**Functionalities of the DDBB:**

# Initial Requirements:

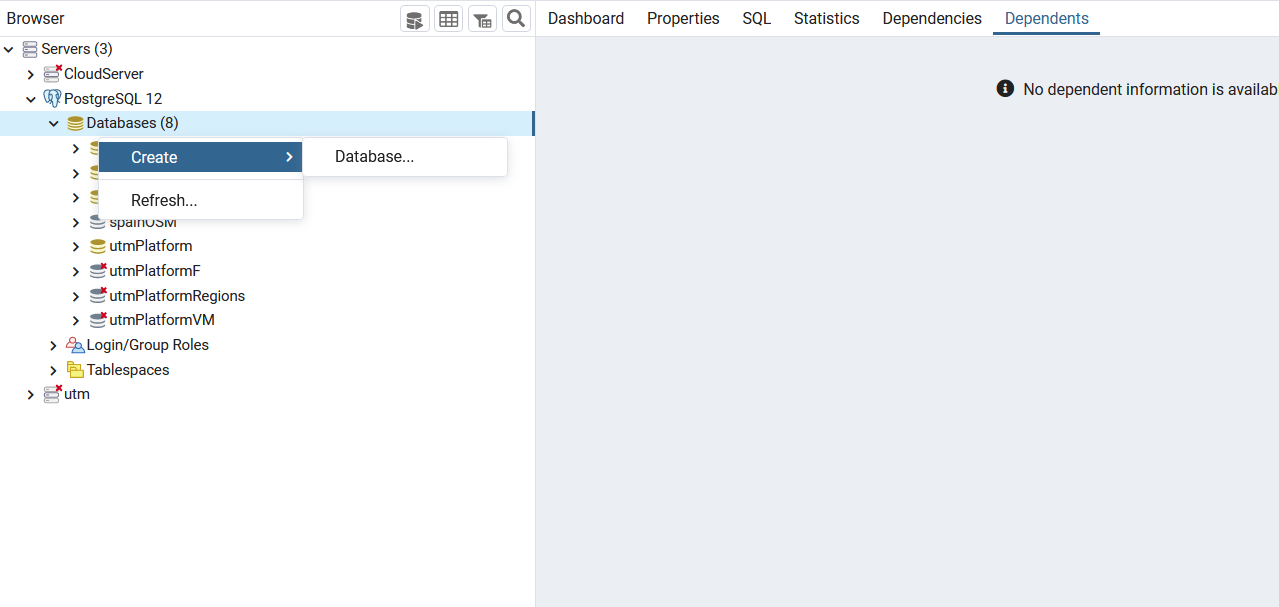
There are some requirements that must be fulfilled to execute the next sections.

These requirements are list next:

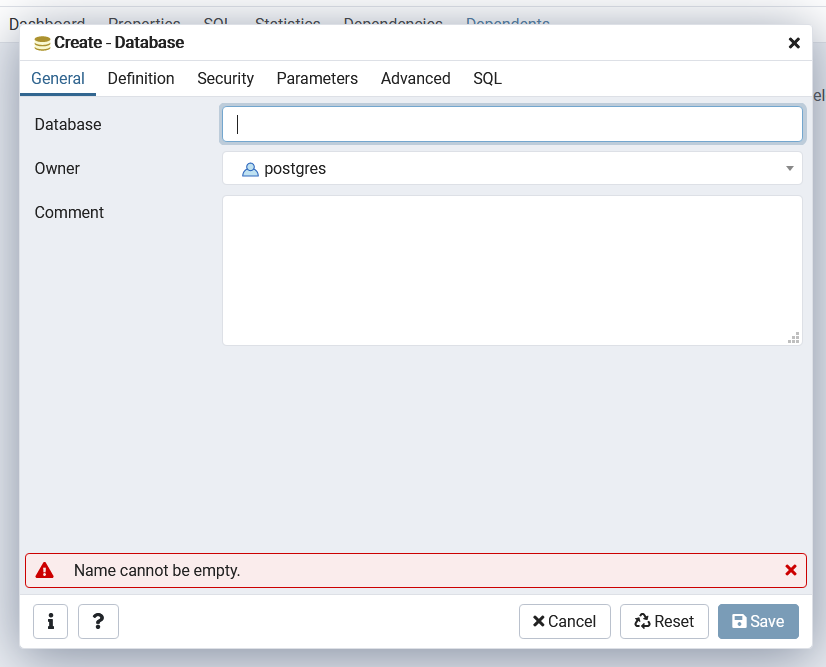
* Python 3.9.4
* psql (PostgreSQL) 12.4
* pgAdmin 4 v4
* Visual Studio Code (or another code editor)

# Connection to DDBB:

First step is to create the ddbb in the server. For this step it is needed to go to pgAdmin and right click on Databases. Then go to Create/Database…



On the property’s specification the name of the DDBB must be defined.



The class implemented to the connection to the DDBB is the pgConnection and the values that must be changed depending into the DDBB are the following:

* HOST = "127.0.0.1"
* PORT = "5432"
* USER = "docker"
* PASSWORD = "docker"
* DB = "gis5"

The values inside “ “ are examples of how the properties have to be written.

To do the connection in the utm schema the next steps have to be followed:

1. Open environment\utm and schema\pgConnection.py
2. Change the values for our DDBB
3. Open environment\utm\schema\00\_main.py
4. Change the database property.

**NOTE**: All the environment/utm schema is required to execute the script.

To do the connection in the osm schema the next steps have to be followed:

1. Open environment\osm\pgConnection.py
2. Change the values for our DDBB
3. Open environment\osm\00\_main.py
4. Change the database property.

**NOTE**: All the environment/utm schema is required to execute the script.

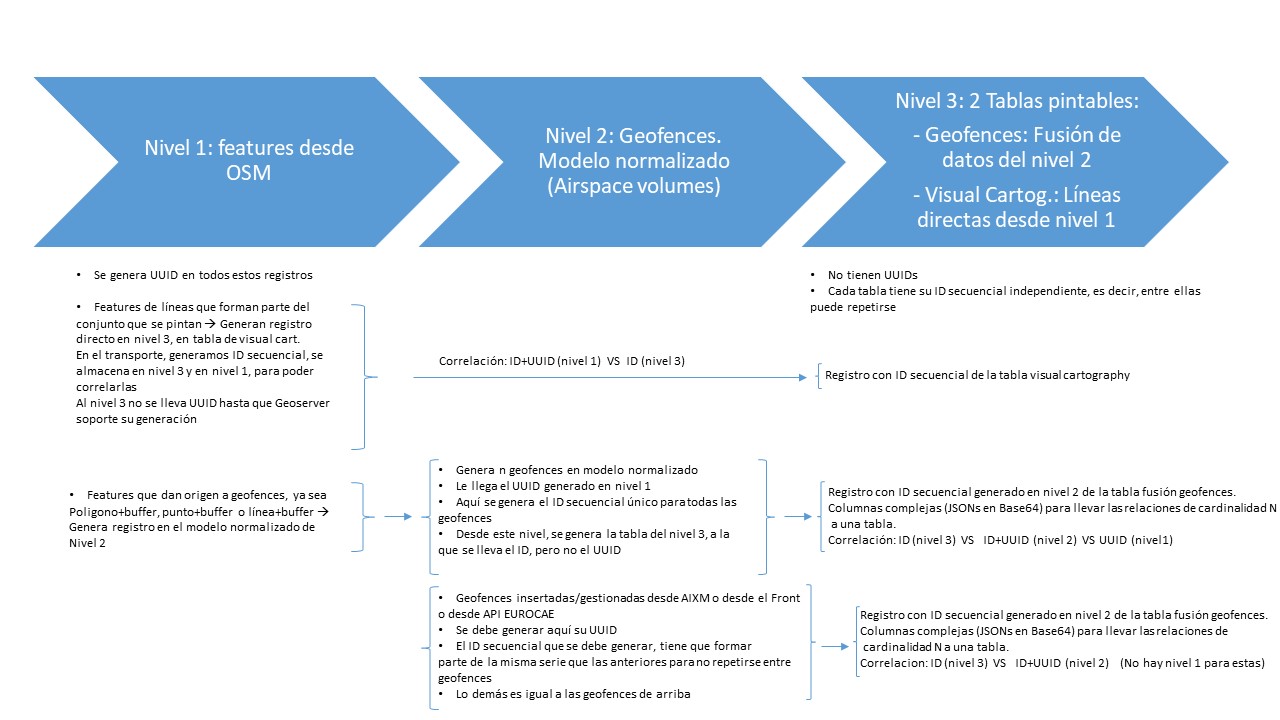
# Create the DDBB schema:

The DDBB is divided on 3 different levels. The first level is the cartography level where the data is downloaded from different services, in this version from OSM. In this level the features are store and some of its information too, such as email, name or idcode. Moreover, in this level, the information related to the open street map levels and its correspondence with the features are stored too.

The second level is the eurocae model for geofences and in this level the geometry data, time period restrictions, metadata, authority information of the geofences, the restrictions and type of geofence layer is stored. This geofences will be created by three different sources. The first one is the first level, in other words, the volume is created from a feature downloaded from open street maps. The second one is an aeronautical xml provided by UTM. The third one is based on geofences not dependents of level 1, this geofences can be created from the webpage from a user or from an api. In this level the relations between tables are not only one to N, but also N to M. Because of this fact, complex columns have been introduced on level three.

The third level is based in fusion tables, one for geofences and one for visual cartography. These tables have been designed to exploit the DDBB information by WFST from the webpage. This level has as objective ease the extraction of information to the map and the introduction of new geofences or the update of old geofences.

A schema of all these levels is shown next.



Apart from these three levels, there are some tables, where layers to create a basic geofence cartography and its information are stored, and some others to store the administrative region levels of Spain.

In order to create this DDBB schema is needed to follow the next steps:

1. Go to 00\_main.py from environments/ddbb
2. Execute the python script changing the variable execCase to 1 in first place, 2 in second and finally to 3.
3. Go to environtment/osm/00\_main.py and define execCase variable by 4.
4. Execute the previous script

**NOTE**: All the environment/ddbb schema is required to execute the script.

# Create a basic cartography of Spain:

This functionality is based on the downloading of some cartography layers from OpenStreetMaps. This action can be done in two main different branches.

The first branch is downloading the data from the overpass api, however there is a big limitation on it. The api can only download a size of data on a specific time and can only do a number of calls in an specific time. For this reason, another option has been implemented.

This second branch is to download the data from OSM by GEOFABRIK website and introduce it to a database.

Although, we have finally use the second option, both of them have been implemented on code and are going to be explained next.

## Download OSM data by Overpass API:

The functions to allow this functionality have been implemented in execCase number 1 and 2 in the 00\_functionalities.py script. This execution case request the cartography layers information to the database and the class/subclass corresponding for OpenStreetMaps. After this request each one of the layers are download and stored in a geojson. In order to do this a python class is defined (Basic\_layers\_Download - A06DownloadBasicLayers.py). The function extracts all the information for each basic layer from geofence\_basic\_layers table and then get its corresponding types and subtypes from osm\_basic\_layers. A loop is done to get this information for each layer and the function to download the data from OSM is execute (one\_subtype\_osm or one\_group\_osm from A03Functionalities.py).

Next the geojson are read and stored on the database. In order to do this a similar process as 4.1 is done. First the function get all the information of the geofences\_basic\_layers and osm\_basic\_layers tables. After that, it do a loop and read for each layer its geojsons and introduce the data stored in this geojson into the database, (this function is store in the class Basic\_layes -A04BasicLayers.py).

To execute this option, follow nest steps:

1. Go to 00\_functionalities.py located on environments/osm folder.
2. Define execCase variable to 1
3. Execute the file
4. When process ends, change execCase to 2.
5. Execute the file again

## Download OSM data by GEOFABRIK website:

This second way to download the spain cartography is based on download a file from geofabrik website and then process it and insert the information interested for us from it to our DDBB.

The scripts used for it are based on the ones used by the Overpass API way but changing the last function (B01InsertfromOSMddbb.py instead of B01Insert\_feature.py). We have included two different functions one for lines and another one for polygons and points. This fact is to ease the process of data.

To execute the case follow the next steps:

1. Go to <http://download.geofabrik.de/> and download the Spain file (Europe/spain) in format .osm.pbf
2. Go to <http://download.geofabrik.de/> and download the Spain file (Africa/Canary Islands) in format .osm.pbf
3. Create a ddbb with the postgis extension (you can execute this command on pgadmin console: CREATE EXTENSION IF NOT EXISTS postgis; )
4. Add hstore extension to ddbb (you can execute this command on pgadmin console: CREATE EXTENSION IF NOT EXISTS hstore; )
5. Download osm2psql from <https://osm2pgsql.org/doc/install.html#installing-on-windows> // <https://osm2pgsql.org/download/windows/>
6. Unzip the release zip and add the downloaded geofabric file to the folder.
7. Then execute: ”osm2pgsql -d spainprueba -U postgres -P 5432 -S default.style --hstore -C 2500 -H localhost -W spain-latest.osm.pbf “
8. Then execute: ”osm2pgsql -d spainprueba -U postgres -P 5432 -S default.style --hstore -C 2500 -H localhost -W canary-islands-latest.osm.pbf “
9. Go to 00\_main.py in environments/osm
10. Define the execCase to 1 then 2, 3 and finally 5.
11. Execute file.

# Functionalities to download individual layers

## Overpass API to download individual layers:

For this first main branch to download OSM data, three different ways have been implemented.

The first way is to download one unique subtype of OSM, the second to download a one unique group (with all its subtypes) of OSM and the last way is to download some subtypes of a group of OSM. Depending on the way chosen the variables have to be defined in a way or in another.

The variables to be defined are the following ones:

* Osmtype: a string with the name of the group of OSM to download
* Osmsubtype: an array of strings with the name of the different subtypes name to be downloaded.
* Coords: if we want to download the features of a square area, we have to introduce the longitude and latitude of two opposite vertices of the square. And if we want to download all the features in spain we introduce [].
* NumberFunc: 1-one subtype; 2-one group; 3-some subtypes of one group.
* Doc: the name of the geojson where the data downloaded is going to be stored. If the download is for a group or some features of a group the name of this document would be this variable plus the name of the group and the subgroup of each subtype downloaded (one document for each subtype).
* Properties: an array with all the properties to be store in string format.

To execute this functionality, follow the next steps:

1. Go to environment/osm and then to 00\_main.py.
2. Change execCase to 4.
3. Check that the point 6/6.1 is uncommented and 6/6.2 is commented.
4. Execute the main script.

**NOTE**: All the environment/osm schema is required to execute the script.

The scripts used in this functionality are the A03Functionalities.py, B03Osm\_conection.py and A01Test.py. In the first script the functions to download a group or a unique subtype are defined. In the second script the functions to create the query and the connection to OSM is defined, three different functions depending if the query is for an area (defined by coordinates), if it is for Spain or if it is for a ccaa. The last script is created to define a function to process all the data downloaded and store the important one on a geojson.

## Introduction of individual OSM layers to DDBB:

This functionality is based on read a geojson and insert the information of the features stored on it into the DDBB. These features have to be store in a cartography layer existing or in a new one. So, the first thing done is to check if the name of the layer is in the data base or not, if it is not insert it into the layers\_cartography table, and then get the layer id. After get the id of the layer the introduction of the feature data into the DDBB is done. There are three options as in the previous section. These options are based on the number of subtypes to introduce, if is only a subtype the functionality one\_subtype is executed, if it is one group the functionality one\_group is executed and if it is some groups of a group the some\_subtype\_1group is executed. All these functionalities are in the script A03Functionalities.py. When one of these functionalities is execute the function insert\_subtype (B01Insert\_feature.py) is execute and in this function a processing and distribution of the data is insert in to the DDBB.

The information needed is the one included in section 3 plus the following one:

Name: the name of the layer.

Goem\_type\_id: the priority geometry type

Description\_layer: the description of the layer

In order to execute this functionality, the next steps have to be followed:

1. Open environment/osm and 00\_functionalities.py.
2. Change execCase to 4.
3. Check that the point 6/6.2 is uncommented and 6/6.1 is commented.
4. Execute the main script.

**NOTE**: All the environment/osm schema is required to execute the script.

An example of the properties for section 2 and 4 is show next:

* osmtype='amenity'
* osmsubtype=['university']
* coords=['40.304740', '-4.040336', '40.619906','-3.363406']
* doc='schools\_university\_amenity'
* properties=['name', 'oneway']
* name='hospital'
* geom\_type\_id= 3
* description\_layer='none'
* name\_view=name
* numberFunc=1 #1:one subtype; 2:one group; 3:some subtypes of one group

**NOTE:** both functionalities can be execute in one pass if we only execute the file once and both 6/6.1 and 6/6.2 are uncommented.

# Insert Spain Regions tables

A script in python has been implemented to generate the spain cartography regions. A schema inside the database must be created and named “Spain”. In this schema 4 tables are created, country, ccaa, provincias, municipios. A readme is in environtments/spain to do the process. All this information is download from IGN and insert into the schema.

To get the schema without download the information from IGN, a sql script is in environment/spain/sql and can be used to generate the schema directly. By using the next command on cmd:

* Connect to ddbb: “Psql -U ‘user’ -h ‘host’ -p ‘port’ ‘databasename’
* Go to Schema spain: “SET search\_path TO spain;”
* Import the schema: “\i ‘directory/spainschema.sql’ ”

To get all this information from schema Spain to the public one, next steps must be followed:

1. Open environtments/spain and 00\_main.py script.
2. Check that all the script is uncommented
3. Run the script.

Note: Please refer document “Procesamiento de la cartografía nacional.docx”

# Upload non-aeronautical layers

## Creation of Volume layers for all basic cartography layers

Once the cartography level is fulfilled, its corresponding geofences must be created. Not all cartography layers generate geofences, the geofence\_basic\_layer table was created to define which layers generate geofences and which are their properties.

A python class, BasicLayersVolumes (A04AddVolumeBasicLayers.py), has been implemented to get all the information needed to create a geofence for each layer. Then a second class is called to get all the features in a layer and generate the corresponding geofence according with the parameters extracted on first class, they are defined on geofences\_basic\_layer table. This second class is FillVolumeFeature (A03VolumeFeature.py). To create the geofences the FillVolumeFeature class call a third class, CommonVolumes (E01CommonVolumes.py), which function is to create the geofence and insert it in to the DDBB.

This distribution has been done in this way to reuse the classes to different functionalities, such as the creation of the aeronautical geofences layers.

To execute this functionality, the next steps must be followed:

1. Open environment/utm and schema/00\_main.py.
2. Change execCase to 1
3. Execute the script

**NOTE**: All the environment/utm schema is required to execute the script.

## Creation of volume layer with custom parameters

A functionality to create a volume layer based on basic cartography but with different properties than the ones in the geofence\_basic\_layer table, has been implemented. This functionality is based on the same classes schema than the one used on previous section (section 5.1), classes FillVolumeFeature and CommonVolumes.

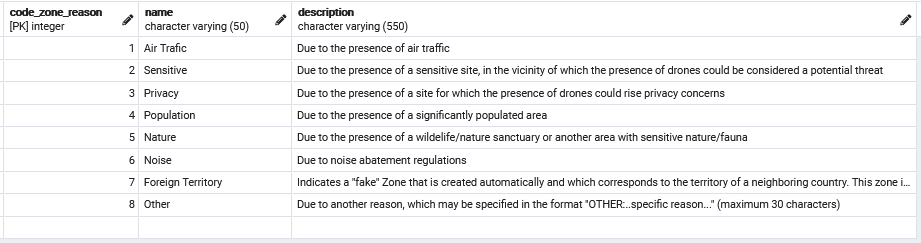
To execute this functionality, the next steps must be followed:

1. Open environment/utm and schema/00\_main.py.
2. Change execCase to 3
3. Define the properties name, buffer, code\_zone, code\_uspace
4. Execute the script

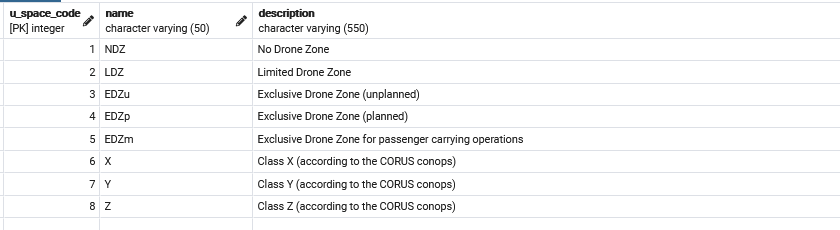
**NOTE**: All the environment/osm schema is required to execute the script.

An example of the properties required:

* Name='aerodromes '
  + Is the name of the basic cartography layer
* Buffer=8000
  + Is the buffer in meters
* Code\_zone=1
  + Is the code zone type id. The list of ids are shown next.



* Code\_uspace=1
  + Is the code uspace class type id. The list of ids are shown next.



# Upload aeronautical layers

This section explains how to create the geofences related to the aeronautical layers. All the information of this geofences are stored on a json provided by client. A class has been implemented to execute the process, AeronauticsVolumes (A05AddVolumesAeronautics.py). This functionality gets the information and create the geofecens calling the previous functions.

Its important to highlight that the json provided by client has geofences from different layers. Some of them were like some layers which already were in the ddbb. These last ones have been integrated in the layer already created.

* ControlledAirspace: 697 geofences. It was created a new layer.
* TemporartFlightRestriction': 191 geofences. It was created a new layer.
* SpecialUseAirspace: 5 geofences. It was created a new layer.
* Emergency: 4 geofences. It was created a new layer.
* School: 2 geofences. They were included in the shools\_volume layer.
* DenselyPopulatedRegion: 1 geofence. It was created a new layer.

These geofences have been created with the properties defined in the json named previously. It is important to say that not all the properties stored on the json were fill or some of them were wrong. This functionality tries to get all the right information but there is a lot of missing information. If it is needed to actualize the information the json has to be actualize and this section has to be executed again. All the geofences on this layer would be deleted and they would be added again if they are on the actualized json.

To execute this functionality, the next steps must be followed:

1. Open environment/utm and schema/00\_main.py
2. Change ExecCase to 2
3. Execute main script

**NOTE**: All the environment/utm schema is required to execute the script.

# Clean geofences duplicated on similar layers.

There are some geofences equivalent in the same layer. This fact is because when a feature is download from openstreetmaps sometimes it is duplicated with minimal differences or there is one which include others inside it. An example to understand this, is the case of Ciudad Universitaria de Madrid. In this case openstreetmaps has a feature for each one of the schools belonging to the universitary campus, but at the same time a global feature with the size of the campus is included in openstreetmap too. For this reason, a clean functionality has been implemented.

The functionality compares geometries in the same layer and delete the ones that are inside of another o has a big percentage of similarity, 98%.

To execute this functionality a class has been implemented, Duplication (A07CheckDuplication.py). And to execute it, next steps must be followed.

1. Open environment/utm and schema/00\_main.py
2. Change ExecCase to 5
3. Execute the script.

**NOTE**: All the environment/utm schema is required to execute the script.

# Add geofences with all restrictions.

A functionality to add a geofence with all type of restrictions has been implemented to have the possibility to do tests and modifications with it. This geofences are added to a layer named sensitive\_volume and all the geofence inside them are false geofences. In this way some tests can be done and then they can be deleted easily.

A class have been implemented, Geofence\_ALLR (A09GeofenceALL.py). This class is based on create a geofences with all the possibilities of properties with the objective of get a xml to defined the standard format of the geofence information. This format is done to know how front has to give us the information to introduced it into the DDBB and to defined it for a future api with the information can to be given by other entity or client.

To execute it follow the next steps:

1. Open environment/utm and schema/00\_main.py
2. Change ExecCase to 4
3. Execute the script

**NOTE**: All the environment/utm schema is required to execute the script.

# Theorical concepts of the DDBB

Some concepts related to the ddbb must be explained to understand the working of this model.

## Fusion tables

As it was explained on section 1, the model have three different levels, and the fusion tables is the third one, the one more external to the model. This fusion tables are the ones exposed to geoserver and for this reason are the ones which would be attack by the web platform to create, modify, or delete geofences.

The geofences table is the one which stores all the information related to each geofence in a unique row, if any property of this row is modify the information is translate to the global model to actualize the corresponding tables.

The relations between them have been defined by some triggers and procedures. When a delete update or insert in the fusion table (called geofences) is done, a trigger is executed automatically and all the information is sent to the EUROCAE schema, having in this way an update of both schemas.

In the other way, when a geofence is created in the EUROCAE schema a procedure have to be execute and the information is store in the fusion table too.

There is not only a table fusion for the geofence, but also there is one for the visual cartography where all the information of the visual cartography layers is stored in order to allow the map to exploit the information only attacking this table.

## Radius and Center Properties

Two properties have been defined as properties of a geofence as complementary properties of the geometry. These properties are the radius and the center of a geometry. The reason of this properties is to give the possibility of modifying circles and runners’ geometries in the map.

When a geometry is created, there are three different types of geometries available, circles runners and polygons. The creation and modification of the geometry in the map would be different depending on the type of the origin geometry. If it is a circle, the geofence would be defined by a point and a radius. If the feature is a runner, the geofence would be defined by a line and a buffer around it. Finally, if the feature is a polygon the geometry would be defined by a sequence of points which define its perimeter.

The center column is a GeometryCollection, geometrical type of postgis and the radius is a number. This peculiar geometry type for the center was defined to able the possibility of introducing in it a point or a line depending on the origin of the geometry.

Some rules have been defined to interpret this information.

* POINT: The point is stored on column center and the radius/buffer is stored on the radius column as a positive value.
* LINE: The line is stored on the column center and the buffer is stored on the radius column as a negative value.
* POLYGON: Only the center of the polygon is stored, the radius column would be empty.

The radius column would show which type of definition is (positive value for circles, negative for lines with buffers and null for polygons).

## Complex Columns

As we said in the first section the relations of the DDBB schema are not only 1 to N but also N to M, and this has as a result the necessity of complex columns in order to get all the information in a unique table that is the one, which would be attack from the map. These columns are composed by a json encoded in base64. To obtain this json some procedures have been implemented, one for each complex column (time\_period, restrictions, code\_uspace\_class\_type, code\_zone\_reason, volume\_limitation, authorityfrom and notificationto). In these procedures some queries are executed to get all the information related to the column and transform it into the json. When the json is created, the procedure encoded it and update the corresponding column of de row evaluated.