, y)    # Mostramos el resultado  print coordenadas  (-6.373443550273573, 39.47330540167159) |

Permite trabajar con cientos de sistemas de coordenadas distintos, realizando cálculos y transformaciones tanto cartográficas como geodésicas mediante las clases Proj y Geod respectivamente.

### Rasterio: para manipular imágenes ráster

El propio nombre ofrece pistas acerca de su cometido: la librería [**Rasterio para Python**](https://pypi.org/project/rasterio/0.13.2/) permite leer, manipular y escribir **archivos de tipo ráster**.

Rasterio es una libreria Python utilizada habitualmente para tratar imágenes de satélite y ráster. Fuente: WikiImages de Pixabay.

Es una buena **alternativa a GDAL** como librería para trabajar con imágenes **ráster** pues permite ejecutar procesos similares escribiendo menos código, con una sintaxis algo más sencilla y elegante.

Algunos artículos cómo el de [Martin en GIS-Blog](https://www.gis-blog.com/benchmark-r-vs-python-rasterio-vs-gdal/)o el de [Andrew Cutts en ACGeospatial](http://www.acgeospatial.co.uk/fastest-image-python/) apuntan a que, si bien el tiempo de ejecución de ambas librerías es similar, en la mayoría de ocasiones GDAL tiene un mejor desempeño que **Rasterio** cuando se trata de trabajar con **imágenes de satélite** o **capas ráster** desde **Python**.

**Rasterio** se construye en base a las funcionalidades de diversas librerías de Python como Numpy o la propia GDAL.

Puedes encontrar toda la información acerca de su funcionamiento y particularidades en su [documentación](https://rasterio.readthedocs.io/en/stable/index.html), bastante completa y con ejemplos gráficos para mayor utilidad.

# Cómo conectar PostGIS y QGIS para importar capas

 por [**Raúl Estévez**](http://www.geomapik.com/author/raul-estevez/)

[5 septiembre, 2019](http://www.geomapik.com/desarrollo-programacion-gis/como-conectar-postgis-qgis-importar-capas-espaciales/)

en [**Análisis GIS**](http://www.geomapik.com/category/analisis-gis/), [**Desarrollo GIS**](http://www.geomapik.com/category/desarrollo-programacion-gis/), [**Tutoriales**](http://www.geomapik.com/category/tutoriales-gis/)

[Home](http://www.geomapik.com/)  [Desarrollo GIS](http://www.geomapik.com/category/desarrollo-programacion-gis/)

1.8k

Vistas

[Compartir](http://www.facebook.com/sharer.php?u=http%3A%2F%2Fwww.geomapik.com%2Fdesarrollo-programacion-gis%2Fcomo-conectar-postgis-qgis-importar-capas-espaciales%2F)[Twitter](https://twitter.com/intent/tweet?text=C%C3%B3mo+conectar+PostGIS+y+QGIS+para+importar+capas+via+%40%40jegtheme&url=http%3A%2F%2Fwww.geomapik.com%2Fdesarrollo-programacion-gis%2Fcomo-conectar-postgis-qgis-importar-capas-espaciales%2F)

Este artículo pretende mostrar paso a paso **cómo conectar** la base de datos [**PostGIS**](https://postgis.net/) con [**QGIS**](https://qgis.org/es/site/), el **Sistema de Información Geográfica** más popular de código libre.

De hecho, la **unión de PostGIS y QGIS** es la base de cualquier **arquitectura GIS** como solución Open Source. Alimentar un **Sistema de Información Geográfica** (en este caso de escritorio) con una **base de datos** es esencial para desarrollar proyectos GIS de cierta envergadura.

## La extensión espacial PostGIS

**PostGIS** es la **extensión espacial** de [**PostgreSQL**](https://www.postgresql.org/), la base de datos relacional Open Source más avanzada y utilizada.

Básicamente, **PostGIS** convierte a **PostgreSQL** en una **base de datos espacial**añadiendo 3 características:

* La capacidad de almacenar y reconocer datos con componente espacial
* La opción de generar índices espaciales
* La capacidad de introducir operaciones de análisis espacial sobre dichos datos almacenados

De hecho, pueden realizarse procesos de **análisis** realmente avanzados y con tiempos de ejecución únicos sobre **PostGIS** mediante**consultas SQL espaciales**. La adición de la extensión PostGIS sobre PostgreSQL habilita más de un millar de funciones espaciales.  
  
Tenemos un artículo explícito sobre [**qué es PostGIS**](http://www.geomapik.com/desarrollo-programacion-gis/que-es-postgis/) para conocer en detalle las posibilidades y ventajas que ofrece este módulo para **PostgreSQL**.

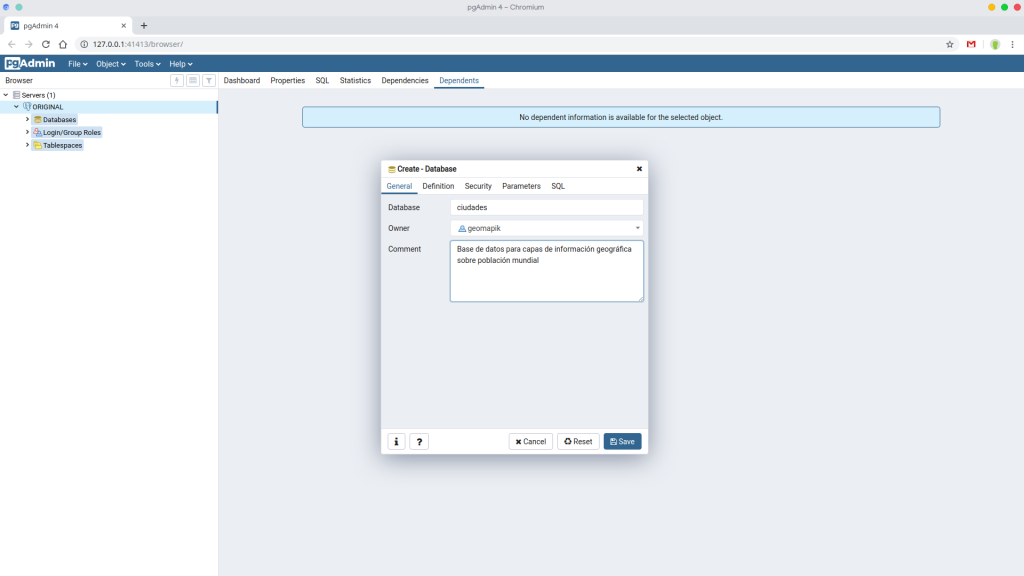
Los principales programas de **software GIS**de escritorio permiten la comunicación y la exportación de datos a **PostGIS**, así como también MapServer, Geoserver y ArcGIS Server para alimentar visores de [webmapping](http://www.geomapik.com/webmapping-gis/librerias-api-web-mapping-javascript/).

## Preparar PostGIS para cargar datos espaciales desde QGIS

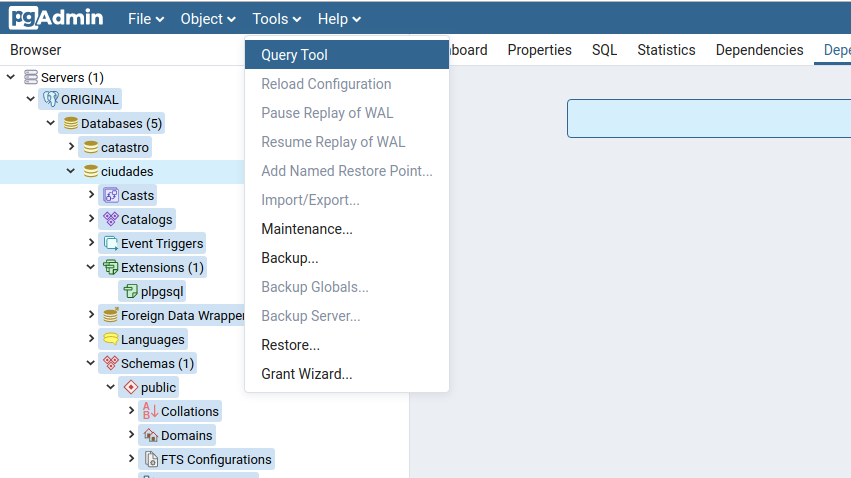
Para operar sobre **PostgreSQL – PostGIS** deberá accederse desde [**pgAdmin**](https://www.pgadmin.org/), la principal interfaz y herramienta de gestión de esta base de datos.

Para este tutorial bastará con habilitar PostgreSQL para que funcione como servidor local en nuestro propio ordenador. Al establecer el servidor deberemos dar los parámetros correctos en pgAdmin.

Lo primero que deberemos hacer es crear una nueva base de datos desde pgAdmin, asignando un nombre y un usuario propietario de dicha base de datos.



Una vez creada, deberemos acceder a la herramienta de creación de consultas sobre nuestra base de datos.

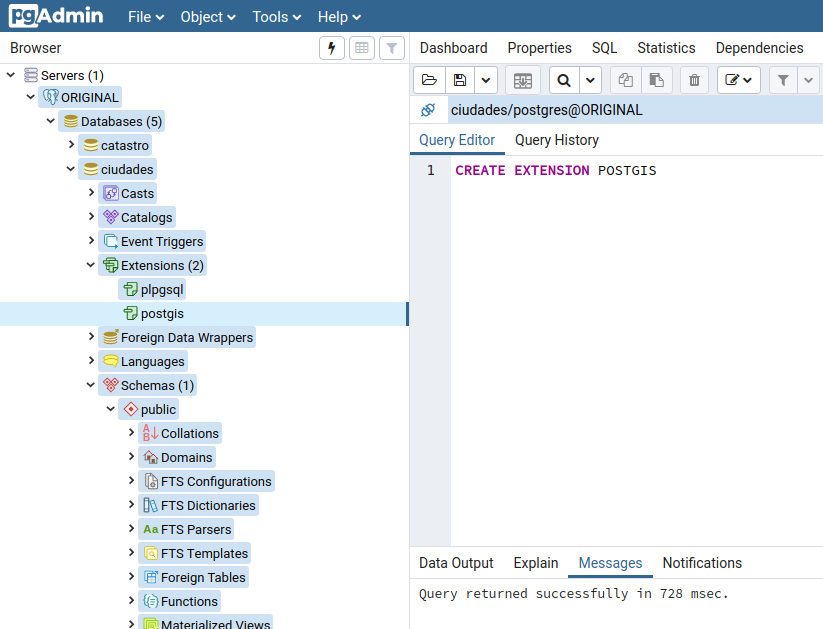


Teniendo disponible la opción de creación de queries, introduciremos la siguiente sentencia SQL para habilitar la creación de la **extensión PostGIS** y poder así almacenar información de carácter espacial.

**SQL** es uno de los [principales lenguajes de programación GIS](http://www.geomapik.com/desarrollo-programacion-gis/5-lenguajes-programacion-gis-imprescindibles/), indispensable para gestionar bases de datos espaciales y necesario en prácticamente cualquier proyecto GIS. Como técnico, analista o programador GIS es indispensable conocerlo.



Podemos comprobar que efectivamente se ha podido **habilitar PostGIS** desde el árbol del menú lateral dirigiéndonos a nuestra base de datos > Extensions > PostGIS.

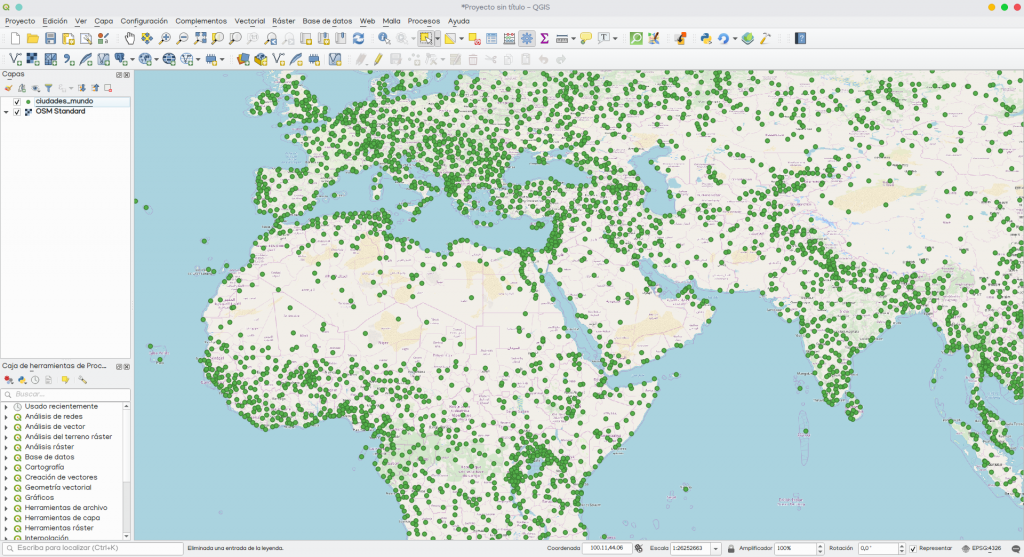


Por ahora, ya con PostGIS habilitado, dejaremos pgAdmin ejecutándose y procederemos a inspeccionar y cargar las capas desde QGIS.

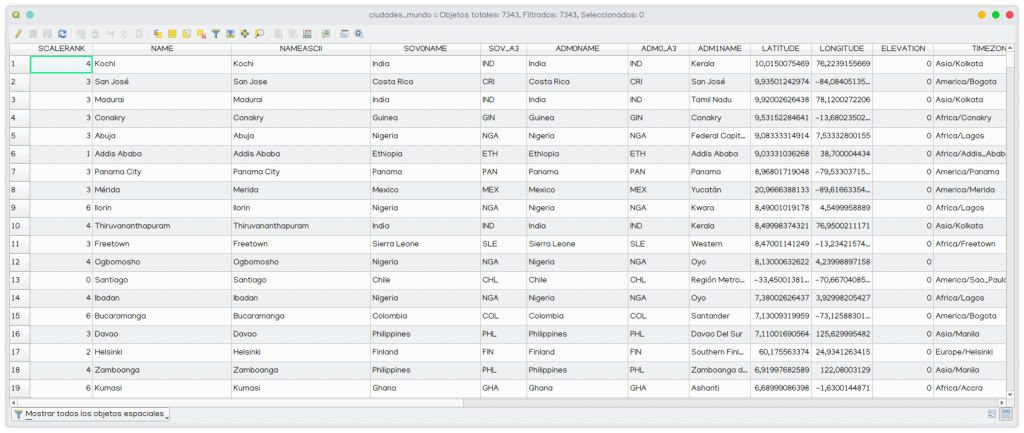
## Cómo importar capas a PostGIS desde QGIS

Para **cargar capas en PostGIS** existen múltiples maneras. Utilizar **QGIS** es tremendamente cómodo pues podremos inspeccionar, editar y generar nuevas capas que serán transferidas a la **base de datos PostGIS** y viceversa.

Lo primero que haremos será cargar nuestra capa en QGIS, en este caso una capa de geometría tipo puntos.



Podemos inspeccionar la tabla de registros de nuestra capa para ver qué tipo de datos contiene.



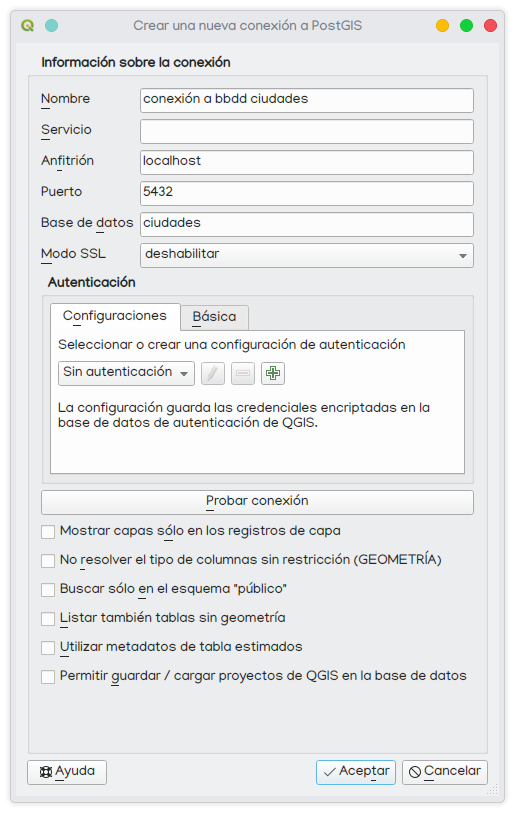
Para proceder a**importar datos a PostGIS desde QGIS** primeramente deberemos habilitar la conexión entre base de datos y nuestro Sistema de Información Geográfica.

### Habilitar la conexión a PostGIS

Nos dirigiremos a Capa > Administrador de Fuentes de Datos > PostgreSQL en el menú lateral. En la interfaz haremos clic en «Nueva» y deberá habilitarse un formulario con distintos campos a rellenar.

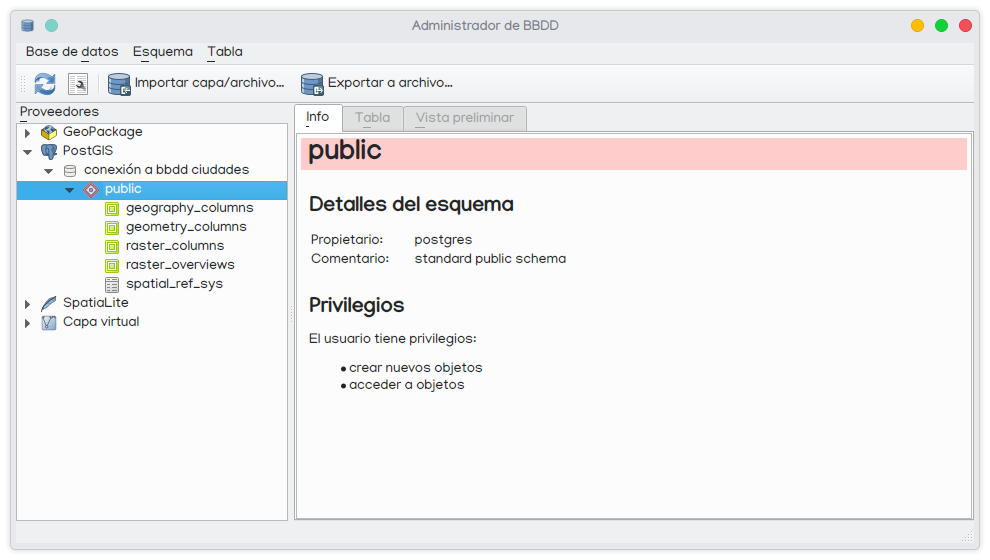
Deberemos introducir los datos de conexión necesarios para establecer el vínculo a nuestra base de datos. Éstos parámetros son:

* el nombre de la conexión (arbitraria, como el usuario desee).
* el host o anfitrión (localhost en nuestro caso) .
* el puerto habilitado para la base de datos (por defecto 5432).
* el nombre de la base de datos dado en pgAdmin (debe coincidir exactamente con el nombre establecido).
* alternativamente los parámetros de identificación de usuario y su contraseña para no tener que introducirlas posteriormente.



Una vez hayamos configurado la **conexión a PostGIS desde QGIS**, aceptaremos y deberemos recibir un mensaje de confirmación de éxito en la conexión.

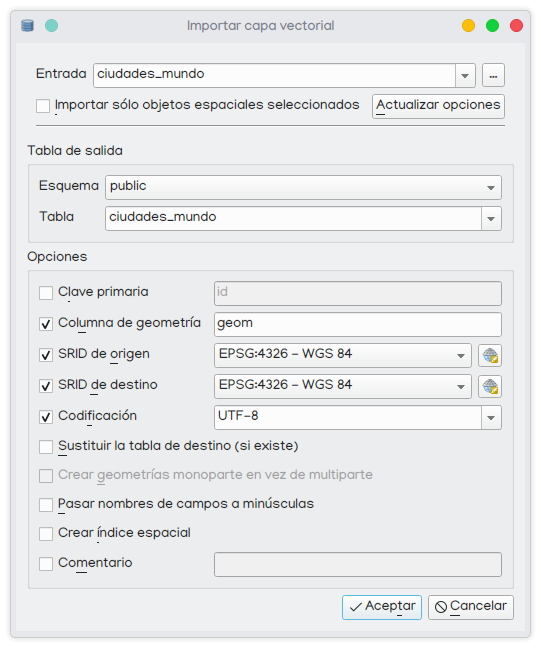
Posteriormente, deberemos dirigirnos al menú Base de Datos > Administrador de Bases de Datos. El administrador es un plugin que forma parte del core de QGIS, viniendo integrado en él con su misma instalación.



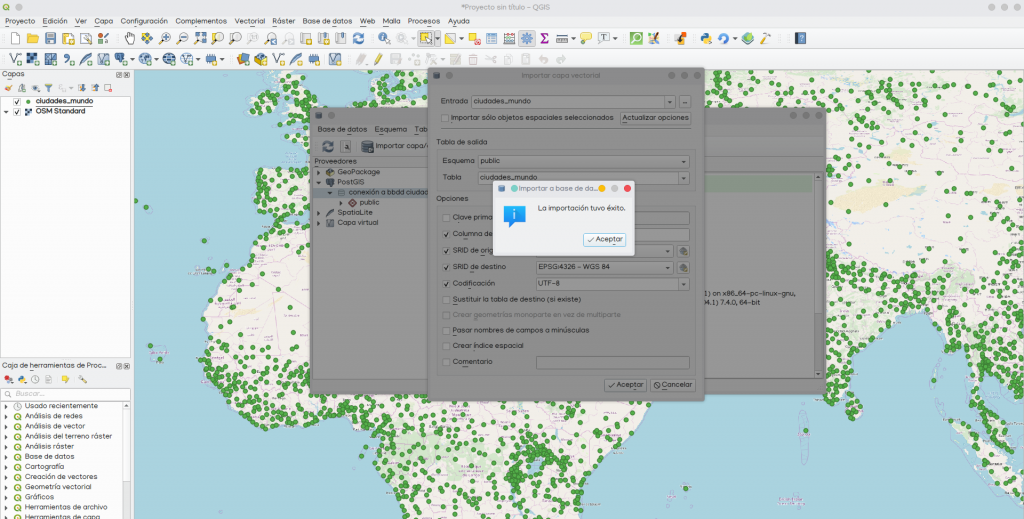
Desde el menú del administrador seleccionaremos la opción de **conexión a PostGIS** y deberá aparecer listada la conexión establecida anteriormente a la base de datos. Deberíamos ser capaces de visualizar el esquema public y las opciones espaciales habilitadas.

### Importar la capa a PostGIS

Finalmente, para **importar una capa a PostGIS desde QGIS** deberemos dirigirnos a la opción del Administrador de Bases de Datos «Importar capa/archivo». Se abrirá un nuevo formulario donde deberemos indicar qué capas cargar en PostGIS y de qué manera.



Seleccionaremos la capa de entrada y el nombre de la tabla que se generará en **PostGIS**. Podemos habilitar la opción de creación de columna de geometría si se nos permite, establecer la codificación y los SRC de origen o destino de la capa, si queremos crear un índice espacial, cierto comentario, etcétera.



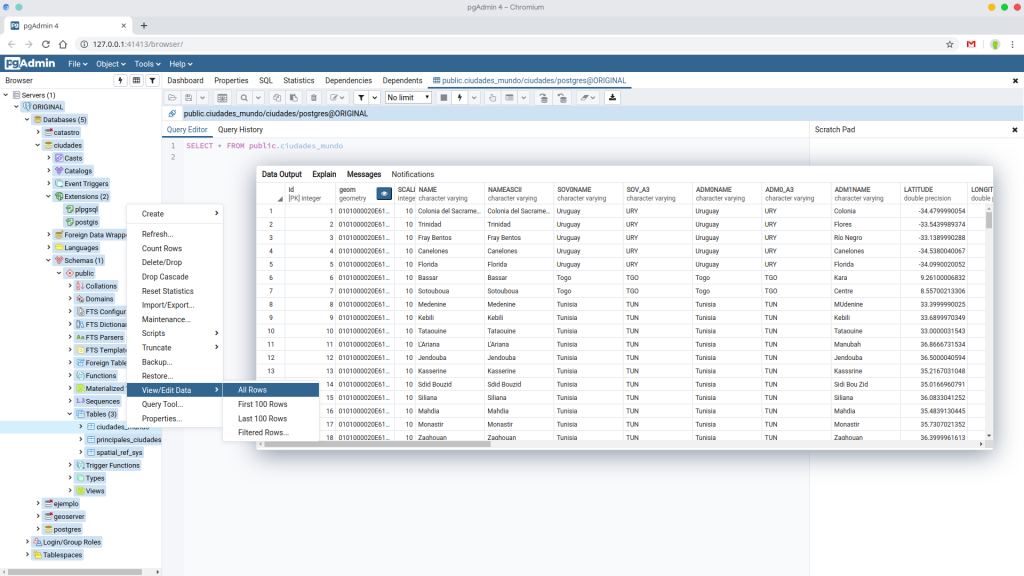
Una vez validados los parámetros de subida y si no existe ningún error en ellos o en la capa, deberíamos recibir un mensaje de confirmación conforme la **importación a PostGIS**se ha realizado con éxito.

## Visualizar las capas subidas en PostGIS

De vuelta a PostGIS, deberíamos poder visualizar la capa cargada desde QGIS. Podremos localizar las capas en el menú lateral izquierdo, en Schemas > Public > Tables.

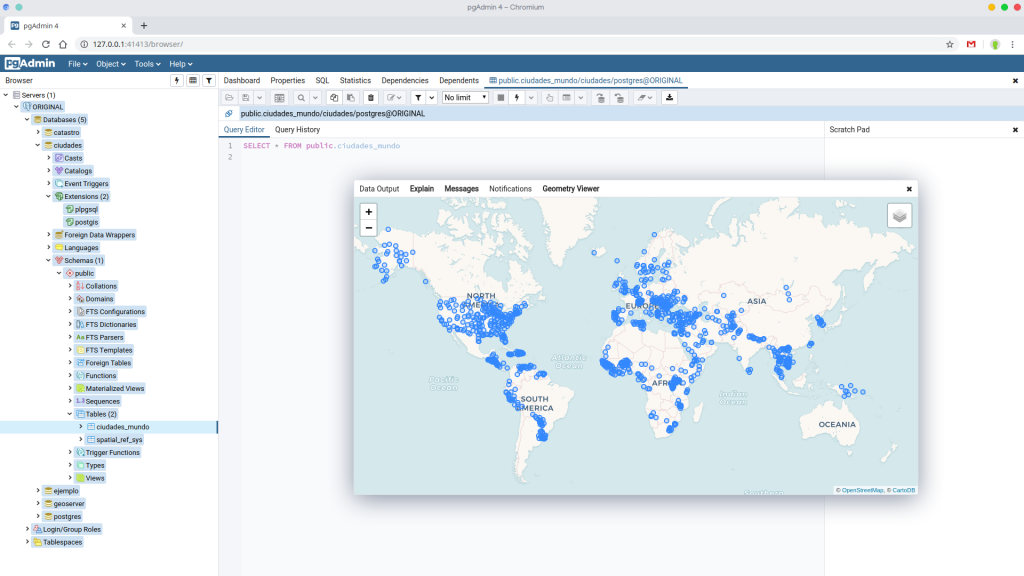
Allí debería aparecer la tabla de sistemas de referencia espaciales y nuestra **capa importada desde QGIS**.

En caso de que no podamos visualizarla, deberemos refrescar la base de datos con la tecla F5 o mediante botón derecho sobre la base de datos > opción Refresh.



Haciendo clic con el botón derecho sobre nuestra capa > View Data > All Rows deberíamos poder ver una ventana la tabla con todos los registros cargados.

Si hemos habilitado la opción de crear la columna de geometría, podremos hacer clic en el botón azul junto a su cabecera para desplegar el visor espacial.

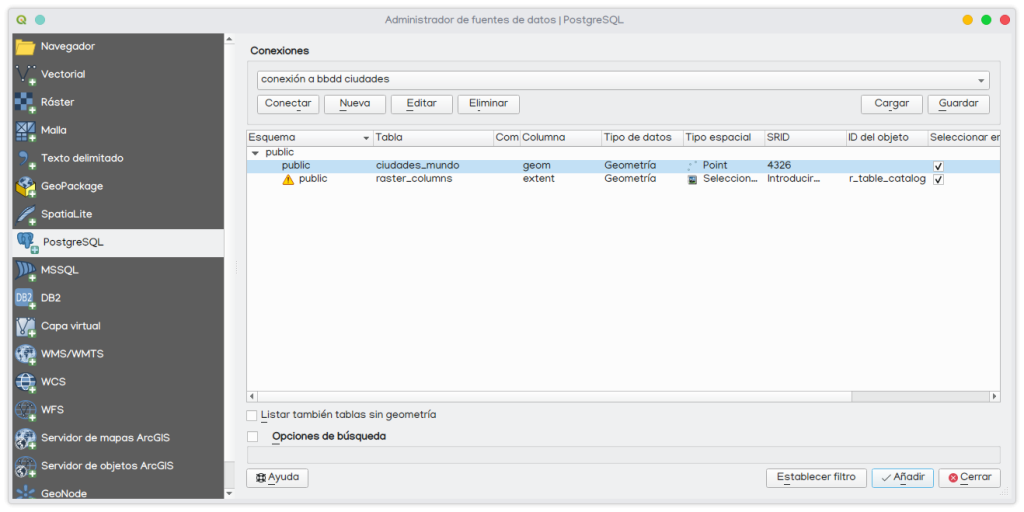


El visor de geometrías en forma de mapa es una característica nueva muy potente e interesante introducida en pgAdmin 4 v3.3. En versiones anteriores de pgAdmin esta opción no está disponible.  
  
Recomendamos actualizar a pgAdmin 4 o, en caso de no tener ninguna versión instalada todavía, instalar directamente esta versión por la gran mejora en su interfaz para trabajar con PostGIS y datos espaciales.

Podremos visualizar, seleccionar e inspeccionar los atributos de la capa tanto desde el visor como desde la tabla alternativamente. Asimismo, podremos realizar cualquier tipo de consulta SQL sobre nuestra capa también a nivel espacial.

## Cómo importar capas a QGIS desde PostGIS

Finalmente, retornando a QGIS, si queremos cargar capas desde una base de datos PostGIS deberemos acceder al Administrador de Fuentes de Datos > opción PostgreSQL y seleccionar la conexión establecida.



Seleccionando la capa deseada, deberemos hacer clic en añadir para **cargar las capas en QGIS**. Pueden cargarse múltiples capas seleccionadas simultáneamente.

En caso de querer cargar o visualizar capas no geométricas deberemos habilitar la opción desde la parte inferior del cuadro de diálogo.

Las capas ya deberían poder verse cargadas en la tabla de contenidos y el lienzo de QGIS para empezar a operar con ellas, editarlas y posteriormente devolverlas a PostGIS.

# Cómo descargar y visualizar cartografía de Catastro en QGIS

 por [**Raúl Estévez**](http://www.geomapik.com/author/raul-estevez/)

[30 agosto, 2019](http://www.geomapik.com/uncategorized/como-descargar-informacion-catastral-parcelas-edificios-qgis/)

en [**Análisis GIS**](http://www.geomapik.com/category/analisis-gis/), [**Recursos GIS**](http://www.geomapik.com/category/recursos-gis/), [**Uncategorized**](http://www.geomapik.com/category/uncategorized/)

[Home](http://www.geomapik.com/)  [Análisis GIS](http://www.geomapik.com/category/analisis-gis/)

3.1k

Vistas

[Compartir](http://www.facebook.com/sharer.php?u=http%3A%2F%2Fwww.geomapik.com%2Funcategorized%2Fcomo-descargar-informacion-catastral-parcelas-edificios-qgis%2F)[Twitter](https://twitter.com/intent/tweet?text=C%C3%B3mo+descargar+y+visualizar+cartograf%C3%ADa+de+Catastro+en+QGIS+via+%40%40jegtheme&url=http%3A%2F%2Fwww.geomapik.com%2Funcategorized%2Fcomo-descargar-informacion-catastral-parcelas-edificios-qgis%2F)

**Descargar cartografía catastral desde QGIS** es más fácil que nunca. Lo que en ocasiones puede parecer un auténtico quebradero de cabeza, ahora está a pocos clics de distancia.

Además, realizar la **descarga de información catastral mediante QGIS** permite ahorrar mucho tiempo, pudiendo obtener gran cantidad de datos cartográficos de Catastro rápidamente, de forma automatizada y repetida.

El hecho de poder descargarla e importarla directamente en QGIS, sin salir de su interfaz, permite visualizarla, inspeccionarla y simbolizarla al insante. Además, nos permitirá exportar mapas personalizados o realizar análisis espacial con ella fácilmente.

## Descargar cartografía catastral con QGIS

El plugin **Spanish Inspire Catastral Downloader** funciona perfectamente y supone una gran ayuda para cualquier técnico o profesional que requiera visualizar, obtener o **generar cartografía con datos de**[**Catastro**](https://www.sedecatastro.gob.es/).

Como informa su autor, Patricio Soriano, en la descripción del mismo, se trata de un **plugin para**[**QGIS**](https://qgis.org/es/site/) que:

Permite descargar datos catastrales de parcelas, edificios y direcciones. La descarga usa el servicio ATOM según la Directiva Inspire

En la web del Catastro encontramos [información acerca de los servicios ATOM](http://www.catastro.minhap.gob.es/webinspire/documentos/inspire-ATOM.pdf). Se trata de un conjunto de servicios que permiten descargar por municipios conjuntos de datos de acuerdo a la normativa INSPIRE.

Básicamente **información catastral**de : Parcelas, Direcciones y Edificios. La información se actualiza cada dos años, según se indica en la propia web del Catastro.

Lógicamente, quedan excluidos de la descarga los territorios del País Vasco y Navarra, ya que ambas Comunidades Autónomas tienen sendos servicios forales a nivel provincial.

### **Instalar el plugin Spanish Inspire Catastral Downloader**

El plugin de descarga de datos catastrales es descargable e instalable desde el propio gestor de complementos de QGIS.



Alternativamente, puede descargarse desde el [repositorio de Github](https://github.com/sigdeletras/Spanish_Inspire_Catastral_Downloader) del autor.

El plugin, una vez instalado, se muestra en el menú Complementos de QGIS además de generar una pequeña barra de herramientas con su icono desde el que acceder a su interfaz.

### **Cómo descargar cartografía catastral en QGIS**

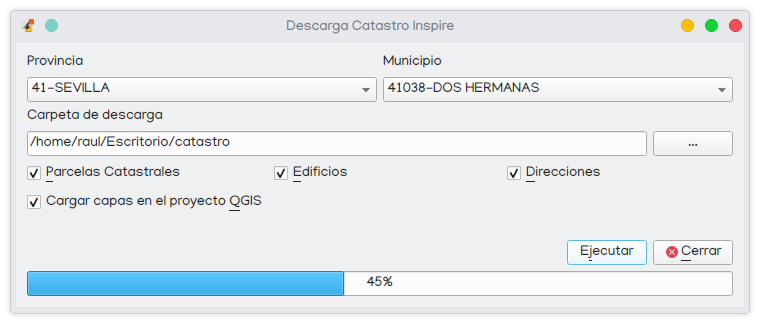
El plugin hace uso de los servicios ATOM mencionados para realizar peticiones de descarga de forma automatizada sin necesidad de recurrir al proceso de forma manual y repetitiva.

El área de descarga es siempre a nivel de municipio y de manera individualizada. Es decir, deberemos ejecutar el proceso del plugin para cada municipio del que necesitemos cartografía catastral.

Para **descargar las capas** correctamente, deberemos indicar en primer lugar la Provincia en la que se encuentre el municipio en cuestión.

Automáticamente, una vez seleccionada la Provincia, el plugin muestra la lista de municipios de la Provincia en el desplegable de la derecha. Seleccionaremos el municipio deseado.

Posteriormente, deberemos introducir una ruta a la carpeta donde deseemos almacenar los datos descargados. Como veremos, el plugin genera una carpeta con el código INE del municipio.



Deberemos indicar qué capas de información catastral queremos descargar:

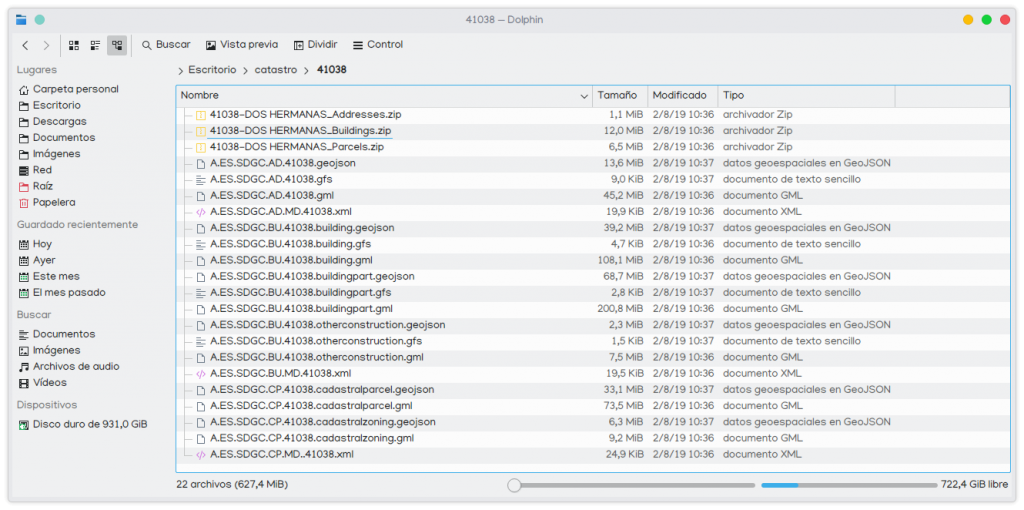
Además, podremos indicar si queremos cargar automáticamente las capas en la Tabla de Contenidos de QGIS tras la ejecución.

### Visualizar e inspeccionar la información de Catastro descargada

#### Tipo de datos y formatos descargados de Catastro

Una vez ejecutado el proceso del plugin, veremos como se ha generado la carpeta con el código del municipio en el directorio indicado.

Accediendo a la carpeta, comprobamos que se han descargado distintos archivos.

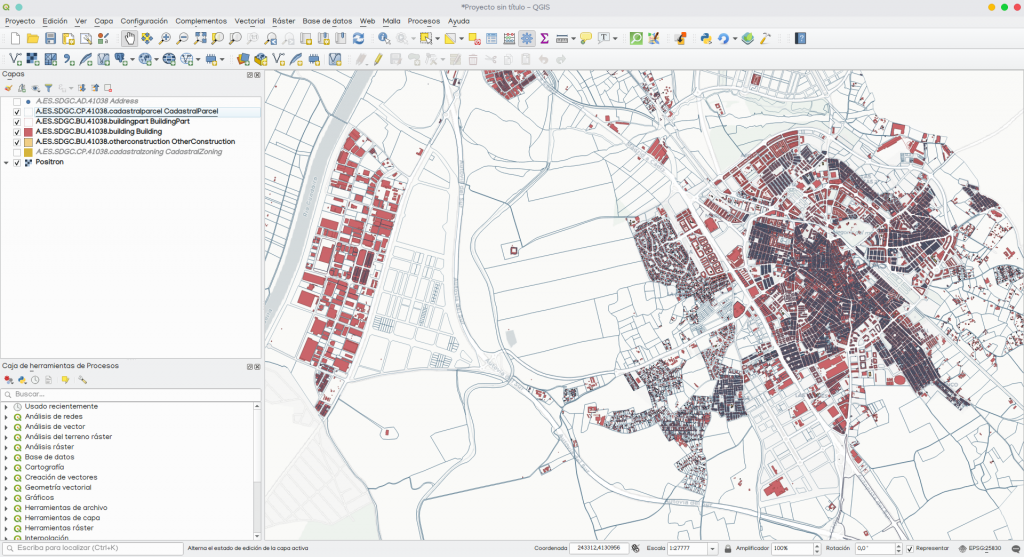


Entre ellos encontramos los ZIP de cada una de las capas marcadas para la descarga y los archivos ya descomprimidos en la carpeta. Cada una de las capas con las geometrías se obtiene en formato GeoJSON, acompañadas de los archivos GML y GFS respectivos

El archivo GML Geography Markup Language) corresponde a un formato para el transporte, intercambio y almacenamiento de información geográfica. Catastro lo usa para mostrar información parcelaria y de edificios de manera ordenada. Si quieres más información [aquí](https://www.certicalia.com/gml-parcela-catastral/que-es-el-gml-parcela-catastral) puedes leer algo más acerca de este tipo de archivo y el por qué de su uso.

No obstante, el archivo que nos interesa para visualizar y operar en QGIS es el [**archivo GeoJSON**](https://geojson.org/), un formato abierto para almacenar información y atributos tanto espaciales como no espaciales de objetos.

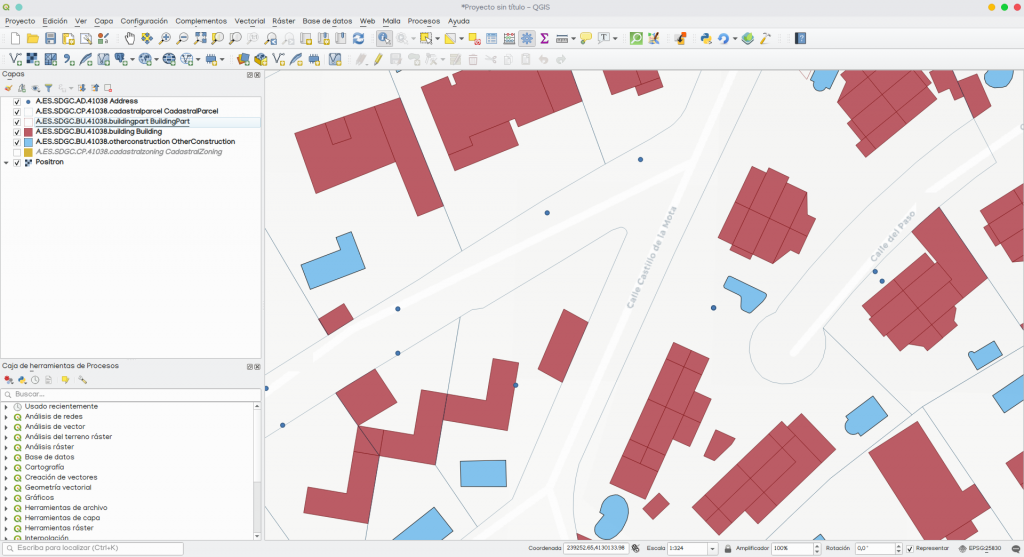
Su uso está muy extendido, especialmente para el intercambio y la visualización de información geográfica en la web. Destaca su ligereza y facilidad de edición en múltiples entornos y programas.



#### Información catastral de cada archivo

En la Tabla de Contenidos de QGIS vemos que se muestran las 6 capas GeoJSON, 1 de ellas de tipo puntual y otras 5 poligonales. Las capas disponibles son:

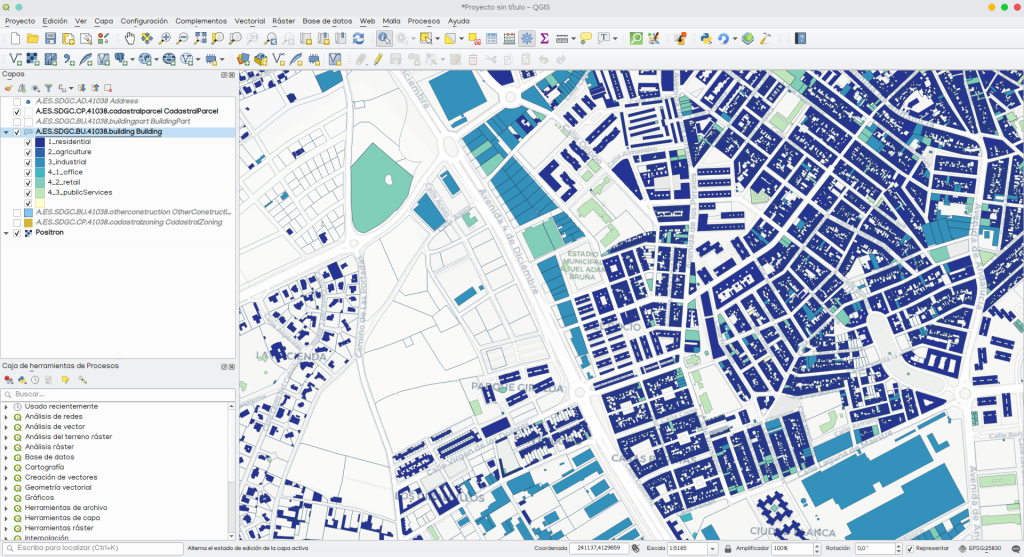
* Address. Puntos con las direcciones. Diferencia entre entradas o portales y parcelas.
* Building. Geometrías de edificios. Cada edificio tiene asociados múltiples campos o atributos de información como: identificador GML, fecha de construcción y finalización, estado de conservación, uso al que se destina el edificio, número de unidades de edificación, superficie en m2…
* Building Part. Capa que contiene las partes en las que se divide cada uno de los edificios del municipio descargado. Destaca como información: el identificador GML, la altura en metros de cada parte y el número de pisos tanto por encima como bajo el nivel de la superficie.
* Other Construction. Capa correspondiente a otros elementos constructivos asociados a los edificios. Habitualmente son piscinas u otros elementos al aire libre.
* Cadastral Parcel. Contiene la geometría y la información de cada parcela de Catastro descargada del municipio. Como atributos cabe destacar el identificador GML, la referencia catastral y la superficie en m2 de la parcela.
* Cadastral Zoning. Contiene las Zonas Catastrales del municipio.



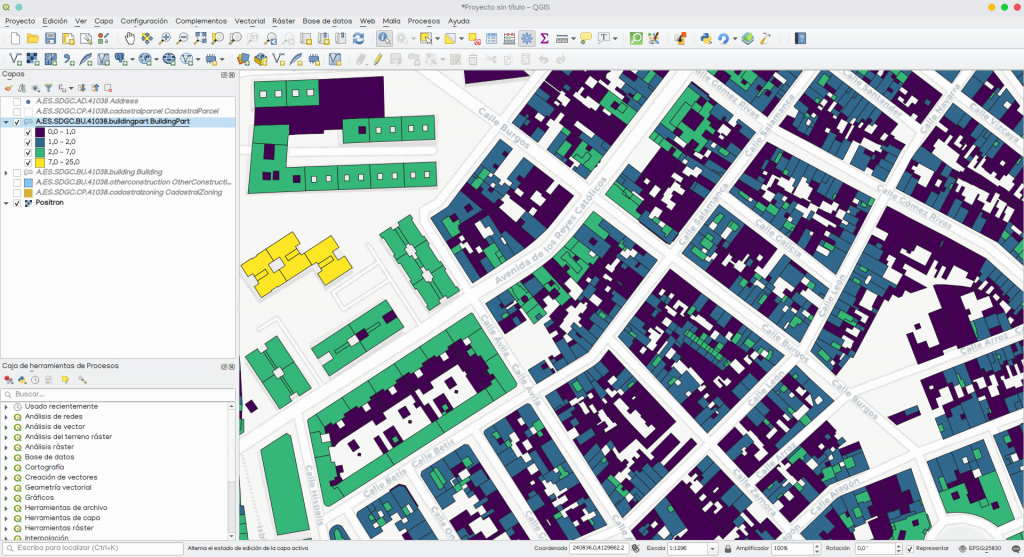
#### Visualización de información por atributos de capa

Puede resultarnos útil realizar múltiples visualizaciones utilizando los distintos campos disponibles.

Un ejemplo puede ser tratar de visualizar el uso al que se destinan los edificios descargados desde Catastro, haciendo uso de la simbología categorizada de la capa en base al campo «currentUse».



También podremos visualizar las partes de edificios según su altura sobre el suelo, para conocer dónde se localizan rápidamente los edificios de mayor altura o conocer la tipología urbana de los distintos barrios de una ciudad.



Evidentemente, no sólo podemos jugar con la simbología de las capas. Existen multitud de análisis espaciales y geométricos en QGIS para obtener información relevante de datos de uso extensivo como los de Catastro. Además, podremos exportar las capas modificadas o generadas y plasmar las visualizaciones en vistas de mapa fácilmente.

Para un análisis más avanzado y automatizado en base a información catastral, recomendamos echar un vistazo a nuestroartículo de [**Goolzoom**: una plataforma para visualizar y obtener insights en base a cartografía y datos catastrales.](http://www.geomapik.com/recursos-gis/goolzom-y-catastro-pro-el-visor-de-catastro-mejorado/)

Esperemos que este artículo haya resultado útil y animamos a los usuarios a utilizar esta genial herramienta para descargar información catastral rápidamente y ser visualizada con QGIS.