QRH : Base de données

Damien GABRIEL

Version : 1 | Juillet 2016

Table des matières

Installation de PostgreSQL / PostGIS

Linux (Manjaro)

Pour installer PostgreSQL et PostGIS sur Manjaro, il faut installer :

```
postgresql
postgresql-docs
postgis
```

On peut aussi installer les paquets suivants pour améliorer le travail avec les bases de données :

```
pgadmin3
umbrello
libpqxx
psqlodbc
python-psycopg2
lua-sql-posqtgres
php-pgsql
haskell-postgresql-libpq
haskell-postgresql-binary
```

Initialisation du Cluster

```
sudo -i -u postgres
initdb --local $LANG -E UTF8 -D '/var/lib/postgres/data'
```

Démarrage et arrêt de PostgreSQL

```
systemctl start postgresql
systemctl stop postgresql
```

Pour que PostgreSQL soit actif à tous les redemarrages on peut aussi utiliser :

```
systemctl enable postgresql
```

Ajout des utilisateurs

Par défaut, l'installation de PostgreSQL n'a que l'utilisateur *postgres*. C'est le seul utilisateur qui peut se connecter a *psql*.

La première étape pour véritablement utiliser PostgreSQL est donc de créer de nouveau utilisateur.

Ici on va créer un utilisateur *lambda* autorisé à créer des bases.

On commence par se connecter à psql : [1]

1. La démarche peut être légèrement différente en fonction des *OS* utilisés. Sous Windows, je pense qu'il suffit de taper directement *psql*

```
su - postgres
psql
```

Une fois dans le shell de psql :

```
CREATE USER lambda;
ALTER ROLE lambda WITH CREATEDB;
ALTER ROLE lambda WITH ENCRYPTED PASSWORD 'omega';
```

Création d'une base de donnée

Pour créer une base de donnée on utilise :

```
created *nom_de_la_base*
```

On peut alors ce connecter avec l'utilisateur *lambda* à la base *db* en utilisant :

```
psql -U lambda -d bd
```

Le shell de PostgreSQL doit affcher sur la première ligne le nom de la base, donc ici *db*.

Utilisation de PostgreSQL et PostGIS

Création du base de données spatiale

Pour créer une base de données spatiale avec PostGIS il faut utiliser les types *geography* si on veut faire des calculs précis (ie sur le geoïde en WGS84).

Pour utiliser PostGIS il faut créer l'extension *postgis* dans la base de données

```
CREATE TABLE demoPoint(
        ptID serial primary key,
        nom character varying(80),
        coordo geography
);
```

Insertion de données

```
INSERT INTO demoPoint(nom, coordo)
VALUES ('LFPO', ST_MAKEPOINT(2.37958, 48.72328));

INSERT INTO demoPoint(nom, coordo)
VALUES ('LFML', ST_MAKEPOINT(5.21500, 43.43667));

INSERT INTO demoPoint(nom, coordo)
VALUES ('LFOR', ST_MAKEPOINT(1.52389, 48.45889));
```

ATTENTION Comme le montre les commandes au-dessus, l'ordre des coordonnées doit d'abord être **E/W** PUIS **N/S**

Calcul de navigation

La fonction **ST_AZIMUTH** permet de calculer des caps en dégrés.

La fonction **ST_DISTANCE** permet de calculer des distances en **mètres**.

Dans tous les cas il faut **faire attention aux modèles utilisés !**. Par exemple, pour de la navigation aérienne il est nécessaire d'être en WGS84 (modèle du GPS).

```
SELECT ST_DISTANCE(c1.coordo, c2.coordo) / 1000 as "Distance (km)",
    CASE WHEN degrees(ST_AZIMUTH(c1.coordo, c2.coordo)) < 0 THEN degrees(ST_AZIMUTH(c1.coord
    ELSE
    degrees(ST_AZIMUTH(c1.coordo, c2.coordo))
    END AS "Cap"
FROM demoPoint c1, demoPoint c2
WHERE c1.nom = 'LFML' AND c2.nom = 'LFPO';
```

Ci-dessous le résultat de la requête précédente :

```
  Distance (km)  |       Cap
-----------------+-----------------
 627.09856039227 | 340.544493840961
(1 row)
```

Dans l'autre sens Paris Orly (LFPO) vers Marseille Provence (LFML) on obtient bien le réciproque :

```
  Distance (km)  |       Cap
-----------------+-----------------
 627.09856039227 | 158.49973504052
(1 row)
```

Requête spatiales

Affichage des coordonnées sous forme de texte

```
SELECT ST_AsText(coordo) from demoPoint where nom='LFOR';
```

Le résultat de la requête est :

```
        st_astext
-------------------------
 POINT(1.52389 48.45889)
(1 row)
```

Trouver les points les plus proches d'un point d'intérêt

```
SELECT nom, ST_AsText(coordo) as coordonnees,
        ST_DISTANCE(coordo,poi)/1000 as Distance_KM
FROM demoPoint,
        (select ST_MakePoint(2,45)::geography as poi) as poi
WHERE ST_DWithin(coordo, poi, 400 * 1000)
ORDER BY ST_Distance(coordo, poi)
LIMIT 10;
```

Le résultat de la requête est :

```
 nom  |       coordonnees       |   distance_km
------+-------------------------+-----------------
 LFML | POINT(5.215 43.43667)   | 310.09213298947
 LFOR | POINT(1.52389 48.45889) | 386.22555702069
(2 rows)
```